MPRI

Abstract Interpretation of Mobile Systems

Jérôme Feret

Département d'Informatique de l'École Normale Supérieure

INRIA, ÉNS, CNRS

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

January, 25/February 1 2024

Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Systèmes mobiles

Un ensemble de composants qui interagissent.
Ces interactions permettent de :

- synchroniser l'exécution de ces composants,
- changer les liaisons entre les composants,
- créer des nouveaux composants.

Le nombre de composants n'est pas borné !

Champs d'application :

- protocoles de communication,
- protocoles cryptographiques,
- systèmes reconfigurables,
- systèmes biologiques,
- . . . .

Démarche

Construction de sémantiques abstraites :

- correctes, automatiques, décidables,
- mais approchées (indécidabilité).

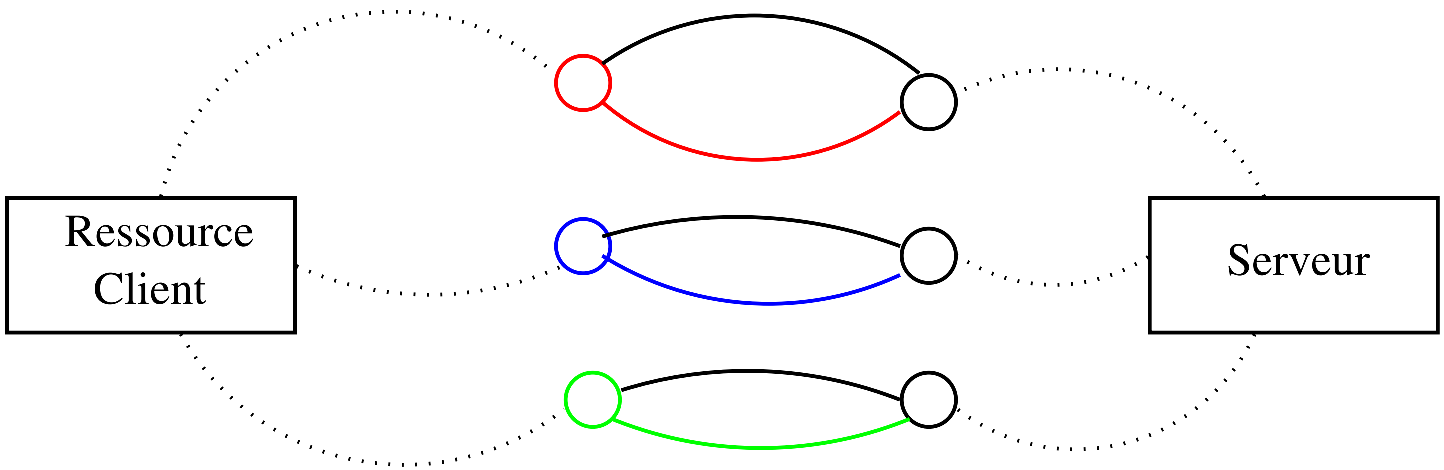
Approche indépendante du modèle:

1. conception d'un META-langage pour encoder les modèles existants ;
2. développement d'analyses au niveau de ce META-langage.

Trois familles d'analyses:

1. analyse des liaisons dynamiques entre les composants :
   (confinement, confidentialité, . . .)
2. dénombrement des composants :
   (exclusions mutuelles, non-épuisement des ressources, . . .)
3. analyse des unités de calculs :
   (absence de conflit, authentification, intégrité des mises à jour, . . .).

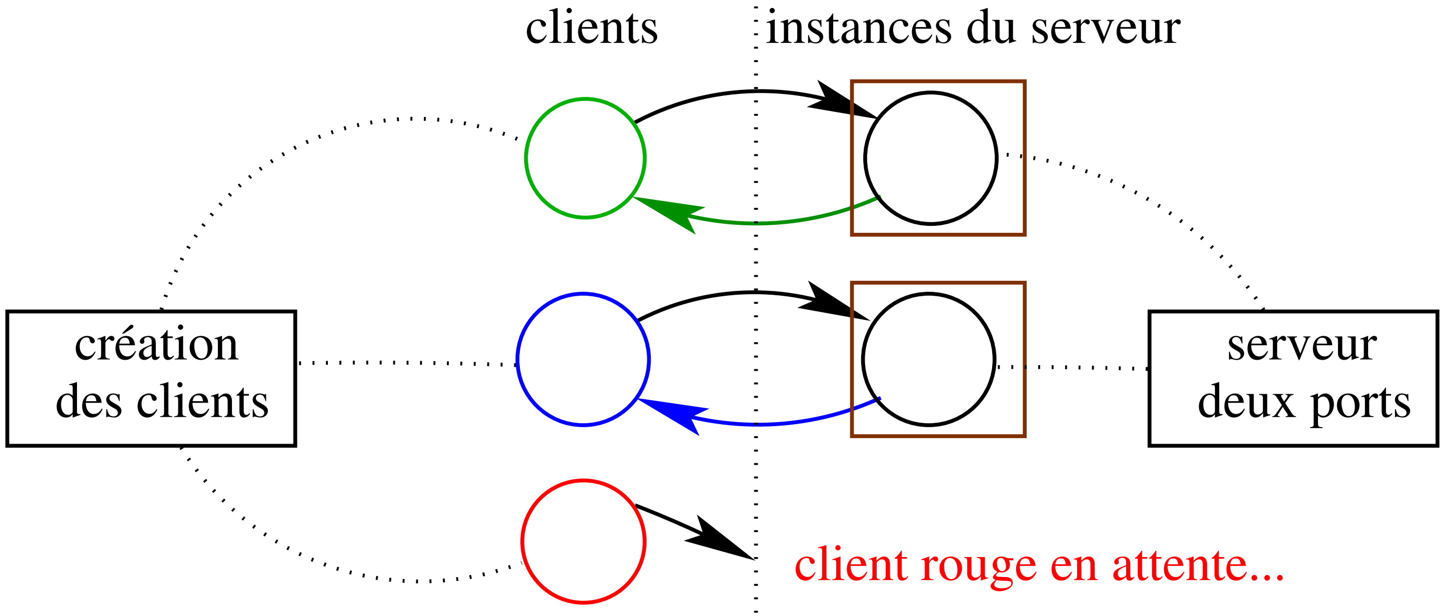
Analyse des liaisons entre les composants :
Quels composants peuvent interagir ?



L'analyse distingue les composants récursifs
Domaines abstraits : relations entre des mots.

Publications : Feret — SAS'00, SAS'01, ESOP'02, JLAP'05

Analyse du nombre des composants :
Borne le nombre de composants



Nouveau domaine abstrait : relations numériques
(invariants affines et intervalles).

Publication : Feret — GETCO'00, JLAP'05

Partitionnement de tâches

Principe :

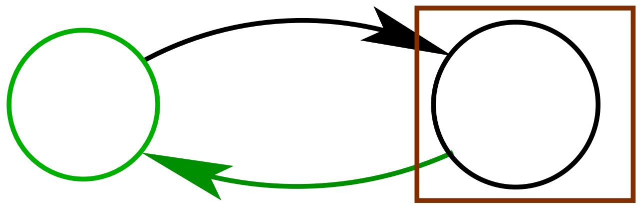
- regrouper les composants en unités de calcul,

  $\Longrightarrow$ grâce à l'analyse des liaisons entre les composants ;
- compter le nombre de composants dans chaque unité de calcul,

  $\Longrightarrow$ grâce à l'analyse de dénombrement.

Intérêt :

- chaque session est isolée,



ce qui permet aux analyses de se focaliser sur chacune des sessions.

Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Mobile system

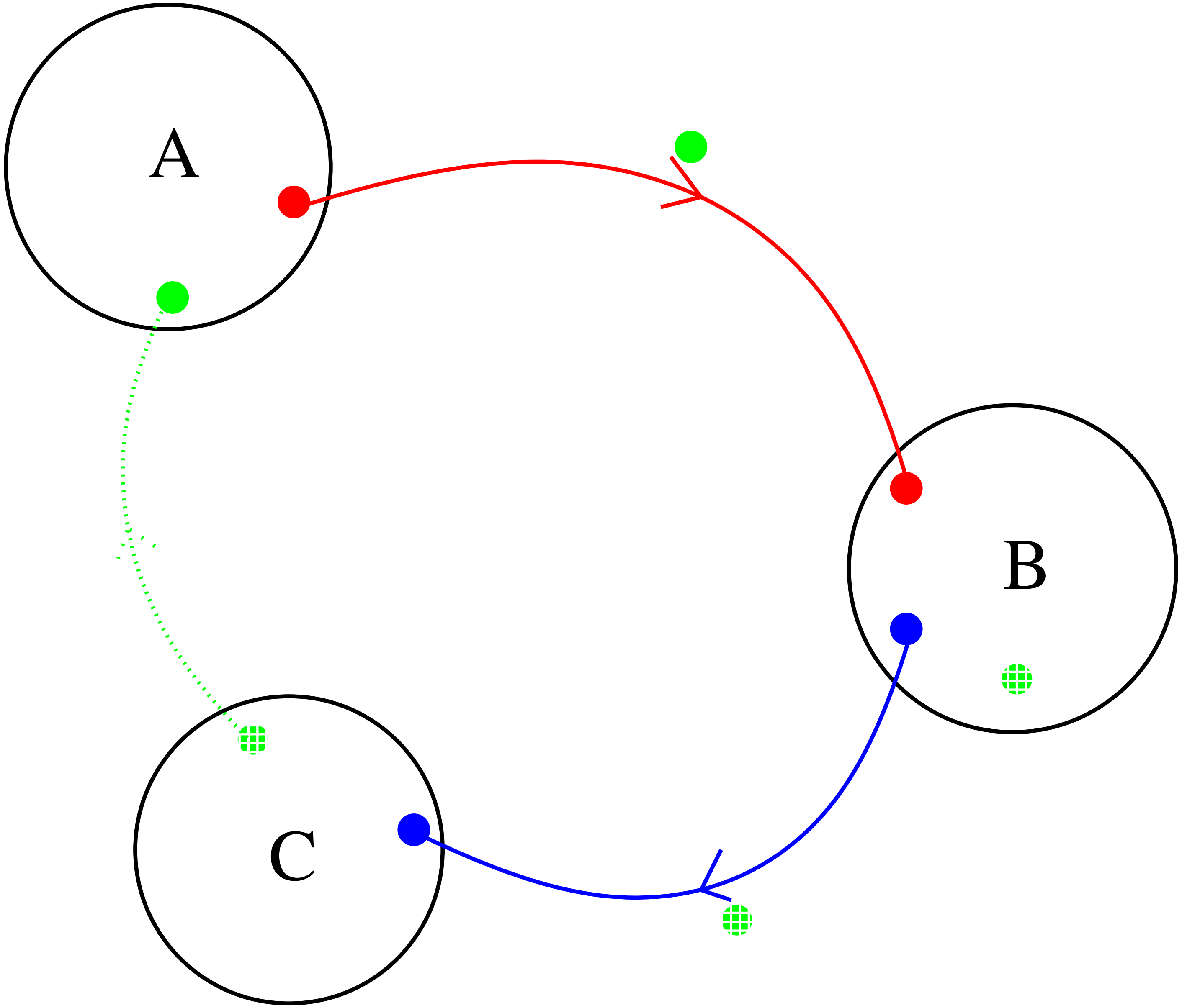
A pool of processes which interact and communicate:

Interactions control:

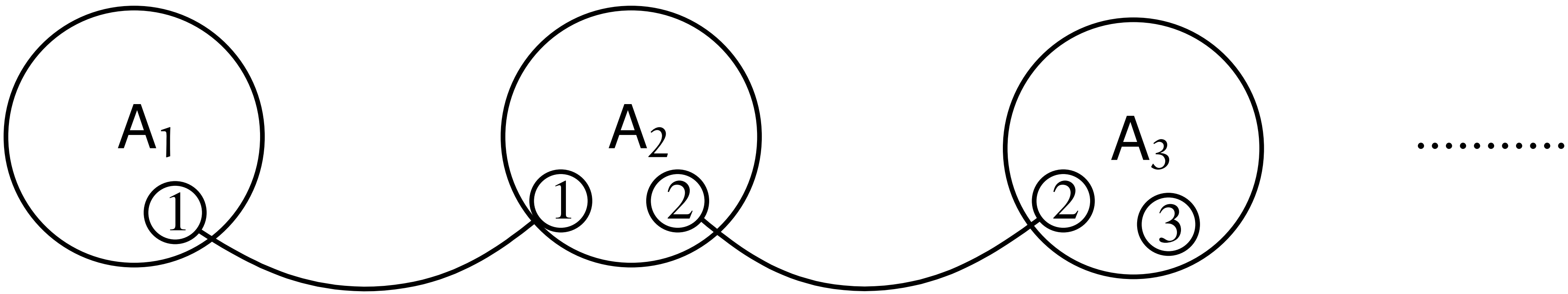
- process synchronization;
- update of link between processes (communication, migration);
- process creation.

The number of processes may be unbounded !

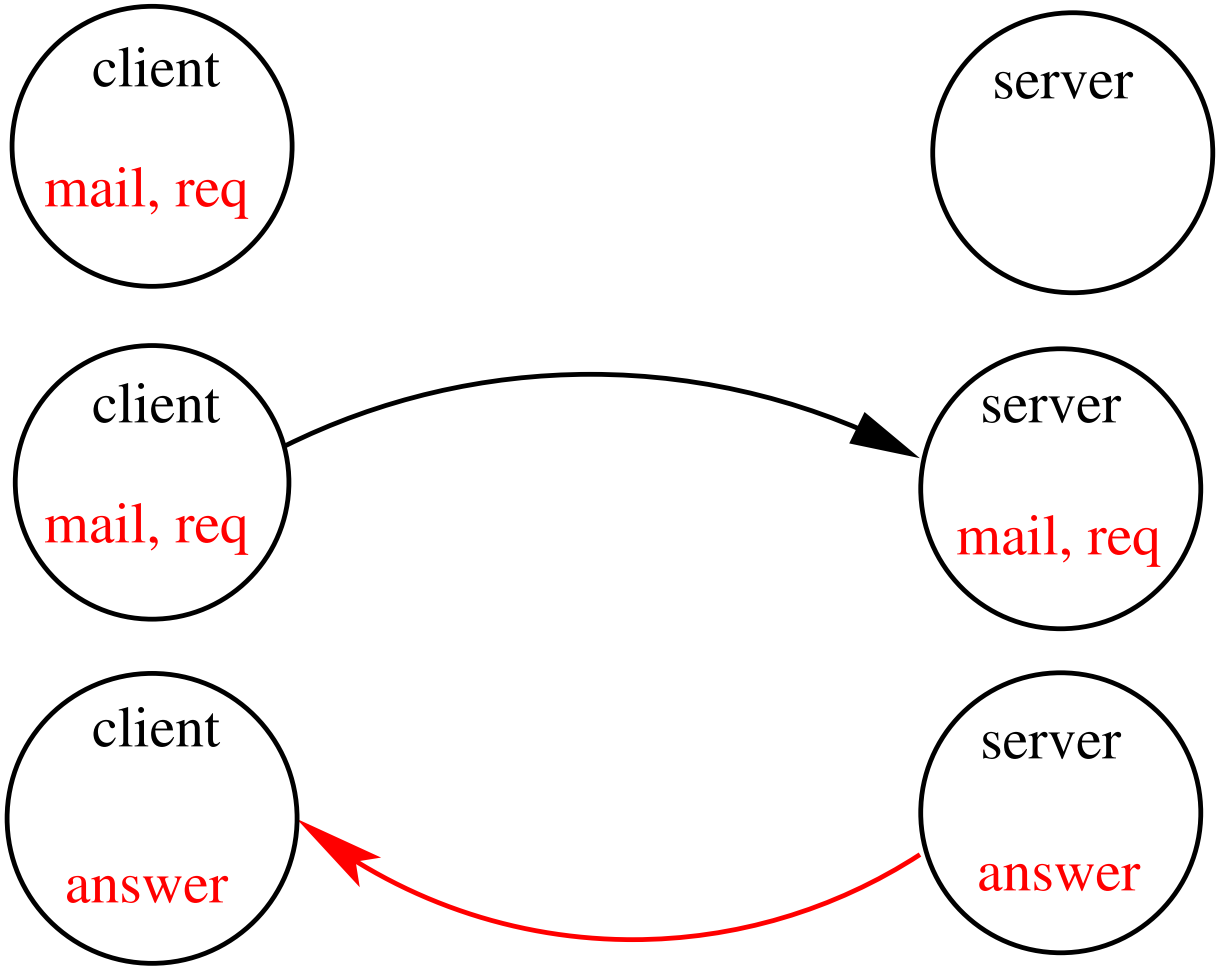
Dynamic linkage of agents



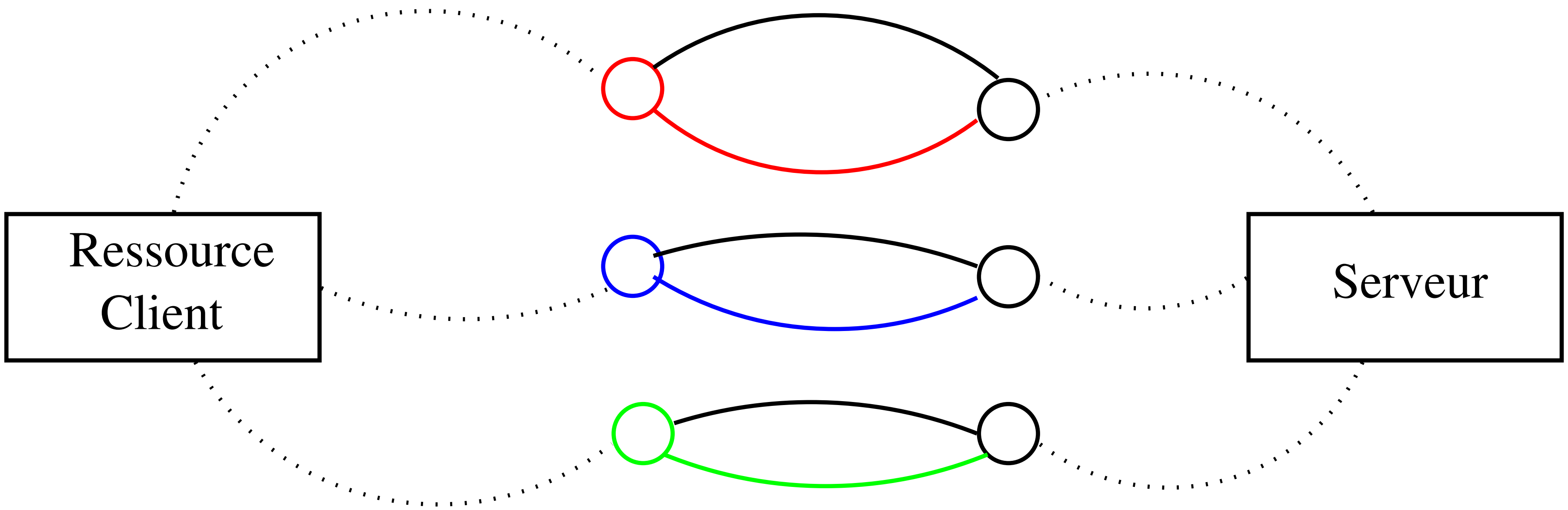
Dynamic creation of agents



A connection:



