Groupe de travail concurrence

Analysis of Mobile Systems by Abstract Interpretation

Jérôme Feret

École Normale Supérieure

http://www.di.ens.fr/~feret

10/03/2005

Introduction

We propose a unifying framework to design

- automatic,
- sound,
- approximate,
- decidable,

semantics to abstract the properties of mobile systems.

Our framework is model-independent:

$\Longrightarrow$ we use a META-language to encode mobility models,

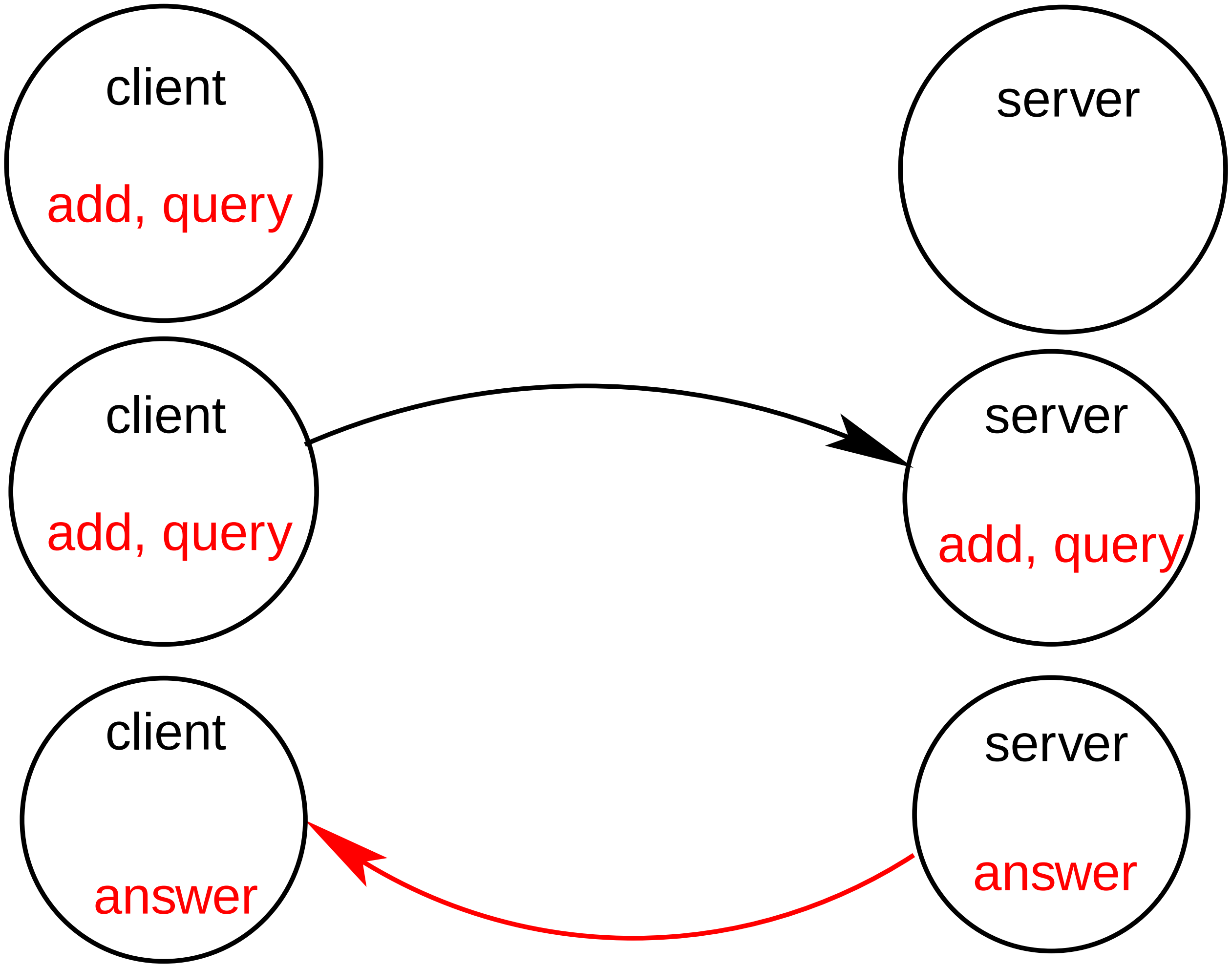
$\Longrightarrow$ we design analyses at the META-language level.

We use the Abstract Interpretation theory.

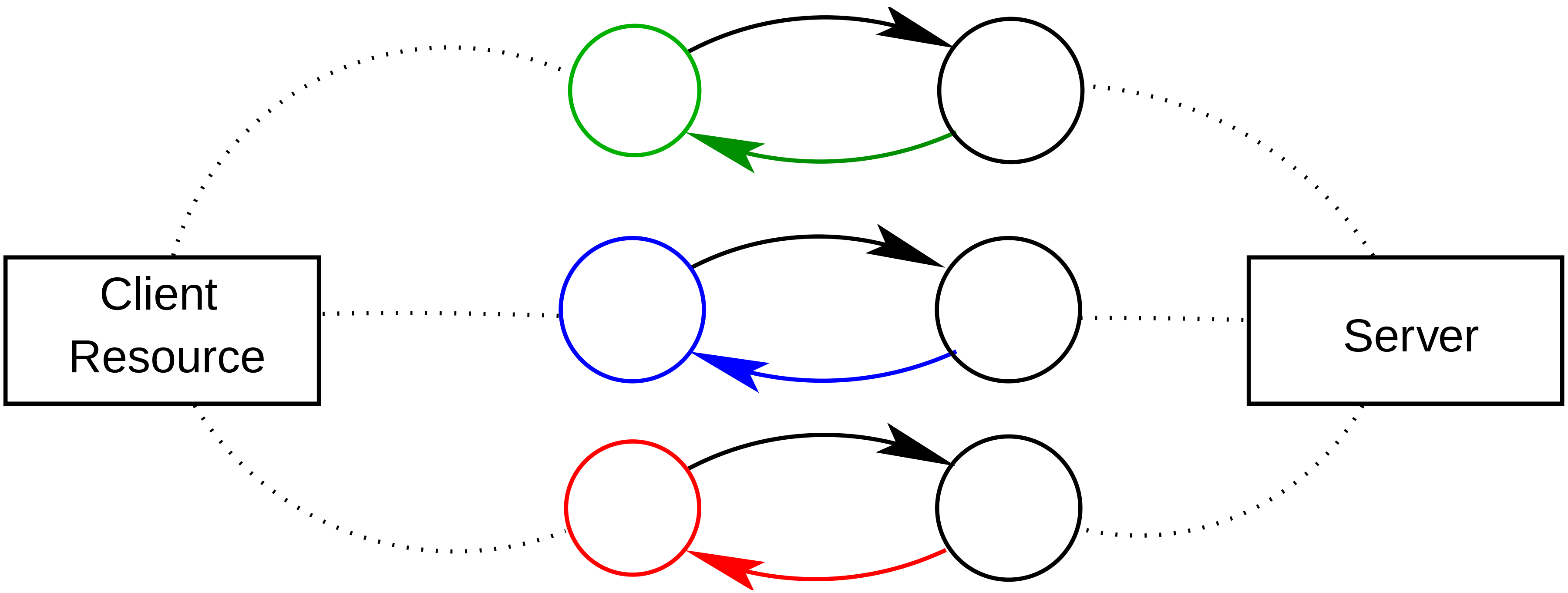
Overview

1. Standard semantics
2. Non-standard semantics
3. META-language
4. Encodings
5. Context encoding

A connection



A network



π-calculus: syntax

Name : infinite set of channel names,
Label : infinite set of labels,

$$P ::= \text{action}.P \mid (P \mid P) \mid (\nu\ x)P \mid \emptyset$$

$$\text{action} ::= c!^{i}[x_1, ..., x_n] \mid c?^{i}[x_1, ..., x_n] \mid *c?^{i}[x_1, ..., x_n]$$

where $n \geqslant 0$, c, x_1, ..., x_n, x, $\in$ *Name* , $i \in$ *Label*.

ν and $?$ are the only name binders.
$fv(P)$: free variables in P,
$bn(P)$: bound names in P.

Transition semantics

A reduction relation and a congruence relation give the semantics of the π-calculus:

- the reduction relation specifies the result of computations:

$$c?^{i}[\overline{y}]Q \mid c!^{j}[\overline{x}]P \xrightarrow{i,j} Q[\overline{y} \leftarrow \overline{x}] \mid P$$

$$*c?^{i}[\overline{y}]Q \mid c!^{j}[\overline{x}]P \xrightarrow{i,j} Q[\overline{y} \leftarrow \overline{x}] \mid *c?^{i}[\overline{y}]Q \mid P$$

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

Congruence relation

- the congruence relation reveals redexes:

  1. some rules make process move inside the syntactic tree (commutativity, associativity of $|$)

  2. some rules handle channel names (α-conversion, extrusion)

Example: syntax

$$\mathcal{S} := (\nu\ \text{port})(\nu\ \text{gen})$$
$$(\textbf{Server} \mid \textbf{Client} \mid \text{gen}!^{6}[])$$

where

$$\textbf{Server} \quad := *\text{port}?^{1}[\textit{info}, \textit{add}](\textit{add}\ !^{2}[\textit{info}])$$

$$\textbf{Client} \quad := *\text{gen}?^{3}[]\ ((\nu\ \textit{data})\ (\nu\ \textit{email})$$
$$(\text{port}!^{4}[\textit{data}, \textit{email}] \mid \text{gen}!^{5}[]))$$

Example: computation

$(\nu\ \text{port})(\nu\ \text{gen})$
$(\textbf{Server} \mid \textbf{Client} \mid \text{gen}!^{6}[])$

