Course on Mobility

`Daniel.Hirschkoff@ens-lyon.fr`

About this course

- focus on the π-calculus: a calculus of mobile processes based on *naming* (cf. R. Milner, Turing award lecture)

About this course

- focus on the π-calculus: a calculus of mobile processes based on *naming* (cf. R. Milner, Turing award lecture)

- π as a $\begin{array}{l}\text{specification}\\ \text{programming}\end{array}$ language

About this course

- focus on the π-calculus: a calculus of mobile processes based on *naming* (cf. R. Milner, Turing award lecture)

- π as a $\begin{array}{l}\text{specification}\\ \text{programming}\end{array}$ language

- more a panorama than a precise technical study of a particular poi

About this course

- focus on the π-calculus: a calculus of mobile processes based on *naming* (cf. R. Milner, Turing award lecture)

- π as a $\begin{array}{l}\text{specification}\\ \text{programming}\end{array}$ language

- more a panorama than a precise technical study of a particular poi

- outline:
  π: definition - types - λ in π - behavioural equivalences

Origins and sources

- predecessors: other process algebras – CSP, CCS

Origins and sources

- predecessors: other process algebras – CSP, CCS
- books:

R. Milner, *Communication and Concurrency*, Prentice Hall

R. Milner, *Communicating and Mobile Systems: the π-calculus*, CUP

D. Sangiorgi, D. Walker, *The π-calculus, a Theory of Mobile Computation*, CUP

Origins and sources

- predecessors: other process algebras – CSP, CCS
- books:

R. Milner, *Communication and Concurrency*, Prentice Hall

R. Milner, *Communicating and Mobile Systems: the π-calculus*, CUP

D. Sangiorgi, D. Walker, *The π-calculus, a Theory of Mobile Computation*, CUP

- notes for the course:
  not a tutorial, more to be used as a reference with the slides

Names and Processes

- nominal calculus:
  an infinite set of *names* (*channels, links, ports*)

$$a, b, \ldots, p, q, r, \ldots, x, y, \ldots$$

- we define *terms* (*processes*)

$$A, B, \ldots, P, Q, \ldots$$

Interaction, reduction, communication

$$P \quad = \quad \overline{a}\langle v\rangle.b(x).\mathbf{0} \quad | \quad a(y).\left(\overline{c}\langle y\rangle.\mathbf{0} \mid \overline{d}\langle y\rangle.\mathbf{0}\right)$$

Interaction, reduction, communication

$$
\begin{array}{rcl}
P & = & \overline{a}\langle v\rangle.b(x).\mathbf{0} \quad | \quad a(y).\big(\overline{c}\langle y\rangle.\mathbf{0} \mid \overline{d}\langle y\rangle.\mathbf{0}\big) \\
& & \downarrow \\
& & b(x).\mathbf{0} \quad | \quad \overline{c}\langle v\rangle.\mathbf{0} \quad | \quad \overline{d}\langle v\rangle.\mathbf{0}
\end{array}
$$

Interaction, reduction, communication

$$\begin{array}{ccc} P & = & \overline{a}\langle v\rangle.b(x).\mathbf{0} \quad | \quad a(y).\big(\overline{c}\langle y\rangle.\mathbf{0} \mid \overline{d}\langle y\rangle.\mathbf{0}\big) \\ & & \downarrow \\ & & b(x).\mathbf{0} \quad | \quad \overline{c}\langle v\rangle.\mathbf{0} \quad | \quad \overline{d}\langle v\rangle.\mathbf{0} \end{array}$$

competition for a resource:

$$Q \quad = \quad a(x).Q_1 \mid a(x).Q_2 \mid \overline{a}\langle v\rangle.\mathbf{0}$$