A network



π-calculus: syntax

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

$$
\begin{array}{rcl}
P & ::= & \text{action}.P \\
 & | & (P \mid P) \\
 & | & (P + P) \\
 & | & (\nu\ x)P \\
 & | & \emptyset
\end{array}
\qquad
\begin{array}{rcl}
\text{action} & ::= & c!^{i}[x_1, ..., x_n] \\
 & | & c?^{i}[x_1, ..., x_n] \\
 & | & *c?^{i}[x_1, ..., x_n]
\end{array}
$$

where $n \geqslant 0$, c, x_1, ..., x_n, x, $\in$ *Name*, $i \in$ *Label*.
ν and $?$ are the only name binders.
$\mathrm{fn}(P)$: free variables in P,
$\mathrm{bn}(P)$: 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:

$$
\begin{array}{rcl}
P+Q & \longrightarrow & P \\
P+Q & \longrightarrow & Q \\
c?^{i}[\overline{y}]Q \mid c!^{j}[\overline{x}]P & \stackrel{i,j}{\longrightarrow} & Q[\overline{y} \leftarrow \overline{x}] \mid P \\
*c?^{i}[\overline{y}]Q \mid c!^{j}[\overline{x}]P & \stackrel{i,j}{\longrightarrow} & Q[\overline{y} \leftarrow \overline{x}] \mid *c?^{i}[\overline{y}]Q \mid P
\end{array}
$$

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

Congruence relation

- the congruence relation reveals redexs:

$$
\begin{array}{rcll}
P \mid Q & \equiv & Q \mid P & & \text{(Commutativity)} \\
P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & & \text{(Associativity)} \\
(\nu\, x)P & \equiv & (\nu\, y)P[x \leftarrow y] & \text{if } y \notin fn(P) & \text{(α-conversion)} \\
(\nu\, x)(\nu\, y)P & \equiv & (\nu\, y)(\nu\, x)P & & \text{(Swapping)} \\
((\nu\, x)P) \mid Q & \equiv & (\nu\, x)(P \mid Q) & \text{if } x \notin fn(Q) & \text{(Extrusion)} \\
(\nu\, x)\emptyset & \equiv & \emptyset & & \text{(Garbage collection)}
\end{array}
$$

Exporting a channel

$(\nu\ a)((\nu\ x)(a?[y].P(x,y)|(\nu\ y)(\nu\ x)a![x].R(x,y)))$

$\equiv$ (α-conversion, swapping and extrusion)

$(\nu\ a)(\nu\ x_1)(\nu\ x_2)(\nu\ y)(a?[y].P(x_1,y)|a![x_2].R(x_2,y))$

$\longrightarrow$

$(\nu\ a)(\nu\ x_1)(\nu\ x_2)(\nu\ y)(P(x_1,x_2)|R(x_2,y))$

$\equiv$ (swapping and extrusion)

$(\nu\ a)(\nu\ x_2)((\nu\ x_1)P(x_1,x_2)|(\nu\ y)R(x_2,y))$

Example: syntax

$$\mathcal{S} := (\nu\ \mathrm{port})(\nu\ \mathrm{gen})\ (\textbf{Server} \mid \textbf{Customer} \mid \mathrm{gen}!^{0}[])$$

where

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

$$\textbf{Customer} := *\mathrm{gen}?^{3}[]\ ((\nu\ \mathit{data})\ (\nu\ \mathit{email})\ (\mathrm{port}!^{4}[\mathit{data}, \mathit{email}] \mid \mathrm{gen}!^{5}[]))$$

Example: computation

$$
\begin{aligned}
& (\nu\ \mathsf{port})(\nu\ \mathsf{gen}) \\
& \quad (\mathbf{Server} \mid \mathbf{Customer} \mid \mathsf{gen}!^{0}[]) \\
\xrightarrow{3,0}\ & (\nu\ \mathsf{port})(\nu\ \mathsf{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1) \\
& \quad (\mathbf{Server} \mid \mathbf{Customer} \mid \mathsf{gen}!^{5}[] \mid \mathsf{port}!^{4}[\mathit{data}_1,\mathit{email}_1]) \\
\xrightarrow{1,4}\ & (\nu\ \mathsf{port})(\nu\ \mathsf{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1) \\
& \quad (\mathbf{Server} \mid \mathbf{Customer} \mid \mathsf{gen}!^{5}[] \mid \mathit{email}_1!^{2}[\mathit{data}_1]) \\
\xrightarrow{3,5}\ & (\nu\ \mathsf{port})(\nu\ \mathsf{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1)(\nu\ \mathit{data}_2)(\nu\ \mathit{email}_2) \\
& \quad (\mathbf{Server} \mid \mathbf{Customer} \mid \mathsf{gen}!^{5}[] \mid \mathit{email}_1!^{2}[\mathit{data}_1] \mid \mathsf{port}!^{4}[\mathit{data}_2,\mathit{email}_2]) \\
\xrightarrow{1,4}\ & (\nu\ \mathsf{port})(\nu\ \mathsf{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1)(\nu\ \mathit{data}_2)(\nu\ \mathit{email}_2) \\
& \quad (\mathbf{Server} \mid \mathbf{Customer} \mid \mathsf{gen}!^{5}[] \mid \mathit{email}_1!^{2}[\mathit{data}_1] \mid \mathit{email}_2!^{2}[\mathit{data}_2])
\end{aligned}
$$

α-conversion

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

$$(\nu\ \mathrm{port})(\nu\ \mathrm{gen})(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_1)$$
$$(\nu\ \mathit{data}_2)(\nu\ \mathit{email}_2)$$
$$(\mathbf{Server} \mid \mathbf{Customer} \mid \mathrm{gen}!^5[]$$
$$\mid \mathit{email}_1!^4[\mathit{data}_1] \mid \mathit{email}_2!^4[\mathit{data}_2])$$

$\sim_\alpha$

$$(\nu\ \mathrm{port})(\nu\ \mathrm{gen})(\nu\ \mathit{data}_2)$$
$$(\nu\ \mathit{email}_1)(\nu\ \mathit{data}_1)(\nu\ \mathit{email}_2)$$
$$(\mathbf{Server} \mid \mathbf{Customer} \mid \mathrm{gen}!^5[]$$
$$\mid \mathit{email}_1!^4[\mathit{data}_2] \mid \mathit{email}_2!^4[\mathit{data}_1])$$

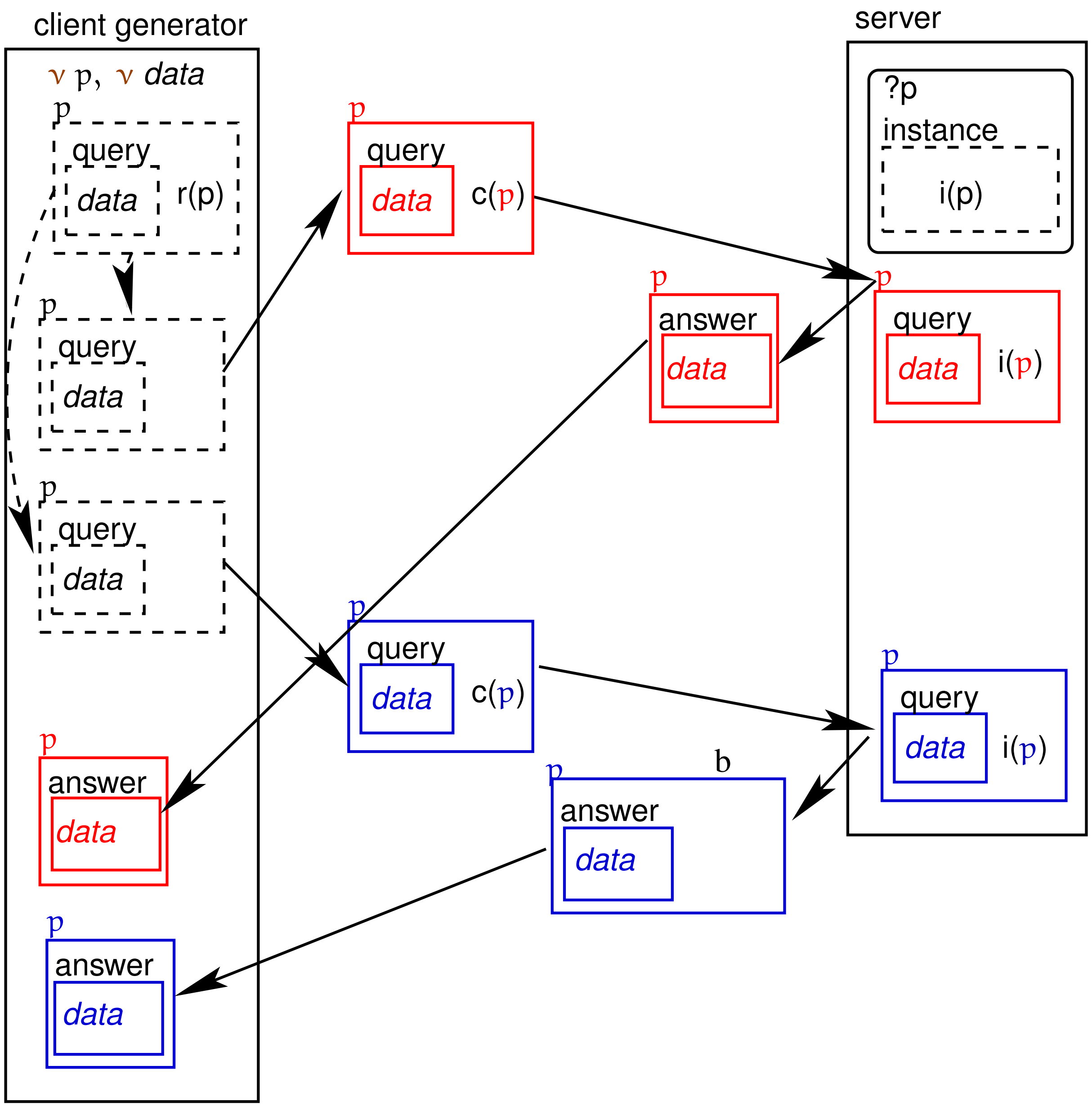
Mobile Ambients

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.

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

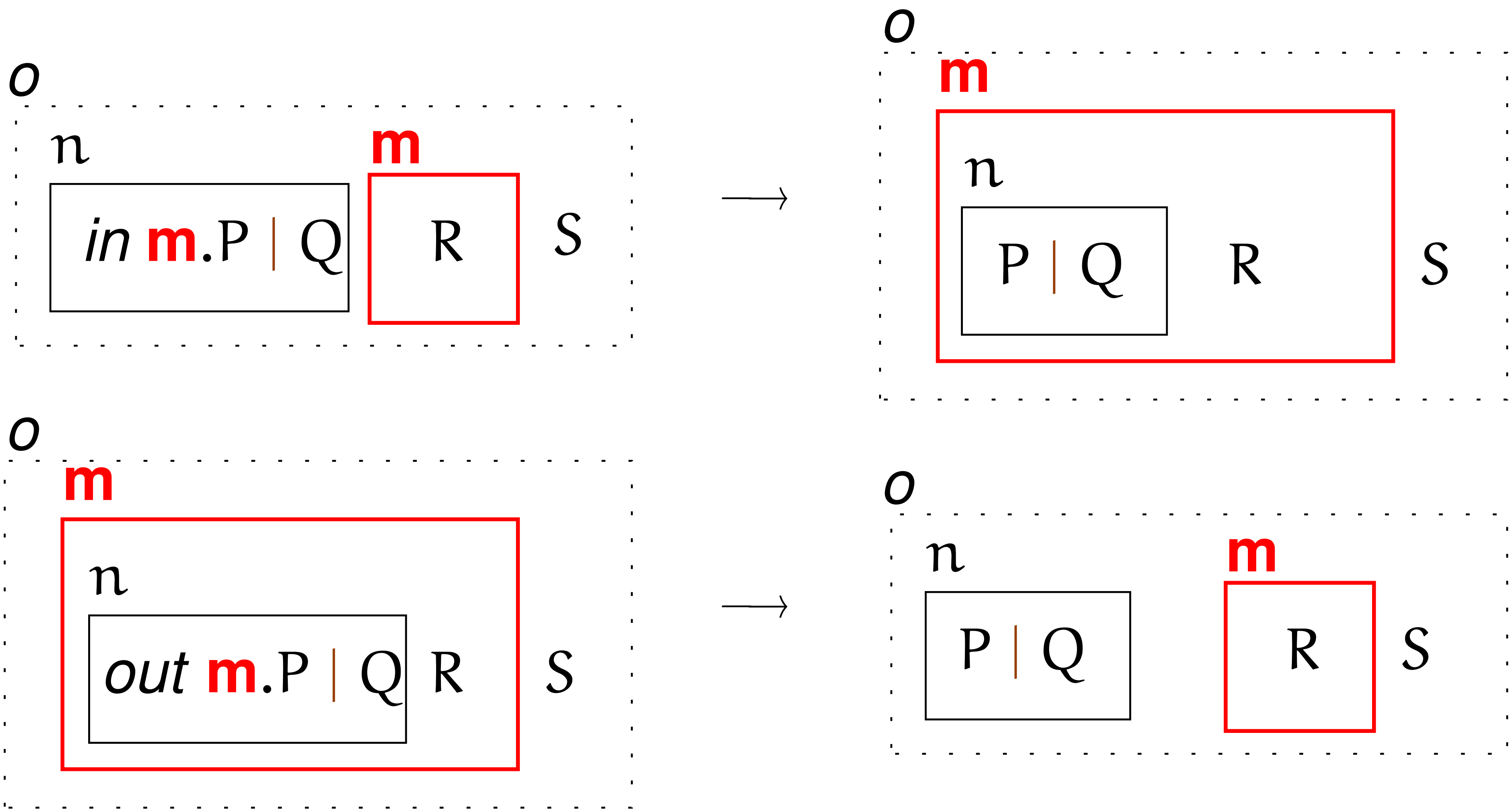
Capability and actions

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

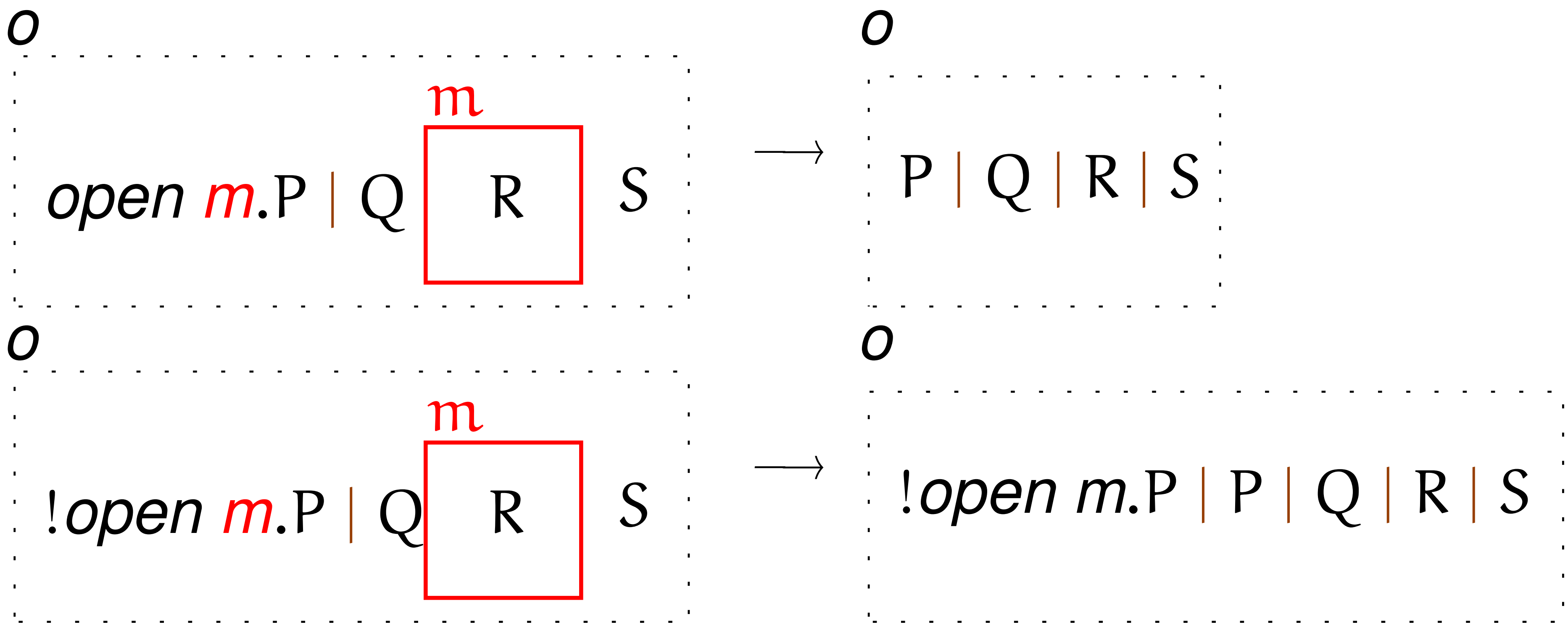
$$
\begin{array}{rcll}
\mathit{io} & ::= & (n)^{l}.P & \text{(input action)} \\
 & | & !(n)^{l}.P & \text{(input action with replication)} \\
 & | & \langle n\rangle^{l} & \text{(async output action)}
\end{array}
$$

