Abstract interpretation of mobile systems

Jérôme Feret
Laboratoire d'Informatique de l'École Normale Supérieure
INRIA, ÉNS, CNRS
http://www.di.ens.fr/~feret

Wednesday, April the 20th
Overview

1. Overview
2. Mobile systems
3. Non standards 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!
Applications:

- communication protocols,
- cryptographic protocols,
- reconfigurable systems,
- biological systems,
- . . . .

Jerôme Feret
Outline

We design abstract semantics:

- sound, automatic, decidable,
- but approximate (indecidability).

Our approach is model-independent:
1. design of a META-language;
2. design of static analysis at the level of the META-language.

Three family of analyses:
1. static analysis of the dynamic links between components:
   (confinment, confidentiality, . . . )
2. counting the number of components:
   (mutual exclusion, non-exhaustion of resources, . . . )
3. static analysis of computation units:
   (race detection, authentication, update integrity,. . . ).
Static analysis of the dynamic links between components
Which components may interact with each other?

The analysis distinguishes between recursive components

Abstract domain: relations among words.

Publications: Feret — SAS’00, SAS’01, ESOP’02, JLP’05
Static analysis of the number of component
Bound the number of components

Abstract domain: numerical relations
(affine and interval invariants).

Publication: Feret — GETCO’00, JLAP’05
Thread partitioning

Idea:
- gather components in computation units,
  \[\Rightarrow\] thanks to the analysis of linkage between components ;
- count the number of components within each computation unit,
  \[\Rightarrow\] thanks to the occurrence counting analysis.

Advantage:
- each session is isolated,

which enables the analysis to focus to each session.
Overview

1. Overview
2. Mobile systems
3. Non standards 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:

client
mail, req

server

client
mail, req

server
mail, req

client
answer

server
answer
A network
\(\pi\)-calculus: syntax

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

\[
P ::= \text{action}.P \\
\mid (P | P) \\
\mid (\nu \; x)P \\
\mid \emptyset
\]

where \(n \geq 0, \ c, \ x_1, \ldots, x_n, \ x, \in \text{Name}, \ i \in \text{Label}.
\nu\) and \(?\) are the only name binders.

\(fn(P)\): free variables in \(P\),
\(bn(P)\): bound names in \(P\).
Transition semantics

A reduction relation and a congruence relation give the semantics of the \( \pi \)-calculus:

- the reduction relation specifies the result of computations:

\[
\begin{align*}
  c?^i[y]Q & \mid c!^j[x]P \overset{i,j}{\rightarrow} Q[y \leftarrow x] \mid P \\
  \star c?^i[y]Q & \mid c!^j[x]P \overset{i,j}{\rightarrow} Q[y \leftarrow x] \mid \star c?^i[y]Q \mid P
\end{align*}
\]

\[
\begin{align*}
  P \rightarrow Q & \quad P' \equiv P \quad P \rightarrow Q \quad Q \equiv Q' \\
  (\nu \; x)P \rightarrow (\nu \; x)Q & \quad P' \rightarrow Q' & \quad P \mid Q \rightarrow P' \mid Q
\end{align*}
\]
Congruence relation

- the congruence relation reveals redexs:

\[ P \parallel Q \equiv Q \parallel P \] (Commutativity)

\[ P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \] (Associativity)

\[ (\nu x)P \equiv (\nu y)(P[x \leftarrow y]) \text{ if } y \not\in fn(P) \] (\(\alpha\)-conversion)

\[ (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P \] (Swapping)

\[ ((\nu x)P) \parallel Q \equiv (\nu x)(P \parallel Q) \text{ if } x \not\in fn(Q) \] (Extrusion)

\[ (\nu x)\varnothing \equiv \varnothing \] (Garbage collection)
Exporting a channel

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

\[\equiv (\alpha\text{-conversion, swapping and extrusion})\]

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

\[\rightarrow\]