$\xrightarrow{3,6}$ $(\nu\ \text{port})(\nu\ \text{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1)$
$(\textbf{Server} \mid \textbf{Client} \mid \text{gen}!^{5}[] \mid \text{port}!^{4}[\mathit{data}_1,\mathit{email}_1])$

$\xrightarrow{1,4}$ $(\nu\ \text{port})(\nu\ \text{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1)$
$(\textbf{Server} \mid \textbf{Client} \mid \text{gen}!^{5}[] \mid \mathit{email}_1!^{2}[\mathit{data}_1])$

$\xrightarrow{3,5}$ $(\nu\ \text{port})(\nu\ \text{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1)(\nu\ \mathit{data}_2)(\nu\ \mathit{email}_2)$
$(\textbf{Server} \mid \textbf{Client} \mid \text{gen}!^{5}[] \mid \mathit{email}_1!^{2}[\mathit{data}_1] \mid \text{port}!^{4}[\mathit{data}_2,\mathit{email}_2])$

$\xrightarrow{1,4}$ $(\nu\ \text{port})(\nu\ \text{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1)(\nu\ \mathit{data}_2)(\nu\ \mathit{email}_2)$
$(\textbf{Server} \mid \textbf{Client} \mid \text{gen}!^{5}[] \mid \mathit{email}_1!^{2}[\mathit{data}_1] \mid \mathit{email}_2!^{2}[\mathit{data}_2])$

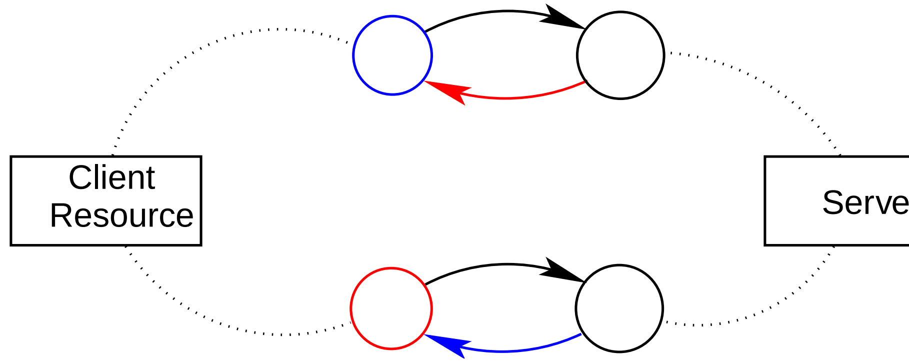
α-conversion

α-conversion destroys the link between names and processes which have declared them:

$$(\nu\ \mathrm{port})(\nu\ \mathrm{gen})(\nu\ data_1)(\nu\ email_1)$$
$$(\nu\ data_2)(\nu\ email_2)$$
$$(\mathbf{Server} \mid \mathbf{Client} \mid \mathrm{gen}!^5[]$$
$$\mid email_1!^4[data_1] \mid email_2!^4[data_2])$$

$$\sim_\alpha$$

$$(\nu\ \mathrm{port})(\nu\ \mathrm{gen})(\nu\ data_2)\ (\nu\ email_1)$$
$$(\nu\ data_1)(\nu\ email_2)$$
$$(\mathbf{Server} \mid \mathbf{Client} \mid \mathrm{gen}!^5[]$$
$$\mid email_1!^4[data_2] \mid email_2!^4[data_1])$$



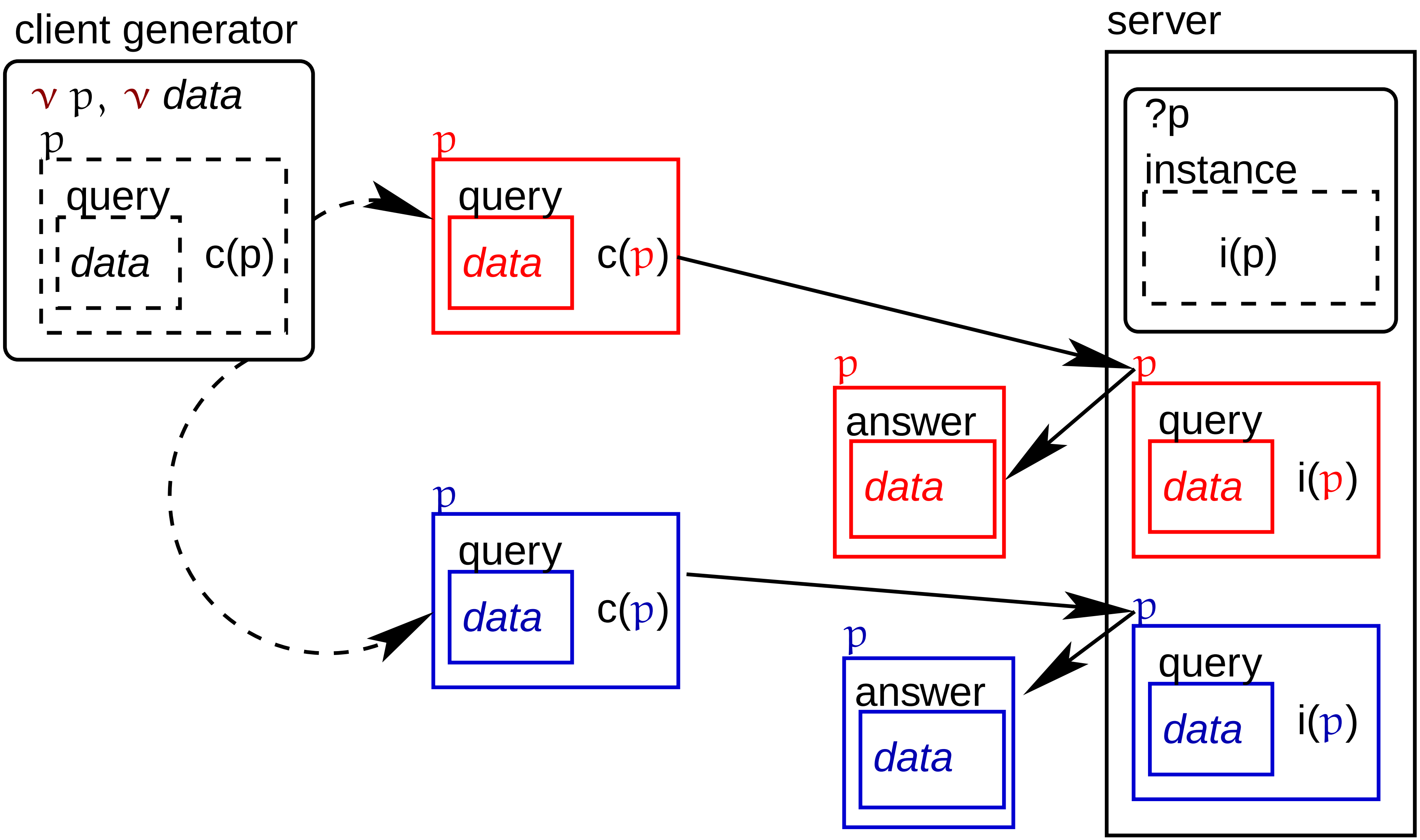
Mobile ambients

Ambients are named boxes containing other ambients (and/or) some agents.

Agents:

- provide capabilities to their surrounding ambients for local migration and other ambient dissolution;
- dynamically create new ambients, names and agents;
- communicate names to each others.

An *ftp*-server



Syntax

Let *Name* be an infinite countable set of ambient names and *Label* an infinite countable set of labels.

$$
\begin{array}{lll}
n & \in \textit{Name} & \text{(ambient name)} \\
l & \in \textit{Label} & \text{(label)} \\
\\
P, Q & ::= (\nu\, n)P & \text{(restriction)} \\
 & \mid\ \mathbf{0} & \text{(inactivity)} \\
 & \mid\ P \mid Q & \text{(composition)} \\
 & \mid\ n^{l}[P] & \text{(ambient)} \\
 & \mid\ M & \text{(capability action)} \\
 & \mid\ \textit{io} & \text{(input/output action)}
\end{array}
$$

