Process Types and Games

for MATHFIT workshop.

Kohei Honda

Edinburgh University
&
Imperial University

[illegible]

1. Introduction.

Objectives

- To pinpoint some of the key ideas of types for mobile processes from a behavioural viewpoint.
- To offer a fresh look on games semantics based on those ideas, with insight on both games and process types.

The π-calculus as a Descriptive Tool (1)

$\lambda \quad M ::= x \mid \lambda x.M \mid MN.$

$\pi \quad P ::= \Sigma \pi_i.P_i \mid P|Q \mid (\nu x)P \mid !P \mid 0.$

with $\pi ::= x(\tilde{y}) \mid \overline{x}\langle \tilde{y} \rangle.$

λ in π

$$[\![x]\!]_u \stackrel{\mathrm{def}}{=} \overline{x}\langle u \rangle.$$

$$[\![\lambda x.M]\!]_u \stackrel{\mathrm{def}}{=} u(x u').[\![M]\!]_{u'}$$

$$[\![MN]\!]_u \stackrel{\mathrm{def}}{=} (\nu f x)\left([\![M]\!]_f \mid \overline{f}\langle x u \rangle \mid [\![x = N]\!]\right)$$

with $[\![x = N]\!] \stackrel{\mathrm{def}}{=} !x(u').[\![N]\!]_{u'}.$

The π-calculus as a Descriptive Tool (2)

λ

$$\begin{tikzcd} \lambda x.M \arrow[d, squiggly] & \bullet & N \arrow[d, squiggly] \\ M[N/x] & & \end{tikzcd}$$

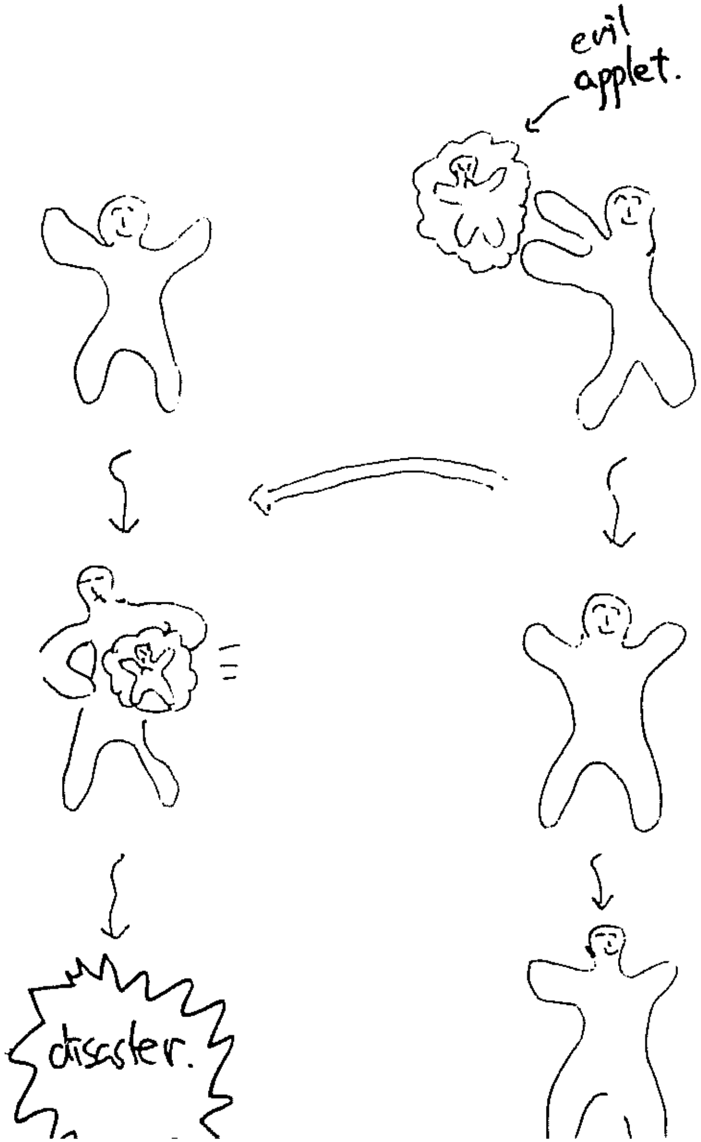
The π-calculus as a Descriptive Tool (3)

λ in π

$$\begin{tikzcd} {[\![\lambda x.M]\!]} \arrow[d, squiggly] & \bullet & {[\![N]\!]} \arrow[d, squiggly] \\ P \arrow[d, squiggly] & & Q \arrow[d, squiggly] \\ P' \arrow[d, squiggly] & & Q' \arrow[d, squiggly] \\ P'' & & Q'' \\ \vdots & & \vdots \end{tikzcd}$$

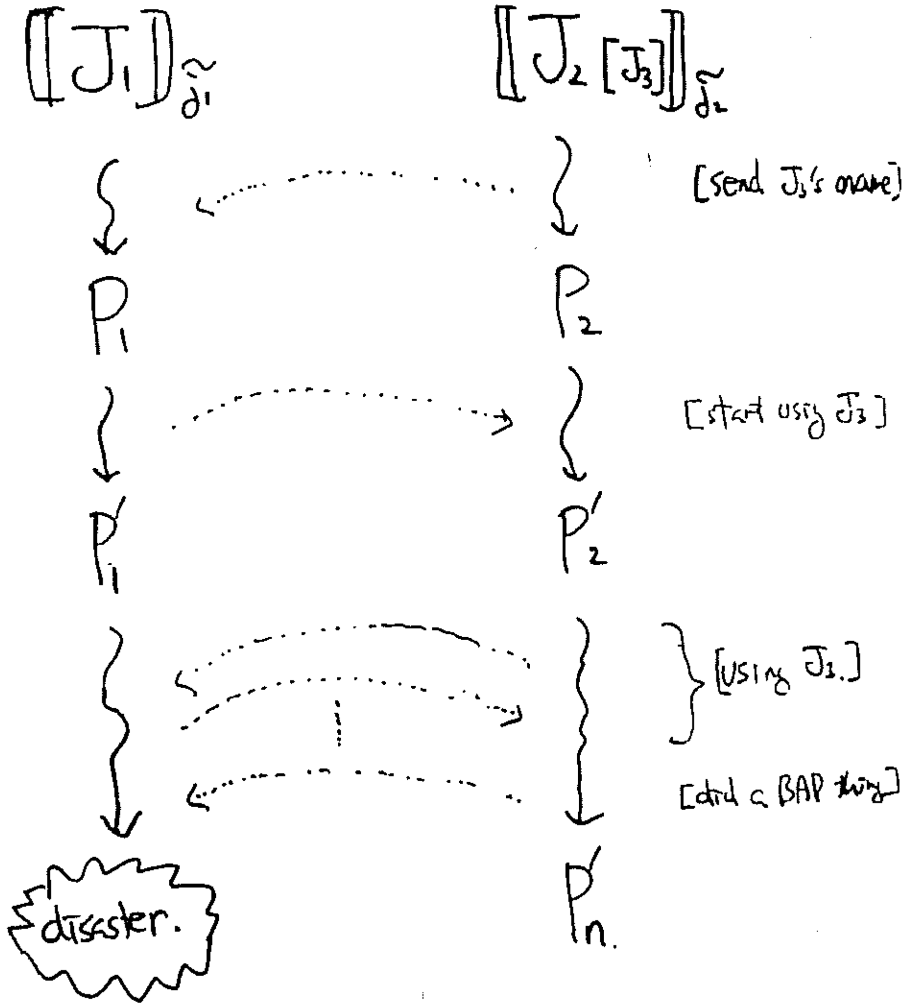
The π-calculus as a Descriptive Tool (4)

Java



The π-calculus as a Descriptive Tool (5)

Java in π



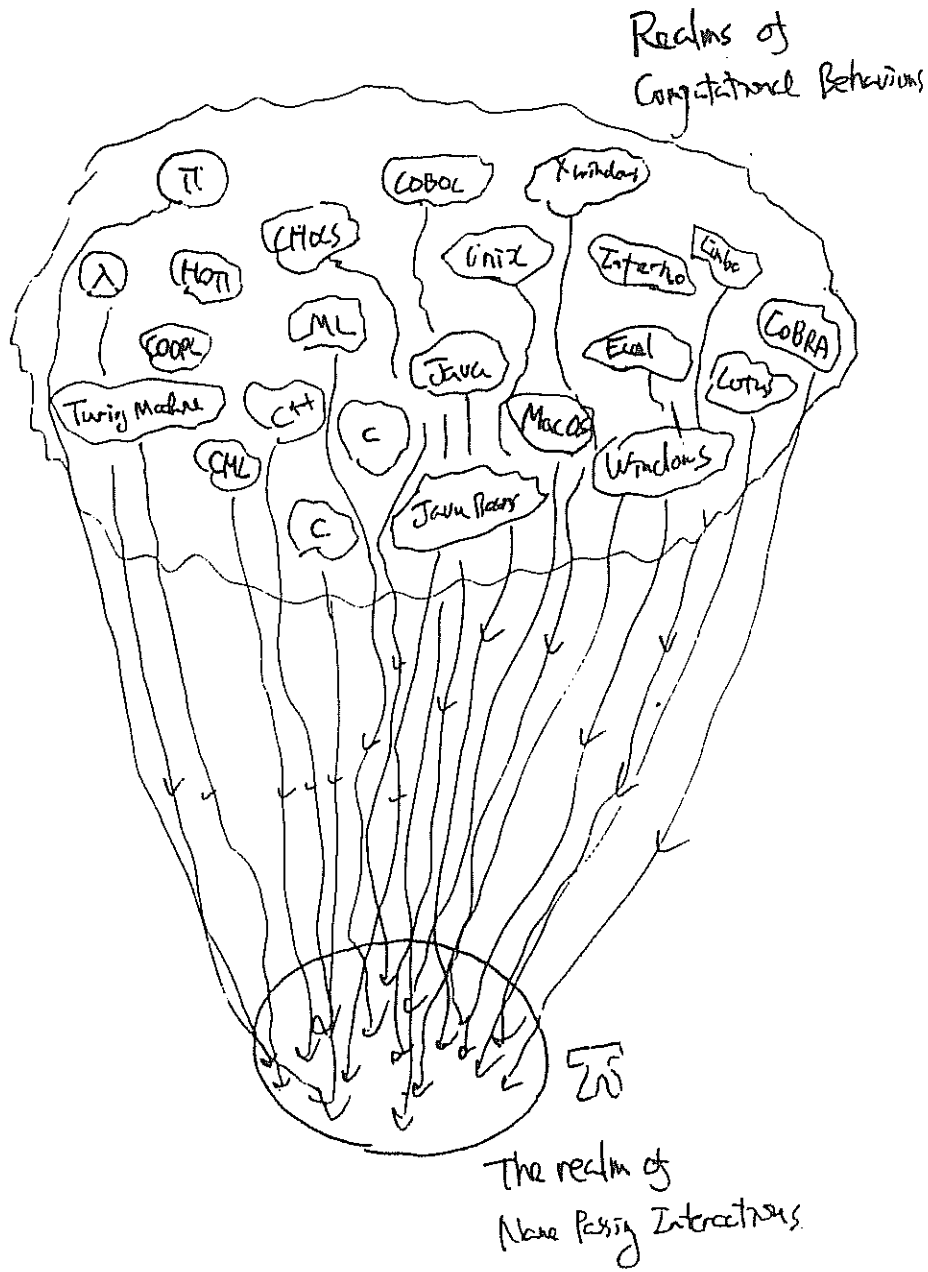
The π-calculus as a Descriptive Tool (6)

* Examples of Representable Computation.

  - λ-calculus [MPW89, Milner90, Milner92, …]
  - Concurrent Object [Walker91]
  - ω-order term passing [Sangiorgi 92]
  - Various data structures [Milner 92, …]
  - Proof Nets [Bellin and Scott 93]
  - Arbitrary "constant" interaction [HY94]
  - strategies on Games [HO95]

  ⋮

The π-calculus as a Descriptive Tool (7)

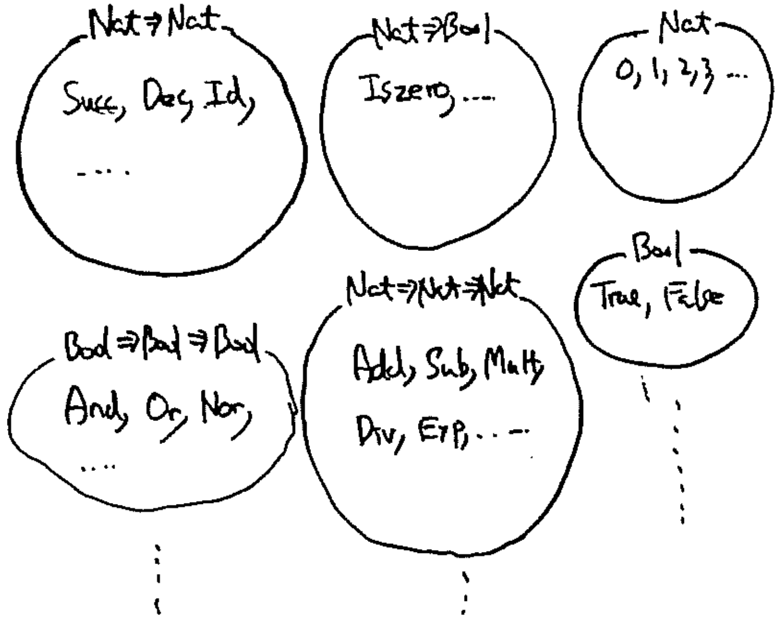


The Role of Types in π-calculus.