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

\[\equiv (\text{swapping and extrusion})\]

\[(\nu a)(\nu x_2)((\nu x_1)P(x_1, x_2)| R(x_2, y))\]
Example: syntax

\[ S := (\nu \text{ port})(\nu \text{ gen}) \]
\[ (\text{Server} \mid \text{Customer} \mid \text{gen}^{0}[[]]) \]

where

\[
\text{Server} := \ast \text{port}^{1}[[\text{info}, \text{add}]](\text{add}^{2}[[\text{info}]])
\]

\[
\text{Customer} := \ast \text{gen}^{3}[[]] ( (\nu \text{ data}) (\nu \text{ email}) \\
(\text{port}^{4}[[\text{data, email}]) \mid \text{gen}^{5}[[]])
\]
Example: computation

$(\nu \text{ port})(\nu \text{ gen})$

$\quad (\text{Server} | \text{Customer} | \text{gen}!^0[])$

$\rightarrow_{3,0} (\nu \text{ port})(\nu \text{ gen})(\nu \text{ data}_1)(\nu \text{ email}_1)$

$\quad (\text{Server} | \text{Customer} | \text{gen}!^5[] | \text{port}!^4[\text{data}_1,\text{email}_1])$

$\rightarrow_{1,4} (\nu \text{ port})(\nu \text{ gen})(\nu \text{ data}_1)(\nu \text{ email}_1)$

$\quad (\text{Server} | \text{Customer} | \text{gen}!^5[] | \text{email}_1!^2[\text{data}_1])$

$\rightarrow_{3,5} (\nu \text{ port})(\nu \text{ gen})(\nu \text{ data}_1)(\nu \text{ email}_1)(\nu \text{ data}_2)(\nu \text{ email}_2)$

$\quad (\text{Server} | \text{Customer} | \text{gen}!^5[] | \text{email}_1!^2[\text{data}_1] | \text{port}!^4[\text{data}_2,\text{email}_2])$

$\rightarrow_{1,4} (\nu \text{ port})(\nu \text{ gen})(\nu \text{ data}_1)(\nu \text{ email}_1)(\nu \text{ data}_2)(\nu \text{ email}_2)$

$\quad (\text{Server} | \text{Customer} | \text{gen}!^5[] | \text{email}_1!^2[\text{data}_1] | \text{email}_2!^2[\text{data}_2])$
\(\alpha\)-conversion

\(\alpha\)-conversion destroys the link between names and processes which have declared them:

\[
(\nu \text{port})(\nu \text{gen})(\nu \text{data}_1)(\nu \text{email}_1)
\]
\[
(\nu \text{data}_2)(\nu \text{email}_2)
\]
\[
(\text{Server} \mid \text{Customer} \mid \text{gen}_5[]
\]
\[
| \text{email}_1^4[\text{data}_1] \mid \text{email}_2^4[\text{data}_2]
\]
\sim_{\alpha}

\[
(\nu \text{port})(\nu \text{gen})(\nu \text{data}_2)
\]
\[
(\nu \text{email}_1)(\nu \text{data}_1)(\nu \text{email}_2)
\]
\[
(\text{Server} \mid \text{Customer} \mid \text{gen}_5[]
\]
\[
| \text{email}_1^4[\text{data}_2] \mid \text{email}_2^4[\text{data}_1]
\]
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

client generator

\[ \nu p, \nu data \]

\[ p \]

query

data \[ c(p) \]

\[ p \]

server

\[ ?p \]

instance

\[ i(p) \]

\[ p \]

query

data \[ i(p) \]

\[ p \]

answer

data

\[ i(p) \]

\[ p \]

query

data

\[ i(p) \]

\[ p \]

answer

data

\[ i(p) \]
Syntax

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

\[
\begin{align*}
n & \in Name \quad \text{(ambient name)} \\
l & \in Label \quad \text{(label)} \\
\end{align*}
\]

\[
P, Q ::= (\nu n)P \quad \text{(restriction)} \\
0 \quad \text{(inactivity)} \\
P | Q \quad \text{(composition)} \\
n'[P] \quad \text{(ambient)} \\
M \quad \text{(capability action)} \\
io \quad \text{(input/output action)}
\]
Capability and actions