Interaction, reduction, communication

$$\begin{tikzcd}[column sep=small] P = & \overline{a}\langle v\rangle.b(x).\mathbf{0} \mid a(y).\big(\overline{c}\langle y\rangle.\mathbf{0} \mid \overline{d}\langle y\rangle.\mathbf{0}\big) \arrow[d] \\ & b(x).\mathbf{0} \mid \overline{c}\langle v\rangle.\mathbf{0} \mid \overline{d}\langle v\rangle.\mathbf{0} \end{tikzcd}$$

competition for a resource:

$$\begin{tikzcd}[column sep=small] & Q = a(x).Q_1 \mid a(x).Q_2 \mid \overline{a}\langle v\rangle.\mathbf{0} \arrow[dl] \arrow[dr] & \\ Q_{1\{x\leftarrow v\}} \mid a(x).Q_2 \mid \mathbf{0} & & a(x).Q_1 \mid Q_{2\{x\leftarrow b\}} \mid \mathbf{0} \end{tikzcd}$$

non confluence

A single entity: names

- prefixes:

$$a(b).\ \text{reception}, \quad \overline{a}\langle b\rangle.\ \text{emission} \qquad \begin{cases} a\text{: } \textit{subject} \\ b\text{: } \textit{object} \end{cases}$$

A single entity: names

- prefixes:

$$a(b).\ \text{reception},\quad \overline{a}\langle b\rangle.\ \text{emission}\qquad \begin{cases} a\text{: } \textit{subject} \\ b\text{: } \textit{object} \end{cases}$$

- communication:

▷ synchronisation on a channel

▷ substitution of a name with a name ($\neq \lambda$)

A single entity: names

- prefixes:

$$a(b).\ \text{reception},\quad \overline{a}\langle b\rangle.\ \text{emission}\qquad \begin{cases} a\text{: } \textit{subject} \\ b\text{: } \textit{object} \end{cases}$$

- communication:

▷ synchronisation on a channel

▷ substitution of a name with a name ($\neq \lambda$)

- often use names like x, y in input object (bound name)

A single entity: names

- prefixes:

$$a(b).\ \text{reception}, \quad \overline{a}\langle b\rangle.\ \text{emission} \qquad \begin{cases} a\text{: } \textit{subject} \\ b\text{: } \textit{object} \end{cases}$$

- communication:
  - ▷ synchronisation on a channel
  - ▷ substitution of a name with a name ($\neq \lambda$)

- often use names like x, y in input object (bound name)

- notation: $\overline{a}\langle b\rangle.\mathbf{0}$ is often written $\overline{a}\langle b\rangle$

Another process

$$\overline{a}\langle c\rangle.\overline{c}\langle v\rangle.\mathbf{0}$$

Another process

$$\overline{a}\langle c\rangle.\overline{c}\langle v\rangle.\mathbf{0} \mid a(x).x(t).\overline{r}\langle t\rangle.\mathbf{0}$$

Another process

$$\begin{array}{c}\overline{a}\langle c\rangle.\overline{c}\langle v\rangle.\mathbf{0} \mid a(x).x(t).\overline{r}\langle t\rangle.\mathbf{0}\\ \downarrow \\ \overline{c}\langle v\rangle.\mathbf{0} \mid c(t).\overline{r}\langle t\rangle.\mathbf{0}\end{array}$$

Another process

$$\overline{a}\langle c\rangle.\overline{c}\langle v\rangle.\mathbf{0} \mid a(x).x(t).\overline{r}\langle t\rangle.\mathbf{0}$$

$$\downarrow$$

$$\overline{c}\langle v\rangle.\mathbf{0} \mid c(t).\overline{r}\langle t\rangle.\mathbf{0}$$

$$\downarrow$$

$$\mathbf{0} \mid \overline{r}\langle v\rangle.\mathbf{0}$$

Another process

$$\overline{a}\langle c\rangle.\overline{c}\langle v\rangle.\mathbf{0} \mid a(x).x(t).\overline{r}\langle t\rangle.\mathbf{0}$$

$$\downarrow$$

$$\overline{c}\langle v\rangle.\mathbf{0} \mid c(t).\overline{r}\langle t\rangle.\mathbf{0}$$

$$\downarrow$$

$$\mathbf{0} \mid \overline{r}\langle v\rangle.\mathbf{0}$$

- a form of *reference passing*
- ▷ object ↪ subject: $\overline{a}\langle c\rangle.\overline{c}\langle v\rangle$, $a(x).x(t).\overline{r}\langle t\rangle$