- (classification) How can we classify name-passing interactive behaviours, i.e. behaviours representabe in π-calculus? What classes ("types") of behaviours can we find in the calculus?

- (safety) Is this program/system in the safe (or correct, relevant, ...) classes of behaviours? Can the safety be preserved compositionally?

A Paradigm.



with operation:

function application.

$$\begin{cases} f:\alpha\Rightarrow\beta \bullet e:\alpha = f\bullet e : \beta. \\ \text{else undefined.} \end{cases}$$

ex. $Succ : Nat\Rightarrow Nat \bullet 2 : Nat = 3 : Nat$

$Succ : Nat\Rightarrow Nat \bullet True : Bool$ (undefined)

$Id : Bool\Rightarrow Bool \bullet 3 : Nat$ (undefined)

Outline of Lectures.

Lecture 1: Understanding Sorting.

* Using the most basic notion of types for π-calculus, we explore the fundamental ideas and subtleties of types for mobile processes from a behavioural viewpoint.

Lecture 2: Games from a π-calculus viewpoint.

* We study the relationship between Games Semantics and process types by showing how games arise as types for mobile processes with a non-trivial behavioural restriction.

2. Understanding Sorting (1)

π-calculus: Syntax.

- Names: $a, b, c, \ldots$ or $x, y, z, \ldots$
- Processes: $P, Q, R, \ldots$

$$P ::= \sum_i \pi_i.P_i \mid P|Q \mid (\nu x)P \mid !P \mid \emptyset.$$

where $\pi ::= x(y_1 \ldots y_n) \mid \overline{x}\langle y_1 \ldots y_n \rangle.$

- Binding: $x(\underline{y}).P$ $\quad (\nu \underline{x})P$ $\quad \boxed{fn(P) \quad \equiv_\alpha}$

- Structural Congruence $\equiv$.

(1) $P \equiv_\alpha Q \Rightarrow P \equiv Q.$

(2) Σ and $|$ are commutative and associative, with $P|\emptyset \equiv P.$

(3) $(\nu x)\emptyset \equiv \emptyset$, $\quad (\nu x y)P \equiv (\nu y x)P$,
$(\nu x)P|Q \equiv (\nu x)(P|Q) \quad x \notin fn(Q).$

(4) $!P \equiv !P|P$

Processes: Examples.

(1) $\overline{a}\langle ef \rangle$ (denoting $\overline{a}\langle ef \rangle.\emptyset$). A process which outputs "ef" via a, and does nothing any more.

(2) $a(xy).\overline{x}\langle y \rangle$. A process which receives two names and sends the second one to the first.

(3) $a(xy).\overline{x}\langle y \rangle + \overline{a}\langle ef \rangle$. This process acts either as (1) or (2).

(4) $(\nu c)\overline{a}\langle c \rangle.c(y).\overline{y}\langle e \rangle$. Sends a new name, gets a name from that new name, and sends "e" to the received name.

(5) $\overline{a}(\nu c).c(y).\overline{y}\langle e \rangle$. This is a shorthand for (4).

(6) $!a(xy).\overline{x}\langle y \rangle$. The replicated version of (2).

(7) $!\overline{a}(\nu c).c(y).\overline{y}\langle e \rangle$. A replicated version of (5), sending a new name at each activation.

Processes: Examples (cont'd).

(8) $!a(x).\,\overline{x}(\nu c).\,\overline{c}\langle e\rangle \mid \overline{a}\langle b\rangle$. This is $\equiv$-congruent to:

$!a(x).\,\overline{x}(\nu c).\,\overline{c}\langle e\rangle \mid a(x).\,\overline{x}(\nu c').\,\overline{c'}\langle e\rangle \mid \overline{a}\langle b\rangle$.

(9) $\overline{a}\langle 1,2\rangle \mid a(x,y).\,\overline{b}\langle x+y\rangle$. We often assume constants like natural numbers are inside $\mathcal{N}$, and use such simple expressions like $x+y$

Π-calculus: Transition.

- Actions: $\alpha, \beta, \gamma, \ldots$

$$\alpha ::= x\langle\tilde{y}\rangle \mid \overline{x}\langle \nu\{\tilde{z}\}.\,\tilde{y}\rangle \mid \tau.$$

$\boxed{\text{Can be omitt... when } \{\tilde{z}\}=\phi}$

with $\{\tilde{z}\} \subseteq \{\tilde{y}\}$. We simply write $\overline{x}\langle\nu\tilde{z}.\tilde{y}\rangle$ for $\overline{x}\langle\nu\{\tilde{z}\}.\tilde{y}\rangle$.

- Transition: $P \xrightarrow{\alpha} Q$, up to $\equiv$.

(IN) $\sum \pi_i.P_i \xrightarrow{x\langle\tilde{w}\rangle} Q_j[\tilde{z}/\tilde{y}]$ if $\pi_j = x(\tilde{y})$.

(OUT) $\sum \pi_i.P_i \xrightarrow{\overline{x}\langle\tilde{y}\rangle} Q_j$ if $\pi_j = \overline{x}\langle\tilde{y}\rangle$.

(BOUT) $P \xrightarrow{\overline{x}\langle\tilde{y}\rangle} Q \Rightarrow (\nu\tilde{z})P \xrightarrow{\overline{x}\langle\nu\tilde{z}.\tilde{y}\rangle} Q$ if $\tilde{z} \lesssim \tilde{y}$.

(τ) $P_1 \xrightarrow{x\langle\tilde{y}\rangle} Q_1,\ P_2 \xrightarrow{\overline{x}\langle\nu\tilde{z}.\tilde{y}\rangle} Q_2 \Rightarrow P_1|P_2 \xrightarrow{\tau} (\nu\tilde{z})(Q_1|Q_2)$ if $\{\tilde{z}\} \cap fn($ $= \phi$

(PAR) $P_1 \xrightarrow{\alpha} Q_1 \Rightarrow P_1|P_2 \xrightarrow{\alpha} Q_1|P_2$ if $bn(\alpha) \cap fn(P_2) = \phi$.

(RES) $P \xrightarrow{\alpha} Q \Rightarrow (\nu x)P \xrightarrow{\alpha} (\nu x)Q$ if x does not occur in α

Transitions: Examples.

(1) $a(xy).\overline{x}\langle y\rangle \xrightarrow{a\langle ef\rangle} \overline{e}\langle f\rangle \xrightarrow{\overline{e}\langle f\rangle} \emptyset.$

(2)
$$\begin{tikzcd}
\overline{a}(\nu c).c(y).\overline{y}\langle e\rangle \arrow[r, "\overline{a}(\nu c.c)"] & c(y).\overline{y}\langle e\rangle \arrow[d, "c\langle f\rangle"] \\
\emptyset. & \overline{f}\langle e\rangle \arrow[l, "\overline{f}\langle e\rangle"']
\end{tikzcd}$$

Note a fresh name is "extruded". By alpha-conversion, we also have a <u>variant transition</u>:

$$\begin{tikzcd}
\overline{a}(\nu c).c(y).\overline{y}\langle e\rangle \arrow[r, "\overline{a}(\nu c'.c')"] & c'(y).\overline{y}\langle e\rangle \arrow[d, "c'\langle f\rangle"] \\
\emptyset & \overline{f}\langle e\rangle \arrow[l, "\overline{f}\langle e\rangle"']
\end{tikzcd}$$

which intuitively has the <u>same meaning</u>: sends a new name, gets "f" via the name, etc.

Transition: Examples.

(3) Let $P \stackrel{\text{def}}{=} !a(x).\overline{x}(\nu c).\overline{c}\langle v\rangle$

$$P \xrightarrow{a\langle e\rangle} P \mid \overline{e}(\nu c).\overline{c}\langle v\rangle \xrightarrow{\overline{e}(\nu c.c)} \overline{c}\langle v\rangle \xrightarrow{\overline{c}\langle v\rangle} \emptyset.$$

(4)
$$\begin{tikzcd}
 & & a(xy).\overline{x}\langle y\rangle + \overline{a}\langle ef\rangle \arrow[dl, "a\langle bc\rangle"'] \arrow[dr, "\overline{a}\langle ef\rangle"] & \\
 & \overline{b}\langle c\rangle \arrow[dl, "\overline{b}\langle c\rangle"'] & & \emptyset \\
\emptyset & & &
\end{tikzcd}$$

(5) $a(xy).\overline{x}(\nu c) \mid \overline{a}\langle ef\rangle \xrightarrow{\tau} \overline{e}(\nu c).\emptyset \xrightarrow{\overline{e}(\nu c.c)} \emptyset$

(6) $\overline{a}\langle 1,2\rangle \mid a(x,y).\overline{b}\langle x+y\rangle$

$$\xrightarrow{\tau} \overline{b}\langle 3\rangle \xrightarrow{\overline{b}\langle 3\rangle} \emptyset.$$

Sorting [Milner 92]

- "How each name is carried by other names".

Ex. In $\bar{x}\langle yz\rangle.\bar{y}\langle z\rangle$, we have a sorting with three sorts S_1, S_2, S_3 such that:

① Name Assignment: $x : S_1$, $y : S_2$, $z : S_3$

② Sorting Function: $S_1 \mapsto S_2 S_3$, $S_2 \mapsto S_3$.

> If $S \mapsto S_1 \ldots S_n$, a name x in S can carry $\langle y_1 \ldots y_n\rangle$ with $y_i : S_i$ for $1 \le i \le n$.

Ex. $x(yz).\bar{y}\langle z\rangle \mid \bar{x}\langle ww\rangle$ is sorted under a sorting with two sorts S_1, S_2 with:

① $x : S_1$, $w : S_2$.

② $S_1 \mapsto S_2 S_2$, $S_2 \mapsto S_2$.

Inference System for Sorting. (1)

- Write $\pi :: \Gamma$ when:

  $\pi = x(y_1 \ldots y_n)$ or $\pi = \bar{x}\langle y_1 \ldots y_n\rangle$

  and $x : S$, $y_i : S_i$ in Γ s.t. $S \mapsto S_1 \ldots S_n$.

- Then P is syntactically well-sorted under Γ when $\Gamma \vdash P$ is derivable in the following system.

$$(\text{sum})\ \frac{\forall i.\ \Gamma_i \vdash P_i,\ \ \pi_i :: \Gamma_i,\ \ \Gamma_i / bn(\pi_i) = \Gamma}{\Gamma \vdash \Sigma\, \pi_i . P_i}$$

$$(\text{par})\ \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad (\text{rest})\ \frac{\Gamma \vdash P}{\Gamma / \{x\} \vdash (\nu x) P}$$

$$(\text{rep})\ \frac{\Gamma \vdash P}{\Gamma \vdash\ !P} \qquad (\text{inact})\ \Gamma \vdash \emptyset.$$

Examples of Sorted Processes (1)

- Fix the sorting Γ with data:

  (1) Sorts: Nat, S_1, S_2 with:

  $0,1,2,\ldots$: Nat, $a : S_1$, $b : S_2$.

  (2) Sorting function: $S_1 \mapsto \mathrm{Nat}\,\mathrm{Nat}$ $\quad S_2 \mapsto S_1$

Then:

$$\Gamma \vdash \bar{b}\langle a\rangle.\bar{a}\langle 1,2\rangle$$

$$\Gamma \vdash (\nu e)\left(e(y).\,\bar{y}\langle a\rangle.\bar{a}\langle 1,2\rangle \mid \bar{e}\langle b\rangle\right)$$

$$\Gamma \vdash (\nu c)\,\bar{b}\langle c\rangle.\bar{c}\langle 1,2\rangle$$

$$\Gamma \vdash b(x).\,\bar{b}\langle x\rangle$$

$$\Gamma \nvdash \bar{b}\langle a\rangle.\bar{a}\langle 1,2\rangle \mid b(x).\,\bar{x}\langle b\rangle.$$

Examples of Sorted Processes (2)

- Process: $\bar{x}\langle 3,5\rangle \mid x(y,z).\,\bar{w}\langle y+z\rangle$ $\qquad \times\ x(w).P$ $\quad \times\ \bar{x}\langle y\rangle.$

  Sorting: $x : S_1$ $\ w : S_2$ $\ 3,5 : \mathrm{Nat}$ with

  $S_1 \mapsto \mathrm{Nat}\,\mathrm{Nat}$ $\quad S_2 \mapsto \mathrm{Nat}$

- Process: $\bar{x}\langle y\rangle.\,\bar{y}\langle 1,2\rangle$ $\qquad \times\ \bar{y}\langle 1\rangle$ $\quad \times\ \bar{x}\langle 3\rangle$

  Sorting: $x : S_1$ $\ y : S_2$ $\ 1,2 : \mathrm{Nat}$ with

  $S_1 \mapsto S_2$ $\quad S_2 \mapsto \mathrm{Nat}\,\mathrm{Nat}$

- Process: $[\![x]\!]u \stackrel{\mathrm{def}}{=} \bar{x}\langle u\rangle$

  $[\![\lambda y.\,y]\!]u \stackrel{\mathrm{def}}{=} u(y\,u').\,\bar{y}\langle u'\rangle.$

  Sorting: $x : \mathrm{Arg}$ $\ u : \mathrm{Fun}$ with

  $\mathrm{Fun} \mapsto \mathrm{Arg} \cdot \mathrm{Fun}.$

  $\mathrm{Arg} \mapsto \mathrm{Fun}.$

//.