\[ M ::= \text{in}^l n.P \quad \text{(can enter an ambient named } n) \]
\[ \quad | \quad \text{out}^l n.P \quad \text{(can exit an ambient named } n) \]
\[ \quad | \quad \text{open}^l n.P \quad \text{(can open an ambient named } n) \]
\[ \quad | \quad !\text{open}^l n.P \quad \text{(can open several ambients named } n) \]

\[ io ::= (n)^l.P \quad \text{(input action)} \]
\[ \quad | \quad !(n)^l.P \quad \text{(input action with replication)} \]
\[ \quad | \quad \langle n \rangle^l \quad \text{(async output action)} \]

The only name binders are \((\nu \_), \(\_\) and \(!\(\_\)).\]
Ambient Migration

\[
\text{in } m.P | Q \quad \text{m} \quad R \quad S \quad \rightarrow \quad \text{n} \quad P | Q \quad R \quad S
\]

\[
\text{out } m.P | Q \quad \text{m} \quad R \quad S \quad \rightarrow \quad \text{m} \quad R \quad n \quad P | Q \quad S
\]
Ambient Dissolution

\[ O \quad \text{open} \ m.P \quad m \quad Q \quad R \quad \rightarrow \quad O \quad P \mid Q \mid R \]

\[ O \quad \text{*open} \ m.P \quad m \quad Q \quad R \quad \rightarrow \quad O \quad \text{*open} \ m.P \mid P \mid Q \mid R \]
Communication

\[
\langle m \rangle | (n).P | Q \quad \rightarrow \quad P_{[n \leftarrow m]} | Q
\]

\[
\langle m \rangle | *(n).P | Q \quad \rightarrow \quad *(n).P | P_{[n \leftarrow m]} | Q
\]
An \textit{ftp}-server

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

\textit{where}

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

\[ C := (\nu q)(\nu p)^{12}[C_1 \mid C_2 \mid C_3] \mid \langle \text{make} \rangle^{20}, \]
\[ C_1 := \text{request}^{13}[\langle q \rangle^{14}], \quad C_2 := \text{open}^{15} \text{instance}, \]
\[ C_3 := \text{in}^{16} \text{server.duplicate}^{17}[\text{out}^{18} p.\langle p \rangle^{19}], \]

\[ S := \text{server}^{1}[S_1 \mid S_2], \quad S_1 := !\text{open}^{2} \text{duplicate}, \quad S_2 := !(k)^{3} \text{instance}^{4}[I], \]

\[ I := \text{in}^{5} k.\text{open}^{6} \text{request}^{7}(\text{rep})^{8}[I_1 \mid I_2], \quad I_1 := \text{answer}^{8}[\langle \text{rep} \rangle^{9}], \quad I_2 := \text{out}^{10} \text{server}^{11}. \]
\[(\nu \text{Pub})(S \mid ! (x)^{11}.C \mid \langle \text{make} \rangle^{21})\]

\[\rightarrow\]

\[(\nu \text{Pub})\left( !(x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^{1}[S_1 \mid S_2] \mid \right.

\left. (\nu q_1)(\nu p_1)p_1^{12} \left[ \text{request}^{13}[\langle q_1 \rangle^{14}] \mid C_2 \mid \text{in}^{16}\text{server.duplicate}^{17}[\text{out}^{18}p_1.\langle p_1 \rangle^{19}] \right] \right)\]

\[\rightarrow\]

\[(\nu \text{Pub})(\nu q_1)(\nu p_1)\]

\[(\nu \text{Pub})(\nu q_1)(\nu p_1)\left( !(x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^{1}[S_1 \mid S_2 \mid p_1^{12} \left[ \text{request}^{13}[\langle q_1 \rangle^{14}] \mid C_2 \mid \text{duplicate}^{17}[\text{out}^{18}p_1.\langle p_1 \rangle^{19}] \right] \right)\]

\[\rightarrow\]