Another process

$$\overline{a}\langle c\rangle.\overline{c}\langle v\rangle.\mathbf{0} \mid a(x).x(t).\overline{r}\langle t\rangle.\mathbf{0}$$

$$\downarrow$$

$$\overline{c}\langle v\rangle.\mathbf{0} \mid c(t).\overline{r}\langle t\rangle.\mathbf{0}$$

$$\downarrow$$

$$\mathbf{0} \mid \overline{r}\langle v\rangle.\mathbf{0}$$

- a form of *reference passing*
- ▷ object ↪ subject: $\overline{a}\langle c\rangle.\overline{c}\langle v\rangle$, $a(x).x(t).\overline{r}\langle t\rangle$
- ▷ *name passing:* the king of France, Google

Another process

$$\overline{a}\langle c\rangle.\overline{c}\langle v\rangle.\mathbf{0} \mid a(x).x(t).\overline{r}\langle t\rangle.\mathbf{0}$$

$$\downarrow$$

$$\overline{c}\langle v\rangle.\mathbf{0} \mid c(t).\overline{r}\langle t\rangle.\mathbf{0}$$

$$\downarrow$$

$$\mathbf{0} \mid \overline{r}\langle v\rangle.\mathbf{0}$$

- a form of *reference passing*
- ▷ object ↪ subject: $\overline{a}\langle c\rangle.\overline{c}\langle v\rangle$, $a(x).x(t).\overline{r}\langle t\rangle$
- ▷ *name passing:* the king of France, Google

- we have *added a context:* $\overline{a}\langle c\rangle.\overline{c}\langle v\rangle.\mathbf{0}$

Another process

$$\overline{a}\langle c\rangle.\overline{c}\langle v\rangle.\mathbf{0} \mid a(x).x(t).\overline{r}\langle t\rangle.\mathbf{0}$$

$$\downarrow$$

$$\overline{c}\langle v\rangle.\mathbf{0} \mid c(t).\overline{r}\langle t\rangle.\mathbf{0}$$

$$\downarrow$$

$$\mathbf{0} \mid \overline{r}\langle v\rangle.\mathbf{0}$$

- a form of *reference passing*
- ▷ object ↪ subject: $\overline{a}\langle c\rangle.\overline{c}\langle v\rangle$, $a(x).x(t).\overline{r}\langle t\rangle$
- ▷ *name passing:* the king of France, Google

- we have *added a context:* $\overline{a}\langle c\rangle.\overline{c}\langle v\rangle.\mathbf{0} \quad \mid a(x).x(t).\overline{r}\langle t\rangle.\mathbf{0}$

this is the way we reason on π-calculus terms

λ versus π

λ: functions that are applied to their arguments (β-reduction)
π: names being exchanged ($\simeq$ β_0-reduction)

λ versus π

λ: functions that are applied to their arguments (β-reduction)
π: names being exchanged ($\simeq$ β_0-reduction)

λ: a term being reduced, an evaluation that is going on
π: a term *in a context*

λ versus π

λ: functions that are applied to their arguments (β-reduction)
π: names being exchanged ($\simeq$ β_0-reduction)

λ: a term being reduced, an evaluation that is going on
π: a term *in a context*

λ: several kinds of reduction

- ▷ strategies (call-by-name, call-by-value,. . .)
- ▷ computing everywhere in the term (rule ξ)

π: reduction only "at top-level", *non deterministically*

Exercise: matching

- some π-calculi include a matching operator:
  $[n = m]\, P$ behaves like P if $n = m$, is stuck otherwise

examples:

▷ $a(x).b(y).[x = y]\, \overline{c}\langle x \rangle$ forwards a name if received twice

▷ $(\boldsymbol{\nu} y)\, a(x).[x = y]\, P$ is equivalent to $\mathbf{0}$

Exercise: matching

- some π-calculi include a matching operator:
  $[n = m]\, P$ behaves like P if $n = m$, is stuck otherwise

examples:

▷ $a(x).b(y).[x = y]\,\overline{c}\langle x\rangle$ forwards a name if received twice

▷ $(\boldsymbol{\nu} y)\, a(x).[x = y]\, P$ is equivalent to $\mathbf{0}$

- is matching encodable in a π-calculus without matching operator?

Restriction operator, $\boldsymbol{\nu}$

$(\boldsymbol{\nu} a)\, P$: the process P in which name a is *private*
(unknown to any other process, unknown to *the context*)

Restriction operator, $\boldsymbol{\nu}$

$(\boldsymbol{\nu} a)\, P$: the process P in which name a is *private*
(unknown to any other process, unknown to *the context*)

other interpretation: create a *new* name a, then execute P

Restriction operator, $\boldsymbol{\nu}$

$(\boldsymbol{\nu}a)\, P$: the process P in which name a is *private*
(unknown to any other process, unknown to *the context*)

other interpretation: create a *new* name a, then execute P

Example: $T \;=\; (\boldsymbol{\nu}a)\,(\overline{a}\langle v\rangle \,|\, a(x).Q_1) \;\;|\;\; a(y).Q_2$

→ no communication with “Q_2”

Restriction operator, $\boldsymbol{\nu}$

$(\boldsymbol{\nu} a)\, P$: the process P in which name a is *private*
(unknown to any other process, unknown to *the context*)

other interpretation: create a *new* name a, then execute P

Example: $T \;=\; (\boldsymbol{\nu} a)\,(\overline{a}\langle v\rangle \,|\, a(x).Q_1) \;\;|\;\; a(y).Q_2$
$\rightarrow$ no communication with "Q_2"

Remarks:

- $\boldsymbol{\nu}$ is a binder: T is α-equivalent to
  $(\boldsymbol{\nu} a')\left(\overline{a'}\langle v\rangle \,|\, a'(x).Q_{1\{a\leftarrow a'\}}\right) \,|\, a(y).Q_2$ (a' *fresh* name)

Restriction operator, $\boldsymbol{\nu}$

$(\boldsymbol{\nu} a)\, P$: the process P in which name a is *private*
(unknown to any other process, unknown to *the context*)

other interpretation: create a *new* name a, then execute P

Example: $\quad T \;=\; (\boldsymbol{\nu} a)\,(\overline{a}\langle v\rangle \,|\, a(x).Q_1) \;\;|\;\; a(y).Q_2$

$\rightarrow$ no communication with "Q_2"

Remarks:

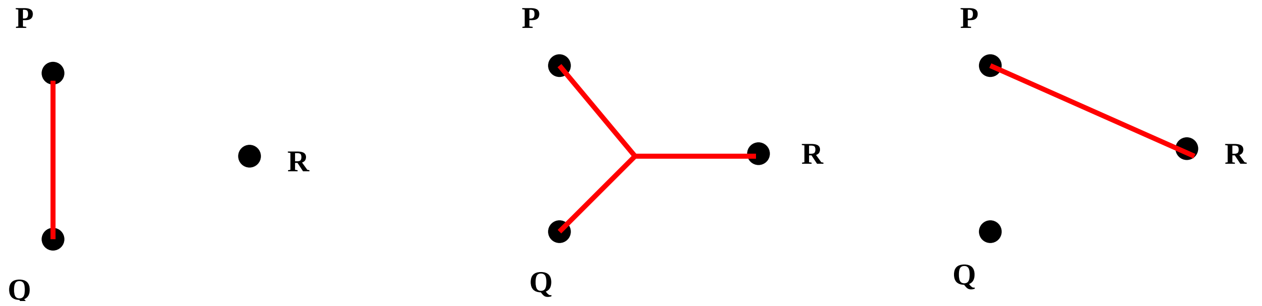
- $\boldsymbol{\nu}$ is a binder: T is α-equivalent to
  $(\boldsymbol{\nu} a')\left(\overline{a'}\langle v\rangle \,|\, a'(x).Q_{1\{a \leftarrow a'\}}\right) \,|\, a(y).Q_2$ $\quad$ (a' *fresh* name)
- $\boldsymbol{\nu}$ has greater priority than $|$