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

Ambient Migration



Ambient Dissolution



Communication

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

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

An *ftp*-server

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

where

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

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

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

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

$$
\begin{aligned}
&(\nu \mathbf{Pub})(\mathbf{S} \mid !(x)^{11}.C \mid \langle \mathsf{make} \rangle^{21})\\
\longrightarrow\\
&(\nu \mathbf{Pub}) \left(\begin{array}{l} !(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2] \mid \\ (\nu\ q_1)(\nu\ p_1)p_1{}^{12}\left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathbf{C}_2 \mid \\ \mathit{in}^{16}\mathsf{server}.\mathsf{duplicate}^{17}[\mathit{out}^{18} p_1.\langle p_1 \rangle^{19}] \end{array} \right] \end{array} \right)\\
\longrightarrow\\
&(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\\
&\left(!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1}\left[\mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12}\left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathbf{C}_2 \mid \\ \mathsf{duplicate}^{17}[\mathit{out}^{18} p_1.\langle p_1 \rangle^{19}] \end{array} \right] \right] \right)\\
\longrightarrow\\
&(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\\
&\left(!(x)^{11}.C \mid \langle \mathsf{make} \rangle^{20} \mid \mathsf{server}^{1}\left[\begin{array}{l} !\mathit{open}^{2}\mathsf{duplicate} \mid \mathbf{S}_2 \mid \mathsf{duplicate}^{17}[\langle p_1 \rangle^{19}] \mid \\ p_1{}^{12}[\mathsf{request}^{13}[\langle q_1 \rangle^{14}] \mid \mathbf{C}_2] \end{array} \right] \right)
\end{aligned}
$$

$$
\begin{aligned}
&(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\\
&\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\begin{array}{l} !\mathit{open}^{2}\mathsf{duplicate} \mid \mathbf{S}_2 \mid \mathsf{duplicate}^{17}[\langle p_1\rangle^{19}] \mid \\ p_1{}^{12}[\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathbf{C}_2] \end{array}\right]\right)\\
\longrightarrow\\
&(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\\
&\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\begin{array}{l} \mathbf{S}_1 \mid !(k)^{3}.\mathsf{instance}^{4}[\mathbf{I}] \mid \langle p_1\rangle^{19} \mid \\ p_1{}^{12}[\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathbf{C}_2] \end{array}\right]\right)\\
\longrightarrow\\
&(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\\
&\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\begin{array}{l} \mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12}[\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathbf{C}_2] \mid \\ \mathsf{instance}^{4}[\mathit{in}^{5}p_1.\mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathsf{I}_1|\mathsf{I}_2)] \end{array}\right]\right)\\
\longrightarrow\\
&(\nu \mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\\
&\left(!(x)^{11}.C \mid \langle \mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\begin{array}{l} \mathbf{S}_1 \mid \mathbf{S}_2 \mid \\ p_1{}^{12}\left[\begin{array}{l} \mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathit{open}^{15}\mathsf{instance} \mid \\ \mathsf{instance}^{4}[\mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathsf{I}_1|\mathsf{I}_2)] \end{array}\right] \end{array}\right]\right)
\end{aligned}
$$

$$
\begin{aligned}
&(\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\\
&\left(!(x)^{11}.C \mid \langle\mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\begin{array}{l}\mathbf{S}_1 \mid \mathbf{S}_2 \mid\\ p_1{}^{12}\left[\begin{array}{l}\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid \mathit{open}^{15}\mathsf{instance} \mid\\ \mathsf{instance}^{4}[\mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathsf{l}_1 \mid \mathsf{l}_2)]\end{array}\right]\end{array}\right]\right)\\
\longrightarrow\\
&(\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\\
&\left(!(x)^{11}.C \mid \langle\mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}\left[\mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12}\left[\begin{array}{l}\mathsf{request}^{13}[\langle q_1\rangle^{14}] \mid\\ \mathit{open}^{6}\mathsf{request}.(\mathit{rep})^{7}(\mathsf{l}_1 \mid \mathsf{l}_2)\end{array}\right]\right]\right)\\
\longrightarrow^{*}\\
&(\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\\
&(!(x)^{11}.C \mid \langle\mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2 \mid p_1{}^{12}[\mathsf{answer}^{8}[\langle q_1\rangle^{9}] \mid \mathit{out}^{10}\mathsf{server}]])\\
\longrightarrow\\
&(\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1)\\
&(!(x)^{11}.C \mid \langle\mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2] \mid p_1{}^{12}[\mathsf{answer}^{8}[\langle q_1\rangle^{9}]])\\
\longrightarrow^{*}\\
&(\nu\mathbf{Pub})(\nu\ q_1)(\nu\ p_1)(\nu\ q_2)(\nu\ p_2)(!(x)^{11}.C \mid \langle\mathsf{make}\rangle^{20} \mid \mathsf{server}^{1}[\mathbf{S}_1 \mid \mathbf{S}_2]\\
&\qquad\qquad \mid p_1{}^{12}[\mathsf{answer}^{8}[\langle q_1\rangle^{9}]] \mid p_2{}^{12}[\mathsf{answer}^{8}[\langle q_2\rangle^{9}]])
\end{aligned}
$$

Shared-memory example
Motivation

We want to describe in the π-calculus a shared-memory in which:

- each process can allocate new cells,
- each authorized process can read the content of a cell,
- each authorized process can write inside a cell, overwriting the former content.

Shared-memory example
Specification

A memory cell will be denoted by three channel names, *cell*, *read*, *write*:

- a channel name *cell* describes the content of the cell:

  the process $cell![data]$ means that the cell *cell* contains the information *data*, this name is internal to the memory (not visible by the user).
- a channel name *read* allows reading requests:

  the process $read![port]$ is a request to read the content of the cell, and send it to the port *port*,
- a channel name *write* allows writing requests:

  the process $write![data]$ is a request to write the information *data* inside the cell.

Shared Memory
Encoding

$\mathsf{System} := (\nu\ \mathsf{create})(\nu\ \mathsf{null})(*\mathsf{create}?[\mathsf{d}].\mathsf{Allocate}(d))$

$$\mathsf{Allocate}(\mathsf{d}) := (\nu\ \mathit{cell})(\nu\ \mathit{write})(\nu\ \mathit{read})\big(\mathsf{init}(\mathit{cell})\ \mid\ \mathsf{read}(\mathit{read},\mathit{cell})\ \mid\ \mathsf{write}(\mathit{write},\mathit{cell})\ \mid\ \mathsf{d}![\mathit{read};\mathit{write}]\big)$$

where

- $\mathsf{init}(\mathit{cell}) := \mathit{cell}![\mathsf{null}]$
- $\mathsf{read}(\mathit{read},\mathit{cell}) := *\mathit{read}?[\mathit{port}].\mathit{cell}?[u].(\mathit{cell}![u]\ \mid\ \mathit{port}![u])$
- $\mathsf{write}(\mathit{write},\mathit{cell}) := *\mathit{write}?[\mathit{data}].\mathit{cell}?[u].\mathit{cell}![\mathit{data}]$

Shared-memory example
Trace example

$(\nu\ \mathrm{create})(\nu\ \mathrm{null})$
$\quad(*\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(d)$
$\quad\mid (\nu\ address)(\nu\ data)\mathrm{create}![address].address?[r;w].w![data].r![address])$
$\longrightarrow$
$(\nu\ \mathrm{create})\ (\nu\ \mathrm{null})\ (\nu\ cell)\ (\nu\ write)\ (\nu\ read)\ (\nu\ address)\ (\nu\ data)$
$\quad(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(read,cell)\ \mid\ \mathrm{write}(write,cell)$
$\quad\mid\ cell![\mathrm{null}]\ \mid\ address![read,write]\ \mid\ address?[r;w].w![data].r![address])$
$\longrightarrow$
$(\nu\ \bar{\mathrm{c}})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(read,cell)\ \mid\ \mathrm{write}(write,cell)$
$\qquad\mid\ cell![\mathrm{null}]\mid\ write![data].read![address])$
$\longrightarrow$
$(\nu\ \bar{\mathrm{c}})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(read,cell)\ \mid\ \mathrm{write}(write,cell)$
$\qquad\mid\ cell![\mathrm{null}]\ \mid\ cell?[u].cell![data]\mid read![address])$

Shared-memory example
Expected Derivation

$(\nu\ \bar{\mathrm{c}})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})$
$\quad \mid\ \mathit{cell}![\mathrm{null}]\ \mid\ \mathit{cell}?[u].\mathit{cell}![\mathit{data}]\ \mid\ \mathit{read}![\mathit{address}])$

$\longrightarrow$

$(\nu\ \bar{\mathrm{c}})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})$
$\quad \mid\ \mathit{cell}![\mathit{data}] \mid \mathit{read}![\mathit{address}])$

$\longrightarrow$

$(\nu\ \bar{\mathrm{c}})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})$
$\quad \mid\ \mathit{cell}![\mathit{data}] \mid \mathit{cell}?[\mathrm{u}].(\mathit{cell}![\mathrm{u}] \mid \mathit{address}![\mathrm{u}]))$

$\longrightarrow$

$(\nu\ \bar{\mathrm{c}})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})$
$\quad \mid\ \mathit{cell}![\mathit{data}] \mid \mathit{address}![\mathit{data}])$

Shared-memory example
Unexpected Derivation

$(\nu\ \bar{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})$
$\quad \mid\ \mathit{cell}![\mathrm{null}]\ \mid\ \mathit{cell}?[u].\mathit{cell}![\mathit{data}] \mid \mathit{read}![\mathit{address}])$

$\longrightarrow$

$(\nu\ \bar{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})$
$\quad \mid\ \mathit{cell}![\mathrm{null}]\ \mid\ \mathit{cell}?[\mathrm{u}].\mathit{cell}![\mathit{data}] \mid \mathit{cell}?[\mathrm{u}].(\mathit{cell}![\mathrm{u}] \mid \mathit{address}![\mathrm{u}]))$

$\longrightarrow$

$(\nu\ \bar{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})$
$\quad \mid\ \mathit{cell}![\mathrm{null}]\ \mid\ \mathit{cell}?[\mathrm{u}].\mathit{cell}![\mathit{data}] \mid \mathit{address}![\mathrm{null}])$

$\longrightarrow$

$(\nu\ \bar{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})$
$\quad \mid\ \mathit{cell}![\mathit{data}] \mid \mathit{address}![\mathrm{null}])$

Shared-memory example
Enforcing synchronisation

$\text{System} := (\nu\ \text{create})(\nu\ \text{null})(*\text{create}?[\text{d}].\text{Allocate}(d))$

$\text{Allocate}(\text{d}) :=$

$$(\nu\ cell)(\nu\ write)(\nu\ read)$$
$$\text{init}(cell)\ |\ \text{read}(read,cell)\ |\ \text{write}(write,cell)\ |\ \text{d}![read;write]$$

where

- $\text{init}(cell) := cell![\text{null}]$
- $\text{read}(read,cell) := *read?[port].cell?[u](cell![u]\ |\ port![u])$
- $\text{write}(write,cell) := *write?[data,ack].cell?[u].(cell![data]\ |\ ack![])$

$$(\nu\ \mathrm{create})(\nu\ \mathrm{null})$$

$$(*\mathrm{create}?[\mathrm{d}].\mathrm{Allocate}(d)$$

$$\mid (\nu\ \mathit{address})(\nu\ \mathit{data})(\nu\ \mathit{ack})$$

$$\mathrm{create}![\mathit{address}].\mathit{address}?[r;w].w![\mathit{data};\mathit{ack}].\mathit{ack}?[].r![\mathit{address}])$$

$$\longrightarrow$$

$$(\nu\ \overline{\mathrm{c}})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell}) \mid\ \mathit{cell}![\mathrm{null}]$$

$$\mid\ \mathit{address}![\mathit{read},\mathit{write}]\ \mid\ \mathit{address}?[r;w].w![\mathit{data};\mathit{ack}].\mathit{ack}?[].r![\mathit{address}])$$

$$\longrightarrow$$

$$(\nu\ \overline{\mathrm{c}})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})$$

$$\mid\ \mathit{cell}![\mathrm{null}] \mid\ \mathit{write}![\mathit{data};\mathit{ack}].\mathit{ack}?[].\mathit{read}![\mathit{address}])$$

$$\longrightarrow$$

$$(\nu\ \overline{\mathrm{c}})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(\mathit{read},\mathit{cell})\ \mid\ \mathrm{write}(\mathit{write},\mathit{cell})$$

$$\mid\ \mathit{cell}![\mathrm{null}]\ \mid\ \mathit{cell}?[u].(\mathit{cell}![\mathit{data}] \mid \mathit{ack}![])$$

$$\mid \mathit{ack}?[].\mathit{read}![\mathit{address}])$$

$$(\nu\ \bar{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(read,cell)\ \mid\ \mathrm{write}(write,cell)$$
$$\mid\ cell![\mathrm{null}]\ \mid\ cell?[u].(cell![data] \mid ack![])$$
$$\mid ack?[].read![address])$$
$$\longrightarrow$$
$$(\nu\ \bar{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(read,cell)\ \mid\ \mathrm{write}(write,cell)$$
$$\mid\ (cell![data] \mid ack![]) \mid ack?[].read![address])$$
$$\longrightarrow$$
$$(\nu\ \bar{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(read,cell)\ \mid\ \mathrm{write}(write,cell)$$
$$\mid\ cell![data] \mid read![address])$$
$$\longrightarrow$$
$$(\nu\ \bar{c})(\ *\mathrm{create}?[d].\mathrm{Allocate}(d)\ \mid\ \mathrm{read}(read,cell)\ \mid\ \mathrm{write}(write,cell)$$
$$\mid\ cell![data] \mid\ address![data])$$

Shared-memory example
Using Mutex

$\text{System} := (\nu\ \text{create})(\nu\ \text{null})(*\text{create}?[\text{d}]\text{Allocate}(d))$

$\text{Allocate}(d) := (\nu\ cell)(\nu\ mutex)(\nu\ nomutex)(\nu\ write)(\nu\ read)(\nu\ lock)(\nu\ unlock)$
$\quad \text{init}(cell,mutex) \mid \text{read}(read,cell) \mid \text{write}(write,cell)$
$\quad \mid \text{lock}(lock,mutex,nomutex) \mid \text{unlock}(unlock,mutex,nomutex)$
$\quad \mid d![read;write;lock;unlock]$

where

$\text{init}(cell,mutex) := cell![\text{null}] \mid mutex![]$
$\text{read}(read,cell) := *read?[port].cell?[u](cell![u] \mid port![u])$
$\text{write}(write,cell) := *write?[data,ack].cell?[u].(cell![data] \mid ack![])$
$\text{lock}(lock,mutex,nomutex) := *lock?[ack].mutex?[].(ack![] \mid nomutex![])$
$\text{unlock}(unlock,mutex,nomutex) := *unlock?[ack].nomutex?[].(ack![] \mid mutex![])$

Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Motivation

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

We design three families of analyses:

1. environment analyses capture dynamic 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 and concurrency properties
   (absence of race condition, authentication, ...).

Non-standard semantics

A refined semantics in where

- recursive instances of processes are identified with unambiguous markers;
- channel names are stamped with the marker of the process which has declared them.

Example: non-standard configuration

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

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

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.

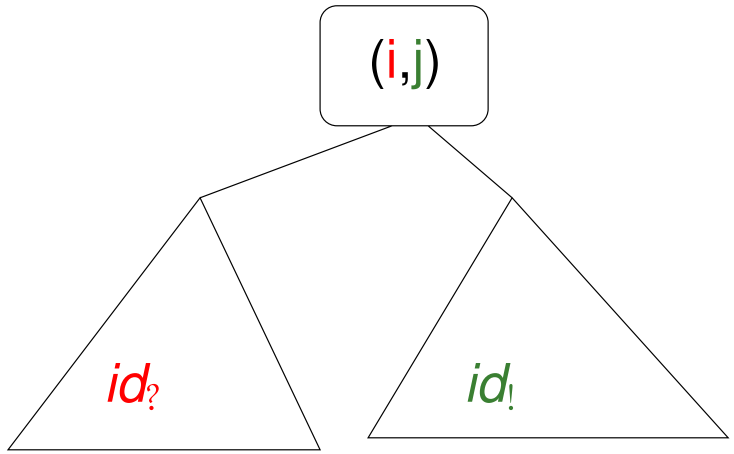
Marker allocation

Markers describe the history of the replications which have led to the creation of the threads.
They are binary trees:

- leaves are not labeled;
- nodes are labeled with a pair $(\mathrm{i}, \mathrm{j}) \in \mathit{Label}^2$.

They are recursively calculated when fetching resources as follows:

$id_* :$



Small step semantics

Small step semantics is given by a transition system:

- an initial configuration;
- three structural reduction rules which simulate the congruence relation;
- four action reduction rules which simulate the transition relation.