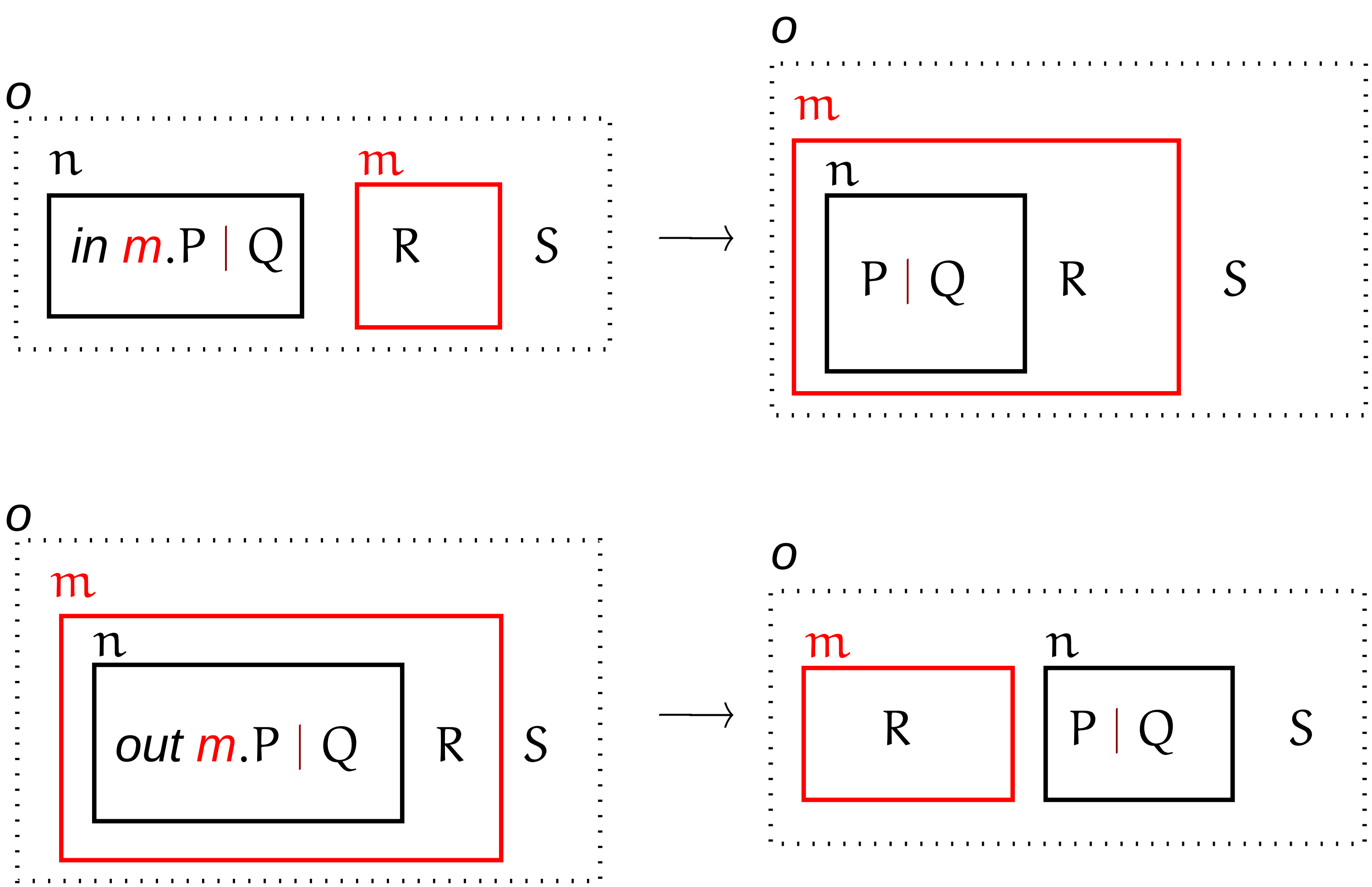
Capability and actions

$$
\begin{array}{llll}
M & ::= & \mathit{in}^l\ n.P & \text{(can enter an ambient named } n\text{)} \\
 & | & \mathit{out}^l\ n.P & \text{(can exit an ambient named } n\text{)} \\
 & | & \mathit{open}^l\ n.P & \text{(can open an ambient named } n\text{)} \\
 & | & !\mathit{open}^l\ n.P & \text{(can open several ambients named } n\text{)}
\end{array}
$$

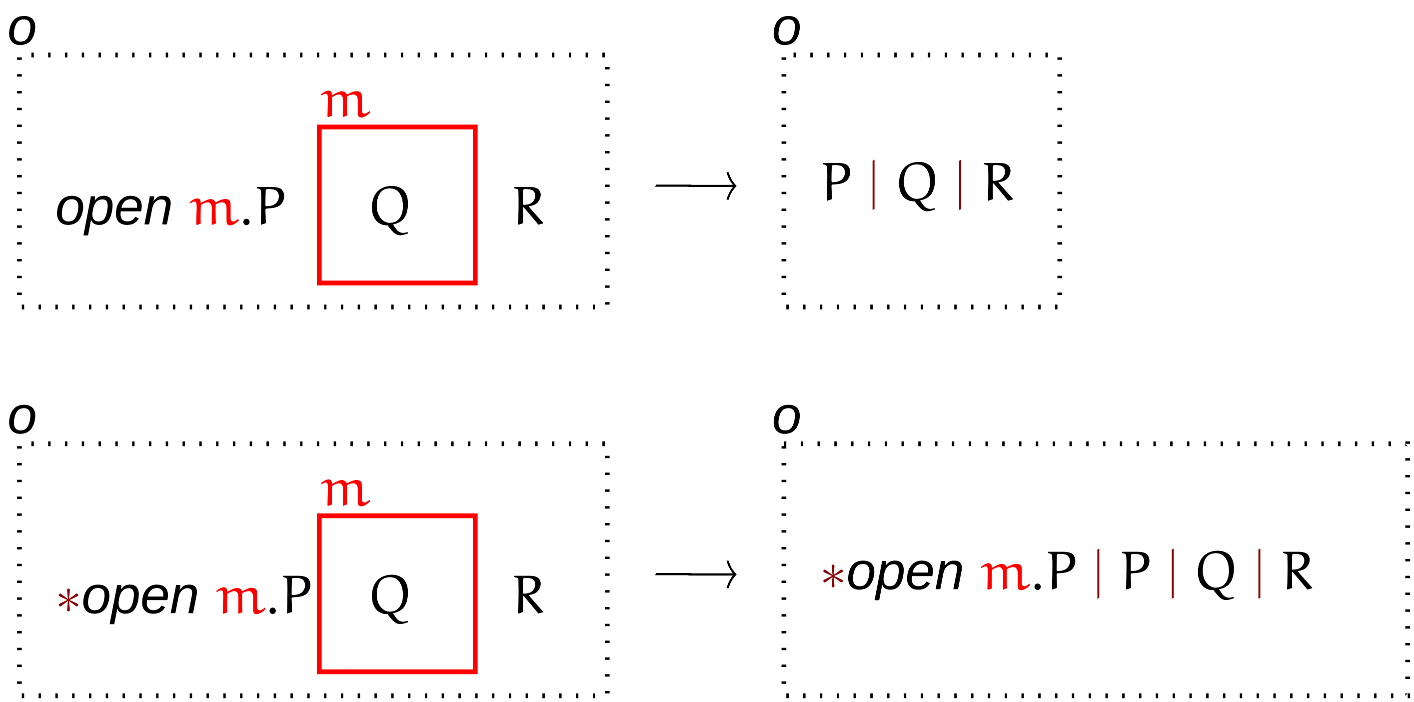
$$
\begin{array}{llll}
\mathit{io} & ::= & (n)^l.P & \text{(input action)} \\
 & | & !(n)^l.P & \text{(input action with replication)} \\
 & | & \langle n \rangle^l & \text{(async output action)}
\end{array}
$$

The only name binders are $(\nu\ \_)$, $(\_)$ and $!(\_)$.

Ambient Migration



Ambient Dissolution



Communication

o

$$\langle m \rangle \mid (n).P \mid Q \longrightarrow P_{[n \leftarrow m]} \mid Q$$

o

$$\langle m \rangle \mid *(n).P \mid Q \longrightarrow *(n).P \mid P_{[n \leftarrow m]} \mid Q$$

An *ftp*-server

$$S := (\nu \mathbf{Pub})(\mathbf{S} \mid !(x)^{11}.C \mid \langle \text{make} \rangle^{21})$$

where

$$\mathbf{Pub} := (\nu\ \text{request})(\nu\ \text{make})(\nu\ \text{server})(\nu\ \text{duplicate})(\nu\ \text{instance})(\nu\ \text{answer}),$$

$$\mathbf{C} := (\nu\ q)(\nu\ p)p^{12}[\mathbf{C}_1 \mid \mathbf{C}_2 \mid \mathbf{C}_3] \mid \langle \text{make} \rangle^{20},$$
$$\mathbf{C}_1 := \text{request}^{13}[\langle q \rangle^{14}],\ \mathbf{C}_2 := \mathit{open}^{15}\text{instance},$$
$$\mathbf{C}_3 := \mathit{in}^{16}\text{server.duplicate}^{17}[\mathit{out}^{18}p.\langle p \rangle^{19}],$$

$$\mathbf{S} := \text{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2],\ \mathbf{S}_1 := !\mathit{open}^{2}\text{duplicate},\ \mathbf{S}_2 := !(k)^{3}.\text{instance}^{4}[\mathbf{I}],$$

$$\mathbf{I} := \mathit{in}^{5}k.\mathit{open}^{6}\text{request}.(\mathit{rep})^{7}(\mathbf{I}_1 \mid \mathbf{I}_2),\ \mathbf{I}_1 := \text{answer}^{8}[\langle \mathit{rep} \rangle^{9}],\ \mathbf{I}_2 := \mathit{out}^{10}\text{server}.$$

$$(\nu \mathbf{Pub})(\mathbf{S} \mid !(x)^{11}.C \mid \langle \mathsf{make} \rangle^{21})$$

$$\longrightarrow$$

$$(\nu \mathbf{Pub}) \left(\begin{array}{l} !(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2] \mid \\ (\nu\ q_1)(\nu\ p_1) p_1{}^{12} \left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathbf{C}_2 \mid \\ \mathit{in}^{16}\mathsf{server}.\mathsf{duplicate}^{17}[\mathit{out}^{18} p_1.\langle p_1 \rangle^{19}] \end{array} \right] \end{array} \right)$$

$$\longrightarrow$$

$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)$$

$$\left(!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1} \left[\mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12} \left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathbf{C}_2 \mid \\ \mathsf{duplicate}^{17}[\mathit{out}^{18} p_1.\langle p_1 \rangle^{19}] \end{array} \right] \right] \right)$$

$$\longrightarrow$$

$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)$$

$$\left(!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1} \left[\begin{array}{l} !\mathit{open}^{2}\mathsf{duplicate} \mid \mathbf{S}_2 \mid \mathsf{duplicate}^{17}[\langle p_1 \rangle^{19}] \mid \\ p_1{}^{12}[\mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathbf{C}_2] \end{array} \right] \right)$$