\[(\nu \text{Pub})(\nu q_1)(\nu p_1)\left( !(x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^{1}[p_1^{12} \left[ \text{open}^{2}\text{duplicate} \mid S_2 \mid \text{duplicate}^{17}[\langle p_1 \rangle^{19}] \right] \left[ \text{request}^{13}[\langle q_1 \rangle^{14}] \mid C_2 \right] \right)\)
\[(\nu \text{Pub})(\nu q_1)(\nu p_1)\]
\[
\left( !(x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^1 \right]
\]
\[
\left( !(x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^1 \right]
\]
\[
\left( \begin{array}{c}
! \text{open}^2 \text{duplicate} \mid S_2 \mid \text{duplicate}^{17}[\langle p_1 \rangle^{19}] \\
p_1^{12}[\text{request}^{13}[\langle q_1 \rangle^{14}] \mid C_2]
\end{array} \right)
\]
\[
\rightarrow
\]
\[
\left( \nu \text{Pub})(\nu q_1)(\nu p_1)\right)
\]
\[
\left( !(x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^1 \right]
\]
\[
\left( \begin{array}{c}
S_1 \mid !(k)^3.\text{instance}^4[l] \mid \langle p_1 \rangle^{19} \\
p_1^{12}[\text{request}^{13}[\langle q_1 \rangle^{14}] \mid C_2]
\end{array} \right)
\]
\[
\rightarrow
\]
\[
\left( \nu \text{Pub})(\nu q_1)(\nu p_1)\right)
\]
\[
\left( !(x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^1 \right]
\]
\[
\left( \begin{array}{c}
S_1 \mid S_2 \mid p_1^{12}[\text{request}^{13}[\langle q_1 \rangle^{14}] \mid C_2] \\
\text{instance}^4[\text{in} p_1.\text{open}^6 \text{request}.(\text{rep})^7(l_1 \mid l_2)]
\end{array} \right)
\]
\[
\rightarrow
\]
\[
\left( \nu \text{Pub})(\nu q_1)(\nu p_1)\right)
\]
\[
\left( !(x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^1 \right]
\]
\[
\left( \begin{array}{c}
S_1 \mid S_2 \\
p_1^{12}[\text{request}^{13}[\langle q_1 \rangle^{14}] \mid \text{open}^{15} \text{instance} \mid \text{instance}^4[\text{open}^6 \text{request}.(\text{rep})^7(l_1 \mid l_2)]
\end{array} \right)
\]
\[(\nu \text{Pub})(\nu \ q_1)(\nu \ p_1) \]
\[
\left( (x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^1 \right] \\
\left[ \begin{array}{c} S_1 \mid S_2 \\ p_1^{12} \left[ \begin{array}{c} \text{request}^{13}\langle q_1 \rangle^{14} \mid \text{open}^{15}\text{instance} \\ \text{instance}^{4}\langle \text{open} \rangle^{6}\text{request.(rep)}^{7}(l_1 \mid l_2) \end{array} \right] \end{array} \right) \]
\[\rightarrow\]
\[(\nu \text{Pub})(\nu \ q_1)(\nu \ p_1) \]
\[
\left( (x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^1 \right] \\
\left[ \begin{array}{c} S_1 \mid S_2 \\ p_1^{12} \left[ \begin{array}{c} \text{request}^{13}\langle q_1 \rangle^{14} \mid \text{open}^{6}\text{request.(rep)}^{7}(l_1 \mid l_2) \end{array} \right] \end{array} \right) \]
\[\rightarrow^*\]
\[(\nu \text{Pub})(\nu \ q_1)(\nu \ p_1) \]
\[
\left( (x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^1 \right] \\
\left[ \begin{array}{c} S_1 \mid S_2 \\ p_1^{12} \left[ \begin{array}{c} \text{answer}^{8}\langle q_1 \rangle^{9} \mid \text{out}^{10}\text{server} \end{array} \right] \end{array} \right) \]
\[\rightarrow\]
\[(\nu \text{Pub})(\nu \ q_1)(\nu \ p_1) \]
\[
\left( (x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^1 \right] \\
\left[ \begin{array}{c} S_1 \mid S_2 \\ p_1^{12} \left[ \begin{array}{c} \text{answer}^{8}\langle q_1 \rangle^{9} \end{array} \right] \end{array} \right) \]
\[\rightarrow^*\]
\[(\nu \text{Pub})(\nu \ q_1)(\nu \ p_1)(\nu \ q_2)(\nu \ p_2) \]
\[
\left( (x)^{11}.C \mid \langle \text{make} \rangle^{20} \mid \text{server}^1 \right] \\
\left[ \begin{array}{c} S_1 \mid S_2 \\ p_1^{12} \left[ \begin{array}{c} \text{answer}^{8}\langle q_1 \rangle^{9} \mid \text{answer}^{8}\langle q_2 \rangle^{9} \end{array} \right] \end{array} \right) \]
We want to describe in the $\pi$-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