Initial configuration

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

Structural rules

$$C \cup \{(P \mid Q, \mathit{id}, E)\} \xrightarrow{\varepsilon} C \cup \{(P, \mathit{id}, E_{|fn(P)}); (Q, \mathit{id}, E_{|fn(Q)})\}$$

$$C \cup \{((\nu\ x)P, \mathit{id}, E)\} \xrightarrow{\varepsilon} C \cup \{(P, \mathit{id}, E[x \longrightarrow (x, \mathit{id})]_{|fn(P)})\}$$

$$C \cup \{(\emptyset, \mathit{id}, E)\} \xrightarrow{\varepsilon} C$$

Communication rules

$$\frac{E_?(y) = E_!(x)}{C \cup \left\{ \begin{array}{l} (y?^i[\overline{y}]P, \mathit{id}_?, E_?); \\ (x!^j[\overline{x}]Q, \mathit{id}_!, E_!) \end{array} \right\} \xrightarrow{(i,j)} C \cup \left\{ \begin{array}{l} (P, \mathit{id}_?, E_?[\overline{y} \to E_![\overline{x}]]_{|fn(P)}); \\ (Q, \mathit{id}_!, E_{!|fn(Q)}) \end{array} \right\}}$$

$$\frac{E_?(y) = E_!(x)}{C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, \mathit{id}_?, E_?); \\ (x!^j[\overline{x}]Q, \mathit{id}_!, E_!) \end{array} \right\} \xrightarrow{(i,j)} C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, \mathit{id}_?, E_?); \\ (P, N((i,j), \mathit{id}_?, \mathit{id}_!), E_?[\overline{y} \to E_!(\overline{x})]_{|fn(P)}); \\ (Q, \mathit{id}_!, E_{!|fn(Q)}) \end{array} \right\}}$$

Choice rules

$$C \cup \{P + Q, \mathit{id}, E\} \xrightarrow{\varepsilon} C \cup \{(P, \mathit{id}, E_{|fn(P)}\}$$
$$C \cup \{P + Q, \mathit{id}, E\} \xrightarrow{\varepsilon} C \cup \{(Q, \mathit{id}, E_{|fn(Q)}\}$$

Coherence

Theorem 1 Standard semantics and small step non-standard semantics are weakly bisimilar.

The main point is to prove that there are no conflicts between markers.

Marker allocation consistency

We denote by *father*(P) the father of P, when it exists, in the syntactic tree of $\mathcal{S}$.

1. the thread $(\mathcal{S}, \varepsilon, \emptyset)$ can only be created at the start of the system computation;
2. a thread $(\mathrm{P}, \mathit{id}, \_)$ such that *father*(P) is not a resource, can only be created by making a thread $(\mathit{father}(\mathrm{P}), \mathit{id}, \_)$ react;
3. a thread $(\mathrm{P}, N((\mathrm{i},\mathrm{j}), \mathit{id}_?, \mathit{id}_!), \_)$ can only be created by making a thread $(\mathrm{P}_\mathrm{j}, \mathit{id}_!, \_)$ react (when P_j denote the syntactic process begining with the syntactic component labeled with j).

This proves marker allocation consistency.

Simplifying markers

We can simplify the shape of the marker without any loss of consistency:

1. replacing each tree by its right comb:

$$\begin{cases} \phi_1(N((i,j), id_1, id_2)) & = \phi_1(id_2).(i,j) \\ \phi_1(\varepsilon) & = \varepsilon \end{cases}$$

2. replacing pairs by their second component:

$$\begin{cases} \phi_2(N((i,j), id_1, id_2)) = \phi_2(id_2).j \\ \phi_2(\varepsilon) = \varepsilon \end{cases}$$

Those simplifications can be seen as an abstraction, they do not loose semantics consistency, but they may abstract away information, in the case of nested resources, by merging information about distinct computation sequences.

Middle semantics

Small step semantics can be analyzed but:

- there are too many transition rules;
- it uses too many kinds of processes.

$\Longrightarrow$ We design a new semantics with only active rules.

(Structural rules are included inside active rules)

Definition

Structural rules:

$$C \cup \{(P \mid Q, \mathit{id}, E)\} \xrightarrow{\varepsilon} C \cup \{(P, \mathit{id}, E_{|fn(P)}); (Q, \mathit{id}, E_{|fn(Q)})\}$$

$$C \cup \{((\nu\ x)P, \mathit{id}, E)\} \xrightarrow{\varepsilon} C \cup \{(P, \mathit{id}, E[x \to (x, \mathit{id})]_{|fn(P)})\}$$

$$C \cup \{(\emptyset, \mathit{id}, E)\} \xrightarrow{\varepsilon} C$$

are a confluent and well-founded transition system,
we denote by $\Longrightarrow$ its limit:

$$a \Longrightarrow b \ \mathit{ssi} \begin{cases} a \to^* b \\ \forall c, b \not\to c. \end{cases}$$

and we define our new transition system by $\xrightarrow{\lambda}{}' = \xrightarrow{\lambda} \circ \Longrightarrow$.

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.

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

Transition system

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

$$C \cup \{(P+Q, id, E)\} \xrightarrow{\varepsilon} (C \cup \beta(P, id, E))$$
$$C \cup \{(P+Q, id, E)\} \xrightarrow{\varepsilon} (C \cup \beta(Q, id, E))$$

Communication rules

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

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

Coherence

Middle semantics and standard semantics are strongly bisimilar, but we still consider too much process: we can also factor choice operations.

For that purpose we restrict our study to the computation sequences in where communication are only made when there are no choice thread instance at top level, and factor choices with communication rules.

Definition

Choice rules are a well-founded transition system,

we denote by $\Longrightarrow$ its non-deterministic limit:

$$a \Longrightarrow b \text{ } \textit{ssi} \begin{cases} a \rightarrow^* b \\ \forall c, b \not\rightarrow c. \end{cases}$$

and we define our new transition system by $\xrightarrow{\lambda}{}' = \xrightarrow{\lambda} \circ \Longrightarrow$.

Extraction function

An extraction function calculates the set of all choices for the set of the thread instances spawned at the beginning of the system execution or after a communication.

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

Transition system

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

$$\frac{E_?(y) = E_!(x),\ \ \mathit{Cont}_P \in \beta(P, id_?, E_?[y_i \mapsto E_!(x_i)]),\ \mathit{Cont}_Q \in \beta(Q, id_!, E_!)}{C \cup \{(y?^i[\overline{y}]P, id_?, E_?), (x!^j[\overline{x}]Q, id_!, E_!)\} \stackrel{(i,j)}{\longrightarrow} (C \cup \mathit{Cont}_P \cup \mathit{Cont}_Q)}$$

$$\frac{E_*(y) = E_!(x),\ \ \mathit{Cont}_P \in \beta(P, N((i,j), id_*, id_!), E_*[y_i \mapsto E_!(x_i)]),\ \mathit{Cont}_Q \in \beta(Q, id_!, E_!)}{C \cup \left\{ \begin{array}{l} (*y?^i[\overline{y}]P, id_*, E_*), \\ (x!^j[\overline{x}]Q, id_!, E_!) \end{array} \right\} \stackrel{(i,j)}{\longrightarrow} \left(C \cup \{(*y?^i[\overline{y}]P, id_*, E_*)\} \cup \mathit{Cont}_P \cup \mathit{Cont}_Q\right)}$$

META-language: intuition

In the π-calculus :

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

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

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

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

- The generic transition rule:

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

  describes communication steps.

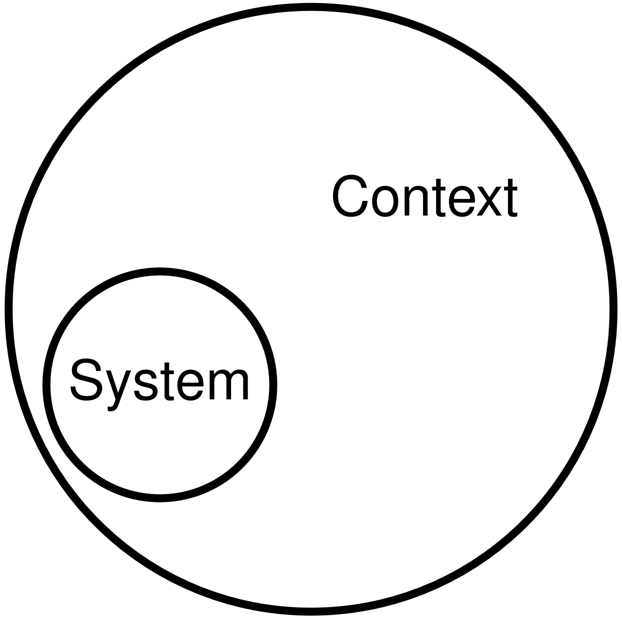
Some rules are more complex (e.g. ambient opening).

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 contains useful information:
   This allows the inference of:
   - more complex properties;
   - some simple properties the proof of which uses complex properties.

Context-free analysis

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*])

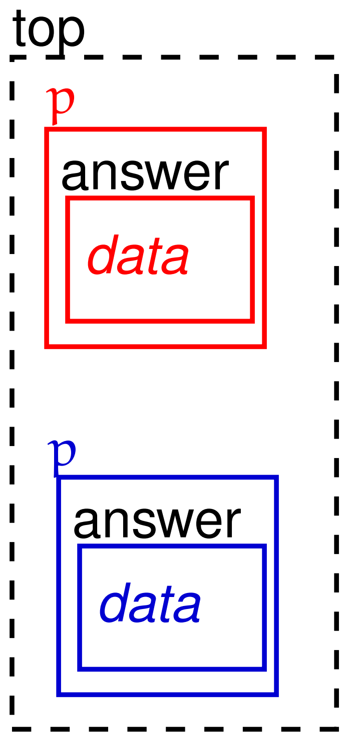
spoil$_k$:= ($*$unsafe?[c]unsafe?[x_1]...unsafe?[x_k]c![x_1,...,x_k])

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

Non-Standard Configuration

We flatly represent system configurations:

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



In migration

$$
\frac{\begin{cases}
\lambda = \left(n^{i}[\bullet], \mathit{id}_1, \mathit{loc}_1, E_1\right), \\
\mu = \left(m^{j}[\bullet], \mathit{id}_2, \mathit{loc}_2, E_2\right), \\
\psi = \left(\mathit{in}^{k} o.P, \mathit{id}_3, \mathit{loc}_3, E_3\right), \\
\mathit{loc}_1 = \mathit{loc}_2,\ \mathit{loc}_3 = (i, \mathit{id}_1),\ E_2(m) = E_3(o),\ \lambda \neq \mu.
\end{cases}}{C \cup \{\lambda; \mu; \psi\} \overset{\mathit{in}(i,j,k)}{\longrightarrow} (C \cup \{\mu\}) \cup (n^{i}[\bullet], \mathit{id}_1, (j, \mathit{id}_2), E_1) \cup \beta\left(P, \mathit{id}_3, \mathit{loc}_3, E_{3|fn(P)}\right).}
$$

out migration

$$\frac{\begin{cases}\lambda = \left(m^{i}[\bullet], \mathit{id}_1, \mathit{loc}_1, E_1\right), \\ \mu = \left(n^{j}[\bullet], \mathit{id}_2, \mathit{loc}_2, E_2\right), \\ \psi = \left(\mathit{out}^{k} o.P, \mathit{id}_3, \mathit{loc}_3, E_3\right), \\ \mathit{loc}_2 = (i, \mathit{id}_1),\ \mathit{loc}_3 = (j, \mathit{id}_2),\ E_1(m) = E_3(o)\end{cases}}{C \cup \{\lambda; \mu; \psi\} \xrightarrow{\mathit{out}(i,j,k)} (C \cup \{\lambda\}) \cup (n^{j}[\bullet], \mathit{id}_2, \mathit{loc}_1, E_2) \cup \beta\left(P, \mathit{id}_3, \mathit{loc}_3, E_{3|fn(P)}\right).}$$

Dissolution

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

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

Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Collecting semantics

$(\mathcal{C}, C_0, \rightarrow)$ is a transition system,
We restrict our study to its collecting semantics:
this is the set of the states that are reachable within a finite transition sequence.

$$\mathcal{S} = \{C \mid \exists i \in C_0,\ i \rightarrow^* C\}$$

It is also given by the least fixpoint of the following $\cup$-complete endomorphism $\mathbb{F}$:

$$\mathbb{F} = \begin{cases} \wp(\mathcal{C}) & \rightarrow \wp(\mathcal{C}) \\ X & \mapsto C_0 \cup \{C' \mid \exists C \in X,\ C \rightarrow C'\} \end{cases}$$

This fixpoint is usually not computable automatically.

Abstract domain

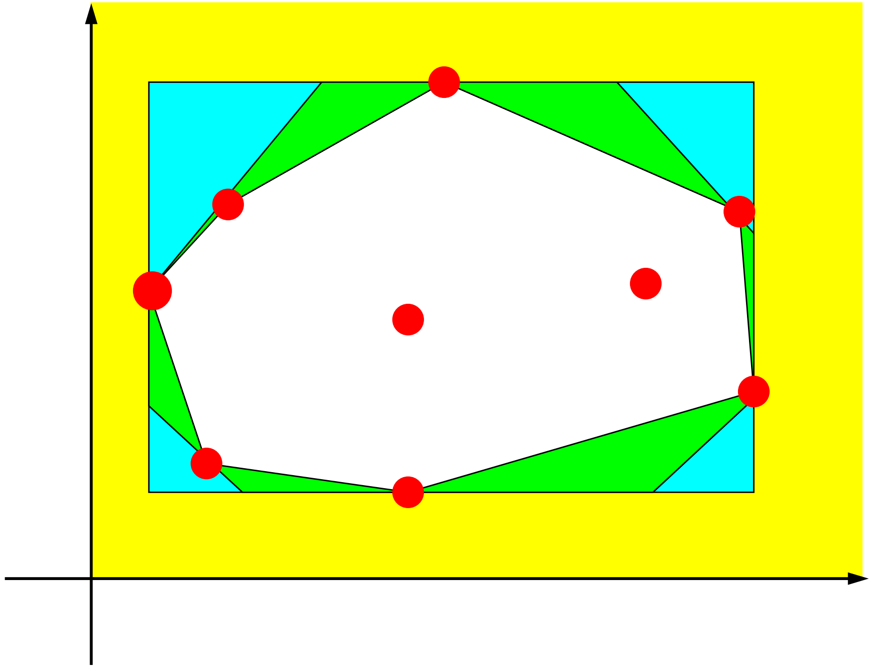
We introduce an abstract domain of properties:

- properties of interest;
- more complex properties used in calculating them.

This domain is often a lattice: $(\mathcal{D}^\sharp, \sqsubseteq, \sqcup, \bot, \sqcap, \top)$ and is related to the concrete domain $\wp(\mathcal{C})$ by a monotonic concretization function γ.

$\forall A \in \mathcal{D}^\sharp$, $\gamma(A)$ is the set of the elements which satisfy the property A.

Numerical domains



- sign approximation;
- interval approximation;
- octagonal approximation;
- polyhedra approximation;
- concrete domain.

Abstract transition system

Let $C_0^\sharp$ be an abstraction of the initial states and $\rightsquigarrow$ be an abstract transition relation, which satisfies $C_0 \subseteq \gamma(C_0^\sharp)$ and the following diagram:

$$\begin{tikzcd}
C^\sharp \arrow[r, "\overset{\lambda}{\rightsquigarrow}"] \arrow[d, "\gamma"'] & \overline{C}^\sharp \arrow[d, "\gamma"] \\
C \arrow[r, "\xrightarrow{\lambda}"'] & \overline{C}
\end{tikzcd}$$

Then, $\mathcal{S} \subseteq \bigcup_{n \in \mathbb{N}} \gamma({\mathbb{F}^\sharp}^n(C_0^\sharp))$,

where $\mathbb{F}^\sharp(C^\sharp) = C_0^\sharp \sqcup C^\sharp \sqcup \left(\bigsqcup_{\textit{finite}} \{\overline{C^\sharp} \mid C^\sharp \rightsquigarrow \overline{C^\sharp}\}\right)$.

Widening operator

We require a widening operator to ensure the convergence of the analysis:

$$\nabla \ : \ D^{\sharp} \times D^{\sharp} \to D^{\sharp}$$

such that:

- $\forall X_1^{\sharp},\ X_2^{\sharp} \in D^{\sharp},\ X_1^{\sharp} \sqcup X_2^{\sharp} \sqsubseteq X_1^{\sharp} \nabla X_2^{\sharp}$
- for all increasing sequence $(X_n^{\sharp}) \in \left(D^{\sharp}\right)^{\mathbb{N}}$, the sequence (X_n^{∇}) defined as
$$\begin{cases} X_0^{\nabla} = X_0^{\sharp} \\ X_{n+1}^{\nabla} = X_n^{\nabla} \ \nabla \ X_{n+1}^{\sharp} \end{cases}$$
is ultimately stationary.

Abstract iteration

The abstract iteration (C_n^∇) of $\mathbb{F}^\sharp$ defined as follows

$$\begin{cases} C_0^\nabla = C_0^\sharp \\ C_{n+1}^\nabla = \begin{cases} C_n^\nabla & \text{if } \mathbb{F}^\sharp(C_n^\nabla) \sqsubseteq C_n^\nabla \\ C_n^\nabla \nabla \mathbb{F}^\sharp(C_n^\nabla) & \text{otherwise} \end{cases} \end{cases}$$

is ultimately stationary and its limit C^∇ satisfies $\mathit{lfp}_\emptyset \mathbb{F} \subseteq \gamma(C^\nabla)$.

Example: Interval widening

We consider the complete $\mathcal{I}$ lattice of the natural number intervals.

$\mathcal{I}$ does not satisfy the increasing chain condition.

Given n a natural number, we use the following widening operator to ensure the convergence of the analyses based on the use of $\mathcal{I}$:

$$\begin{cases} [|a;b|] & \nabla & [|c;d|] & = & [|\mathit{min}\{a;c\};\infty|[& \text{if } d > \mathit{max}\{n;b\} \\ I & \nabla & J & = & I \sqcup J & \text{otherwise} \end{cases}$$

Composing two abstractions

Given two abstractions $(\mathcal{D}^\sharp, \gamma, C_0^\sharp, \rightsquigarrow, \nabla)$ and $(\mathcal{D}^\sharp, \gamma, C_0^\sharp, \rightsquigarrow, \nabla)$, and a reduction $\rho : \mathcal{D}^\sharp \times \mathcal{D}^\sharp \longrightarrow \mathcal{D}^\sharp \times \mathcal{D}^\sharp$ which satisfy:

$$\forall (A, A) \in \mathcal{D}^\sharp \times \mathcal{D}^\sharp,\ \gamma(A) \cap \gamma(A) \subseteq \gamma(a) \cap \gamma(a) \text{ where } (a, a) = \rho(A, A).$$

Then $(\mathcal{D}^\sharp, \gamma, C_0^\sharp, \rightsquigarrow, \nabla)$ where:

- $\mathcal{D}^\sharp = \mathcal{D}^\sharp \times \mathcal{D}^\sharp$;
- ∇ is pair-wisely defined;
- $\gamma(A, A) = \gamma(A) \cap \gamma(A)$;
- $C_0^\sharp = \rho(C_0^\sharp, C_0^\sharp)$;
- $(A, A) \rightsquigarrow \rho(C, C)$
  if $B \rightsquigarrow C$ and $B \rightsquigarrow C$ and $(B, B) = \rho(A, A)$

is also an abstraction.

Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Generic environment analysis

For each subset V of variables, we introduce a generic abstract domain $\mathcal{G}_V$ to describe the markers and the environments which may be associated to a syntactic component the free name of which is V:

$$\wp(\mathit{Id} \times (V \to (\mathit{Name} \times \mathit{Id}))) \xleftarrow{\gamma_V} \mathcal{G}_V.$$

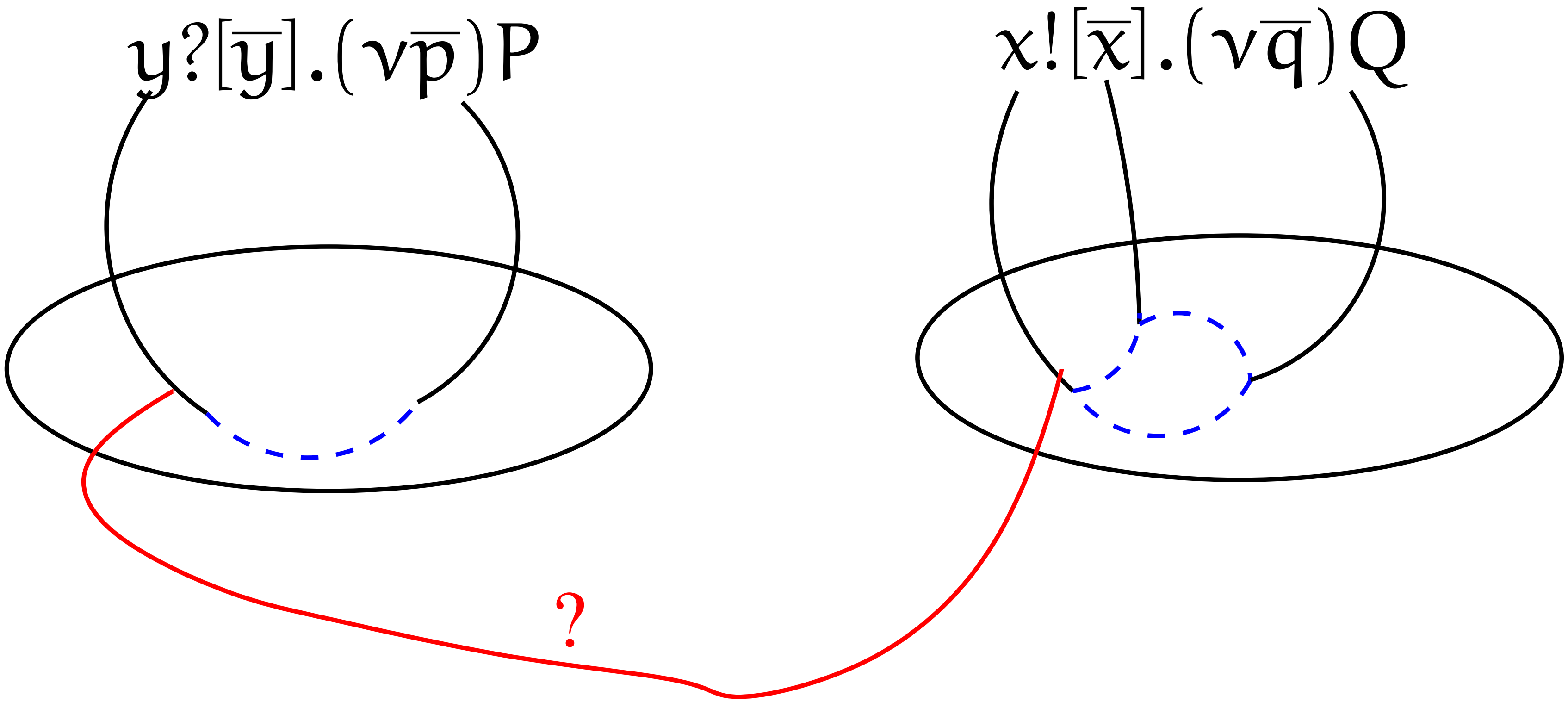
The abstract domain $C^\sharp$ is then the set:

$$C^\sharp = \prod_{p \in \mathcal{P}} \mathcal{G}_{\mathrm{fn}(p)}$$

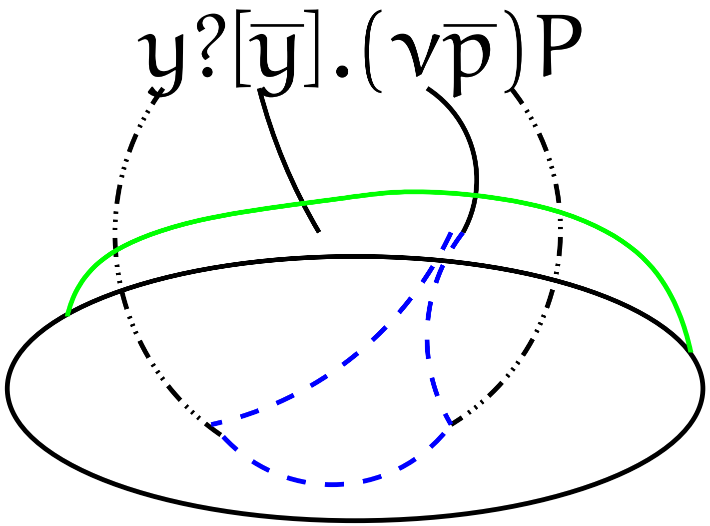
related to $\wp(C)$ by the concretization γ:

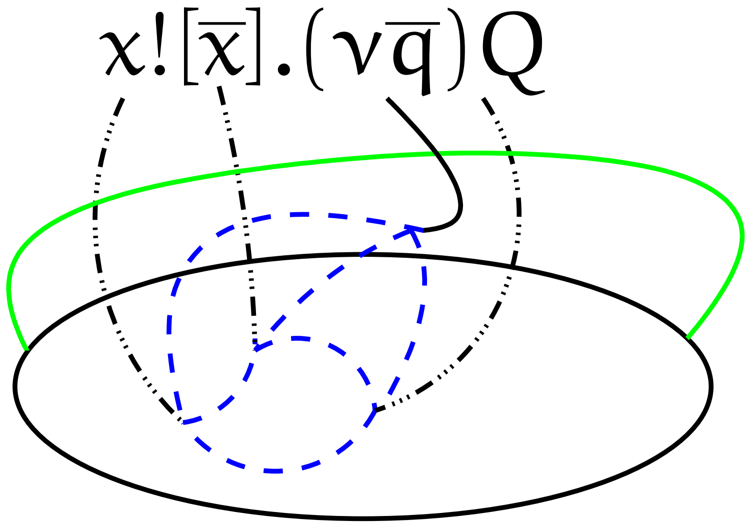
$$\gamma(f) = \{C \mid (p, \mathit{id}, E) \in C \implies (\mathit{id}, E) \in \gamma_{\mathrm{fn}(p)}(f_p)\}.$$