Inference System for Sorting (2)

- Basic properties:
  - $\Gamma \vdash P \wedge P \xrightarrow{*} P' \Rightarrow \Gamma \vdash P'$.
  - $\Gamma \vdash P \Rightarrow P \not\xrightarrow{*} \mathrm{Error}$. $a(x).P \mid \bar{a}\langle bc\rangle.Q$ etc.

- Let:

$$P_1 \stackrel{\mathrm{def}}{=} \bar{b}\langle a\rangle.\bar{a}\langle 1,2\rangle$$

$$P_2 \stackrel{\mathrm{def}}{=} \bar{b}\langle a\rangle.\bar{a}\langle 1,2\rangle \mid (\nu c)\, c.\, \bar{b}\langle b\rangle.$$

Then P_1 and P_2 have exactly the same behaviour, and P_1 is well-sorted, but P_2 can never be sorted under any Γ.

Understanding Sorting (1)

- A plan: Take simple Γ and P such that $\Gamma \vdash P$. What does Γ say about the transitions P owns?

> Let $\Gamma \vdash P$. If
>
> $$P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} P_2 \xrightarrow{\alpha_3} P_3 \ldots\ldots$$
>
> What can we say about $\alpha_1, \alpha_2, \ldots$?
>
> Or about $P_1, P_2, P_3, \ldots$?

Understanding Sorting (2)

- From now on, we say

$$\boxed{P \text{ respects } \Gamma}$$

for an intuitive notion of

"P's interactive behaviour (not syntax) conforms to the specification given by Γ."

Sorting Behaviourally. (1)

- We fix Γ as the following sorting:
  - Sorts are S_0, S_1 and S_2 with
    $a : S_0, \quad e : S_1, \quad f : S_2$
  - Sorting function is:
    $S_0 \mapsto S_1 S_2, \quad S_1 \mapsto S_2.$
- We can check:

$$\Gamma \vdash \bar{a}\langle ef \rangle . \bar{e}\langle f \rangle$$

$$\Gamma \vdash (\nu c)\, \bar{a}\langle cf \rangle . \bar{c}\langle f \rangle$$

$$\Gamma \vdash a(xy) . \bar{x}\langle y \rangle$$

etc.

Sorting Behaviourally (2)

- Output (1). Take $P \stackrel{\mathrm{def}}{=} \overline{a}(ef).\overline{e}\langle f\rangle$.

  Then $\Gamma \vdash P$ and:

$$\boxed{P \xrightarrow{\overline{a}(ef)} P' \xrightarrow{\overline{e}\langle f\rangle} 0.}$$

- Output (2). Take $Q \stackrel{\mathrm{def}}{=} (\nu c)\,\overline{a}\langle cf\rangle.\overline{c}\langle f\rangle$.

  Then $\Gamma \vdash Q$ and:

$$\boxed{Q \xrightarrow{\overline{a}\langle \nu c.\, cf\rangle} Q' \xrightarrow{\overline{c}\langle f\rangle} 0.}$$

Sorting Behaviourally (3)

- Observation:

> If P respects Γ and $P \xrightarrow{\alpha} P'$ with
>
> $\alpha = \overline{x}\langle \nu\tilde{z}.\, y_1 .. y_n\rangle$ where $\{\tilde{z}\} \cap fn(\Gamma) = \emptyset$,
>
> then
>
> [1] $x : S$ with $S_1 \mapsto S_1 \ldots S_n$ in Γ and
>
> $y_i : S_i \Leftrightarrow y_i \notin \{\tilde{z}\}$.
>
> [2] P' again respects Γ expanded
>
> with $y_i : S_i$ for all $y_i \in \{\tilde{z}\}$.

* Exercise: Prove this for "$\Gamma \vdash P$" as the notion of "P respects Γ".

Sorting Behaviourally (4)

- Silent action. Take

$$R \stackrel{\mathrm{def}}{=} (\nu c)(c(x y).\overline{x}\langle y\rangle \mid \overline{c}\langle e f\rangle).$$

Then $\Gamma \vdash R$ and:

$$\boxed{R \xrightarrow{\tau} R' \xrightarrow{\overline{e}\langle f\rangle} R''.}$$

- Here we observe:

> If P respects Γ and $P \xrightarrow{\tau} P'$, then P' respects Γ again.

(this is just subject reduction).

Sorting Behaviourally (5)

- So we may conclude:

> If P respects Γ, we have:
>
> [1] All actions of P conform to the name usage specified by Γ; and
>
> [2] The derivative of each action respects Γ again, expanded with "new" names brought by the action.

- Let us call tentatively this idea, General Principle of Sorting (GPS for short).

Sorting Behaviourally (6)

- Input (1). Let $R \stackrel{\text{def}}{=} a(xy).\overline{x}\langle y\rangle$.

Then $\Gamma \vdash R$ and:

$$\boxed{\begin{array}{l} R \xrightarrow{a\langle ef\rangle} R' \xrightarrow{\overline{e}\langle f\rangle} \emptyset. \\ R \xrightarrow{a\langle cf\rangle} R'' \xrightarrow{\overline{c}\langle f\rangle} \emptyset. \end{array}}$$

- So can we say:

If P respects Γ, and if $P \xrightarrow{x\langle y_1 \ldots y_n\rangle} P'$, then $x:S$ s.t. $S \mapsto S_1..S_n$ and $y_i : S_i \Leftrightarrow y_i \in fn(\Gamma)$, and P' respects Γ expanded with fresh names. ?

- But:

Sorting Behaviourally (7)

- Input (2). With $R \stackrel{\text{def}}{=} a(xy).\overline{x}\langle y\rangle$,

$$\boxed{R \xrightarrow{a\langle af\rangle} R''' \xrightarrow{\overline{a}\langle f\rangle} \emptyset.}$$

This does NOT conform to GPS:

[1] $a\langle af\rangle$ and $\overline{a}\langle f\rangle$ violate Γ; $\overline{a}\langle f\rangle$ even violates the arity constraint.

[2] R''' is not well-sorted under Γ.

- What happened: You cannot choose input values ($a\langle af\rangle$), and a bad input can cause You to behave badly ($\overline{a}\langle f\rangle$).

*Exercise: Show a bad input can cause a run-time error in a well-sorted process

Sorting Behaviourally (8)

- Thus, for input, we only have:

> If P respects Γ and $P \xrightarrow{x(y_1..y_n)} P'$
> where $x:S$ s.t. $S \mapsto S_1..S_n$ and $y_i:S_i$
> iff $y_i \in fn(\Gamma)$, then P' again respects
> Γ expanded with fresh names in $\{\tilde{y}\}$.

(Actually we also have: P's input action obeys the arity constraint given by Γ!)

Sorting Behaviourally summary

- Thus the new GPS becomes:

> If P respects Γ, we have:
>
> [1] All actions of P except inputs conform to the name usage of Γ; for input, only arity is guranteed.
>
> [2] The result of all non-input actions again respects Γ (with possible expansion); The result of input conforming to Γ respects Γ (with possible expansion).

- So Γ specifies P's possible actions and their consequences assuming the environment behaves well (i.e. does not communicate bad values).

A Bit of Formalisation (1) Preparation, 1.

- Question: Formally, what Γ specifies?
- Answer: Possible actions (both of the process and of the environment) and the resulting specifications:

$$\Gamma \xrightarrow{\alpha} \Gamma' \cdots$$

$$\underset{\{a,b,c\}}{\bullet} \xrightarrow{\alpha} \underset{\{a,b,c\} \cup \text{New Names in } \alpha}{\bullet} \cdots$$

- To have a rigorous treatment of "new names" in input, we let each input explicitly mention them:

$$(*) \quad \alpha ::= x\langle \nu\{\tilde{z}\}.\tilde{y}\rangle \;\Big|\; \overline{x}\langle \nu\{\tilde{z}\}.\tilde{y}\rangle \;\Big|\; \tau.$$

where $\{\tilde{z}\} \subseteq \{\tilde{y}\}$. We again simply write $x\langle \nu\tilde{z}.\tilde{y}\rangle$ etc.

- Notations: $fn(\alpha)$ (resp. $bn(\alpha)$) is the set of free names (resp. bound names) in α, and $sbj(x\langle\nu\tilde{z}.\tilde{y}\rangle) = x$

A Bit of Formalisation (2) Preparation, 2.

<u>Definition.</u>

(i) A name passing labelled transition system is a usual lts with actions given as (*) and states denoted $p, q, r, \ldots$ such that:

- Each state p is given a finite set of names, called its surface, written $surf(p)$.
- Whenever $p \xrightarrow{\alpha} q$, we have:

[1] $fn(\alpha) \subseteq surf(p)$,

[2] $bn(\alpha) \cap surf(p) = \emptyset$, and

[3] $surf(q) = surf(p) \cup bn(\alpha)$.

$$\underset{A}{\overset{p}{\bullet}} \xrightarrow{\alpha} \underset{B}{\overset{q}{\bullet}}$$

(ii) A process is the root state of a name-passing lts which is a tree whose all edges are directed towards leaves w.r.t. $\xrightarrow{\alpha}$. $p, q, r, \ldots$ again range over processes. //

Bisimulations.

Notation.

$\rightarrow$, $\overset{s}{\Longrightarrow}$, $\overset{\alpha}{\Longrightarrow}$, $\overset{\hat{\alpha}}{\Longrightarrow}$. (See [Milner 89]).

Definition. (bisimulations).

(i) Given a name-passing lts, a relation $\mathcal{R}$ over its states is a strong bisimulation if, whenever $p\mathcal{R}q$, we have: [1] $sort(p) = sort(q)$, [2] If $p \xrightarrow{\alpha} p'$ then $q \xrightarrow{\alpha} q'$ s.t. $p'\mathcal{R}q'$, and [3] the symmetric case of [2]. We write $p \sim q$ if two are related by such $\mathcal{R}$.

(ii) Similarly we define a weak bisimulation, replacing the second $\xrightarrow{\alpha}$ in [2] above with $\overset{\hat{\alpha}}{\Longrightarrow}$, similarly for [3]. We write $p \approx q$ if p and q are related by a weak bisimulation. //

A Bit of Formalisation (3) Formalising $\Gamma \xrightarrow{\alpha} \Gamma'$ 1.

Definition. (Allowance and Consequence)

(i) $\alpha \prec \Gamma$ ("α respects Γ") iff either

[1] $\alpha = \tau$

[2] $\alpha = x^{\updownarrow}\langle \nu\tilde{z}.\, y_1 \cdots y_n \rangle$ such that

- $\{\tilde{z}\} \cap fn(\Gamma) = \phi$.
- $x : S$ in Γ where $S \mapsto S_1 \ldots S_n$ and $y_i : S_i \Leftrightarrow y_i \notin \{\tilde{z}\}$.

(ii) Γ after α ("the consequence of α in Γ") is defined iff $\alpha \prec \Gamma$ and, when defined, has the value as follows.

[1] Γ after $\tau = \Gamma$ always.

[2] Γ after $x^{\updownarrow}\langle \nu\tilde{z}.\, y_1 \cdots y_n \rangle = \Gamma \bullet \{y_i : S_i\}_{y_i \in \{\tilde{z}\}}$.

$\left(\Gamma \bullet \{x_i : S_i\}\right.$ with $fn(\Gamma) \cap \{\tilde{x}\} = \phi$ is
the result of adding $x_i : S_i$ to Γ $\left.\right)$

Allowance and Consequence: Examples.