System := (ν create)(ν null)(*create?[d].Allocate(d))

Allocate(d) :=

(ν cell)(ν write)(ν read)

(init(cell) | read(read,cell) | write(write,cell) | d![read;write])

where

- init(cell) := cell![null]
- read(read,cell) := *read?[port].cell?[u].(cell![u] | port![u])
- write(write,cell) := *write?[data].cell?[u].cell![data]
Shared-memory example
Trace example

(ν create)(ν null)
  (*create?[d].Allocate(d)
   | (ν address)(ν data)create![address].address?[r;w].w![data].r![address])
→
(ν create) (ν null) (ν cell) (ν write) (ν read) (ν address) (ν data)
  ( *create?[d].Allocate(d) | read(read,cell) | write(write,cell)
   | cell![null] | address![read,write] | address?[r;w].w![data].r![address])
→
(ν ń)( *create?[d].Allocate(d) | read(read,cell) | write(write,cell)
   | cell![null] | write![data].read![address])
→
(ν ń)( *create?[d].Allocate(d) | read(read,cell) | write(write,cell)
   | cell![null] | cell?[u].cell![data] | read![address])
Shared-memory example

Expected Derivation

\((\nu \overline{c})( \ast \text{create}?[d].\text{Allocate}(d) \mid \text{read}(\text{read},\text{cell}) \mid \text{write}(\text{write},\text{cell}) \mid \text{cell}![\text{null}] \mid \text{cell}[u].\text{cell}![\text{data}] \mid \text{read}![\text{address}])\)

\rightarrow

\((\nu \overline{c})( \ast \text{create}?[d].\text{Allocate}(d) \mid \text{read}(\text{read},\text{cell}) \mid \text{write}(\text{write},\text{cell}) \mid \text{cell}![\text{data}] \mid \text{read}![\text{address}])\)

\rightarrow

\((\nu \overline{c})( \ast \text{create}?[d].\text{Allocate}(d) \mid \text{read}(\text{read},\text{cell}) \mid \text{write}(\text{write},\text{cell}) \mid \text{cell}![\text{data}] \mid \text{cell}[u].(\text{cell}![u] \mid \text{address}![u]))\)

\rightarrow

\((\nu \overline{c})( \ast \text{create}?[d].\text{Allocate}(d) \mid \text{read}(\text{read},\text{cell}) \mid \text{write}(\text{write},\text{cell}) \mid \text{cell}![\text{data}] \mid \text{address}![\text{data}])\)
Shared-memory example

Unexpected Derivation

\( (\nu \bar{c})( \ast \text{create}\?[d].\text{Allocate}\,(d) \mid \text{read}(\text{read,cell}) \mid \text{write}(\text{write,cell}) \mid \text{cell}![\text{null}] \mid \text{cell}?[u].\text{cell}![\text{data}] \mid \text{read}![\text{address}]) \rightarrow \)