$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)$$
$$\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\begin{array}{l} !\mathit{open}^{2}\mathsf{duplicate} \mid \mathbf{S}_2 \mid \mathsf{duplicate}^{17}[\langle p_1\rangle^{19}] \mid \\ p_1{}^{12}[\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathbf{C}_2] \end{array}\right]\right)$$

$$\longrightarrow$$

$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)$$
$$\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\begin{array}{l} \mathbf{S}_1 \mid !(k)^{3}.\mathsf{instance}^{4}[\mathbf{I}] \mid \langle p_1\rangle^{19} \mid \\ p_1{}^{12}[\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathbf{C}_2] \end{array}\right]\right)$$

$$\longrightarrow$$

$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)$$
$$\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\begin{array}{l} \mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12}[\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathbf{C}_2] \mid \\ \mathsf{instance}^{4}[\mathit{in}^{5} p_1.\mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathbf{I}_1 \mid \mathbf{I}_2)] \end{array}\right]\right)$$

$$\longrightarrow$$

$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)$$
$$\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\begin{array}{l} \mathbf{S}_1 \mid \mathbf{S}_2 \mid \\ p_1{}^{12}\left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathit{open}^{15}\mathsf{instance} \mid \\ \mathsf{instance}^{4}[\mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathbf{I}_1 \mid \mathbf{I}_2)] \end{array}\right] \end{array}\right]\right)$$

$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)$$

$$\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1} \left[\begin{array}{l} \mathbf{S}_1 \mid \mathbf{S}_2 \mid \\ p_1{}^{12}\left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathit{open}^{15}\mathsf{instance} \mid \\ \mathsf{instance}^{4}[\mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathbf{l}_1|\mathbf{l}_2)] \end{array}\right]\end{array}\right]\right)$$

$$\rightarrow$$

$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)$$

$$\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12}\left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \\ \mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathbf{l}_1 \mid \mathbf{l}_2) \end{array}\right]\right]\right)$$

$$\rightarrow^{*}$$

$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)$$

$$(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12}[\mathsf{answer}^{8}[\langle q_1\rangle^{9}] \mid \mathit{out}^{10}\mathsf{server}]])$$

$$\rightarrow$$

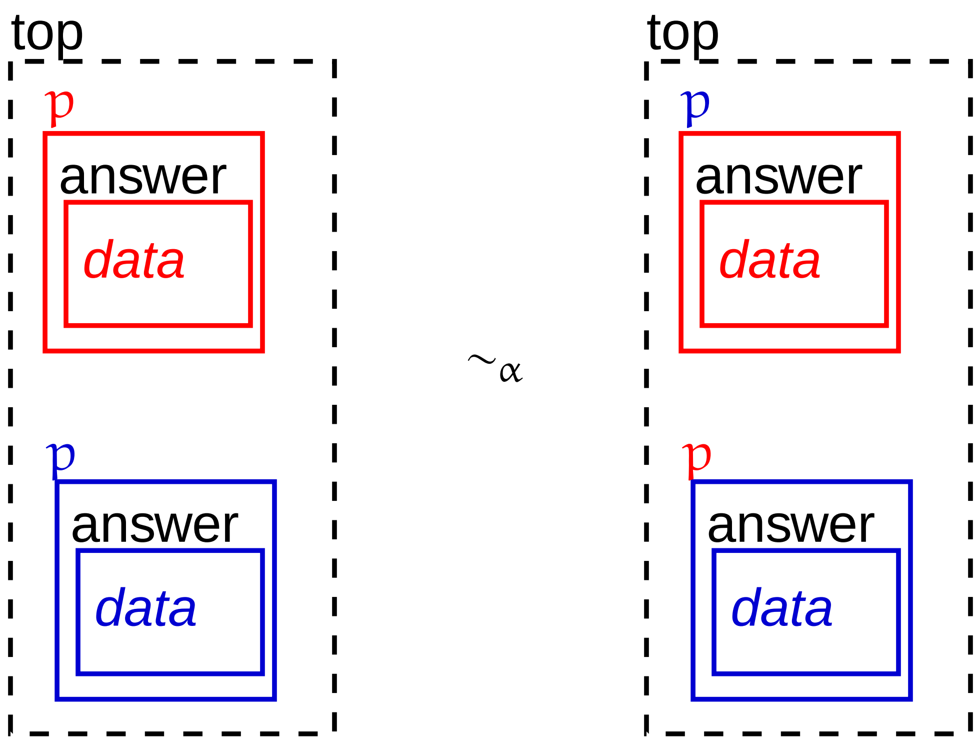
$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)$$

$$(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2] \mid p_1{}^{12}[\mathsf{answer}^{8}[\langle q_1\rangle^{9}]])$$

$$\rightarrow^{*}$$

$$(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)(\nu\ q_2)(\nu\ p_2)(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2] \mid p_1{}^{12}[\mathsf{answer}^{8}[\langle q_1\rangle^{9}]] \mid p_2{}^{12}[\mathsf{answer}^{8}[\langle q_2\rangle^{9}]])$$

α-conversion



Overview

1. Standard semantics
2. Non-standard semantics
3. META-language
4. Encodings
5. Context encoding

Non-standard semantics

A refined semantics where:

- each recursive instance of processes is identified with an unambiguous marker;
- each name is stamped with the marker of the process which has declared this name.

Example: non-standard configuration

$(\textbf{Server} \mid \textbf{Client} \mid \text{gen}!^5[] \mid \mathit{email}_1!^2[\mathit{data}_1] \mid \mathit{email}_2!^2[\mathit{data}_2])$

$$\left\{\begin{array}{l}
\left(1, \varepsilon, \left\{\text{port} \mapsto (\text{port}, \varepsilon)\right.\right) \\
\left(3, \varepsilon, \left\{\begin{array}{ll}\text{gen} & \mapsto (\text{gen}, \varepsilon) \\ \text{port} & \mapsto (\text{port}, \varepsilon)\end{array}\right.\right) \\
\left(2, \mathit{id}'_1, \left\{\begin{array}{ll}\mathit{add} & \mapsto (\mathit{email}, \mathit{id}_1) \\ \mathit{info} & \mapsto (\mathit{data}, \mathit{id}_1)\end{array}\right.\right) \\
\left(2, \mathit{id}'_2, \left\{\begin{array}{ll}\mathit{add} & \mapsto (\mathit{email}, \mathit{id}_2) \\ \mathit{info} & \mapsto (\mathit{data}, \mathit{id}_2)\end{array}\right.\right) \\
\left(5, \mathit{id}_2, \left\{\text{gen} \mapsto (\text{gen}, \varepsilon)\right.\right)
\end{array}\right\}$$

Marker properties

1. Marker allocation must be consistent:

   Two instances of the same process cannot be associated to the same marker during a computation sequence.

2. Marker allocation should be robust:

   Marker allocation should not depend on the interleaving order.

Extraction function

An extraction function calculates the set of the thread instances spawned at the beginning of the system execution or after a computation step.

$$
\begin{aligned}
\beta((\nu\ n)P, \mathit{id}, E) &= \beta\,(P, \mathit{id}, (E[n \mapsto (n, \mathit{id})]))\\
\beta(\emptyset, \mathit{id}, E) &= \emptyset\\
\beta(P \mid Q, \mathit{id}, E) &= \beta(P, \mathit{id}, E) \cup \beta(Q, \mathit{id}, E)\\
\beta(y?^{i}[\overline{y}].P, \mathit{id}, E) &= \{(y?^{i}[\overline{y}].P, \mathit{id}, E_{|fv(y?^{i}[\overline{y}].P)})\}\\
\beta(*y?^{i}[\overline{y}].P, \mathit{id}, E) &= \{(*y?^{i}[\overline{y}].P, \mathit{id}, E_{|fv(*y?^{i}[\overline{y}].P)})\}\\
\beta(x!^{j}[\overline{x}].P, \mathit{id}, E) &= \{(x!^{j}[\overline{x}].P, \mathit{id}, E_{|fv(x!^{j}[\overline{x}]P)})\}
\end{aligned}
$$

Transition system

$$C_0(\mathsf{S}) = \beta(\mathsf{S}, \varepsilon, \emptyset)$$

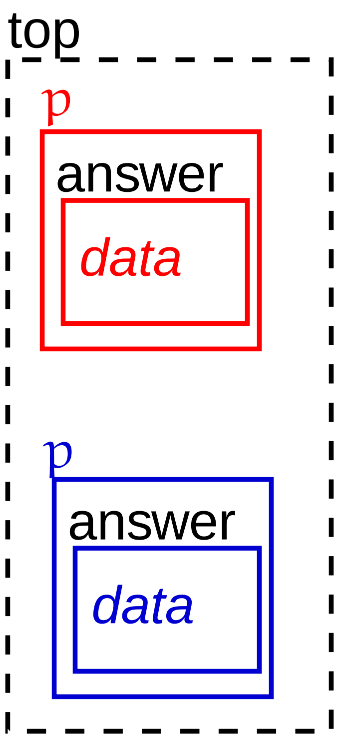
$$\frac{E_?(y) = E_!(x)}{C \cup \left\{ \begin{array}{l} (y?^i[\overline{y}]P, id_?, E_?), \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{i,j} (C \cup \beta(P, id_?, E_?[y_i \mapsto E_!(x_i)]) \cup \beta(Q, id_!, E_!))}$$

$$\frac{E_*(y) = E_!(x)}{C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, id_*, E_*), \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \xrightarrow{i,j} \left(\begin{array}{l} \quad \cup\{(*y?^i[\overline{y}]P, id_*, E_*)\} \\ C \cup \beta(P, \mathcal{N}((i,j), id_*, id_!), E_*[y_i \mapsto E_!(x_i)]) \\ \quad \cup \beta(Q, id_!, E_!) \end{array} \right)}$$

where $\mathcal{N}$ is the tree constructor.