(1) $\Gamma \stackrel{\mathrm{def}}{=} a{:}s_1,\ s_1 \mapsto s_2,\ s_2 \mapsto s_3 s_1.$ Then:

$$\alpha \prec \Gamma \iff \alpha \in \{a\langle \nu b.b\rangle,\ \overline{a}\langle \nu b.b\rangle,\ \tau\}.$$

$$\Gamma \text{ after } a^{\updownarrow}\langle \nu b.b\rangle = \Gamma \circ \{b{:}s_2\}.$$

(2) $\Gamma' \stackrel{\mathrm{def}}{=} a{:}s_1,\ b{:}s_2,\ s_1 \mapsto s_2,\ s_2 \mapsto s_3 s_1.$ Then:

$$\alpha \prec \Gamma \iff \alpha \in \{a^{\updownarrow}\langle b\rangle,\ a^{\updownarrow}\langle \nu c.c\rangle,\ b^{\updownarrow}\langle \nu c.ca\rangle,\ b^{\updownarrow}\langle \nu ce.ce\rangle,\ \tau\}.$$

$$\Gamma \text{ after } b^{\updownarrow}\langle \nu ce.ce\rangle = \Gamma \circ \{c{:}s_3,\ e{:}s_1\}.$$

(3) $\Gamma'' \stackrel{\mathrm{def}}{=} a{:}s_1,\ e{:}s_1,\ b{:}s_2,\ c{:}s_3.$ Then:

$$\alpha \prec \Gamma \iff \alpha \in \{a^{\updownarrow}\langle b\rangle,\ a^{\updownarrow}\langle \nu f.f\rangle,\ b^{\updownarrow}\langle \nu f.fa\rangle,\ b^{\updownarrow}\langle \nu fg.fg\rangle,\ e^{\updownarrow}\langle b\rangle,\ e^{\updownarrow}\langle \nu f.f\rangle,\ b^{\updownarrow}\langle \nu f.fe\rangle,$$

A Bit of Formalisation (4) forming $\Gamma \xrightarrow{a} \Gamma'$, 2.

Definition. (behavioural sorting)

For each Γ, the behavioural sorting from Γ, denoted $\mathrm{beh}(\Gamma)$, is the process corresponding to the node Γ° in the following name-passing lts:

(i) $\mathrm{sub}(\Gamma^{\circ}) = \mathrm{fn}(\Gamma)$.

(ii) $\Gamma^{\circ} \xrightarrow{\alpha} \Delta^{\circ}$ iff:

[1] $\alpha \prec \Gamma$ and

[2] $\Delta = \Gamma \text{ after } \alpha$. //

Note: $\mathrm{beh}(\Gamma)$ is more abstract than Γ, i.e. there are Γ and Δ s.t. $\Gamma \neq \Delta$ but $\mathrm{beh}(\Gamma) = \mathrm{beh}(\Delta)$ (Exercise: Find

A Bit of Formalisation (5) Formalising $\Gamma \xrightarrow{a} \Gamma'$, 3.

Proposition.

For any Γ, $beh(\Gamma)$ is strongly deterministic in the following sense: if $beh(\Gamma) \xrightarrow{s} p$, then [1] $p \xrightarrow{\tau} p'$ implies $p' \sim p$, and [2] $p \xrightarrow{\alpha} p'_{1,2}$ implies $p'_1 \sim p'_2$ for any α.

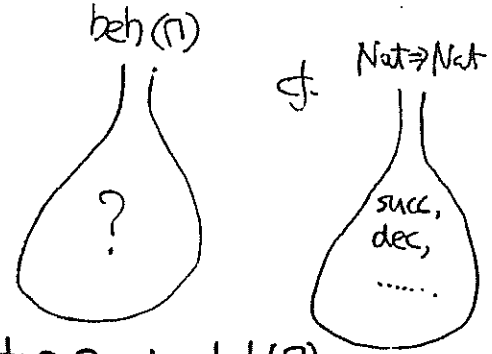
Proof: [1] is direct from definition while [2] is because after is a (partial) function. ∎

* Determinacy says $beh(\Gamma)$ is quite tractable as a specification.

3. Understanding Sorting (2)

Sorted Processes, Semantically (1)

• Question: what are in the bag?



• Answer: behaviours conforming to beh(Γ).

Definition.

A relation $\mathcal{R}$ between processes and $\{s \mid s = \mathrm{beh}(\Gamma) \text{ for some } \Gamma\}$ is a simulation when $p\mathcal{R}s$ implies:

[1] $\mathrm{surf}(p)\,\mathcal{R}\,\mathrm{surf}(s)$, and

[2] If $p \xrightarrow{\alpha} p'$ then $s \xrightarrow{\alpha} s'$ and $p'\mathcal{R}s'$, for each α.

A simulation $\mathcal{R}$ is a conformance when moreover

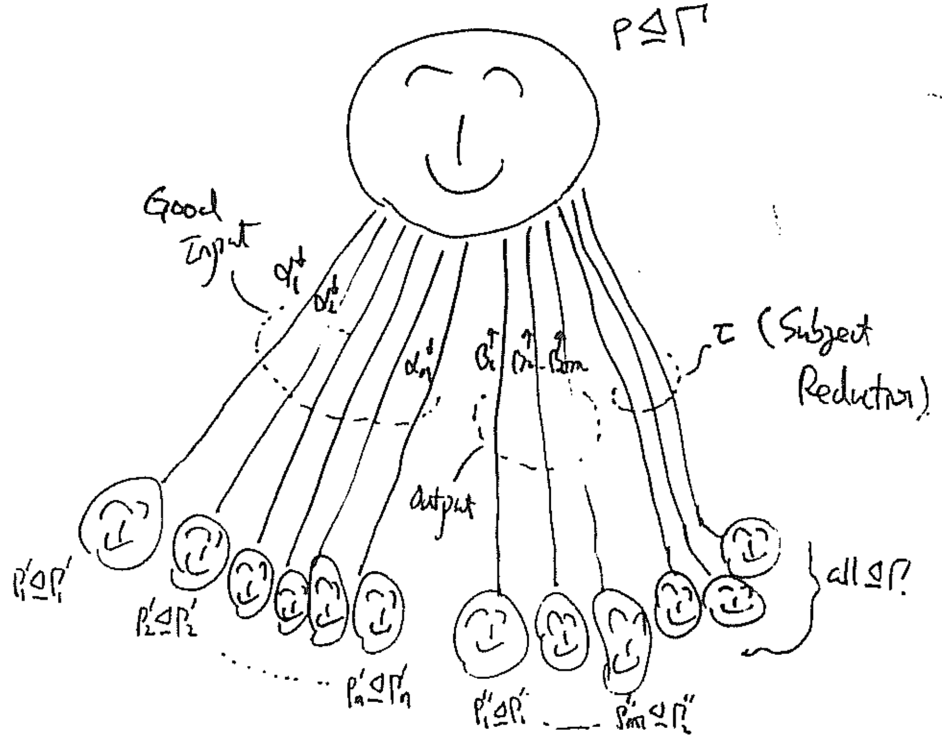
[3] If $s \xrightarrow{\alpha^{\downarrow}} s'$ and p has input at $\mathrm{sbj}(\alpha)$, then $p \xrightarrow{\alpha}$.

$\boxed{\text{You should get all legitimate input values.}}$

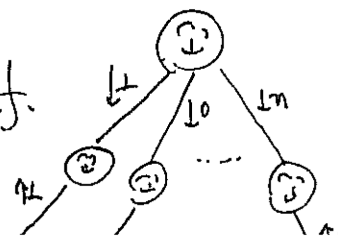
We write $p \trianglelefteq \mathrm{beh}(\Gamma)$ if $p\mathcal{R}\mathrm{beh}(\Gamma)$ for a conformance $\mathcal{R}$. //

Sorted Processes, Semantically (2).

• Conformance:



cf.



succ in Nat⇒Nat in CPO. ("bad input" is e.g. true, false, 3.1415, etc.)

Sorted Processes Semantically (3).

- We have now types for sorting.

Definition. (behavioural types for sorting).

We define $[\![\Gamma]\!]$, the behavioural type corresponding to Γ, as follows.

$$[\![\Gamma]\!] \stackrel{\text{def}}{=} \{P \mid P \trianglelefteq \mathrm{beh}(\Gamma)\}.$$

$X, Y, \ldots$ range over these collections.

We write $p : X$ when $p \in X$. //

* $[\![\Gamma]\!]$ gives the collection of all well-typed interactions w.r.t. Γ.

Universe of Sorting (1) Operations, I.

- We wish to define, e.g. $p:X \mid q:X$, $(\nu x)(p:X)$ etc. How? We expect:

  - $p:X \mid q:X$ is defined and is in X again.
  - $(\nu x)(p:X)$ is always defined and is in $X/x \stackrel{\text{def}}{=} [\![\Gamma/\{x\}]\!]$ with $X = [\![\Gamma]\!]$.
  - $!(p:X)$ is always defined and is in X again.
  - $\sum \alpha_i.(p_i:X_i)$ is defined iff $\sum \alpha_i.\mathrm{beh}(\Gamma_i) \trianglelefteq \mathrm{beh}(\Delta)$ for some Δ where $X_i = [\![\Gamma_i]\!]$ for each i.

But we first need operations on individual processes (cf. $f \bullet e = f(e)$).

Universe of Sorting (2) Operations, 2.

Definition (sum) [Milner 92]

Given $\{\alpha_i\}_{i\in I}$ and $\{P_i\}_{i\in I}$ such that (1) $bn(\alpha_i) \subseteq surf(P_i)$ for each i, and (2) $surf(P_i)\setminus bn(\alpha_i) = surf(P_j)\setminus bn(\alpha_j)$ for each i, j, we define $\sum_{i\in I} \alpha_i . P_i$ as the standard sum construction, with $surf(\sum_{i\in I} \alpha_i P_i) = \bigcup surf(P_i)\setminus bn(\alpha_i)$. //

Remark.

- Note the surface is empty if $I = \phi$.
- We sometimes write $\alpha.P + \alpha'.P' + \alpha''.P''$ or $\sum \alpha_i.P_i + \sum \beta_j.Q_j \cdots$ to denote a sum. //

Universe of Sorting (3) Operations, 3.

Notation: Given $A \subseteq \{\vec{y}\}$, $\nu A . \overline{x}\langle \nu B . \vec{y}\rangle \overset{\text{def}}{=} \overline{x}\langle \nu A \cup B . \vec{y}\rangle$.

We write $\alpha^{\downarrow}$ (resp. $\alpha^{\uparrow}$) to indicate α is input (resp. output).

Definition (hiding) [MPW89].

Let $P : X$ for some X and A be a finite set of names. Assume, w.l.o.g., $P \overset{\text{def}}{=} \sum \alpha_i . P_i'$ s.t. $bn(\alpha_i) \cap A = \phi$ for each i. Then $(\nu A) P$ is defined inductively as:

$$(\nu A) P \overset{\text{def}}{=} \sum_{fn(\alpha_i^{\downarrow})\cap A = \phi} \alpha_i^{\downarrow} . (\nu A) P_i' \qquad (*)$$

$$+ \sum_{sbj(\alpha_i) \notin A} (\nu A \cap fn(\alpha_i) . \alpha_i^{\uparrow}) . (\nu A \setminus fn(\alpha_i)) P_i'$$

$$+ \sum_{\alpha_i = \tau} \tau . (\nu A) P_i,$$

setting $surf((\nu A)P) = surf(P)\setminus A$. //

(*) $fn(\alpha_i^{\downarrow}) \cap A = \phi$ says (1) the subject is not hidden, and

Universe of Sorting (4) Operations, 4.

Notation. (i) If $A \supseteq \mathrm{surf}(\rho)$, we write ρ^A for the same process as ρ except its surface is A (avoiding the capture of names).

(ii) We write $x\langle \nu A.\tilde{y}\rangle \leftrightarrow \bar{x}\langle \nu A'.\tilde{y}'\rangle$ when $A\sigma \supseteq A'$ and $\tilde{y}\sigma = \tilde{y}'$ for some permutation σ on names which is identity except on $A \cup A'$ (such σ is unique if it exists).

Definition. (expansion). ([MPW89], adapted to early transition).

Given a family $\{P_i : X\}_{i\in I}$ for some X, assume w.l.o.g. $P_i \stackrel{\mathrm{def}}{=} \sum_{j\in J_i} \alpha_{ij}.P_{ij}$ with $\mathrm{bn}(\alpha_{ij}) \cap \mathrm{surf}(P_{i'}) = \emptyset$ whenever $i \neq i'$. Then $\prod P_i$ is given by the following inductive formula:

$$\prod_{i\in I} P_i \stackrel{\mathrm{def}}{=} \sum_{\substack{i\in I\\ j\in J_i}} \alpha_{ij}.\prod_{m\in I\cup\{ij\}\setminus\{i\}} P_m^{\mathrm{surf}(P_{ij})} + \sum_{\substack{\alpha_{ij}\leftrightarrow\alpha_{kl}\\ i\neq k}} \tau.(\nu\,\mathrm{bn}(\alpha_{ij})\sigma)\prod_{\substack{m\in I\cup\{ij,kl\}\\ \setminus\{i,k\}}} P'_m,$$

where (i) $\mathrm{surf}(\prod_{i\in I} P_i) = \bigcup_{i\in I}\mathrm{surf}(P_i)$ (ii) σ in the second sum is the permutation associated with $\alpha_{ij}\leftrightarrow\alpha_{kl}$ mentioned in Notation, and (iii) P'_m in the second sum is given by: $P'_m \stackrel{\mathrm{def}}{=} P_m^{\mathrm{surf}(P_{ij})\sigma}$ when $m \neq ij$, $P'_m \stackrel{\mathrm{def}}{=} P_{ij}\sigma$ when $m = ij$. //

Universe of Sorting (5) Operations, 5.

Remark

- Hiding and expansion are a simple adaptation of the standard construction [MPW89] (which is for the late transition) to the early, well-sorted setting, enjoying basic algebraic properties such as associativity.
- The well-definedness of these operations (inductive invariance of the initial conditions) is verified by induction on the depth of the (resulting) tree. //

Notation.