\( (\nu \bar{c})( \ast \text{create}\?[d].\text{Allocate}\,(d) \mid \text{read}(\text{read,cell}) \mid \text{write}(\text{write,cell}) \mid \text{cell}![\text{null}] \mid \text{cell}?[u].\text{cell}![\text{data}] \mid \text{cell}?[u].(\text{cell}![u] \mid \text{address}![u])) \rightarrow \)

\( (\nu \bar{c})( \ast \text{create}\?[d].\text{Allocate}\,(d) \mid \text{read}(\text{read,cell}) \mid \text{write}(\text{write,cell}) \mid \text{cell}![\text{null}] \mid \text{cell}?[u].\text{cell}![\text{data}] \mid \text{address}![\text{null}]) \rightarrow \)

\( (\nu \bar{c})( \ast \text{create}\?[d].\text{Allocate}\,(d) \mid \text{read}(\text{read,cell}) \mid \text{write}(\text{write,cell}) \mid \text{cell}![\text{data}] \mid \text{address}![\text{null}]) \)
Shared-memory example
Enforcing synchronisation

System := (ν create)(ν null)(∗create?[d].Allocate(d))

Allocate(d) :=
(ν cell)(ν write)(ν read)
  init(cell) | read(read,cell) | write(write,cell) | d![read;write]

where

• init(cell) := cell![null]
• read(read,cell) := ∗read?[port].cell?[u](cell![u] | port![u])
• write(write,cell) := ∗write?[data,ack].cell?[u].(cell![data] | ack![])

Jérôme Feret 38 Wednesday, April the 20th
\((\nu \text{create})(\nu \text{null})\)
\((\star \text{create}\,[d].\text{Allocate}(d)\)
\quad | \quad (\nu \text{address})(\nu \text{data})(\nu \text{ack})\)
\quad | \quad \text{create!}[\text{address}].\text{address?}[r;w].w![\text{data};\text{ack}].\text{ack?}[].r![\text{address}])\)
\rightarrow
\((\nu \overline{c})(\star \text{create}\,[d].\text{Allocate}(d)\quad | \quad \text{read}(\text{read},\text{cell})\quad | \quad \text{write}(\text{write},\text{cell})\quad | \quad \text{cell!}[\text{null}]\)
\quad | \quad \text{address!}[\text{read},\text{write}]\quad | \quad \text{address?}[r;w].w![\text{data};\text{ack}].\text{ack?}[].r![\text{address}])\)
\rightarrow
\((\nu \overline{c})(\star \text{create}\,[d].\text{Allocate}(d)\quad | \quad \text{read}(\text{read},\text{cell})\quad | \quad \text{write}(\text{write},\text{cell})\quad | \quad \text{cell!}[\text{null}]\quad | \quad \text{write!}[\text{data};\text{ack}].\text{ack?}[].\text{read!}[\text{address}])\)
\rightarrow
\((\nu \overline{c})(\star \text{create}\,[d].\text{Allocate}(d)\quad | \quad \text{read}(\text{read},\text{cell})\quad | \quad \text{write}(\text{write},\text{cell})\quad | \quad \text{cell!}[\text{null}]\quad | \quad \text{cell?}[u].(\text{cell!}[\text{data}]\quad | \quad \text{ack!}[])\quad | \quad \text{ack?}[].\text{read!}[\text{address}])\)
\( (\nu \bar{c})( \ast \text{create}[\bar{d}] \cdot \text{Allocate}(\bar{d}) \mid \text{read}(\text{read},\text{cell}) \mid \text{write}(\text{write},\text{cell}) \mid \text{cell}[\text{null}] \mid \text{cell}[u] \cdot (\text{cell}[\text{data}] \mid \text{ack}[]) \mid \text{ack}[] \cdot \text{read}![\text{address}] ) \)

\[ \rightarrow \]

\( (\nu \bar{c})( \ast \text{create}[\bar{d}] \cdot \text{Allocate}(\bar{d}) \mid \text{read}(\text{read},\text{cell}) \mid \text{write}(\text{write},\text{cell}) \mid \text{cell}[\text{data}] \mid \text{ack}[] \mid \text{ack}[] \cdot \text{read}![\text{address}] ) \)

\[ \rightarrow \]