Name extrusion

the object of an output is a restricted name

$$(\boldsymbol{\nu} c)\,(P \mid \overline{a}\langle c\rangle.Q) \mid a(x).R \quad\to\quad (\boldsymbol{\nu} c)\,(P \mid Q \mid R_{\{x\leftarrow c\}}) \quad\equiv\quad (\boldsymbol{\nu} c)\,(P|R_{\{x\leftarrow c\}}) \mid Q$$

if $c \notin \mathsf{fn}(Q)$



→ ‘network topology’ is changing along computation

Exercise: localised π

- grammar so far: $P \;::=\; \mathbf{0} \mid P_1 \mid P_2 \mid a(b).P \mid \overline{a}\langle b\rangle.P \mid (\boldsymbol{\nu} n)\, P$

Exercise: localised π

- grammar so far: $P \ ::= \ \mathbf{0} \mid P_1 \mid P_2 \mid a(b).P \mid \overline{a}\langle b\rangle.P \mid (\boldsymbol{\nu} n)\, P$
- localised π: in $a(b).P$, b can only be used in *output*

$\hookrightarrow$ why the name *"localised π"*?

(consider a term of the form $(\boldsymbol{\nu} n)\, P$)

The polyadic π-calculus

- possibility of exchanging *name tuples*:

$$\overline{a}\langle u, v\rangle.P \mid a(x, y).Q \quad \longrightarrow \quad P \mid Q_{\{x,y \leftarrow u,v\}}$$

The polyadic π-calculus

- possibility of exchanging *name tuples*:

$$\overline{a}\langle u, v\rangle.P \mid a(x, y).Q \quad \longrightarrow \quad P \mid Q_{\{x,y \leftarrow u,v\}}$$

- remark: “type” errors

$$\overline{a}\langle u, v, w\rangle.P \mid a(x, y).Q \quad \longrightarrow \quad \text{??}$$

The polyadic π-calculus

- possibility of exchanging *name tuples*:

$$\overline{a}\langle u, v\rangle.P \mid a(x, y).Q \quad \longrightarrow \quad P \mid Q_{\{x,y \leftarrow u,v\}}$$

- remark: “type” errors

$$\overline{a}\langle u, v, w\rangle.P \mid a(x, y).Q \quad \longrightarrow \quad ??$$

- notation:

$a().P$ (resp. $\overline{a}\langle\rangle.P$) is written $a.P$ (resp. $\overline{a}.P$): cf. $\boxed{\text{CCS}}$

Booleans in the polyadic π-calculus

- an *abstraction:* true $\overset{\text{def}}{=} (t, f).\bar{t}$

cf. Milner's tutorial on π, abstractions and concretions

Booleans in the polyadic π-calculus

- an *abstraction:* true $\overset{\text{def}}{=}$ $(t, f).\bar{t}$

  cf. Milner's tutorial on π, abstractions and concretions

- the value true located at b: $\text{true}_b \overset{\text{def}}{=} b(t, f).\bar{t}$

Booleans in the polyadic π-calculus

- an *abstraction:* true $\overset{\text{def}}{=} (t, f).\bar{t}$

  cf. Milner's tutorial on π, abstractions and concretions

- the value true located at b: $\text{true}_b \overset{\text{def}}{=} b(t, f).\bar{t}$