Hereafter we use the following familiar notations:

$$(\nu x)P \stackrel{\mathrm{def}}{=} (\nu\{x\})P.$$

$$P_1 \mid P_2 \stackrel{\mathrm{def}}{=} \prod_{i\in\{1,2\}} P_i.$$

$$!P \stackrel{\mathrm{def}}{=} \prod_{i\in\omega} P_i \quad \text{where } P_i = P \text{ for each } i\in\omega. \ //$$

Universe of Sorting (6) Algebra.

• Operators on sorted processes are given by:

$$\Sigma \alpha_i.(p_i : X_i) \stackrel{def}{=} \Sigma \alpha_i.p_i : \Sigma \alpha_i.X_i$$

If $\Sigma \alpha_i.\mathrm{beh}(X_i) \trianglelefteq \Delta$ for some i; then $\Sigma \alpha_i.X_i$ is the minimum such.

$$(p : X) \mid (q : X) \stackrel{def}{=} (p \mid q) : X.$$

$$(\nu x)(p : X) \stackrel{def}{=} \nu\{x\}p : X/x$$

(X/x is given before.)

$$!(p : X) \stackrel{def}{=} (!p) : X.$$

$$0 : X$$

(the nullary operator, 0 being the inaction).

Proposition

All operators above are well-defined.

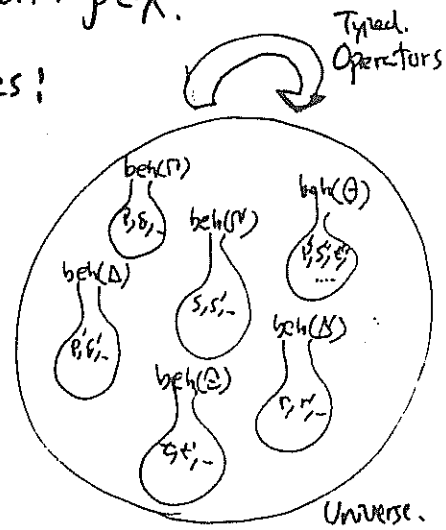
Proof: First show $\mathrm{beh}(\Gamma) \mid \mathrm{beh}(\Gamma) = \mathrm{beh}(\Gamma)$ etc. and form conformances. ∎

Universe of Sorting (summary).

• We have now a universe of typed processes given by:

- the collection of types: $\{X \mid X = [\![\Gamma]\!] \text{ for some } \Gamma\}$.
- typed processes: $p : X$ with $p \in X$.
- operators on typed processes:

$$p : X \mid p : X$$
$$(\nu x)(p : X)$$
$$!(p : X)$$
$$\Sigma \alpha_i.(p_i : X_i)$$
$$0 : X$$



• The universe offers a sound model for the sorted π-calculus via a natural interpretation.

• Deeper study of such a model will be discussed in other occasions.

	λ	π
Type is what:	A collection of functions.	A collection of name passing interactive behaviours.
Classify:	functions.	interactive behaviours.
Typical Type constructors	Arrow types $\alpha \Rightarrow \beta$ (product, sum, recursion, ...)	Dynamic specification of behaviour (can vary a lot.)
Fundamental Operator	Function application $f \bullet e$ ($\sim$ functional composition $g \circ f$)	Process-theoretic operations (e.g. expansion) and their combination.
Underlying/applied mathematical structures/theories	Domain theory Category Theory (CCCs, PCCs ...) Logics	Domain theory Category theory (cf. Action Structures, Games) Logics (cf. HM-logic) (and perhaps more?)
Applications.	Design/analysis of (sequential)	Design/analysis/understanding of general

What We Have Got

* Basic understanding of the behavioural aspects of sorting.
* A basis for further study of behavioural types for mobile processes!
  - Refined Sortings. (IO, Linearity, Polymorphism, ..)
  - Integration and new ideas, starting from the understanding on dynamics, rather than from syntax.

⇨ - Relationship to functional universes via e.g. games.

4

Games from a π-calculus viewpoint (1)

A Typed λν-Calculus and its Encoding.

- λν-calculus with types and constants.

$$\alpha ::= \mathrm{Nat} \mid \alpha \Rightarrow \beta \mid t \mid \mu t.\alpha \quad \text{(as tree unfoldings)}$$

$$M^\alpha ::= x^\alpha \mid (\lambda x^\alpha . M^\beta)^{\alpha\Rightarrow\beta} \mid (M^{\alpha\Rightarrow\beta} N^\alpha)^\beta \mid n^{\mathrm{Nat}} \mid \mathrm{succ}^{\mathrm{Nat}\Rightarrow\mathrm{Nat}}$$

$$V^\alpha ::= \text{not of form } MN.$$

$$[\beta_v]\ (\lambda x^\alpha . M^\beta)V^\alpha \to M^\beta[V^\alpha/x^\alpha]. \quad [\mathrm{succ}]\ \mathrm{succ}\, n \to n+1.$$

$$[\mathrm{app}]\ M \to M' \Rightarrow MN \to M'N \wedge NM \to NM'.$$

- λν in π ([Milner 90], Polyadic version).

$\boxed{\begin{array}{l}\underline{\mathrm{Thm}} \text{ (comp. adeq.)} \\ M \Downarrow n \Leftrightarrow ([M])_u \Downarrow \xrightarrow{\bar{u}\langle n\rangle}\end{array}}$

$$([x^{\mathrm{Nat}}])_u \stackrel{\mathrm{df}}{=} \bar{u}\langle x\rangle \qquad ([x^{\alpha\Rightarrow\beta}])_u \stackrel{\mathrm{df}}{=} \bar{u}(\nu c).!c(eu').\bar{x}\langle eu'\rangle.$$

$$([\lambda x^\alpha . M^\beta])_u \stackrel{\mathrm{df}}{=} \bar{u}(\nu c).!c(xu').([M^\beta])_{u'}.$$

often $\mathrm{ap}\langle f,x,u\rangle$

$$([M^{\alpha\Rightarrow\beta} N^\alpha])_u \stackrel{\mathrm{df}}{=} (\nu f x)\big(([M^{\alpha\Rightarrow\beta}])_f \mid ([N])_x \mid f(c).x(e).\bar{c}\langle eu\rangle\big)$$

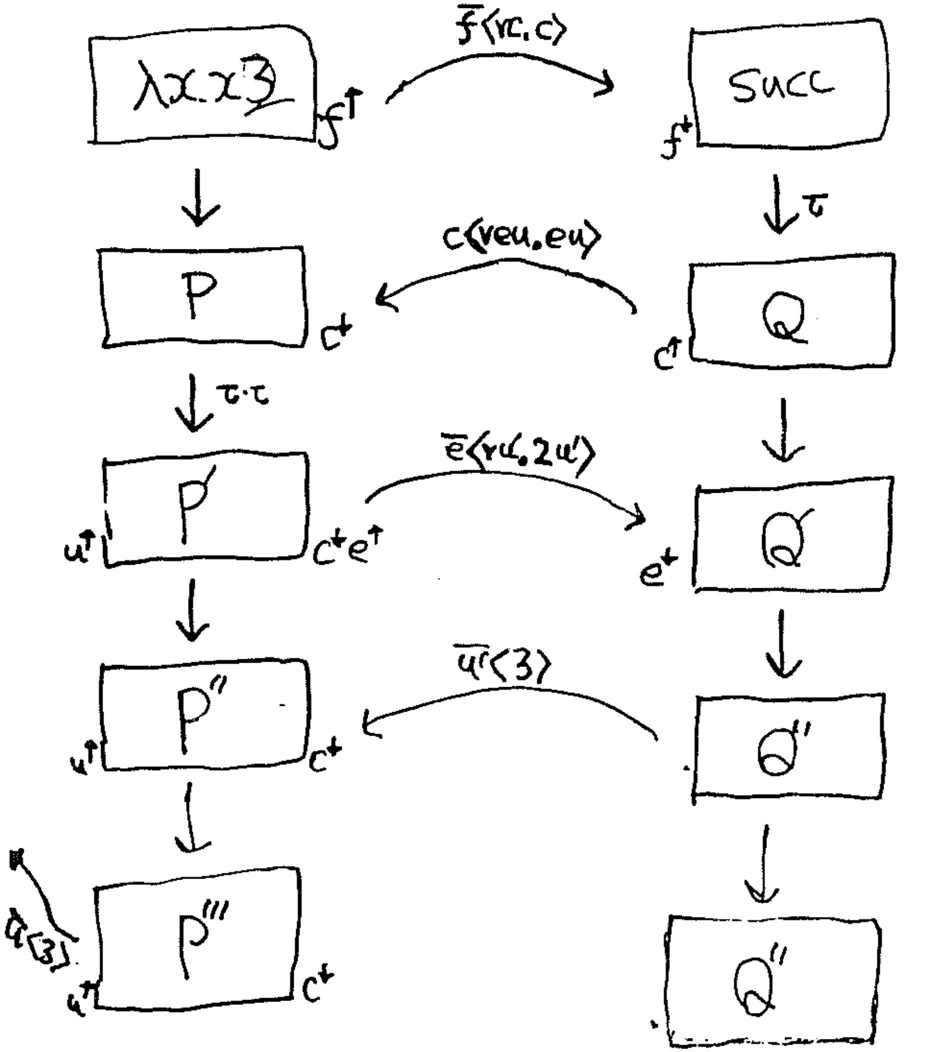
$$([n])_u \stackrel{\mathrm{df}}{=} \bar{u}\langle n\rangle \qquad ([\mathrm{succ}])_u \stackrel{\mathrm{df}}{=} \bar{u}(\nu c).!c(n,u').\bar{u'}\langle n+1\rangle. \ //$$

A λv-interaction.

$(4.7)\ (\llbracket M \rrbracket_f \mid \llbracket N \rrbracket_x \mid ap(f,x,u))$

(vfx)

$$\left(\llbracket \lambda x^{Nat \Rightarrow Nat}.\, x2 \rrbracket_f \quad \begin{smallmatrix}\cdots\cdots\rightarrow \\ \leftarrow\cdots\cdots\end{smallmatrix} \quad \left(\llbracket succ \rrbracket_x \mid f(c).x(e).\overline{c}\langle eu\rangle\right)\right)$$



Understanding λv-Interaction.

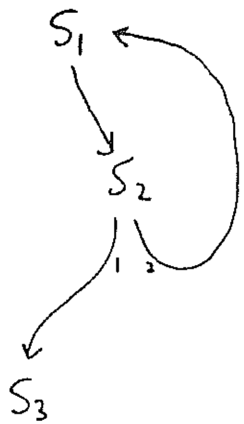
- Plan: To give an exact characterisation of well-typed λv-interaction, i.e. those interactions between $\llbracket M \rrbracket_f$ and $\llbracket N \rrbracket_x \mid ap(f,x,u)$, using basic elements of games semantics.
- Concretely we give an incremental behavioural specification as follows:
  - [1] Basic sorting.
  - [2] Refinement of sorting
    - IO-modes.
    - Linearity.
    - Strict Alternation and πI-interaction.
  - [3] Global Constraints.
    - Determinacy and Switching Condition.
    - Innocence and Well-bracketing.
- We finally turn the resulting universe of process types into a category of call-by-value computation, linking the world of interaction and that of function.