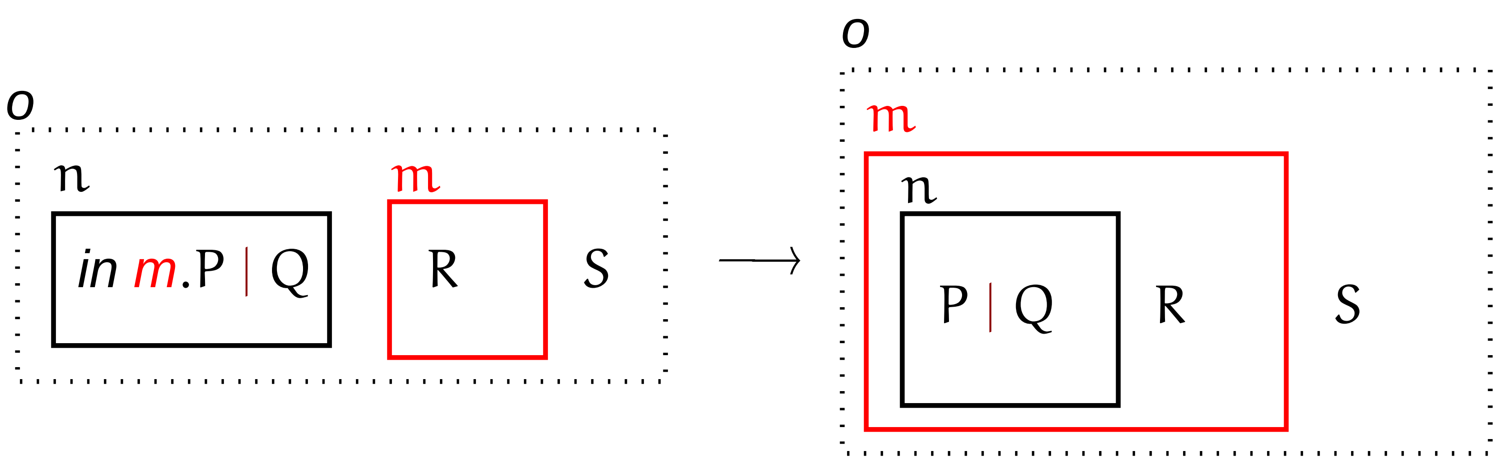
Non-standard semantics for mobile ambients

$$\left\{\begin{array}{l}
(\mathrm{p}^{12}[\bullet], id_0, (\mathsf{top}, \varepsilon), [\mathrm{p} \mapsto (\mathrm{p}, id_0)]) \\
(\mathrm{p}^{12}[\bullet], id_1, (\mathsf{top}, \varepsilon), [\mathrm{p} \mapsto (\mathrm{p}, id_1)]) \\
(\mathsf{answer}^{8}[\bullet], id'_0, (12, id_0), \emptyset) \\
(\mathsf{answer}^{8}[\bullet], id'_1, (12, id_1), \emptyset) \\
\hline
(\langle rep \rangle^{9}, id'_0, (8, id'_0), [rep \mapsto (data, id_0)]) \\
(\langle rep \rangle^{9}, id'_1, (8, id'_1), [rep \mapsto (data, id_1)])
\end{array}\right.$$



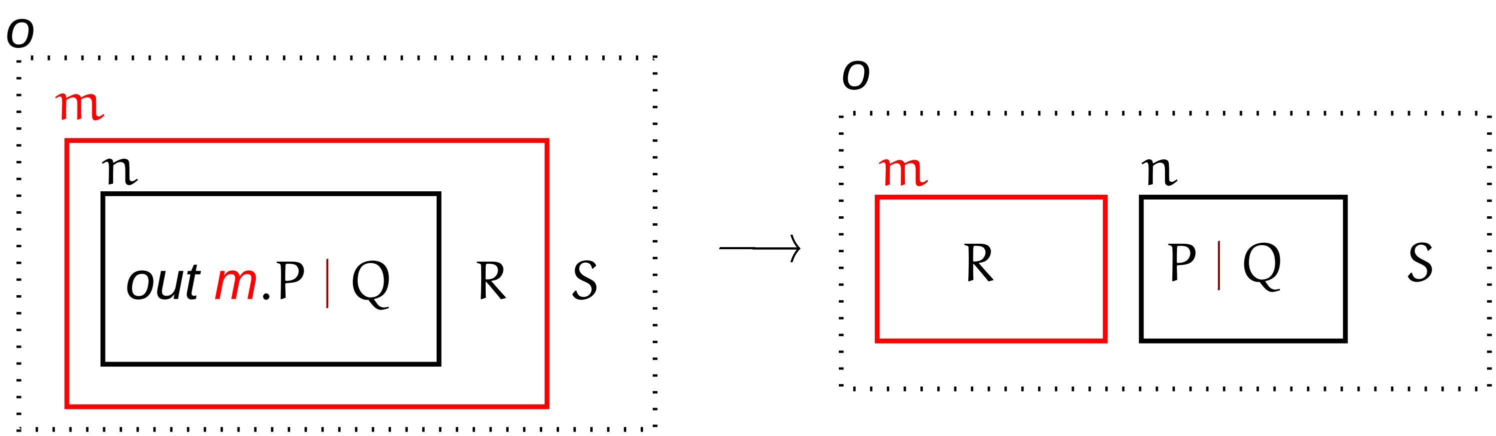
In migration

$$\frac{\begin{cases}\lambda = \left(n^{i}[\bullet], id_1, loc_1, E_1\right), \\ \mu = \left(m^{j}[\bullet], id_2, loc_2, E_2\right), \\ \psi = \left(in^{k} o.P, id_3, loc_3, E_3\right), \\ loc_1 = loc_2,\ loc_3 = (i, id_1),\ E_2(m) = E_3(o),\ \lambda \neq \mu.\end{cases}}{C \cup \{\lambda; \mu; \psi\} \stackrel{in(i,j,k)}{\longrightarrow} (C \cup \{\mu\}) \cup (n^{i}[\bullet], id_1, (j, id_2), E_1) \cup \beta\left(P, id_3, loc_3, E_{3|fv(P)}\right).}$$



out migration

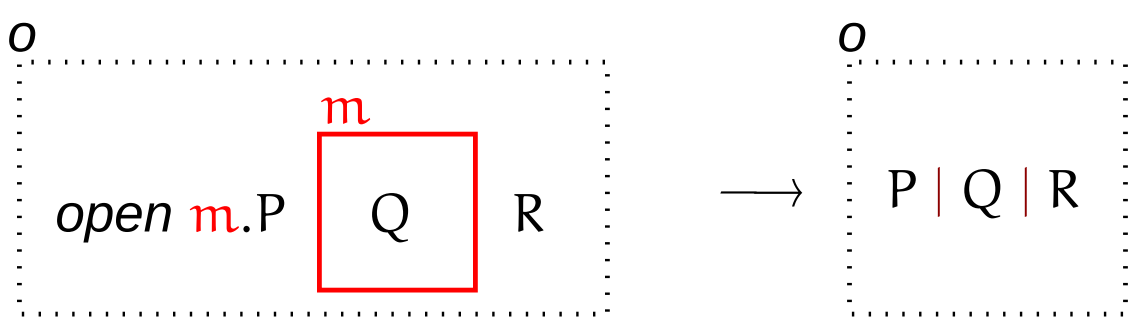
$$\frac{\begin{cases}\lambda = \left(m^i[\bullet], id_1, loc_1, E_1\right), \\ \mu = \left(n^j[\bullet], id_2, loc_2, E_2\right), \\ \psi = \left(out^k o.P, id_3, loc_3, E_3\right), \\ loc_2 = (i, id_1),\ loc_3 = (j, id_2),\ E_1(m) = E_3(o)\end{cases}}{C \cup \{\lambda;\mu;\psi\} \overset{out(i,j,k)}{\longrightarrow} (C \cup \{\lambda\}) \cup (n^j[\bullet], id_2, loc_1, E_2) \cup \beta\left(P, id_3, loc_3, E_{3|fv(P)}\right).}$$



Dissolution

$$\frac{\begin{cases} \lambda = \left(\mathit{open}^{i} m.P, \mathit{id}_1, \mathit{loc}_1, E_1\right) \\ \mu = \left(n^{j}[\bullet], \mathit{id}_2, \mathit{loc}_2, E_2\right), \\ \mathit{loc}_1 = \mathit{loc}_2,\ E_1(m) = E_2(n), \end{cases}}{C \cup \{\lambda; \mu\} \overset{\mathit{open}(i,j)}{\longrightarrow} (C \setminus A) \cup A' \cup \beta\left(P, \mathit{id}_1, \mathit{loc}_1, E_{1|fv(P)}\right)}$$

$$\text{where } \begin{cases} A = \{(a, \mathit{id}, \mathit{loc}, E) \in C \mid \mathit{loc} = (j, \mathit{id}_2)\} \\ A' = \{(a, \mathit{id}, \mathit{loc}_2, E) \mid (a, \mathit{id}, (j, \mathit{id}_2), E) \in C\}. \end{cases}$$



Overview

1. Standard semantics
2. Non-standard semantics
3. META-language
4. Encodings
5. Context encoding

Toward a unifying framework

Several models depending on the application field:

- π-calculus (implicit mobility)
- join-calculus (locality)
- spi-calculus (cryptographic primitives)
- ambient-calculus (explicit mobility)
- BIO-ambients (biological systems)
- ...

Key-idea : Propose a META-language and design reachability analyses at the META-language level.