- test:

$$\texttt{if } b \texttt{ then } P \texttt{ else } Q \overset{\text{def}}{=} \bar{b}\langle t, f\rangle.(t.P \mid f.Q)$$

Booleans in the polyadic π-calculus

- an *abstraction:* true $\overset{\text{def}}{=}$ $(t, f).\bar{t}$

  cf. Milner's tutorial on π, abstractions and concretions

- the value true located at b: $\text{true}_b \overset{\text{def}}{=} b(t, f).\bar{t}$

- test:

$$\texttt{if } b \texttt{ then } P \texttt{ else } Q \overset{\text{def}}{=} \bar{b}\langle t, f\rangle.(t.P \mid f.Q)$$

better $\mapsto$ $\overset{\text{def}}{=} (\boldsymbol{\nu} t)(\boldsymbol{\nu} f)\,\bar{b}\langle t, f\rangle.(t.P \mid f.Q)$

Exercises

- write π-calculus terms for boolean $\neg$ and $\wedge$ operators

Exercises

- write π-calculus terms for boolean $\neg$ and $\wedge$ operators

- how can we 'program' the diadic π-calculus in the monadic π-calculus?

$$\overline{a}\langle u, v\rangle.P \mid a(x, y).Q \quad \longrightarrow \quad P \mid Q_{\{x,y \leftarrow u,v\}}$$

Replication

- to have a Turing-complete model (and in particular to be able to define a programming language), one has to have a form of recursion

Replication

- to have a Turing-complete model (and in particular to be able to define a programming language), one has to have a form of recursion

- replication: $\boxed{!P}$

  stands for *as many copies of P as you wish in parallel*

  $(!P \text{ ``}{=}\text{''} \; P \,|\, P \,|\, P \,|\, \ldots)$

Replication

- to have a Turing-complete model (and in particular to be able to define a programming language), one has to have a form of recursion

- replication: $\boxed{!P}$

stands for *as many copies of P as you wish in parallel*

$(!P \text{ “=” } P \,|\, P \,|\, P \,|\, \ldots)$

- examples:

▷ $\overline{a}\langle v\rangle.P \mid !a(x).Q \quad \longrightarrow \quad P \mid Q_{\{x \leftarrow v\}} \mid !a(x).Q$

Replication

- to have a Turing-complete model (and in particular to be able to define a programming language), one has to have a form of recursion

- replication: $\boxed{!P}$

stands for *as many copies of P as you wish in parallel*

$(!P \text{ ``$=$''} \; P \,|\, P \,|\, P \,|\, \ldots)$

- examples:

▷ $\overline{a}\langle v\rangle.P \mid !a(x).Q \quad \longrightarrow \quad P \mid Q_{\{x \leftarrow v\}} \mid !a(x).Q$

▷ let $T \overset{\text{def}}{=} !\overline{c}\langle x\rangle \mid !c(y)\,, \qquad T \longrightarrow T$

→ *the replication operator brings* persistence

Replication and persistence

- persistent data

$$\mathsf{true}_b \overset{\mathsf{def}}{=} \,!b(t, f).\bar{t}$$

Replication and persistence

- persistent data

$$\mathsf{true}_b \overset{\mathsf{def}}{=} \; !b(t,f).\bar{t}$$

- a resource: server for boolean $\vee$

$$!l(b_1,b_2,r).(\boldsymbol{\nu} b)\left(!b(t,f).(\boldsymbol{\nu} f')\left(\overline{b_1}\langle t,f'\rangle \mid f'.\overline{b_2}\langle t,f\rangle\right) \;\mid\; \overline{r}\langle b\rangle \right)$$

The language so far

$$P \quad ::= \quad \mathbf{0} \mid P_1 \,|\, P_2 \mid\, !P \mid a(b).P \mid \overline{a}\langle b\rangle.P \mid (\boldsymbol{\nu} a)\, P$$

this π-calculus is:

- monadic
- synchronous
- with replication

but there exist several other variations/extensions