Sorting as a Tree

- $S_1 \mapsto S_2 \quad S_2 \mapsto S_3 S_4 S_5$

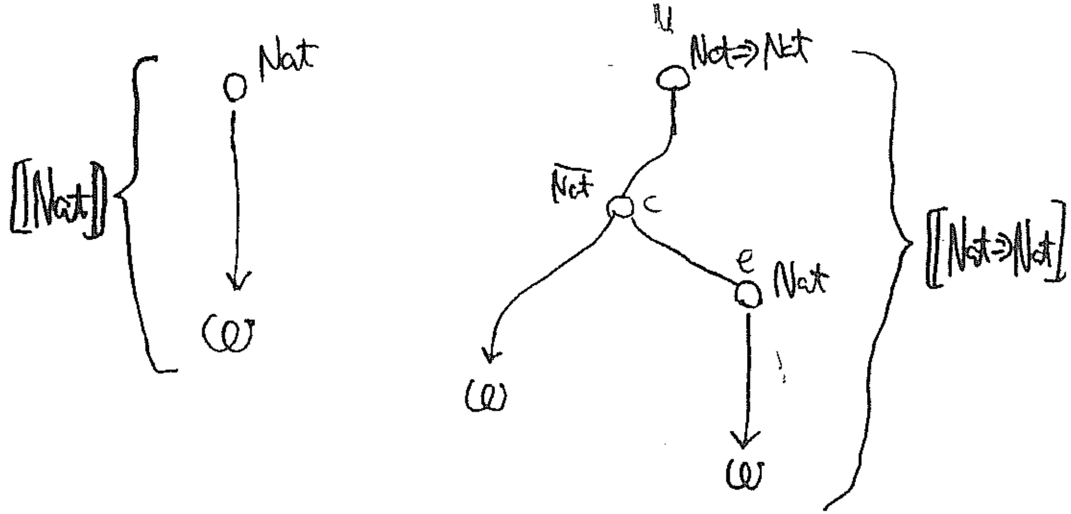
or simply:

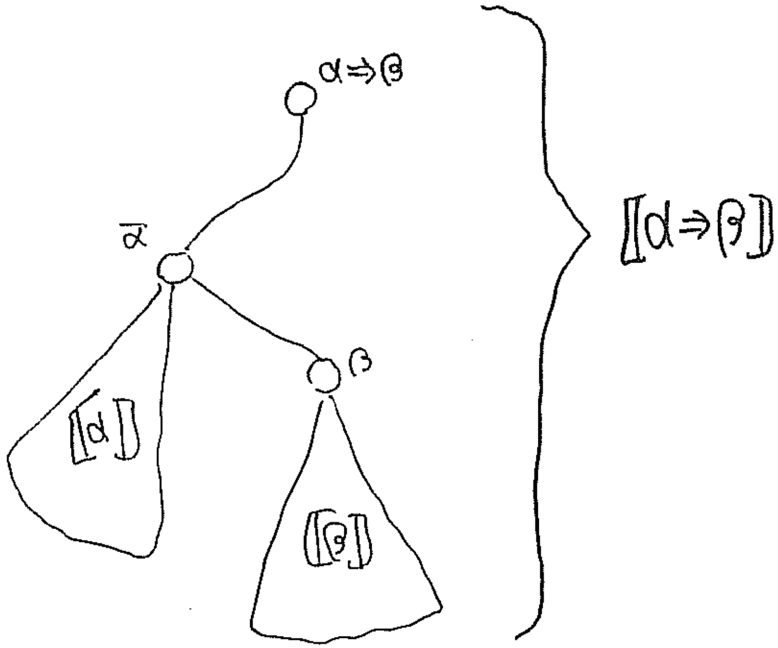
- $S_1 \mapsto S_2 \quad S_2 \mapsto S_3 S_1$



or simply:

Sorting for λ_v-Interaction (1) basic sorting.

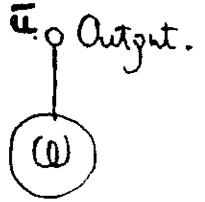




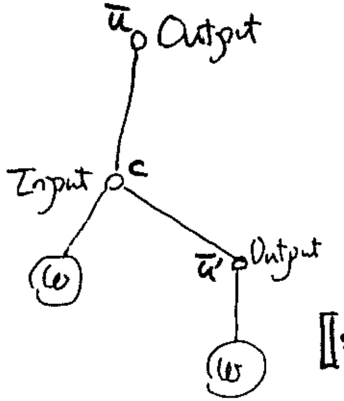
Sorting for λv-Interaction (2) refinement, 1.

• Add further constraints on interaction;

[1] IO-modes.



$[\![n]\!]_u \stackrel{\mathrm{def}}{=} \underline{\overline{u}}\langle n\rangle$



$[\![\mathrm{succ}]\!]_u = \underline{\overline{u}}(\nu c).\underline{!c}(n,u').\underline{\overline{u'}}\langle n+1\rangle$

[2] Linearity and Replication.

$$[\![\mathrm{succ}]\!]_u = \underbrace{\overline{u}}_{\text{Linear}}(\nu c).\underbrace{!c}_{\text{Replicated}}(n,u').\underbrace{\overline{u'}}_{\text{Linear}}\langle n+1\rangle$$

$$[\![x^{\alpha\Rightarrow\beta}]\!]_u = \underbrace{\overline{u}}_{\text{Linear}}(\nu c).!\underline{c}(eu').\underline{\overline{x}}(eu')$$

Replicated.

$$[\![MN]\!]_u = (\nu f x)([\![M]\!]_{\underline{f}} \mid [\![N]\!]_{\underline{x}} \mid \underline{f}(c).\underline{x}(y).\underbrace{\underline{\overline{c}}}_{\text{Replicated.}}\langle yu\rangle)$$

Linear

πI-transformation.

• Sangiorgi [Sangiorgi 96] showed we can represent many structures (including λ) by πI interaction;

$$\alpha ::= x\langle \nu\tilde{y}.\tilde{y}\rangle \mid \overline{x}\langle \nu\tilde{y}.\tilde{y}\rangle \mid \tau.$$

where $\tilde{y}$ is a sequence of <u>distinct</u> names (if we have constants, such as 1, 2, ..., these are not bound).

• Syntactically this means the prefix becomes;

$\rightarrow$ $\pi ::= \overline{x}(\nu\tilde{y}) \mid x(\tilde{y})$,

again excluding constant outputs.

• In particular, the forwarder $!a(xy).\overline{b}\langle xy\rangle$ becomes, under our λv-sorting, the following dynamic link;

$$a \overset{2}{\mapsto} b \stackrel{\mathrm{def}}{=} !a(xy).\overline{b}(\nu x'y').(x' \overset{2}{\mapsto} x \mid y' \overset{1}{\mapsto} y).$$

$$e \overset{1}{\mapsto} f \stackrel{\mathrm{def}}{=} \underbrace{\underline{e}}_{\text{Linear.}}(x).\overline{f}(\nu x').x' \overset{2}{\mapsto} x,$$

which is nothing but copy-cat. Using them we set;

$$(\![x^{\alpha\Rightarrow\beta}]\!)_u \stackrel{\mathrm{def}}{=} \overline{u}(\nu c).c \overset{2}{\mapsto} x.$$

$$ap(f,x,u) \stackrel{\mathrm{def}}{=} f(c).x(e).\overline{c}(\nu e'u').\begin{pmatrix} u' \overset{1}{\mapsto} u \\ e' \overset{2}{\mapsto} e \end{pmatrix},$$

which again gives computationally adequate encoding. //

Sorting for λv-Interaction (3) refinement, 2.

[3] Strict Alternation.

$$(\nu f x)\big(\underbrace{[\![M]\!]_f}_{(1)} \,\big|\, \underbrace{[\![N]\!]_x \mid \mathrm{ap}(f, x, u)}_{(2)}\big)$$

The interaction between (1) and (2) is always

(from (1)'s viewpoint)

$$\xrightarrow{\text{output}}$$
$$\xleftarrow{\text{input}}$$
$$\vdots$$

* For this property to hold within a nested expressions, one needs the π-I transformation.

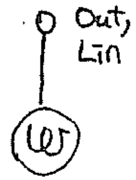
Essentially, πI-transformation prohibits the change of positions of an interaction point;

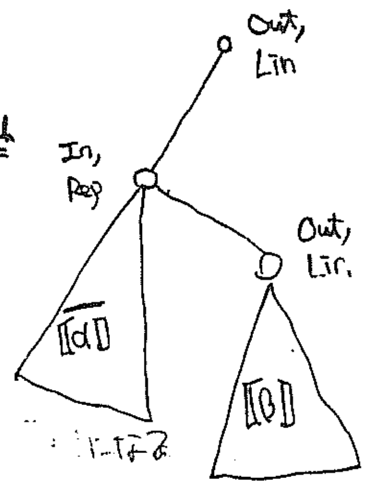
Thus interactions follow syntactic structures precisely. //

Sorting of λ-Interaction (4) λv-sorting, 1. statics.

Definition.

For each λv-type α, the corresponding λv-sorting, again denoted $[\![\alpha]\!]$ is defined as follows.

$[\![Nat]\!] \stackrel{\mathrm{def}}{=}$ 

$[\![\alpha \Rightarrow \beta]\!] \stackrel{\mathrm{def}}{=}$ 

where $\overline{[\![\alpha]\!]}$ is the result of reversing In/Out labels. Γ, Δ, ... range over λv-sortings, with appropriate name assignment. //

Sorting of $\lambda\nu$-Interaction (5) $\lambda\nu$-sorting, 2. dynamics.

Definition (the dynamics of $\lambda\nu$-sorting).

Given Γ (with appropriate name assignment), its annotation is a tuple $\langle \Gamma, \tilde{x}, \delta \rangle$ with $\delta \in \{odd, even\}$ and $\{\tilde{x}\} \subseteq fn(\Gamma)$, denoted $\Gamma^{\tilde{x}}_{\delta}$. Then we set:

$\tilde{x}$ are linear, dispensed names; δ is for strict alternation.

[i] $\alpha^{\downarrow} < \Gamma^{\tilde{x}}_{even}$ iff $\alpha = a\langle \nu\tilde{z}.\, y_1 \dots y_n \rangle$

(1) $a : S^{IN}$ s.t. $S \mapsto S_1 \dots S_n$ as well as $a \notin \{\tilde{x}\}$ — I/O-mode & linearity.

(2) $y_i \notin \{\tilde{z}\} \Leftrightarrow y_i : \mathrm{Nat}$, $\{\tilde{z}\} \cap fn(\Gamma) = \phi$. — πI-interaction

We then set:

$\Gamma^{\tilde{x}}_{even}$ after $\alpha \overset{def}{=} (\Gamma \cdot \{y_i : \alpha_i\})^{\tilde{x}'}_{odd}$

where $\tilde{x}' = \tilde{x}a$ if S^{Lin} while $\tilde{x}' = \tilde{x}$ if S^{Rep}. — strict alternation and linearity.

[ii] $\alpha^{\uparrow} < \Gamma^{\tilde{x}}_{odd}$ iff $\alpha = \bar{a}\langle \nu\tilde{z}.\, y_1 \dots y_n \rangle$ with the same conditions except, in (1), we have $a : S^{OUT}$. Then

$\Gamma^{\tilde{x}}_{odd}$ after $\alpha \overset{def}{=} (\Gamma \cdot \{y_i : \alpha_i\})^{\tilde{x}'}_{even}$ with $\tilde{x}'$ as in [i].

[iii] $\tau < \Gamma^{\tilde{x}}_{\delta}$ always, and $\Gamma^{\tilde{x}}_{\delta}$ after $\tau = \Gamma^{\tilde{x}}_{\delta}$. //

Sorting of $\lambda\nu$-Interaction (6) $\lambda\nu$-sorting, 3.

- Let the name-passing lts $\Gamma^{\tilde{x}}_{\delta} \xrightarrow{\alpha} \Gamma'^{\tilde{x}'}_{\delta'}$ be given by $surf(\Gamma^{\tilde{x}}_{\delta}) = fn(\Gamma)$ and $\Gamma^{\tilde{x}}_{\delta} \xrightarrow{\alpha} \Gamma'^{\tilde{x}'}_{\delta'} \Leftrightarrow \alpha :: \Gamma^{\tilde{x}}_{\delta} \wedge \Gamma'^{\tilde{x}'}_{\delta'} = \Gamma^{\tilde{x}}_{\delta}$ after α. We can now introduce:

Definition. (behavioural $\lambda\nu$-sorting)

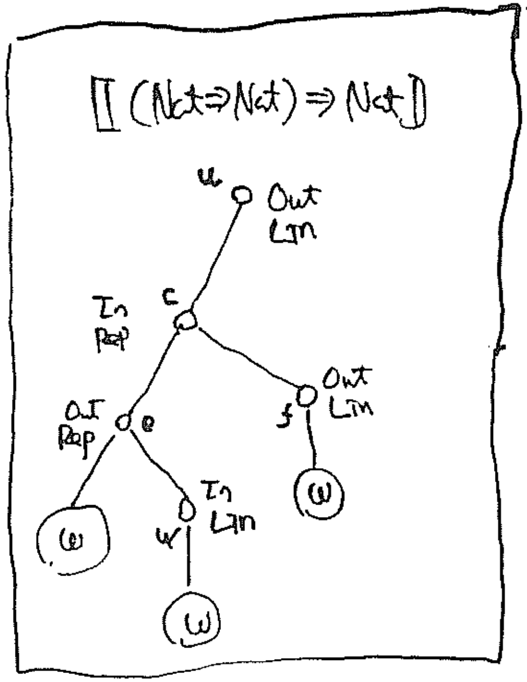
Given a $\lambda\nu$-type α, the behavioural sorting from α with name u, denoted $beh(\alpha)_u$, is given by the process unfolding the lts $\Gamma^{\tilde{x}}_{\delta} \xrightarrow{a} \Gamma'^{\tilde{x}'}_{\delta'}$ starting from the state $[\![\alpha]\!]^{\phi}_{odd}$ with the name assignment: $u : S$ for the root S of $[\![\alpha]\!]$ and $n : \omega$ for each new. //

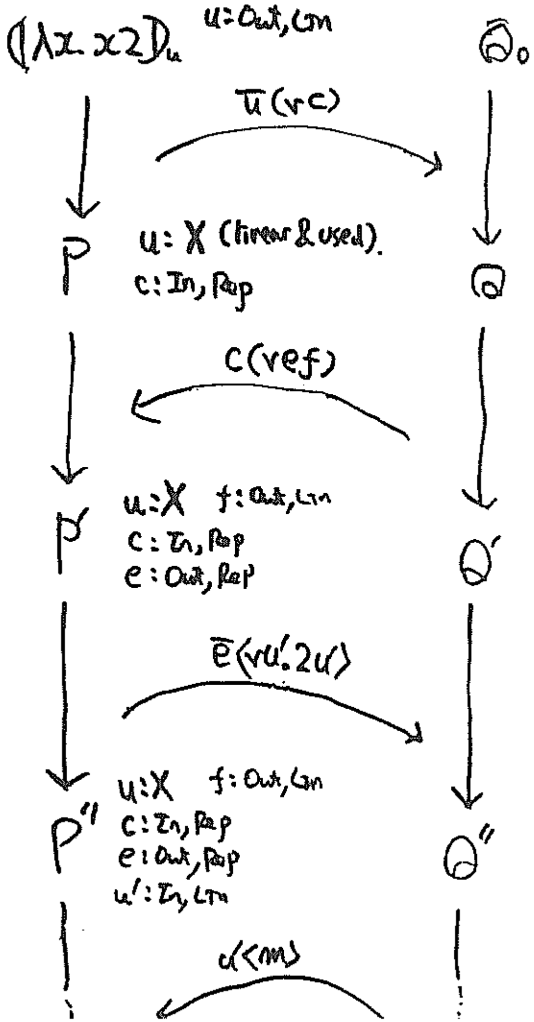
* "Odd" corresponds to "odd turn"; similarly for "even".