Advantages of the META-language

1. each analysis at the META-language level provides an analysis for each encoded model;
2. the META-language avoids the use of congruence and α-conversion:
   Fresh names are allocated according to the local history of each process.
3. names contain useful information:
   This allows the inference of:
   - more complex properties;
   - some simple properties the proof of which uses complex properties.

META-language: intuition

In the π-calculus :

- each program point $a?[y]P$ is associated with a partial interaction:

$$(\textit{in}, [a], [y], \textit{label}(P))$$

- each program point $b![x]Q$ is associated with a partial interaction:

$$(\textit{out}, [b, x], [], \textit{label}(Q))$$

- The generic transition rule:

$$((\textit{in}, \textit{out}), [X_1^1 = X_1^2], [Y_1^1 \leftarrow X_2^2])$$

  describes communication steps.

Some rules are more complex (e.g. ambient opening).

META-syntax: Generic partial interactions

Let $\mathcal{A}$ be a set of generic partial interaction names.
For each partial interaction name $pi \in \mathcal{A}$:

- a type $type(pi) \in \{computation; replication; migration\}$
  specifies the behavior of the threads that compute partial interactions;
- a number of parameters $n\text{-}args(pi) \in \mathbb{N}$:
  the number of the names that are relevant in the global interaction;
- a number of variables $n\text{-}vars(pi) \in \mathbb{N}$:
  the number of the variables bound when computing the partial interaction.

For instance, in the π-calculus:

- out_2: $type(out_2) = computation$, $n\text{-}args(out_2) = 3$, $n\text{-}vars(out_2) = 0$.
- $fetch_2$: $type(fetch_2) = replication$, $n\text{-}args(fetch_2) = 1$, $n\text{-}vars(fetch_2) = 2$.
- $move$: $type(move) = migration$, $n\text{-}args(move) = 2$, $n\text{-}vars(move) = 1$.

META-syntax: Interacting components

A rule is given by a tuple:

$$\mathcal{R} = (n, \textit{components}, \textit{compatibility}, \textit{v-passing}, \textit{broadcast}),$$

where:

- $n \in \mathbb{N}$,

  n denotes the number of the components that are involved in the global interaction;

- $\textit{components} \in [\![1; n]\!] \to \mathcal{A}$,

  $\textit{components}(k)$ is the name of the partial interaction that is computed by the k-th component.

Only the first component may be of type *replication*, in such a case the second component is of type *computation*.

META-syntax: Formal rules: Symbolic variables

We introduce some sets of variables:

1. $\mathcal{V}_{\mathcal{R}}^{I} = \{I^k \mid 1 \leq k \leq n\}$,
   I_k denotes the identity of the k-th interacting component;
2. $\mathcal{V}_{\mathcal{R}}^{X} = \left\{X_l^k \;\middle|\; 1 \leq k \leq n, 1 \leq l \leq \textit{n-args}(\textit{components}(k))\right\}$,
   X_l^k denotes the value of the l-th argument of the k-th component;
3. $\mathcal{V}_{\mathcal{R}}^{Y} = \left\{Y_l^k \;\middle|\; 1 \leq k \leq n, 1 \leq l \leq \textit{n-vars}(\textit{components}(k))\right\}$,
   Y_l^k denotes the l-th variable that is bound in the k-th component.

Example

In the case of the π-calculus,

- two program points: $a?^{p_1}[x].P$ and $b!^{p_2}[y].Q$,
- two threads: $t_1 = (p_1, id_1, E_1)$ and $t_2 = (p_2, id_2, E_2)$,
- t_1 is associated with the sequences:
  - of arguments $[a]$,
  - of new variables $[x]$;
- t_2 is associated with the sequences:
  - of arguments $[b; y]$,
  - of new variables $[]$;
- $I^1 \mapsto (p_1, id_1)$, $I^2 \mapsto (p_2, id_2)$,
- $X_1^1 \mapsto E_1(a)$; $X_1^2 \mapsto E_2(b)$, $X_2^2 \mapsto E_2(y)$;
- $Y_1^1 \mapsto x$.

META-syntax: Formal rules: Synchronization, communication

A rule is given by a tuple:

$$\mathcal{R} = (n, \textit{components}, \textit{compatibility}, \textit{v-passing}, \textit{broadcast}),$$

where:

- *compatibility* $\in \wp\left(\mathcal{V}_{\mathcal{R}}^{I} \cup \mathcal{V}_{\mathcal{R}}^{X}\right)^{2}$,
  compatibility encodes synchronization constraints,
  ex: $X_1^1 = X_1^2$ means $E_1(a) = E_2(b)$;
- *v-passing* $\in \mathcal{V}_{\mathcal{R}}^{Y} \rightarrow \mathcal{V}_{\mathcal{R}}^{I} \cup \mathcal{V}_{\mathcal{R}}^{X}$,
  v-passing encodes name passing,
  ex: $[Y_1^1 \rightarrow X_2^2]$ means that the variable x is bound to $E_2(y)$ in P;
- *broadcast* $\in \mathcal{V}_{\mathcal{R}}^{I} \rightarrow \mathcal{V}_{\mathcal{R}}^{I} \cup \mathcal{V}_{\mathcal{R}}^{X}$,
  broadcast encodes re-addressing,

System syntax
Labeling

Label: a finite set of labels.
$Label_p$: a subset of *Label*.
Elements in $Label_p$ tag program points.
Elements in $Label \setminus Label_p$ tag values.

$\mathcal{V}$ is a set of variables.

Each program point is associated with an interface:

$$\mathrm{I} : Label_p \rightarrow \wp(\mathcal{V})$$

System syntax: Continuation

A thread in a continuation is described by:

- a program point $\mathfrak{p} \in \mathit{Label}_{\mathfrak{p}}$,
- a partial map mapping some variables $v \in \mathcal{V}$ into value labels $\mathfrak{l} \in \mathit{Label}$. this map describes fresh values (new ambients, restricted names,...)

A potential continuation is described by an element in

$$\wp(\mathit{Label}_{\mathfrak{p}} \times (\mathcal{V} \rightharpoonup \mathit{Label})).$$

The set of the potential continuations is described by an element in:

$$\wp(\wp(\mathit{Label}_{\mathfrak{p}} \times (\mathcal{V} \rightharpoonup \mathit{Label})))$$

to model internal non-determinism.

System syntax: Partial interaction

$$pi = (s, (\textit{parameter}_i), (\textit{bound}_i), \textit{constraints}, \textit{continuation})$$

where

- s is a partial interaction name in $\mathcal{A}$;
- $(\textit{parameter}_i) \in \mathcal{V}^{\textit{n-args}(s)}$,
  variables that are bound to insightful values;
- $(\textit{bound}_i) \in \mathcal{V}^{\textit{n-vars}(s)}$,
  variables that are bound during the computation step;
- $\textit{constraints} \subseteq \{v \diamond v' \mid (v, v') \in \mathcal{V}^2,\ \diamond \in \{=; \neq\}\}$,
  positive and negative matchings.
- $\textit{continuation} \in \wp(\wp(\textit{Label}_p \times (\mathcal{V} \rightharpoonup \textit{Label})))$.
  a description of the potential continuations.

System syntax

The syntax of the system is defined by a triple:

$$(\mathrm{I}, \mathit{init}, \mathit{interaction})$$

where

1. $\mathrm{I} : \mathit{Label}_{\mathrm{p}} \rightarrow \wp(\mathcal{V})$ maps each program point to their interface;
2. $\mathit{init} \in \wp(\wp(\mathit{Label}_{\mathrm{p}} \times (\mathcal{V} \rightharpoonup \mathit{Label})))$ describes initial states;
3. *interaction* maps each program point to a set of partial interaction.

We have some constraints about variable usage.

System configuration

A system is a set of threads.

Each thread is a triple (p, id, E):

- $p \in \mathit{Label}_p$ is a program point;
- id is a marker in $\mathcal{M}$;
- $E \in I(p) \to \mathit{Label} \times \mathcal{M}$ is an environment.

$\mathcal{M}$ is the set of trees the leaves of which are ε and the n-ary nodes of which are in Label_p^n.

Operational semantics

Here are the different steps of an interaction:

- *interaction enabling*:
  - matching guards,
  - checking interface compatibility;
- *interaction computation*:
  - removing threads;
  - computing dynamic data:
    (markers, name passing, fresh variables);
- readdressing: applying the broadcast substitution to the whole system.

Operational semantics: Exhibited action

A thread (p, id, E) may compute any partial interaction

$$(s, (parameter_i), (bound_i), constraints, continuation)$$

if and only if:

- $(s, (parameter_i), (bound_i), constraints, continuation) \in interaction(p)$, (The thread contains the partial interaction);
- $\forall (a \diamond b) \in constraints,\ E(a) \diamond E(b)$, (Matching constraints are satisfied).

Operational semantics: Global synchronizations

- $\mathcal{R} = (n, \textit{components}, \textit{compatibility}, \textit{v-passing}, \textit{broadcast})$
- $(t^k, \textit{id}^k, E^k)$
- $(\textit{param}_l^k)$

Threads may synchronize their computation if and only if:

$$\forall a, b \in \textit{compatibility},\ \sigma(a) = \sigma(b),$$

where $\sigma : \begin{cases} X_l^k & \mapsto E^k(\textit{param}_l^k) \\ I^k & \mapsto (p^k, \textit{id}^k). \end{cases}$.

Operational semantics:
Marker computation

- $\mathcal{R} = (n, \textit{components}, \textit{compatibility}, \textit{v-passing}, \textit{broadcast})$
- $(t^k, \textit{id}^k, E^k)$

The marker of the first component is updated if and only if it computes a partial interaction of type *replication* .

The new marker is:

$$((t^1, \ldots, t^n), \textit{id}^1, \ldots, \textit{id}^n).$$

Operational semantics:
Removing threads

Threads that compute partial interactions of type *computation* or *migration* are removed.