Abstract communication

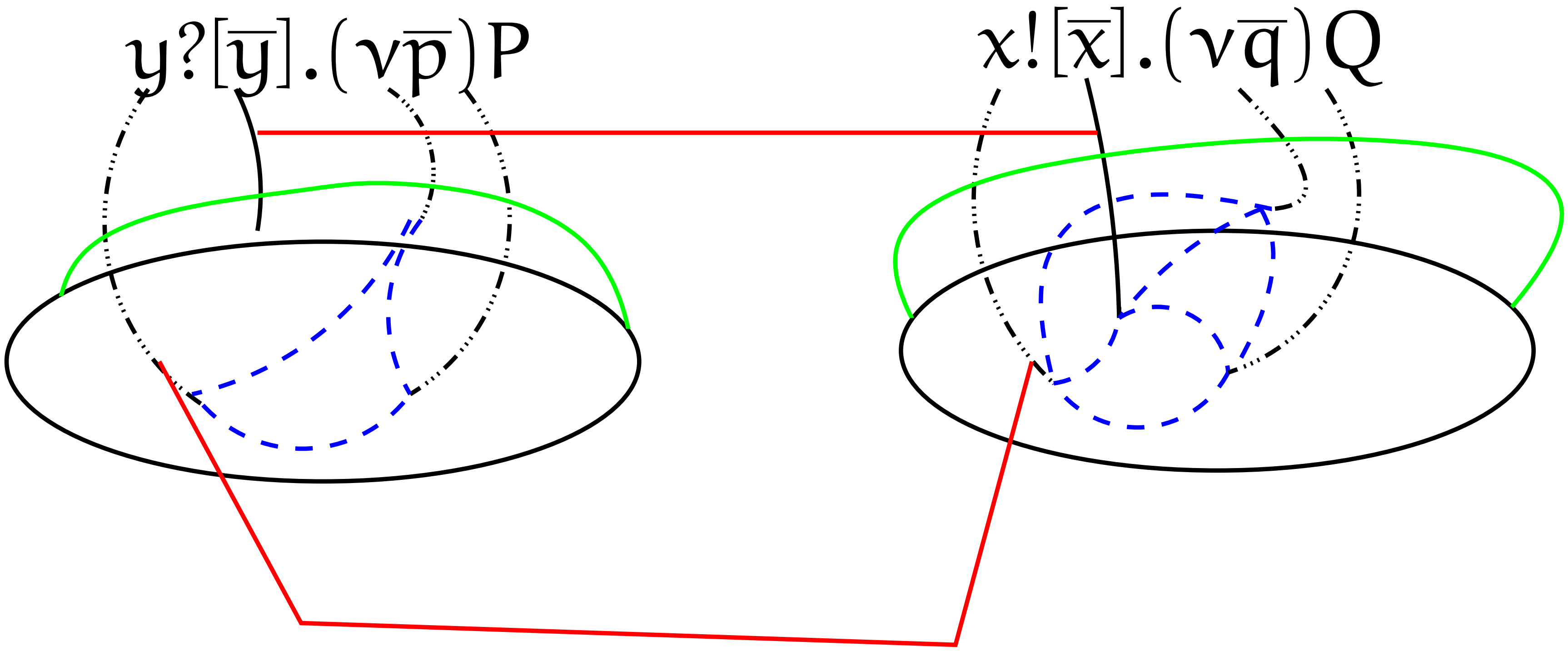


Extending environments

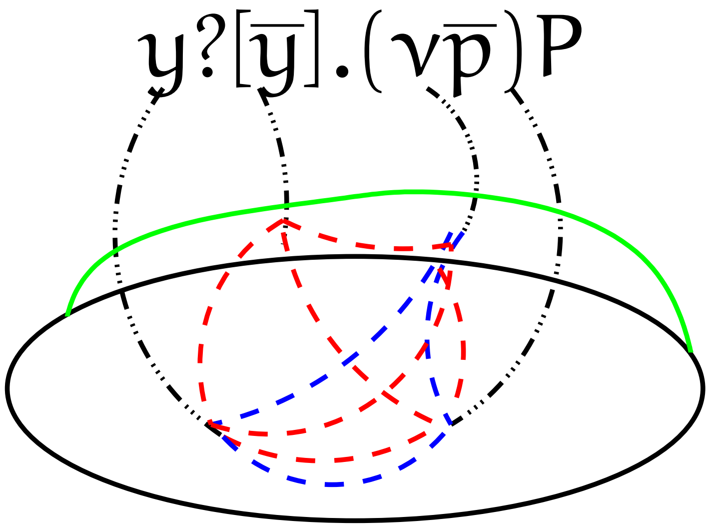


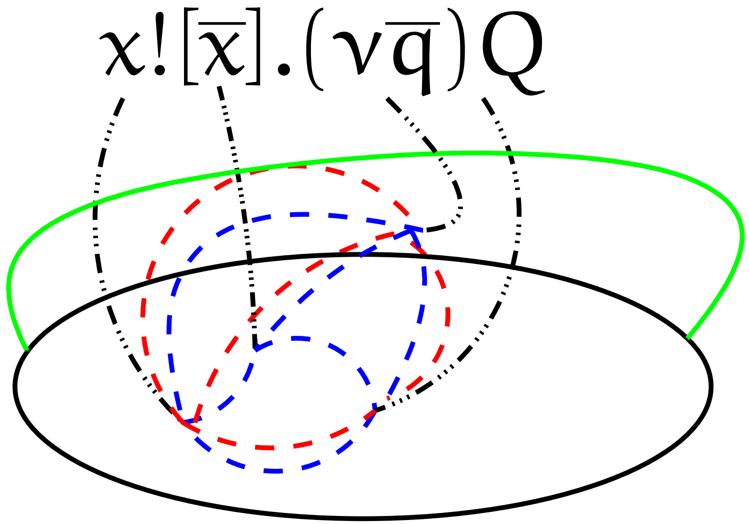


Synchronizing environments



Propagating information



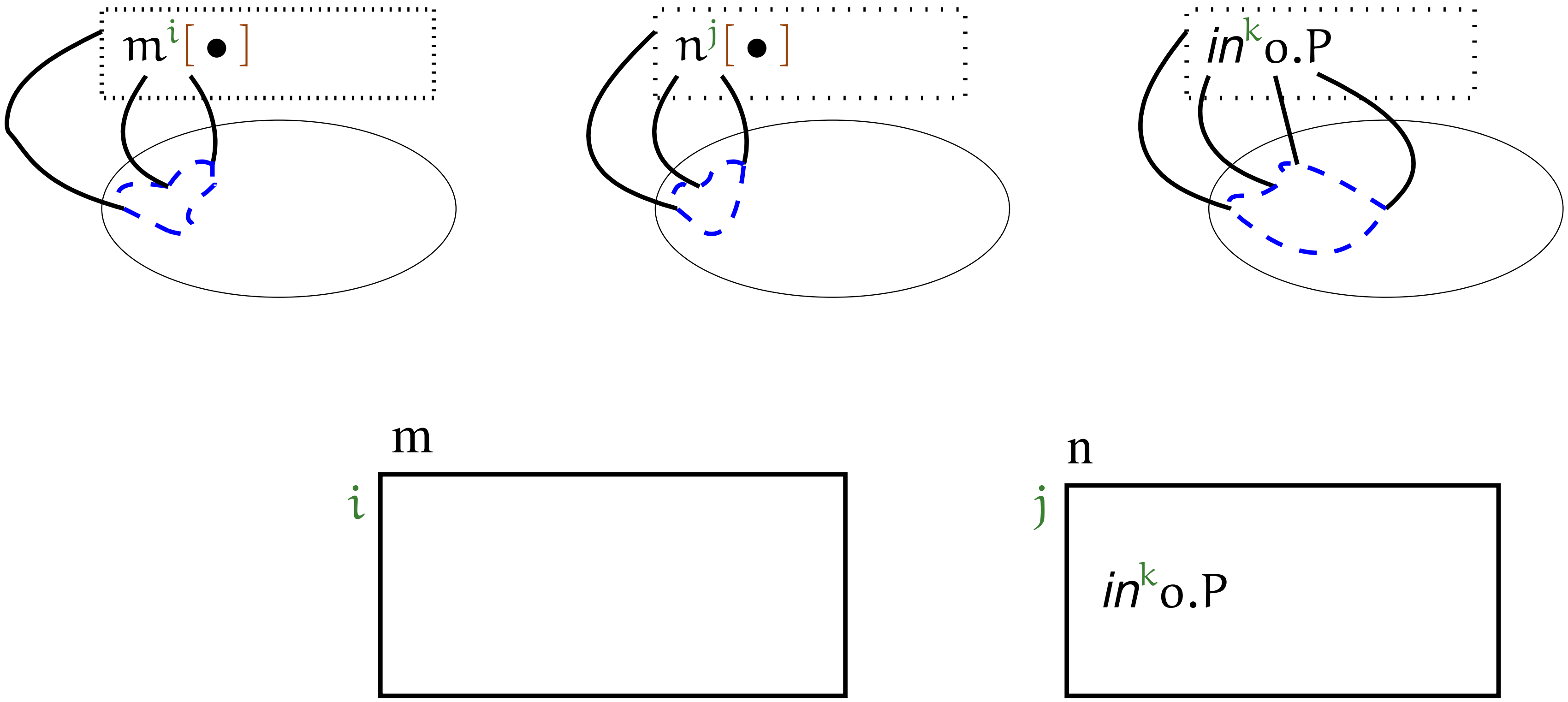


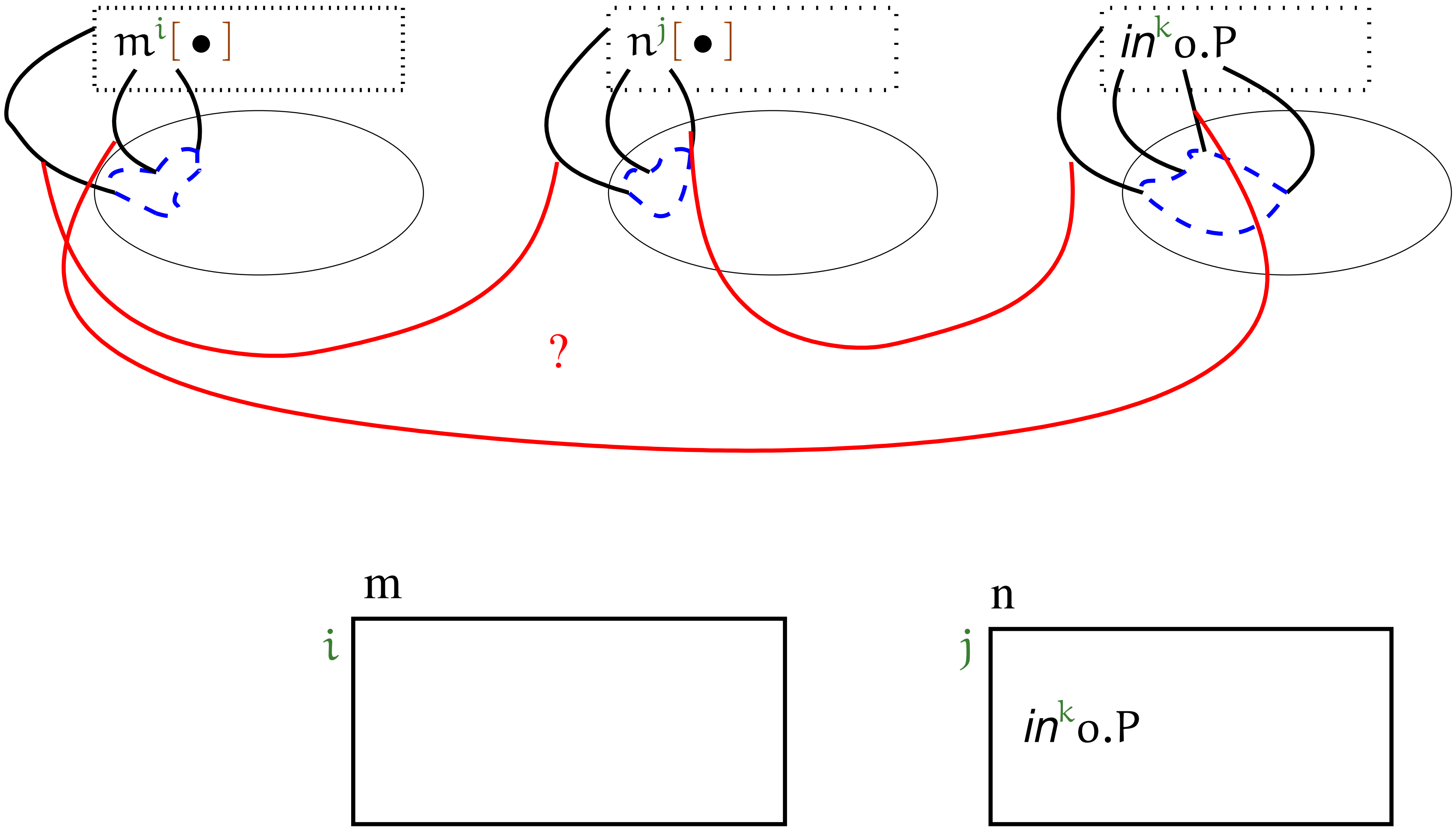
Generic primitives

We only require abstract primitives to:

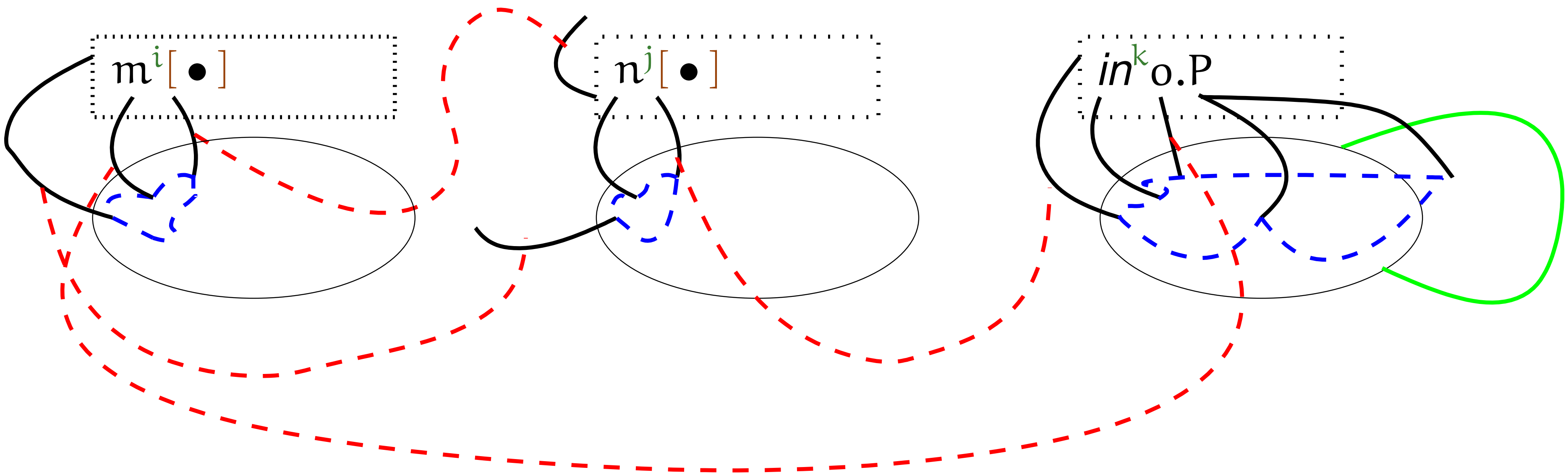
1. extend an environment domain,
2. gather the description of the linkage of two syntactic agents,
3. synchronize variables,
4. separate two descriptions,
5. restrict an environment domain.

About mobile ambients

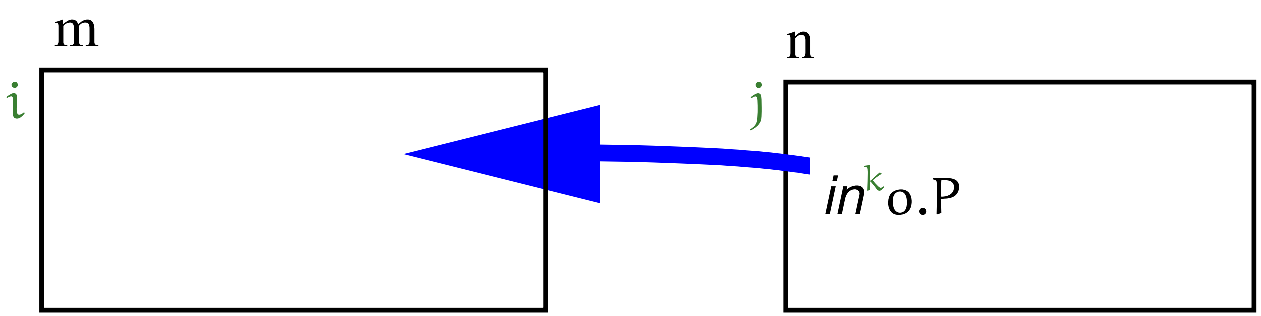




m^i[●]
n^j[●]
in^k o.P
?
m
i
n
j
in^k o.P



m^i[•]
n^j[•]
in^k o.P



m
i
n
j
in^k o.P

Control flow analyses

We abstract for each variable x and each name restriction $\nu\, y$ the set of marker pairs $(\mathrm{id}_x, \mathrm{id}_y)$ such that the channel opened by the instance of the restriction $\nu\, y$ tagged with the marker id_y may be communicated to the variable x of a thread tagged by the marker id_x.

Let $Id^\sharp$ be an abstract domain of properties about marker pairs.

$$\gamma_{Id^2} : Id^\sharp \to \wp(Id^2)$$

$$\mathcal{G}_V = V \times Name \to Id^\sharp$$

$\gamma_V(a^\sharp)$ is the set of marker/environment pairs (id_x, E) such that:

$$\forall x \in V, E(x) = (y, id_y) \implies (id_x, id_y) \in \gamma_{Id^2}(a^\sharp(x, y)).$$

Regular approximation

We approximate the shape of the markers which may be associated to channel names linked to variables, and syntactic components, without relations among them.
We use the following abstract domain:

$$\wp(\Sigma) \times \wp(\Sigma) \times \wp(\Sigma \times \Sigma) \times \{\textit{true;false}\}.$$

$\gamma(I, F, T, b)$ is defined by $\gamma_1(I) \cap \gamma_2(F) \cap \gamma_3(T) \cap \gamma_4(b)$ where:

- $\gamma_1(I) = \{u \in \Sigma^* \mid |u| > 0 \Rightarrow u_1 \in I\}$,
- $\gamma_2(F) = \{u \in \Sigma^* \mid |u| > 0 \Rightarrow u_{|u|} \in F\}$,
- $\gamma_3(T) = \{u \in \Sigma^* \mid \forall a, b \in \Sigma^*, \lambda, \mu \in \Sigma,\ u = a.\lambda.\mu.b \Rightarrow (\lambda, \mu) \in T\}$,
- $\gamma_4(b) = \begin{cases} \Sigma^+ & \text{if } b = 0 \\ \Sigma^* & \text{otherwise.} \end{cases}$

Domain complexity is $O(n.|\Sigma|)$ and maximum iteration number is $O(n^4.|\Sigma|)$.

Comparison between channel and agent markers

We capture the difference between the occurrence number of letters in such two markers.

$$Id^2 = (\Sigma \to (\mathbb{Z} \cup \{\top\})) \cup \{\bot\}$$

γ_{Id^2} is defined as follows:

$$\gamma_{Id^2}(\bot) = \emptyset$$
$$\gamma_{Id^2}(f) = \{(u, v) \in (\Sigma^*)^2 \mid \forall \lambda,\ f(\lambda) \in \mathbb{Z} \implies |u|_\lambda - |v|_\lambda = f(n)\}.$$

Domain complexity is $O(|\Sigma|)$ and maximum iteration number is $O(n^3.|\Sigma|)$.

Several trade-offs

1. 0-cfa (0-CFA): $Id^\sharp = \{\bot; \top\}$,

   Cf [Nielson *et al.*:CONCUR'98], [Hennessy and Riely:HLCL'98].

2. Confinement (CONF): $Id^\sharp = \{\bot, =, \top\}$,

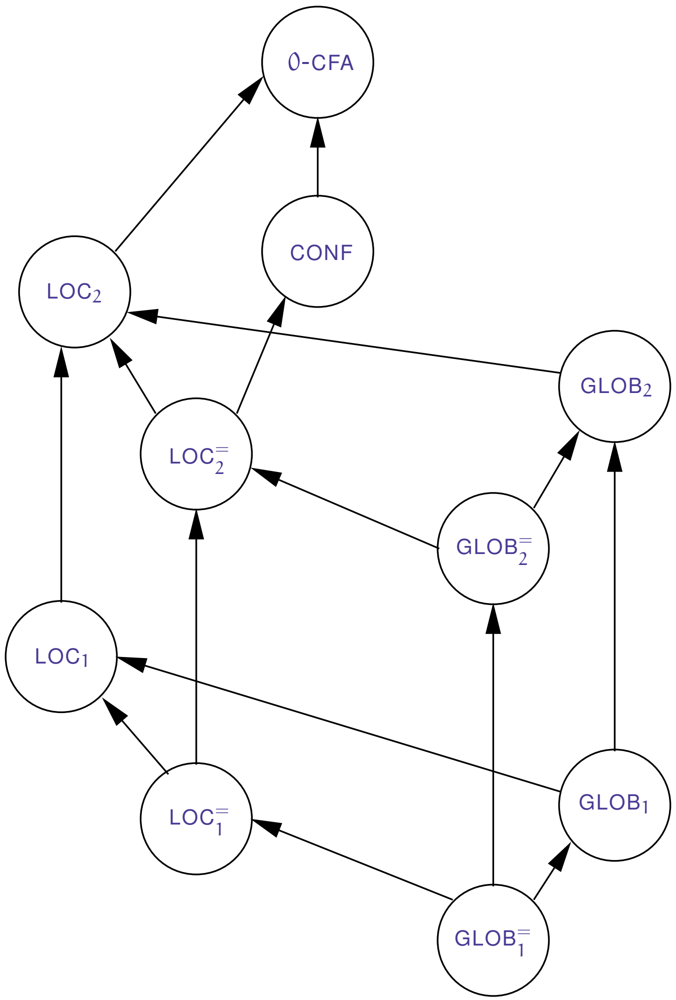
   Cf [Cardelli *et al.*:CONCUR'00].

3. Algebraic comparisons: we use the product between regular approximation and relational approximation.

   We can tune the complexity:

   - by capturing all numerical relations (GLOB_i), or only one relation per literal (LOC_i).
   - by choosing the set of literals among *Label* ($i = 2$)or *Label*^{2} ($i = 1$).

Abstract semantics hierarchy



where

$$A \longrightarrow B$$

means that there exists $\alpha : A \longrightarrow B$, such that for any system $\mathcal{S}$,

$$\alpha(\llbracket \mathcal{S} \rrbracket^\sharp_A) \sqsubseteq_B \llbracket \mathcal{S} \rrbracket^\sharp_B.$$

Example: 0-CFA

Pi.s.a III : a Pi-calculus Static Analyzer - Mozilla

File Edit View Go Bookmarks Tools Window Help

(# port)(# gen)
(*port?^{1}[info,add](add!^{2}[info])
| *gen?^{3}[](# data)(# email)(port!^{4}[data,email] | gen!^{5}[])
| gen!^{6}[])

main menu - control flow analysis

Pi-s.a. Version 3.24, last Modified Fri November 19 2004
Pi-s.a. is an experimental prototype for academic use only.

Analysis result

We detect that threads at program point 2 as the following shape:

$$\left(2, (3,6)(3,5)^{n}(1,4), \begin{cases} \textit{add} & \mapsto \left(\textit{email}, (3,6)(3,5)^{n}\right) \\ \textit{info} & \mapsto \left(\textit{data}, (3,6)(3,5)^{n}\right) \end{cases}\right)$$

Example: non-uniform result

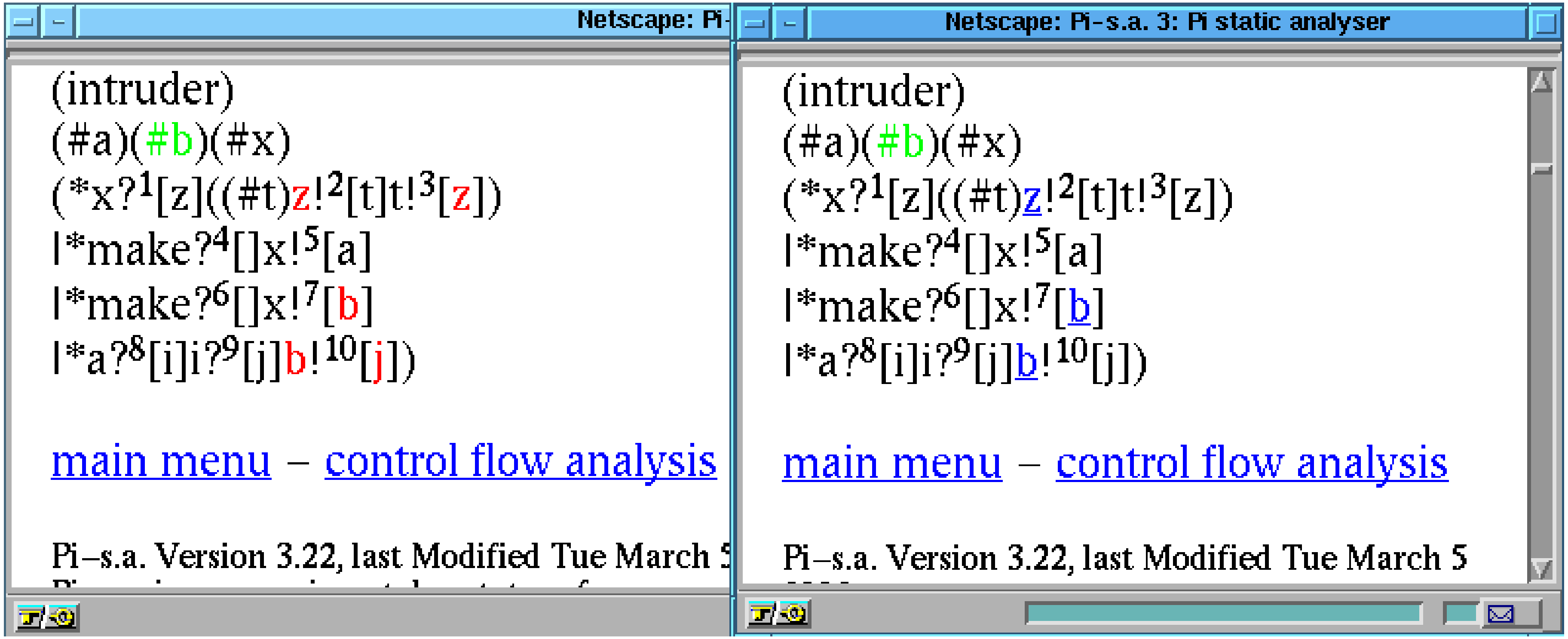
Pi.s.a III : a Pi-calculus Static Analyzer - Mozilla

```
( *port?^1[info,add](add!^2[info])
| *gen?^3[](# data)(# email)(port!^4[data,email] | gen!^5[])
| gen!^6[] )
```

Start --> (3,6)A
A --> (3,5)A + (1,4)B
B --> END

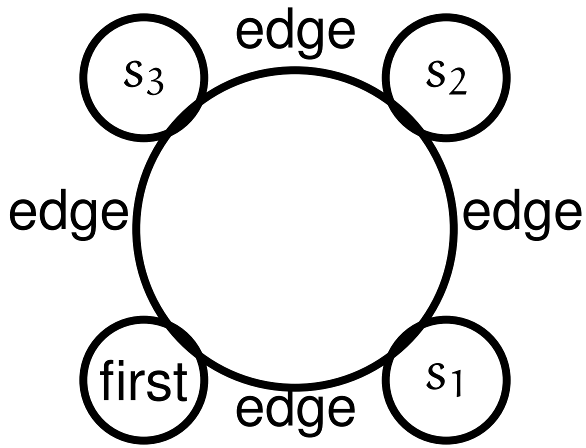
Start --> (3,6)A
A --> END + (3,5)A

(3,6) = (3,6)
(3,5) = (3,5)