1	2	3	4	5	....
Out	In	Out	In	Out	...... //

Sorting of λν-Interaction (6) λν-Sorting, 3.

Example: the interaction of $(\!|\lambda x^{Nat\Rightarrow Nat}.\, x2|\!)$ which lies within $beh(Nat \Rightarrow Nat)_u$.





Comments:

* "e" will be used many times if x occurs several times, e.g. $\lambda x.\, x(x2)$. So Rep is correct.
* Only "well-typed" actions are

Two Global Constraints (1) Determinacy.

- At present, we have the following process p s.t. $p \trianglelefteq beh(Nat \Rightarrow Nat)_u$:

$$\overline{u}(\nu c).\, c(\nu e.e3).\, (\overline{e}\langle 6\rangle + \overline{e}\langle 7\rangle),$$

which returns 6 or 7 nondeterministically after receiving 3.

- To eliminate such a behaviour, we use the following axiom.

Definition (determinacy).

A process $p \trianglelefteq beh(Nat \Rightarrow Nat)_u$ is O-deterministic if $p \xrightarrow{s} p'$ implies:

[1] If $p' \xrightarrow{a} p''_{1,2}$, then $p''_1 \sim p''_2$.

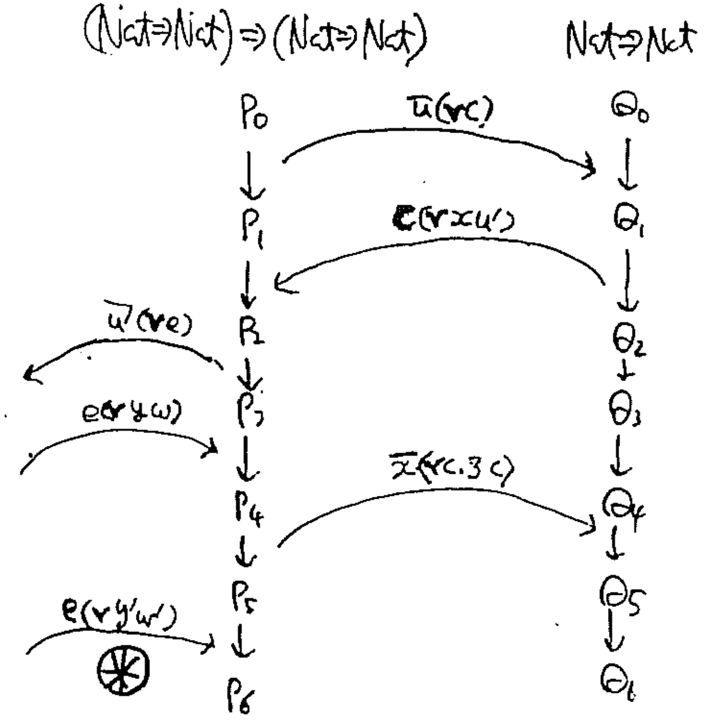
[2] If $p' \xrightarrow{\tau} p''$, then $p'' = \tau.p'$.

[3] If $p' \xrightarrow{\uparrow\alpha}$ and $p' \xrightarrow{\uparrow\beta}$, then $\alpha \equiv_\alpha \beta$. //

* Determinacy is a global constraint on behaviour, not specifiable by allowance and consequence.

Two Grobal Constraints (2) switching, 1.

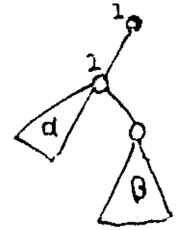
- Consider the following interaction sequence allowed in $\mathrm{Beh}((\mathrm{Nat} \Rightarrow \mathrm{Nat}) \Rightarrow (\mathrm{Nat} \Rightarrow \mathrm{Nat}))_u$:



- Notice that, when composed with argument on the right, the "visible" interaction is no longer strictly alternating, even though the whole interaction is.
- This is because of the input ⊛, which "at the left component" NOT strictly alternating.
- How can we amend this?

Two Global Constraints (3) Switching (2)

- Given $p \trianglelefteq \mathrm{Beh}(\alpha \Rightarrow \beta)_u$, visible actions of p belong to either α or β except the first two actions. We call α and β, the components of an action.



<u>Definition</u> (switching condition)

Given $p \trianglelefteq \mathrm{Beh}(\alpha \Rightarrow \beta)_u$, if $p \overset{s}{\Longrightarrow} \overset{l_1}{\Rightarrow} \overset{l_2}{\Rightarrow}$ and the componts of l_1 and l_2 are different, then we say p switches at sl_1 by l_2. Then p switches properly if, whenever p switches at sl_1 by l_2, l_1 is input and l_2 is output, for any sl_1 and l_2. //

* The input action ⊛ in the example does not obey the above condition.

* On the other hand it is clear this condition is enough to guarantee strict alternation after composition. //

Some Terminology.

- In games semantics, one uses the following terminology, with good reasons (see later):

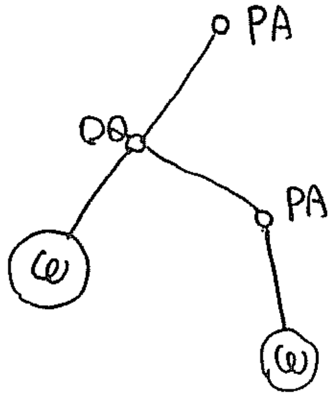
Player : output.

Opponent : input

Answer : action with linear usage.

Question : action with non-linear usage.

So the tree for Nat ⇒ Nat becomes:

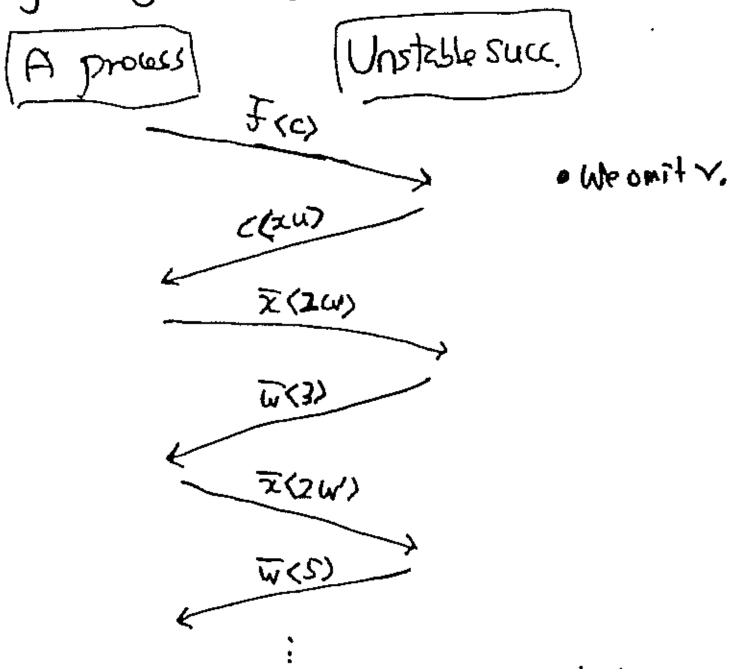


5

Games from a π-calculus viewpoint (2).

Determinacy is not enough.

- Think of the following O-deterministic interaction:



- Why $[\![\mathrm{succ}]\!]_u \overset{\mathrm{def}}{=} \bar{u}(\nu c).!c(nw).\bar{w}\langle n+1\rangle$ never behaves like this?
- Because it is "banged" at c; whenever it is asked at c, it reacts with a fresh copy of the code.
- This idea is formalised using innocence, the ingenious idea due to Hyland and Ong.

Innocence (1) Views.

Notation: Given $s = s_1 l_1 s_2 l_2 s_3$ which is in $\mathrm{Beh}(\alpha)_u$, we write

$$s_1 \; l_1 \; s_2 \; l_2 \; s_3$$

when the subject of l_2 is bound by the object of l_1, e.g.

$$s_1 \; \underline{\bar{e}(\nu c)} \; s_2 \; \underline{\bar{c}\langle 3\rangle} \; s_3 .$$

We then say l_1 justifies l_2.

Definition (view) ([HO94, McCusker96])

Given s in some $\mathrm{Beh}(\alpha)_u$, we define $\ulcorner s \urcorner$ and $\llcorner s \lrcorner$ by:

$$\ulcorner \varepsilon \urcorner = \llcorner \varepsilon \lrcorner = \varepsilon \qquad \ulcorner l_1 \urcorner = \llcorner l_1 \lrcorner = l_1$$

$$\ulcorner s_1 l_1 s_2 l_2^{\mathrm{IN}} \urcorner = \ulcorner s_1 \urcorner l_1 l_2 \qquad \ulcorner s\, l^{\mathrm{out}} \urcorner = \ulcorner s \urcorner l^{\mathrm{out}}$$

$$\llcorner s_1 l_1 s_2 l_2^{\mathrm{OUT}} \lrcorner = \llcorner s_1 \lrcorner l_1 l_2 \qquad \llcorner s\, l^{\mathrm{IN}} \lrcorner = \llcorner s \lrcorner l^{\mathrm{IN}}. \quad //$$

* $\ulcorner s \urcorner$ is often called the P-view of s, while $\llcorner s \lrcorner$ is the O-view. Views are contexts by which a process determines its action.

Examples of Justification and View.

• Interaction by:

$$[\![\lambda x.\lambda y.x+y]\!]_u$$

assuming addition is incorporated as a constant. Below we omit ν as before.

"add3" begins.

$\overline{u}(c).\, c\langle x u'\rangle.\, \overline{x}(e).\, e\langle 3\rangle.\, \overline{u'}(c').\, c'(yw).\, \overline{y}(f).\, \underline{f\langle 5\rangle}.\, \overline{w}\langle 8\rangle.$

$c'(y'w').\, \overline{y'}(f').\, f'\langle 6\rangle.\, \overline{w'}\langle 9\rangle.$

"add5" begins.

$c(x'u'').\, \overline{x'}(e').\, e'\langle 5\rangle.\, \overline{u''}(c'').\, \ldots\ldots$

A new "add3" begins.

$c(x''u''').\, \overline{x''}(e'').\, e''\langle 3\rangle.\, \overline{u'''}(c''').\, c'''(y''w'').\, \overline{y''}(f'').\, \underline{f''\langle 5\rangle}.\, \overline{w''}\langle 8\rangle.$

• Note two P-views, one starting from $f\langle 5\rangle$ and another from $f''\langle 5\rangle$ is the same up to $\equiv_\alpha$.

Innocence (2) Visibility and Innocence.

Definition

(i) (visibility) Given $s l^{IN}$ (resp. $s l^{OUT}$) in $Beh(\alpha)_u$, l is visible from s when l is justified from $\ulcorner s \urcorner$ (resp. $\llcorner s \lrcorner$). A sequence s in $Beh(\alpha)_u$ satisfies the visibility condition when each action in s is either free or visible from the preceding sequence.

(ii) (Innocence, [HO94]) A process $p \trianglelefteq Beh(\alpha)_u$ is innocent when it satisfies the visibility condition and, if

$$p \stackrel{s_1 l_1^{IN}}{\Longrightarrow} p_1 \;\wedge\; p \stackrel{s_2 l_2^{IN}}{\Longrightarrow} p_2 \;\wedge$$
$$\ulcorner s_1 l_1 \urcorner \equiv_\alpha \ulcorner s_2 l_2 \urcorner \;\wedge\; p_1 \stackrel{l_1'^{OUT}}{\Longrightarrow},$$

then we have

$$p_2 \stackrel{l_2'^{OUT}}{\Longrightarrow} \quad \text{s.t. } \ulcorner s_1 l_1 l_1' \urcorner \equiv_\alpha \ulcorner s_2 l_2 l_2' \urcorner. \quad //$$

* Innocence says:

"The same context, the same action."

Exercise: Show visibility implies proper switching...

Well-bracketing.