Operational semantics: Shared environment

Given:

- $i \in [\![1; n]\!]$: an index;
- $(t^k)_{1 \leq k \leq n} = (p^k, \mathit{id}^k, E^k)_{1 \leq k \leq n}$: interacting threads;
- $(\mathit{bd}_l)_l$: sequence of variables (will be bound in the i-th thread);
- $(\mathit{param}_l^k)_{k,l}$: thread parameters;
- $\mathit{communications} \in \mathcal{V}_{\mathcal{R}}^{Y} \rightarrow \mathcal{V}_{\mathcal{R}}^{X} \cup \mathcal{V}_{\mathcal{R}}^{I}$: name-passing formal description;

We define the environment

$$\overline{E^i} = E^i[\mathit{bd}_j \mapsto \sigma(\mathit{communications}(Y_j^i))],$$

where $\sigma = \begin{cases} X_l^k \mapsto E^k(\mathit{param}_l^k) \\ I^k \mapsto (p^k, \mathit{id}^k). \end{cases}$

Operational semantics:
Launching a continuation

Given:

- $\mathit{id} \in \mathcal{M}$: a marker,
- $E_d \in V_d \to \mathit{Label} \times \mathcal{M}$: an environment,
- $(q, E_s) \in \mathit{Label}_p \times (V \rightharpoonup \mathit{Label})$: a continuation thread;

We update the environment definition:

$$\overline{E} = \begin{cases} V_d \cup \mathit{Dom}(V_s) & \to \mathit{Label} \times \mathcal{M} \\ x & \mapsto \begin{cases} (E_s(x), \mathit{id}) & \text{if } x \in V_s \\ E_d(x) & \text{if } x \in V_d \setminus V_s. \end{cases} \end{cases}$$

Operational semantics: Broadcast value passing

Given:

- $(t^k)_{1\leq k\leq n} = (p^k, \mathit{id}^k, E^k)$: interacting threads;
- $(\mathit{param}^k_l)_{k,l}$: thread parameters;
- $\mathit{broadcast} \in \mathcal{V}^I_{\mathcal{R}} \rightharpoonup \mathcal{V}^X_{\mathcal{R}} \cup \mathcal{V}^I_{\mathcal{R}}$: substitution formal description;

We define the support $\mathcal{D}$ of the substitution:

$$\mathcal{D} = \{(p^k, \mathit{id}^k) \mid \exists k,\ I^k \in \mathit{Dom}(\mathit{broadcast})\}.$$

We compute the substitution $\tau \in \mathit{Label} \times \mathcal{M} \to \mathit{Label} \times \mathcal{M}$ such that:

1. $\forall x \in (\mathit{Label} \times \mathcal{M}) \setminus \mathcal{D},\ \tau(x) = x$;
2. $\forall x \in \mathcal{D},\ \tau(x) \in \{\sigma(\mathit{broadcast}(X^k_l)) \mid x = (p^k, \mathit{id}^k)\}$,

   where $\sigma : \begin{cases} I^{k'} \mapsto (p^{k'}, \mathit{id}^{k'}) \\ X^{k'}_{l'} \mapsto E^{k'}(\mathit{param}^{k'}_{l'}). \end{cases}$

Marker consistency

The META-language does not guarantee marker consistency.
We give some sufficient conditions that ensure marker consistency.

Conditions about partial interactions

1. *replication* partial interactions:

   In $(n, \textit{components}, \textit{compatibility}, \textit{v-passing}, \textit{broadcast})$,

   If:

   $$\textit{type}(\textit{components}(i)) = \textit{replication},$$

   then:

   $$\begin{cases} i = 1 \\ \textit{type}(\textit{components}(2)) = \textit{computation}; \end{cases}$$

2. *migration* partial interactions:

   In $(s, (\textit{parameter}_i), (\textit{bound}_i), \textit{constraints}, \textit{continuation})$,

   If:

   $$\textit{type}(s) = \textit{migration};$$

   Then:

   $$\forall C \in \textit{continuation}, C \textit{ matches } \{(p, \emptyset)\}.$$

Syntactic forest

We define $\sim$ over *Label*$_p$ as the strongest equivalence relation that satisfies: $a \sim b$ as soon as there exists a partial interaction

$$(s, (\textit{parameter}_i), (\textit{bound}_i), \textit{constraints}, \textit{continuation}) \in \textit{interaction}(a),$$

such that both *type*(s) = *migration* and $\{(b, \emptyset)\} \in$ *continuation*.

We require that:

1. In each continuation choice, there is at most one computation thread per equivalence class,
2. The equivalence class of initial threads are unreachable with a *computation* or a *replication* partial interaction,
3. There is at most one equivalence class from which we can reach a given equivalence class with a *computation* or a *replication* partial interaction.

Fresh values

A label never occurs in continuations of distinct program points.

Overview

1. Standard semantics
2. Non-standard semantics
3. META-language
4. Encodings
5. Context encoding

Expressiveness

We encode:

- π-calculus (implicit mobility)
- join-calculus (locality)
- spi-calculus (cryptographic primitives)
- ambient-calculus (explicit mobility)
- BIO-ambients (biological systems)

We deal with:

- internal choices/external choices,
- guarded replication/recursive definition,
- location, migration, and dissolution,
- term construction and term destruction,
- safe migration and channeled communication across boundaries.

Ambients encoding: Partial interaction

Partial interaction names:

$\{$*input*; *fetch*; *output*; *in*; *out*; *open*; *ropen*; *mov-ambient*; *dis-ambient*$\}$;

$$(\textit{type}, \textit{n-args}, \textit{n-vars}) : \begin{cases} \textit{input} & \mapsto (\textit{computation}, 1, 1) \\ \textit{fetch} & \mapsto (\textit{replication}, 1, 1) \\ \textit{output} & \mapsto (\textit{computation}, 2, 0) \\ \textit{in} & \mapsto (\textit{computation}, 2, 0) \\ \textit{out} & \mapsto (\textit{computation}, 2, 0) \\ \textit{open} & \mapsto (\textit{computation}, 2, 0) \\ \textit{ropen} & \mapsto (\textit{replication}, 2, 0) \\ \textit{mov-ambient} & \mapsto (\textit{migration}, 2, 1) \\ \textit{dis-ambient} & \mapsto (\textit{computation}, 2, 0); \end{cases}$$

Ambients encoding: in migration

$$\textit{in} = (3, \textit{component}, \textit{synchronization}, \textit{communication}, \textit{global})$$

where

1. $\textit{component} = \begin{cases} 1 \mapsto \textit{mov-ambient} \\ 2 \mapsto \textit{mov-ambient} \\ 3 \mapsto \textit{in}; \end{cases}$
2. $\textit{synchronization} = \{X_1^1 = X_1^2; X_1^3 = I^1; X_2^2 = X_2^3\}$;
3. $\textit{communication} = [Y_1^1 \leftarrow I^2]$;
4. $\textit{global} = \emptyset$.

Ambients encoding: out migration

$$out = (3, component, synchronization, communication, global)$$

where

1. $component = \begin{cases} 1 \mapsto \textit{mov-ambient} \\ 2 \mapsto \textit{mov-ambient} \\ 3 \mapsto \textit{out}; \end{cases}$
2. $synchronization = \{X_1^1 = I^2; X_1^3 = I^1; X_2^2 = X_2^3\};$
3. $communication = [Y_1^1 \leftarrow X_1^2];$
4. $global = \emptyset$.

Ambients encoding: dissolution

$$open = (2, component, synchronization, communication, global)$$

where

1. $component = \begin{cases} 1 \mapsto open \\ 2 \mapsto \textit{dis-ambient} \end{cases}$
2. $synchronization = \{X_1^1 = X_1^2; X_2^1 = X_2^2\}$;
3. $communication = \emptyset$;
4. $global = [I^2 \mapsto X_1^2]$.

Ambients encoding: communication

$$com = (2, component, synchronization, communication, global)$$

where

1. $component = \begin{cases} 1 \mapsto input \\ 2 \mapsto output \end{cases}$
2. $synchronization = \{X_1^1 = X_1^2\}$;
3. $communication = [Y_1^1 \mapsto X_2^2]$;
4. $global = \emptyset$.

Ambients encoding: Continuations

$$\begin{aligned}
\beta(n^{i}[P], E_s) &= \beta(P, E_s[\textit{loc} \mapsto i]) \cup \{(i, E_s)\} \\
\beta(P \mid Q, E_s) &= \beta(P, E_s) \cup \beta(Q, E_s) \\
\beta((\nu^{l} n)P, E_s) &= \beta(P, E_s[n \mapsto l]) \\
\beta(M, E_s) &= \{(M, E_s)\} \\
\beta(\textit{io}, E_s) &= \{(\textit{io}, E_s)\} \\
\beta(\mathbf{0}, E_s) &= \emptyset
\end{aligned}$$

Ambients encoding: Program points

- $n^l[P]$: $\begin{cases} I(l) = \{\textit{loc}; n\} \\ \textit{interaction}(l) = \left\{ \begin{array}{l} (\textit{mov-ambient}, [\textit{loc}; n], [\textit{loc}], \emptyset, \{\{(l, \emptyset)\}\}); \\ (\textit{dis-ambient}, [\textit{loc}; n], [], \emptyset, \{\emptyset\}) \end{array} \right\}. \end{cases}$

- In $a^l n.P$: $\begin{cases} I(l) = \{\textit{loc}; n\} \cup (fv(P)) \\ \textit{interaction}(l) = \{(a, [\textit{loc}; n], [], \emptyset, \{\beta(P, \emptyset)\})\}; \end{cases}$