Netscape: Pi-
(intruder)
(#a)(#b)(#x)
(*x?1[z]((#t)z!2[t]t!3[z])
|*make?4[]x!5[a]
|*make?6[]x!7[b]
|*a?8[i]i?9[j]b!10[j])
main menu – control flow analysis
Pi–s.a. Version 3.22, last Modified Tue March 5
Netscape: Pi-s.a. 3: Pi static analyser
(intruder)
(#a)(#b)(#x)
(*x?1[z]((#t)z!2[t]t!3[z])
|*make?4[]x!5[a]
|*make?6[]x!7[b]
|*a?8[i]i?9[j]b!10[j])
main menu – control flow analysis
Pi–s.a. Version 3.22, last Modified Tue March 5

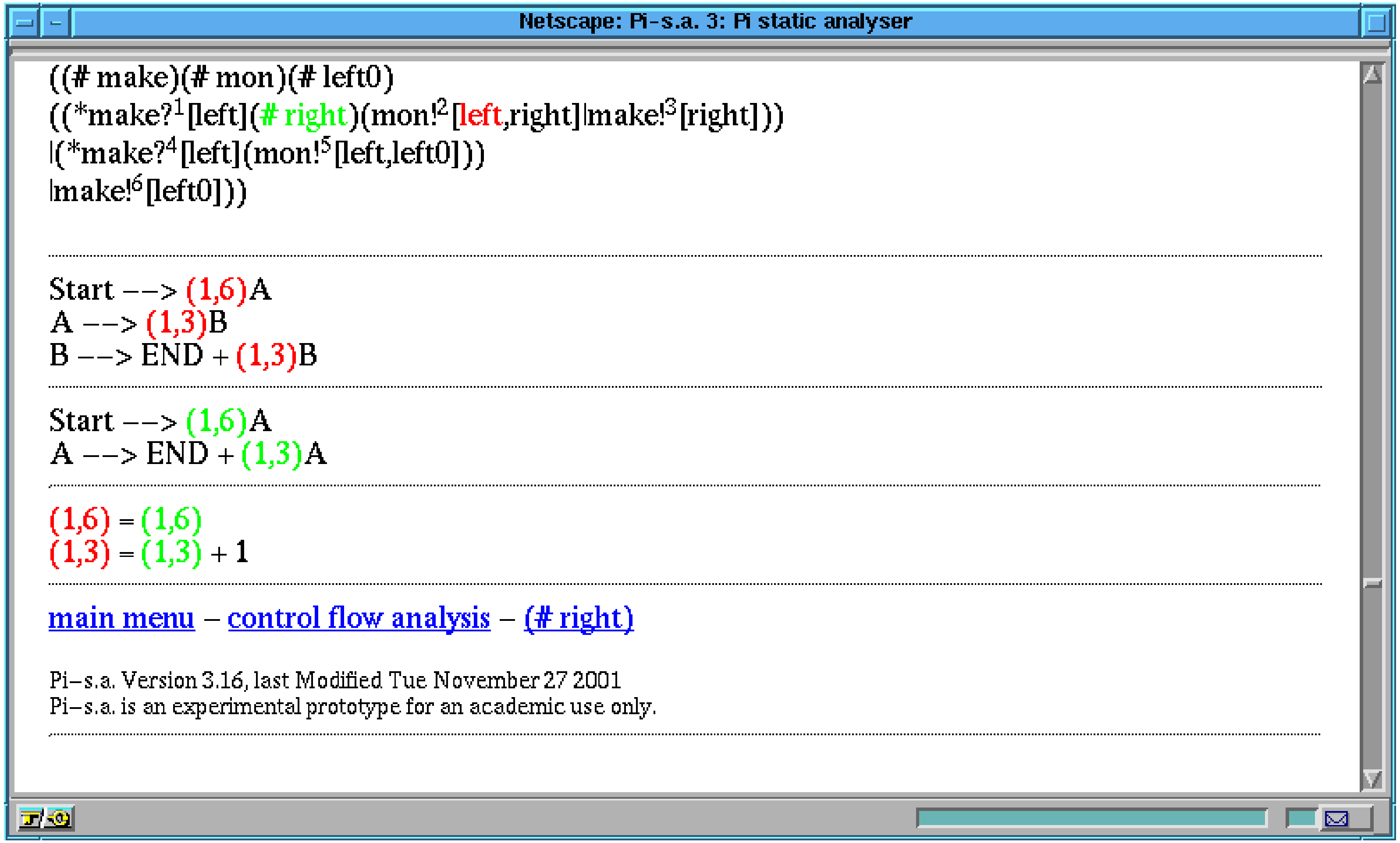
Example: the ring of processes

$$
\begin{array}{l}
(\nu\ \mathrm{make})(\nu\ \mathrm{edge})(\nu\ \mathrm{first}) \\
\quad (*\mathrm{make}?^{1}[\mathit{last}](\nu\, \mathit{next}) \\
\qquad\qquad\qquad (\mathrm{edge}!^{2}[\mathit{last}, \mathit{next}] \\
\qquad\qquad\qquad \mid\ \mathrm{make}!^{3}[\mathit{next}]) \\
\quad \mid\ *\mathrm{make}?^{4}[\mathit{last}](\mathrm{edge}!^{5}[\mathit{last}, \mathrm{first}]) \\
\quad \mid\ \mathrm{make}!^{6}[\mathrm{first}])
\end{array}
$$



$\underline{\sharp}(1,3) + 1 =$
$\underline{\sharp}(1,3)$

Example: Algebraic properties



Example

We detect that:

$$\begin{cases} (p^{12}[\bullet], (11,20)^m.(11,21), \_, [p \mapsto (p, (11,20)^m.(11,21))]) \\ (\text{answer}^{8}[\bullet], (3,19).(11,20)^n.(11,21), (12, (11,20)^n.(11,21), \_) \\ (\langle \textit{rep} \rangle^{9}, \_, (8, (3,19).(11,20)^p.(11,21), [\textit{rep} \mapsto (\textit{data}, (11,20)^p.(11,21))])) \end{cases}$$

We deduce that each packet exiting the server has the following structure:

$(p.(11,20)^n.(11,21))$

$(11,20)^n.(11,21)$

answer

$(3,19).(11,20)^n.(11,21)$

$(\textit{data}, (11,20)^n.(11,21))$

Limitations

Two main drawbacks:

1. we only prove equalities between Parrikh's vectors, some more work is needed in order to prove equalities of words;
2. we only capture properties involving comparison between channel name and agent markers:

$$
\begin{array}{l}
(\nu\ \text{make})(\nu\ \text{edge})(\nu\ \text{first})(\nu\ \text{first}) \\
\quad (*\text{make}?^{1}[\textit{last}](\nu\,\textit{next}) \\
\qquad\qquad\qquad (\text{edge}!^{2}[\textit{last},\textit{next}] \\
\qquad\qquad\qquad \mid\ \text{make}!^{3}[\textit{next}]) \\
\quad \mid\ *\text{make}?^{6}[\textit{last}](\text{edge}!^{7}[\textit{last},\text{first}]) \\
\quad \mid\ \text{make}!^{8}[\text{first}]) \\
\quad \mid\ \text{edge}?[x,y][x =^{9} y][x \neq^{10} \text{first}]\text{Ok}!^{11}[]
\end{array}
$$

we cannot infer that 11 is unreachable.

Dependency analysis between names

We describe equality and inequality relations between the names linked to variables.

$$\mathcal{G}_V = \left\{ (A, R) \;\middle|\; \begin{array}{l} A \textit{ is a partition of } V \\ R \textit{ is a symetric anti-reflexive relation on } A \end{array} \right\}.$$

$\mathcal{G}_V$ is related to $\wp(\mathit{Id} \times (V \to (\mathit{Name} \times \mathit{Id})))$ by the following concretization function:

$$\gamma_V((A, R)) = \left\{ (\mathit{id}, E) \;\middle|\; \begin{array}{l} \forall \mathcal{X} \in A,\ \{x, y\} \subseteq \mathcal{X} \implies E(x) = E(y) \\ (\mathcal{X}, \mathcal{Y}) \in R \implies \forall x \in \mathcal{X}, y \in \mathcal{Y},\ E(x) \neq E(y) \end{array} \right\}$$

$\Longrightarrow$ implicit closure of relations and information propagation.

Dependency analysis between markers

We describe equality and inequality relations between the markers of threads and the names linked to variables.

$$\mathcal{G}_V = \left\{ (A, R) \;\middle|\; \begin{array}{l} A \textit{ is a partition of } V \uplus \{\mathit{id}_p\} \\ R \textit{ is a symetric anti-reflexive relation on } A \end{array} \right\}.$$

$\mathcal{G}_V$ is related to $\wp(\mathit{Id} \times (V \to (\mathit{Name} \times \mathit{Id})))$ by the following concretization function:

$$\gamma_V((A, R)) = \left\{ (\mathit{id}, E) \;\middle|\; \begin{array}{l} \forall \mathcal{X} \in A,\ x \in V,\ \{\mathit{id}_p, x\} \subseteq \mathcal{X} \implies \mathit{id} = \mathit{snd}(E(x)) \\ \forall \mathcal{X} \in A,\ x, y \in V,\ \{x, y\} \subseteq \mathcal{X} \implies \mathit{snd}(E(x)) = \mathit{snd}(E(y)) \\ \forall (\mathcal{X}, \mathcal{Y}) \in R,\ y \in V, \\ \quad \mathit{id}_p \in \mathcal{X} \textit{ and } y \in \mathcal{Y} \implies \mathit{id} \neq \mathit{snd}(E(y)) \\ \forall (\mathcal{X}, \mathcal{Y}) \in R,\ x, y \in V, \\ \quad x \in \mathcal{X} \textit{ and } y \in \mathcal{Y} \implies \mathit{snd}(E(x)) \neq \mathit{snd}(E(y)) \end{array} \right\}$$

$\implies$ implicit closure of relations and information propagation.

Global numerical analysis

We abstract relations between all the name markers and all the names linked to variables, and the thread markers:
For each $V \subseteq \mathit{Name}$, we introduce the set

$$\mathcal{X}_V = \{p^{\lambda} \mid \lambda \in \Sigma\} \cup \{c^{(\lambda,\nu)} \mid \lambda \in \Sigma \cup \mathit{Name},\ \nu \in V\}$$

The domain $\mathcal{G}_V$ is then the set of the affine relations system among $\mathcal{X}_V$ related to the concrete domain by the following concretization:

$$\gamma_V(\mathcal{K}) = \left\{ (\mathit{id}, E) \;\middle|\; \begin{pmatrix} p^{\lambda} \longrightarrow |\mathit{id}|_{\lambda} \\ x^{(y,\nu)} \longrightarrow (y = \mathit{first}(E(\nu))) \\ x^{(\lambda,\nu)} \longrightarrow |\mathit{snd}(E(\nu))|_{\lambda} \end{pmatrix} \textit{ satisfies } \mathcal{K} \right\}.$$

Pair-wise numerical analysis

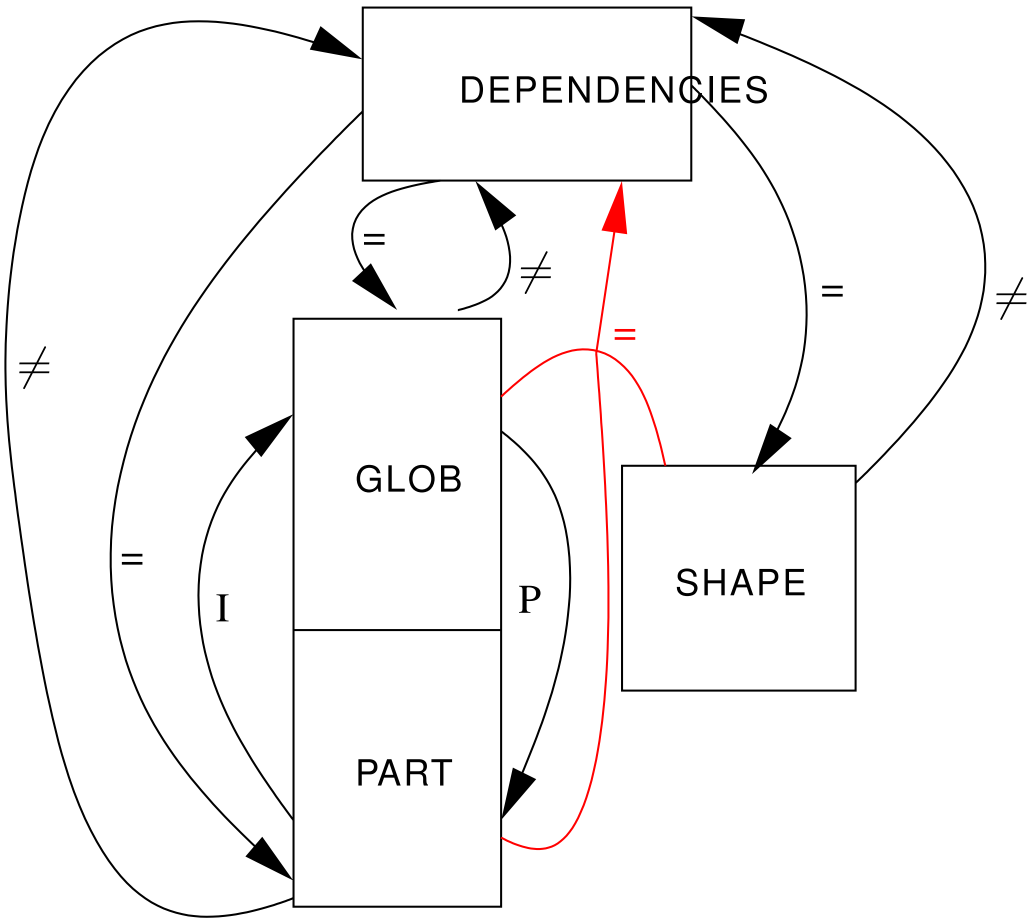
We compare pair-wisely markers, having partitioned in accordance with the name creations having created the names.
Let Φ be a linear form defined on $\mathbb{R}^{\Sigma}$, for each $V \subseteq \textit{Name}$, the domain $\mathcal{G}_V$ is a pair of function (f, g):

$$\begin{aligned} f &: V \cup \textit{Name} \to \{ \textit{Affine subspace of } \mathbb{R}^2\}, \\ g &: (V \cup \textit{Name})^2 \to \{ \textit{Affine subspace of } \mathbb{R}^2\}, \end{aligned}$$

the concretization $\gamma_V(f, g)$ is given by:

$$\left\{ (\textit{id}, E) \;\middle|\; \begin{array}{l} E(x) = (y, \textit{id}_y) \implies (\Phi((|\textit{id}|_\lambda)_{\lambda \in \Sigma}), \Phi((|\textit{id}_y|_\lambda)_{\lambda \in \Sigma})) \in f(x, y) \\ \begin{cases} E(x) = (y, \textit{id}_y) \\ E(x') = (y', \textit{id}_y') \end{cases} \implies (\Phi((|\textit{id}_y|_\lambda)_{\lambda \in \Sigma}), \Phi((|\textit{id}_y'|)|_\lambda)_{\lambda \in \Sigma}) \in g((x, y), (x', y')) \end{array} \right\}$$

Reduction



Example

$(\nu\ \text{make})(\nu\ \text{edge})(\nu\ \text{first})$
$\quad(*\text{make}?^1[\textit{last}](\nu\,\textit{next})\ (\text{edge}!^2[\textit{last},\textit{next}] \mid \text{make}!^3[\textit{next}])$
$\quad\mid\ *\text{make}?^6[\textit{last}](\text{edge}!^7[\textit{last},\text{first}])$
$\quad\mid\ \text{make}!^8[\text{first}])$
$\quad\mid\ \text{edge}?[x,y][x=^9 y][x \neq^{10} \text{first}]\text{Ok}!^{11}[]$

we first prove in global abstraction that:

$$f(2)\ \textit{satisfies}\ \begin{cases} c^{(1,3),\textit{next}} = c^{(1,3),\textit{last}} + c^{\textit{next},\textit{last}} \\ c^{\text{first},\textit{last}} + c^{\textit{next},\textit{last}} = 1 \end{cases}$$

$$f(7)\ \textit{satisfies}\ \begin{cases} c^{\textit{next},\textit{last}} + c^{\text{first},\textit{last}} = 1 \\ c^{\text{first},\text{first}} = 1 \end{cases}$$

Example

We then prove in pair-wise analysis that in process 9, x and y are respectively linked to names created by some instance of the restrictions :

1. (ν first) and (ν first),
2. (ν first) and (ν *next*),
3. (ν *next*) and (ν *next*) but distinct instances,
4. (ν *next*) and (ν first).

so, the matching pattern $[x = y]$ is satisfiable only in the first case !!!