- Another significant constraint which λν-interaction obeys is well-bracketing. Let us call "Req" actions, "Questions", and "Ch" actions "Answers". Then the example of interactions (of $(\!|\lambda x.\lambda a.x+5|\!)_u$) becomes:

$$\underset{A}{\overline{u}(c)}.\underset{Q}{c\langle u'\rangle}.\underset{Q}{\overline{x}(e)}.\underset{A}{e\langle 3\rangle}.\underset{A}{\overline{u'}(c')}.\underset{Q}{c'(y\,w)}.\underset{Q}{\overline{y}(f)}.\underset{A}{f\langle 5\rangle}.\underset{A}{\overline{w}\langle 8\rangle}.$$

$$\underset{Q}{c'(y'w')}.\underset{Q}{\overline{y'}(f')}.\underset{A}{f'\langle 7\rangle}.\underset{A}{\overline{w'}\langle 9\rangle}.$$

- Let us say, when we have $s_1 \underset{Q}{l_1} s_2 \underset{A}{l_2} s_3$, l_2 answers l_1.

Now we define!

<u>Definition</u> (well-bracketing)

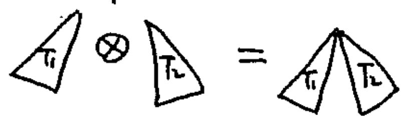
A sequence s in $beh(\alpha)_u$ is well-bracketing when a later asked question is always answered first. $\rho \trianglelefteq beh(\alpha)_u$ is well-bracketing if all sequences in ρ is well-bracketing. ■

Constructing Category (1)

- We extend our structures so that they can model open terms. The resulting universe is then turned into a category of call-by-value computation, with rich internal structures.

<u>Definition.</u>

(i) (extended λν-sorting) We extend sorting trees by closing them by the constructor ⊗ given by:



,

as well as ⇒, given as before, and the constant $\bullet^{\mathrm{out,in}}$ of a singleton tree. Γ, Δ, ... range over extended sorting trees.

(ii) (open λν-sorting). An open λ-sorting is a forest of form $\overline{\Gamma} \uplus \Delta$ where $\overline{\Gamma}$ is the result of reversing input/output. We write $\Gamma^u \to \Delta^v$ to denote an open λ-sorting whose roots are assigned names u and v, respectively.

Constructing a Category (2)

Notation.

Given p and q such that $\mathrm{surf}(p) \cup \mathrm{surf}(q) = A$, if $p^A | q^A : X$ for some (basic) sorting X, then we write: $p \circ q$ for $(\nu\, \mathrm{surf}(p) \cap \mathrm{surf}(q))(p|q)$. //

Definition. (Λ_ν)

Λ_ν is given by the following data:

(i) Types: All open λ-sortings of form $\Gamma^u \to \Delta^v$.

(ii) Typed Processes: We define $p : \Gamma^u \to \Delta^v$ by:

$$p : \Gamma^u \to \Delta^v \Leftrightarrow p \trianglelefteq \mathrm{beh}(\Gamma^u \to \Delta^v) \wedge p \text{ innocent} \wedge p \text{ deterministic} \wedge p \text{ well-bracketed}.$$

where $\mathrm{beh}(\Gamma^u \to \Delta^v)$ is given as before, except we reverse "odd" and "even" when constructing $\alpha \prec \Gamma^u \to \Delta^v$.

(iii) Typed Operations: Given pairwise distinct u, v, w,

$$p : \Gamma^u \to \Delta^v \,;\, q : \Delta^v \to \Xi^w = p \circ q : \Gamma^u \to \Xi^w. \quad //$$

Constructing Category (3)

Proposition.

The operation ; in Λ_ν is well-defined, i.e. whenever $p : \Gamma^u \to \Delta^v$ and $q : \Delta^v \to \Xi^w$ for pairwise distinct u, v, w, we have $p \circ q : \Gamma^u \to \Xi^w$.

Proof: This is essentially because innocence is preserved by the construction. See [HY97]. ∎

Remark.

(i) Types in Λ_ν can be made more abstract: we can even start from general λ_ν-sorting structures without mentioning syntactic types.

(ii) The encoding also enjoys the property that any opened input becomes immediately active. This can be incorporated as "contingency completeness" [HO94], though we omit it. //

Constructing Category. (4).

<u>Definition</u> (Category of $\lambda\nu$-interaction).
$Cat(\Lambda_\nu)$ is given by the following data:

[1] Objects: All extended λ-sorting trees.

[2] Arrows: $f : \Gamma \to \Delta$ iff, for some $\rho_0 : \Gamma^u \to \Delta^v$,

$$f \stackrel{\text{def}}{=} \{(uv)\rho \mid \rho : \Gamma^u \to \Delta^v,\ \rho \approx \rho_0\},$$

where $(uv)\rho$ is abstraction [Milner 92], i.e. a map from two distinct names to a denotation of ρ. The composition is given by:

$$f ; g \stackrel{\text{def}}{=} (u)(w)\left(f[uv] \circ g[vw]\right)$$

where u, v, w are pairwise distinct and operations are pointwise given. //

<u>Remark</u>: Since all processes are σ-deterministic, we can simply truncate all τ-actions. This gives (via abstraction) the representative for each arrow. //

Constructing Category (5)

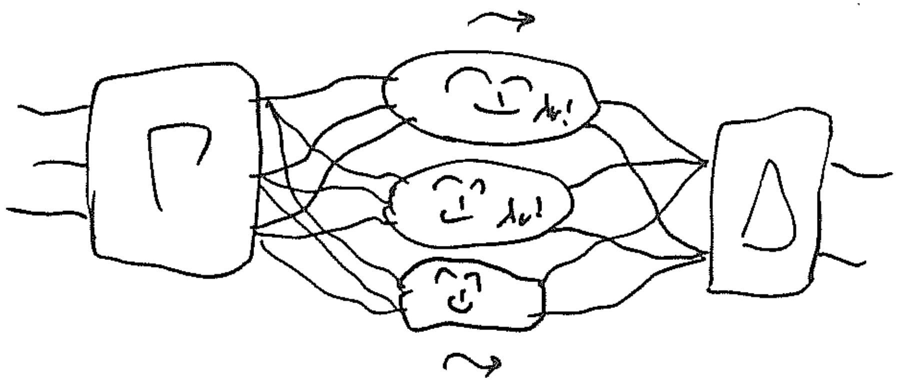
- We finally conclude by stating:

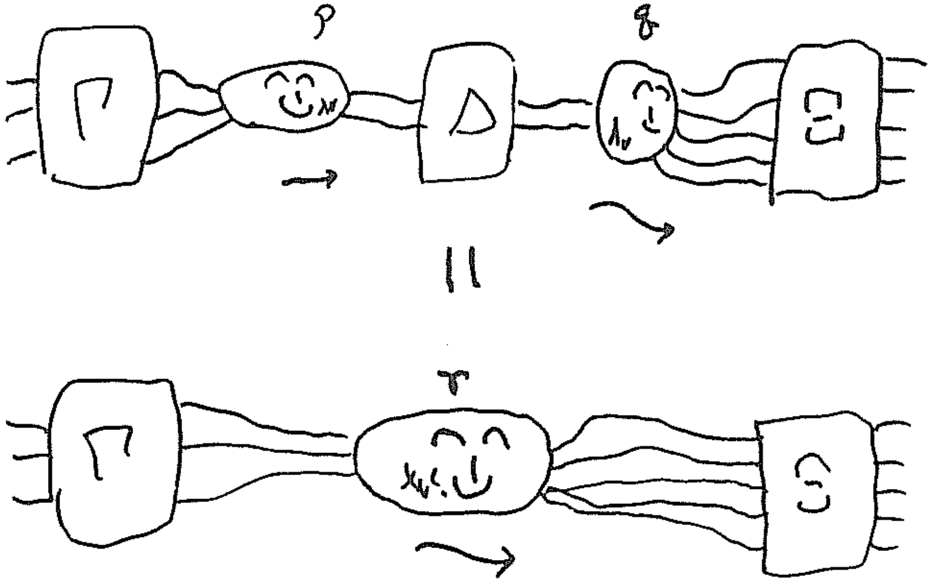
<u>Proposition</u>
$Cat(\Lambda_\nu)$ is a category.

Proof: See [HY97]. Note associativity directly comes from that of 1. The identity is given by the dynamic link. ∎

- $Cat(\Lambda_\nu)$ offers all basic order/type structures to soundly model e.g. PCF_ν. With a slight extension, we can also own a category suitable to model FPC. For related results, see [HY97, AHM98, FH98]. //

Cat(Λ_v).





$p \circ q \approx r$.

6 Conclusion.

What We Have Done.

- In the present lectures, we saw:
  - A sorting Γ is about dynamics; more precisely, it specifies possible interactive behaviours of both processes and their environments.
  - A large universe of types for sortings, analogous in construction to universes of function types which however classifies interactions rather than functions.
  - How games arise as significant instances of theories of types for mobile processes, offering a structural embedding of function types into process types.
- These would give a deeper understanding on the existing subjects; they would also offer a basis for our study in future.

Further Directions.

(1) Application of the methods/ideas in Part I/II to the study of process types both in theory and practice.

- the precise understanding of the behavioural content of existing syntactic notions of types.
- integration of existing ideas based on semantic understanding.
- development of new (syntactic) type disciplines starting from behavioural analysis.
- the use of behavioural notions of types for description/analysis of conspicuous computation/interaction structures.

Further Directions (contd)

(2) Further study of functional computation and types in the context of process types, through basic encodings and games. Hopefully the presented approach will lead to a new, effective common forum for the two worlds.

Also the convergence of basic structures in Hyland-Ong games and process theory is notable; deeper inspection of the role and the status of the π-calculus for the general semantic study is due.

Notes.

1. A more general form of actions follows, with which all our present apparatus work as before without change:

$$\alpha ::= x\langle \nu A\, \{y_i\}_{i\in I}\rangle \mid \overline{x}\langle \nu A\, \{y_i\}_{i\in I}\rangle \mid \tau.$$

where I ranges over an arbitrary set, and A is a subset of names in $\{y_i\}_{i\in I}$. Making the collection of names to be a proper class and each surface being allowed to be any set of names, the whole constructions work as they are.

We note such constructions are useful for the study of certain typed π-calculi. These actions also underlie various games universes.

(2) We list the definition of contingency completeness omitted from the main section.

<u>Definition.</u>

A process $p \trianglelefteq beh(\alpha)_u$ which is innocent is contingently complete when $p \overset{s}{\Longrightarrow} p'$ and $beh(\alpha)_u \overset{s l^{IN}}{\Longrightarrow}$ with $s l^{IN}$ satisfying the visibility condition, implies $p' \overset{l}{\Longrightarrow}$. The case when a process is on an open λ-sorting is similarly defined. //

Not only does this conform to our λμ-encoding but this gives a simpler representation of processes via innocent functions since only output behaviours matter once a process is contingently complete. //

References

In the following, the titles of the papers referred to in the copy of the slides is given.

- `[AHM 98]` Abramsky, S., Honda, K. and McCusker, G. Fully Abstract Game Semantics for General Reference. *LICS'98*, IEEE, 1998.
- `[FH 98]` Fiore, M. and Honda, K. Recursive Types in Games: Axiomatics and Process Representation. *LICS'98*, IEEE, 1998.
- `[HO 94]` Hyland, M. and Ong, L., On Full Abstraction for PCF: I, II and III. 130 pages. ftp-able at `theory.doc.ic.ac.uk/papers/Ong`, 1994.
- `[HO 95]` Hyland, M. and Ong, L., Pi-calculus, dialogue games and PCF, Proceedings of 7th Annual ACM Conference on Functional Programming Languages and Computer Architecture, June 26-28, 1995.
- `[HY 97]` Honda, K. and Yoshida, N. Game-theoretic Analysis of Call-by-value Computation. *Proc. of ICALP'97*, volume 1256 of *Lecture Notes in Computer Science*. pp.225–236, Springer-Varlag, 1997. (Full version available by from `ftp.dcs.ed.ac./export/kohei/cbvfull..gz`.)
- `[MPW 89]` Milner, R., Parrow, J.G. and Walker, D.J., A Calculus of Mobile Processes, *Information and Computation* 100(1), pp.1–77, 1992.
- `[Milner 89]` Milner, R., Communication and Concurrency, 1989, Prentice-Hall.
- `[Milner 90]` Milner, R., Functions as Processes. *Mathematical Structure in Computer Science*. 2(2), pp.119–146, 1992.
- `[Milner 92]` Milner, R., Polyadic π-Calculus: a tutorial. *Proceedings of the International Summer School on Logic Algebra of Specification*, Marktoberdorf, 1992.
- `[Sangiorgi 96]` Sangiorgi, D. π-calculus, internal mobility, and agent-passing calculi. *TCS*, 167(2):235–271, 1996.
- `[Walker 91]` Walker, D. Objects in the π-calculus. *Information and Computation*, ukftpVol. 116, pp.253–271, 1995.

I also note that I once made an annotated bibliography of basic papers on π-calculus as a guidance to those who are new to the field. This may be useful if you are not familiar with the literature on π-calculus in general.

- Honda, K., Selected Bibliography on Mobile Processes, 1998. Available from `http://www.cs.auc.dk/mobility/`.