- In $(n)^l.P(\textit{act} = \textit{input})$ or $!(n)^l.P(\textit{act} = \textit{fetch})$:

$$\begin{cases} I(l) = \{\textit{loc}\} \cup fv(P) \setminus \{n\} \\ \textit{interaction}(l) = \{(\textit{act}, [\textit{loc}], [n], \emptyset, \{\beta(P, \emptyset)\})\}, \end{cases}$$

- In $\langle n \rangle^l$: $\begin{cases} I(l) = \{\textit{loc}; n\} \\ \textit{interaction}(l) = \{(\textit{output}, [\textit{loc}; n], [], \emptyset, \{\emptyset\})\}. \end{cases}$

Encoding choices

1. External choices may be encoded:
   (each program point is associated with a set of partial interactions).

2. Internal choices may be encoded:
   (each partial interaction is associated with a set of continuation choice).

Explicit recursion

$(\textit{type}, \textit{n-args}, \textit{n-vars}) : \textit{rec} \mapsto (\textit{replication}, 1, 1), \textit{unfold} \mapsto (\textit{computation}, 2, 0);$

$$\textit{rec} = (2, \textit{component}, \textit{synchronization}, \textit{communication}, \textit{global})$$

where

1. $\textit{component} = \left\{ 1 \mapsto \textit{rec}, 2 \mapsto \textit{unfold}; \right.$
2. $\textit{synchronization} = \{X_1^1 = X_2^2\};$
3. $\textit{communication} = [Y_1^1 \mapsto X_1^2];$
4. $\textit{global} = \emptyset.$

- A^p: $I(p) = \{\textit{loc}; A\}$ and $\textit{interaction}(p) = \{\textit{unfold}, [\textit{loc}; A], [], \emptyset, \{\emptyset\}\}$,
- let^p $A^\alpha = Q$ in P: $I(p) = \{A\} \cup (fv(Q))$
  and $\textit{interaction}(p) = \{\textit{rec}, [A], [\textit{loc}], \emptyset, \beta(Q, \emptyset)\}$.

Term handling

We use a heap modelisation:

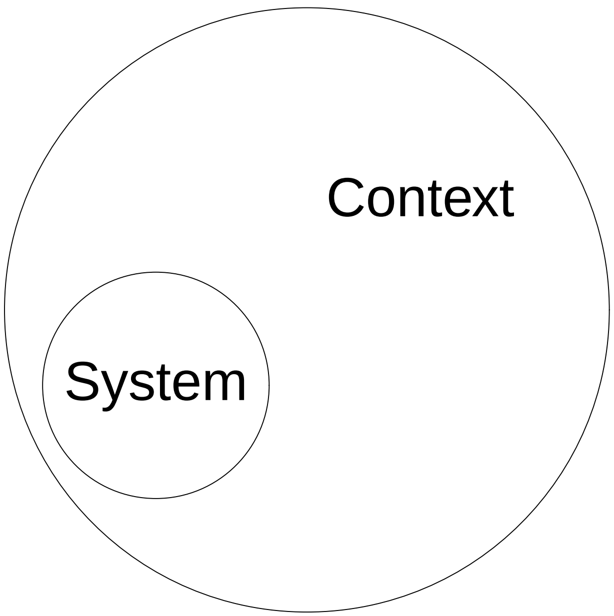
- we associate a thread that may compute a *migration* partial interaction to each subterm;
- the term constructor is given by the name of the partial interaction;
- children subterms are given by the partial interaction parameters;
- we do not consume the thread when computing the partial interaction.

Overview

1. Standard semantics
2. Non-standard semantics
3. META-language
4. Encodings
5. Context encoding

Context independent semantics

Analyzing interaction between a system and its unknown context.



The context may

- spy the system, by listening to message on unsafe channel names;
- spoil the system, by sending message via unsafe channel names.

Nasty context

Context := (ν unsafe) (**new**
| **spy**$_0$ | ... | **spy**$_n$
| **spoil**$_0$ | ... | **spoil**$_n$)

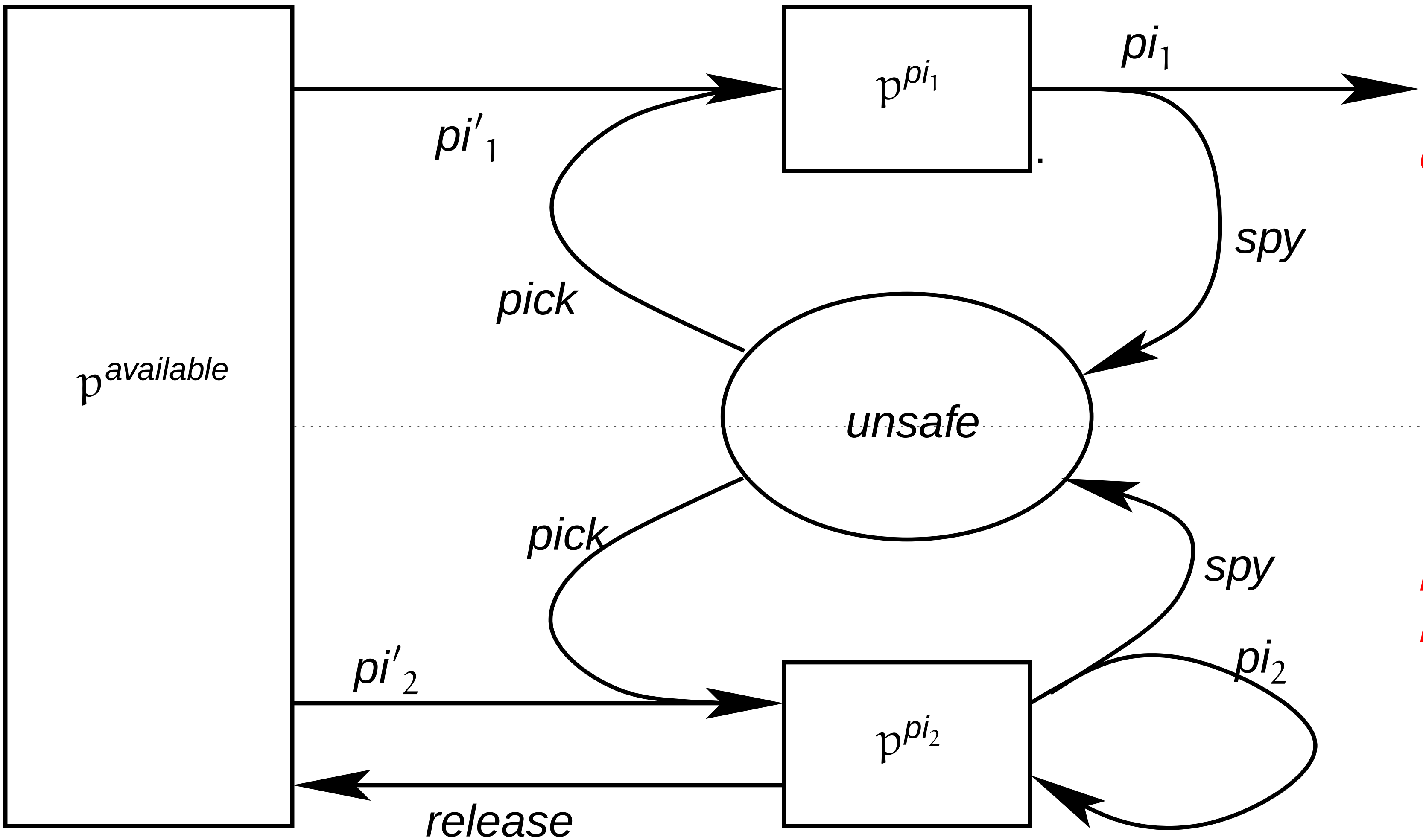
where

new := ($*$(ν *channel*)$*$unsafe![*channel*])

spoil$_k$:= ($*$unsafe?[c]unsafe?[x_1]...unsafe?[x_k]c![x_1,...,x_k])

spy$_k$:= ($*$unsafe?[c]c?[x_1,...,x_k] (($*$unsafe![x_1]) | ... | ($*$unsafe![x_k])))

Context instance



computatio

migration
replication

Coherence

1. This context encoding is sound for any encoding model;
2. This context encoding is complete for the π-calculus.

Incompleteness

The approach is incomplete.

1. In mobile ambients:

$$(\nu^{l_1}\text{secret})(\nu^{l_2}\text{a})(\nu^{l_3}\text{b})(\text{a}^1[\text{b}^2[\textit{in}^3 c.\langle\text{secret}\rangle^4] \mid c^5[\textit{in}^6 \text{a}.\textit{open}^7 \text{b}]])$$

2. in the spi-calculus:

$$(\nu^{\alpha}\text{secret})c^1\langle M\rangle.\text{let}^2\ x = \text{getmessage}(M)\ \text{in let}^3\ y = \text{th}_i^n(M)\ \text{in}\ \overline{\text{test}}^4\langle\text{secre}$$

Future Works
Enriching the META-language

- symmetric communication (fusion calculus),

  $\Longrightarrow$ theoretical problem;

- term defined up to an equational theory (applied pi),

  $\Longrightarrow$ analyzing cryptographic protocols with XOR;

- higher order communication;

  $\Longrightarrow$ agents may communicate running programs;

  $\Longrightarrow$ agents may duplicate running programs.

To be continued

We focus on reachability properties.
We distinguish between recursive instances of components.

We design three families of analyses:

1. environment analyses capture dynamic topology properties
   (non-uniform control flow analysis, secrecy, confinement, . . .)
2. occurrence counting captures concurrency properties
   (mutual exclusion, non exhaustion of resources)
3. thread partitioning mixes both dynamic topology and concurrency properties
   (absence of race conditions, authentication, . . .).