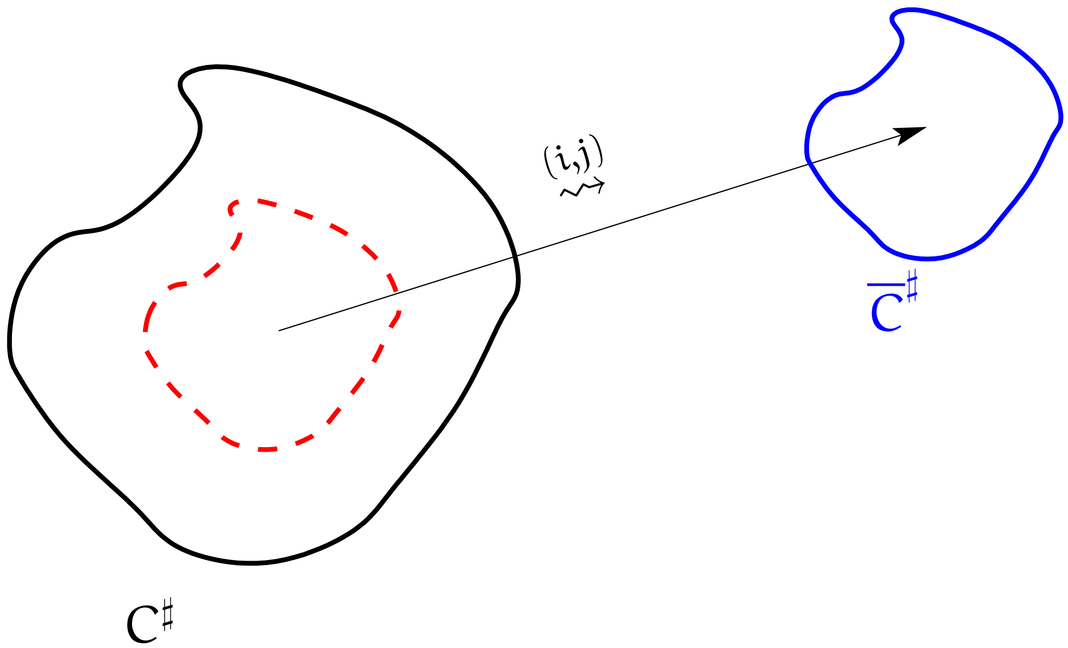
Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Intuition

$$
\left\{
\begin{array}{l}
\left(1, \varepsilon, \left\{ \mathrm{port} \mapsto (\mathrm{port}, \varepsilon) \right.\right) \\
\left(3, \varepsilon, \left\{ \begin{array}{ll} \mathrm{gen} & \mapsto (\mathrm{gen}, \varepsilon) \\ \mathrm{port} & \mapsto (\mathrm{port}, \varepsilon) \end{array} \right.\right) \\
\left(2, id_1', \left\{ \begin{array}{ll} add & \mapsto (email, id_1) \\ info & \mapsto (data, id_1) \end{array} \right.\right) \\
\left(2, id_2', \left\{ \begin{array}{ll} add & \mapsto (email, id_2) \\ info & \mapsto (data, id_2) \end{array} \right.\right) \\
\left(5, id_2, \left\{ \mathrm{gen} \mapsto (\mathrm{gen}, \varepsilon) \right.\right)
\end{array}
\right\}
$$

Abstract transition



Abstract domains

We design a domain for representing numerical constrains between

- the number of occurrences of processes $\sharp(i)$;
- the number of performed transitions $\underline{\sharp}(i,j)$.

We use the product of

- a non-relational domain:
  $\Longrightarrow$ the interval lattice;
- a relational domain:
  $\Longrightarrow$ the lattice of affine relationships.

Interval narrowing

An exact reduction is exponential.
We use:

- Gaus reduction: $\begin{cases} x+y+z=1 \\ x+y+t=2 \end{cases} \Longrightarrow \begin{cases} x+y+z=1 \\ t-z=1 \end{cases}$
- Interval propagation: $\begin{cases} x+y+z=3 \\ x \in [|0;\infty|[\\ y \in [|0;\infty|[\\ z \in [|0;\infty|[\end{cases} \Longrightarrow \begin{cases} x+y+z=3 \\ x \in [|0;3|] \\ y \in [|0;\infty|[\\ z \in [|0;\infty|[\end{cases}$
- Redundancy introduction: $\begin{cases} x+y-z=3 \\ x \in [|1;2|[\end{cases} \Longrightarrow \begin{cases} x+y-z=3 \\ y-z \in [|1;2|] \\ x \in [|1;2|] \end{cases}$

to get a cubic approximated reduction.

Example: non-exhaustion of resources

Netscape: Pi-s.a. 3: Pi static analyser

```
((# make)(# server)(# port)
((*make?^{1:1}[](# address)(# request)
     (
      (*address?^{2:[|0;+oo|[}[]server!^{3:[|0;+oo|[}[address,request])
     |
      address!^{4:[|0;+oo|[}[]
     |
      make!^{5:[|0;1|]}[]))
 |
  (*server?^{6:1}[email,data]
     (port?^{7:[|0;+oo|[}[](# deal)(
               deal!^{8:[|0;3|]}[data]
              |
               deal?^{9:[|0;3|]}[rep](email!^{10:[|0;+oo|[}[rep] | port!^{11:[|0;3|]}[]))
     +
      email!^{12:[|0;+oo|[}[]))
| port!^{13:[|0;1|]}[] | port!^{14:[|0;1|]}[] | port!^{15:[|0;1|]}[]
| make!^{16:[|0;1|]}[]))
```

main menu

Example: exhaustion of resources

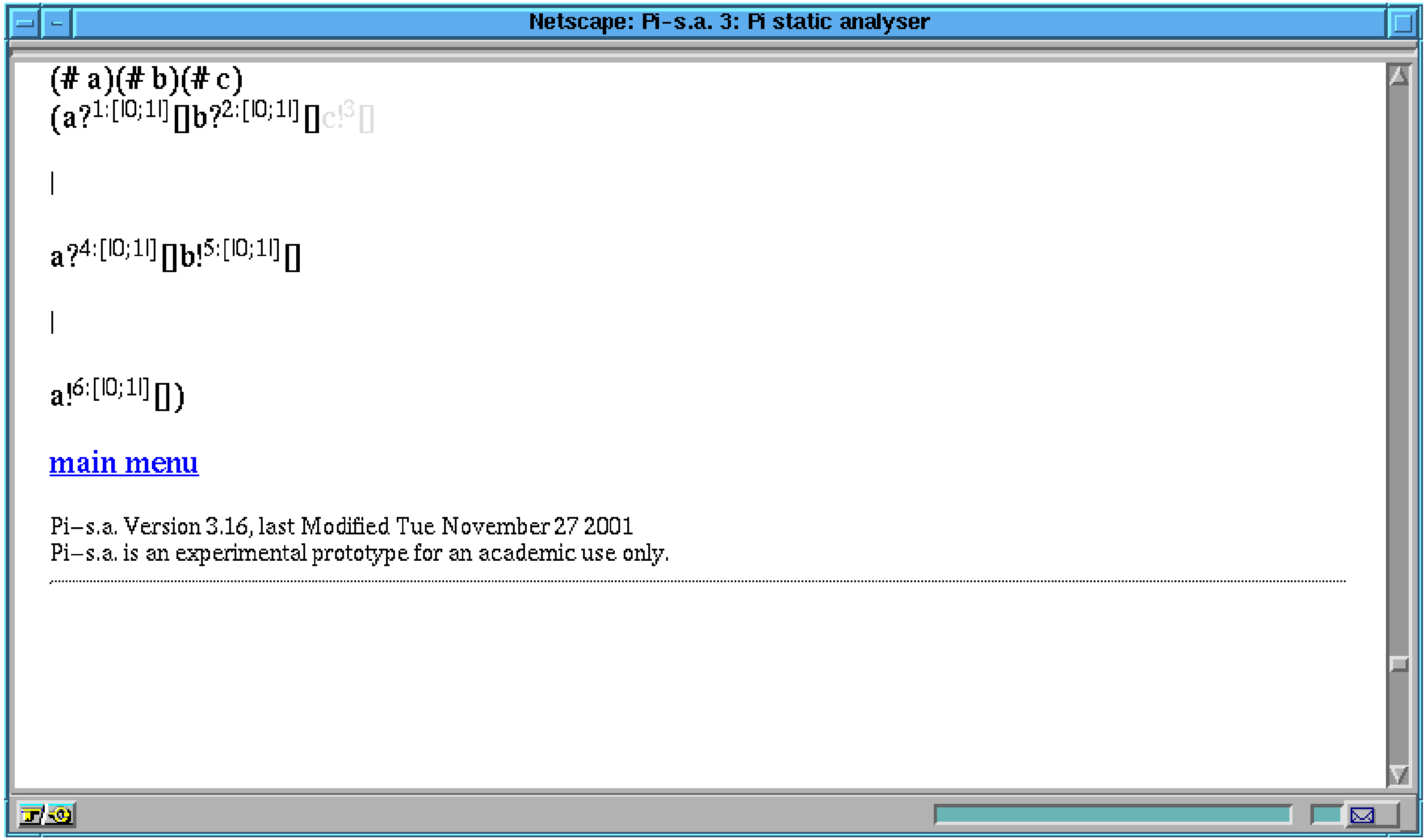
Netscape: Pi-s.a. 3: Pi static analyser

```
((# make)(# server)(# port)(# deal)
((*make?^{1:1}[](# address)(# request)
      (
       (*address?^{2:[|0;+oo|[}[]server!^{3:[|0;+oo|[}[address,request])
      |
       address!^{4:[|0;+oo|[}[]
      |
       make!^{5:[|0;1|]}[]))
|
 (*server?^{6:1}[email,request]
      (port?^{7:[|0;+oo|[}[]( port!^{8:[|0;3|]}[] | deal!^{9:[|0;+oo|[}[request] | deal?^{10:[|0;+oo|[}[rep]email!^{11:[|0;+oo|[}[rep])
      +
       email!^{12:[|0;+oo|[}[]
      ))
| port!^{13:[|0;1|]}[] | port!^{14:[|0;1|]}[] | port!^{15:[|0;1|]}[] | make!^{16:[|0;1|]}[]))
```

[main menu]()

Pi–s.a. Version 3.16, last Modified Tue November 27 2001
Pi–s.a. is an experimental prototype for an academic use only.

Example: mutual exclusion



Example: token ring

Pi.s.a III : a Pi-calculus Static Analyzer - Mozilla

```
(# make)(# mon)(# left0)
( (*make?^1:1[left](# right)(mon!^2:[|0;+oo|[[left,right] | make!^3:[|0;1|][right]))
| (*make?^4:1[left](mon!^5:[|0;1|][left,left0]))
| make!^6:[|0;1|][left0]

| (*mon?^7:1[prev,next]
      (*prev?^8:[|0;+oo|[[](# crit)
            ( crit!^9:[|0;1|][] | (crit?^10:[|0;1|][]next!^11:[|0;1|][]))))
| left0!^12:[|0;1|][])
```

Comparison

- Non relational analyses.
  [Levi and Maffeis: SAS’2001]

- Syntactic criteria.
  [Nielson *et al.*:SAS’2004]

- Abstract multisets.
  [Nielson *et al.*:SAS’1999,POPL’2000]

- Finite control systems.
  [Dam:IC’96],[Charatonik *et al.*:ESOP’02]

Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Computation unit

Gather threads inside an unbounded number of dynamically created computation units.
Then detect mutual exclusion inside each computation unit.

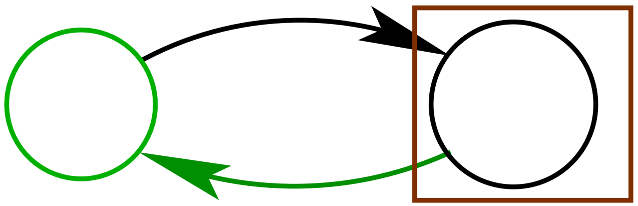
Each thread is associated with a computation unit, which is left as a parameter of:

- the model
- and the properties of interest.

For instance:

- in the π-calculus, the channel on which the input/output action is performed;
- in ambients, agent location and the location of its location [Nielson:POPL'2000].

Thread partitioning



Thread partitioning

We gather threads according to their computation unit.
We count the occurrence number of threads inside each computation unit.

To simulate a computation step, we require:

- to relate the computation units of:
  1. the threads that are consumed;
  2. the threads that are spawned.

  This may rely on the model structure (ambients) or on a precise environment analysis (other models).
- an occurrence counting analysis:

  to count occurrence of threads inside each computation unit.

Concrete partitioning

B: a finite set of indice.
We define the set of computation units as:

$$\mathit{unit} \stackrel{\Delta}{=} \mathrm{B} \longrightarrow \mathit{Label} \times \mathit{Id}.$$

give-index maps each program point p to a function $\mathit{give\text{-}index}(\mathrm{p}) \in \mathrm{B} \longrightarrow \mathrm{fn}(\mathrm{p})$.

Given a thread $\mathrm{t} = (\mathrm{p}, \mathit{id}, \mathrm{E})$, we define its computation unit $\mathit{give\text{-}unit}(\mathrm{t})$ as:

$$\mathit{give\text{-}unit}(\mathrm{t}) = [\mathrm{b} \in \mathrm{B} \longrightarrow \mathrm{E}(\mathit{give\text{-}index}(\mathrm{p})(\mathrm{b}))].$$

Abstract computation unit

There may be an unbounded number of computation units.

To get a decidable abstraction, we merge the description of the computation units that have the same labels.

We define:

$$\mathsf{UNIT}^\sharp \triangleq \mathcal{B} \to \mathit{Label}.$$

The abstraction function:

$$\Pi_{\mathit{unit}} \in \begin{cases} \mathit{unit} & \to \mathsf{UNIT}^\sharp \\ [b \in \mathcal{B} \mapsto (l_b, \_)] & \mapsto [b \mapsto l_b]; \end{cases}$$

maps each computation unit to an abstract one.

Abstract domain

Our main domain is a Cartesian product:

$$\mathcal{C}^{\sharp}_{part} \stackrel{\Delta}{=} \left(\Pi_{p \in \mathcal{L}_p} \mathcal{G}_{fn(p)}\right) \times \left(\mathsf{UNIT}^{\sharp} \to \mathcal{N}_{\mathcal{L}_p}\right).$$

The set $\gamma_{part}(\mathsf{ENV}, \mathsf{CU})$ contains any configuration $(v, C) \in \Sigma^* \times \mathcal{S}$ that satisfies:

1. $(v, C) \in \gamma_{\mathsf{ENV}}(\mathsf{ENV})$;
2. for any computation unit $u \in \mathit{unit}$, there exists a function

$$t \in \{(0) \in \mathbb{N}^{\mathcal{L}_p}\} \cup \left(\gamma_{\mathcal{N}_{\mathcal{L}_p}}(\mathsf{CU}(\Pi_{unit}(u)))\right)$$

   such that:

$$t(p) = \mathit{Card}(\{(p, \mathit{id}, E) \in C \mid \mathit{give\text{-}unit}(p, \mathit{id}, E) = u\}).$$

Balance molecule

To simulate an abstract computation step,

we compute an abstract molecule that describes:

- both the n threads that are interacting;
- and the m threads that are launched;

we also collect any information about the values in computation units:

- each thread is launched in a computation unit. Each value occurring in this computation unit may either be fresh, or may come from interacting threads;

  (we take into account these constraints in the abstract molecule).

Admissible relations

Then, we consider any potential choice for:

1. the equivalence relation among the computation unit of the $(n + m)$ threads involved in the computation step;
2. abstract computation units associated to each thread.

Each choice induces some constraints about:

- the control flow;
- the number of threads inside computation units;

We use these constraints to:

1. check that this choice is possible;
2. refine control flow and occurrence counting information;

Then, we simulate the computation step.

Shared-memory example

A memory cell will be denoted by three channel names, *cell*, *read*, *write*:

- the channel name *cell* describes the content of the cell:

  the process $cell![data]$ means that the cell *cell* contains the information *data*, this name is internal to the memory (not visible by the user).

- the channel name *read* allows reading requests:

  the process $read![port]$ is a request to read the content of the cell, and send it to the port *port*,

- the channel name *write* allows writing requests:

  the process $write![data]$ is a request to write the information *data* inside the cell.

Implementation

$\text{System} := (\nu\ \text{create})(\nu\ \text{null})(*\text{create}?[\text{d}].\text{Allocate}(d))$

$\text{Allocate}(\text{d}) :=$

$$(\nu\ \mathit{cell})(\nu\ \mathit{write})(\nu\ \mathit{read})$$
$$\text{init}(\mathit{cell}) \mid \text{read}(\mathit{read},\mathit{cell}) \mid \text{write}(\mathit{write},\mathit{cell}) \mid \text{d}![\mathit{read};\mathit{write}]$$

where

- $\text{init}(\mathit{cell}) := \mathit{cell}![\text{null}]$
- $\text{read}(\mathit{read},\mathit{cell}) := *\mathit{read}?[\mathit{port}].\mathit{cell}?[u](\mathit{cell}![u] \mid \mathit{port}![u])$
- $\text{write}(\mathit{write},\mathit{cell}) := *\mathit{write}?[\mathit{data},\mathit{ack}].\mathit{cell}?[u].(\mathit{cell}![\mathit{data}] \mid \mathit{ack}![])$

Absence of race conditions

The computation unit of a thread is the name of the channel on which it performs its i/o action.

We detect that there is never two simultaneous outputs on a channel opened by an instance of a (ν *cell*) restriction.

Other Applications

By choosing appropriate settings for the computation unit, it can be used to infer the following causality properties:

- authentication in cryptographic protocols;
- absence of race conditions in dynamically allocated memories;
- update integrity in reconfigurable systems.

Overview

1. Overview
2. Mobile systems
3. Non standard semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion

Conclusion

We have designed generic analyses:

- automatic, sound, terminating, approximate,
- model independent (META-language),
- context independent.

We have captured:

- dynamic topology properties:
  absence of communication leak between recursive agents,
- concurrency properties:
  mutual exclusion, non-exhaustion of resources,
- combined properties:
  absence of race conditions, authentication (non-injective agreement).

Future Work I
Enriching the META-language

- 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;
- Using our framework to describe and analyze mobility in industrial applications (ERLANG).

Future works II
High level properties

Fill the gap between:

- low level properties captured by our analyses;
- high level properties specified by end-users.

Our goal:

- check some formula in a logic [Caires and Cardelli:IC'2003/TCS'2004]
- still distinguishing recursive instances
  $\neq$ [Kobayashi:POPL'2001]

Future works III
Analyzing probabilistic semantics

In a biological system, a cell may die or duplicate itself. The choice between these two opposite behaviors is controlled by the concentration of components in the system.
$\Longrightarrow$ a reachability analysis is useless.

- Using a semantics where the transitions are chosen according to probabilistic distributions:

  $\Longrightarrow$ (e.g token-based abstract machines [Palamidessi:FOSSACS'00])

- Existing analyses consider finite control systems [Logozzo:SAVE'2001,Degano *et al.*:TSE'2001]

- We want to design an analysis for capturing the probabilistic behavior of unbounded systems.