\( (\nu \bar{c})( \ast \text{create}[\bar{d}] \cdot \text{Allocate}(\bar{d}) \mid \text{read}(\text{read},\text{cell}) \mid \text{write}(\text{write},\text{cell}) \mid \text{cell}[\text{data}] \mid \text{read}![\text{address}] ) \)

\[ \rightarrow \]

\( (\nu \bar{c})( \ast \text{create}[\bar{d}] \cdot \text{Allocate}(\bar{d}) \mid \text{read}(\text{read},\text{cell}) \mid \text{write}(\text{write},\text{cell}) \mid \text{cell}[\text{data}] \mid \text{address}![\text{data}] ) \)
Shared-memory example
Using Mutex

System ::= (\nu \text{create})(\nu \text{null})(\star \text{create}\left[d\right]\text{Allocate}(d))
Allocate(d) ::= (\nu \text{cell})(\nu \text{mutex})(\nu \text{nomutex})(\nu \text{write})(\nu \text{read})(\nu \text{lock})(\nu \text{unlock})
  \quad \text{init}(\text{cell,mutex}) \mid \text{read}(\text{read,cell}) \mid \text{write}(\text{write,cell})
  \mid \text{lock}(\text{lock,mutex,nomutex}) \mid \text{unlock}(\text{unlock,mutex,nomutex})
  \mid d!\left[\text{read;write;lock;unlock}\right]

where
\quad \text{init}(\text{cell,mutex}) ::= \text{cell}!\left[\text{null}\right] \mid \text{mutex}![]
\quad \text{read}(\text{read,cell}) ::= \star \text{read}\left[\text{port}\right].\text{cell}?!\left[u\right](\text{cell}!\left[u\right] \mid \text{port}!\left[u\right])
\quad \text{write}(\text{write,cell}) ::= \star \text{write}\left[\text{data,ack}\right].\text{cell}?!\left[u\right].(\text{cell}!\left[\text{data}\right] \mid \text{ack}![])
\quad \text{lock}(\text{lock,mutex,nomutex}) ::= \star \text{lock}\left[\text{ack}\right].\text{mutex}?!\left[]\right.(\text{ack}![] \mid \text{nomutex}![])
\quad \text{unlock}(\text{unlock,mutex,nomutex}) ::= \star \text{unlock}\left[\text{ack}\right].\text{nomutex}?!\left[]\right.(\text{ack}![] \mid \text{mutex}![])
Overview

1. Overview
2. Mobile systems
3. Non standards 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 which

- 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

\[
\begin{align*}
\text{(Server} \mid \text{Client} \mid \text{gen!}^5[] \mid \text{email}_1 \downarrow^2 [\text{data}_1] \mid \text{email}_2 \downarrow^2 [\text{data}_2])
\end{align*}
\]
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 \((i, j) \in Label^2\).

They are recursively calculated when fetching resources as follows:

\[
id_* : \\
\begin{array}{c}
\text{(i,j)} \\
\end{array}
\begin{array}{c}
\text{id}_r \\
\end{array}
\begin{array}{c}
\text{id}_l \\
\end{array}
\]
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

\[ C \cup \{(P \mid Q, \text{id}, E)\} \xrightarrow{\varepsilon} C \cup \{(P, \text{id}, E_{|fn(P)}); (Q, \text{id}, E_{|fn(Q)})\} \]

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

\[ C \cup \{((\emptyset, \text{id}, E)\} \xrightarrow{\varepsilon} C \]
Communication rules

\[
E?(y) = E_1(x)
\]

\[
C \cup \left\{ (y?^i \overline{y}, P, id?, E?); (x!^j \overline{x}, Q, id!, E_1) \right\} \xrightarrow{(i,j)} C \cup \left\{ (P, id?, E?[\overline{y} \rightarrow E_1(\overline{x})]|fn(P)); (Q, id!, E_1|fn(Q)) \right\}
\]

\[
E?(y) = E_1(x)
\]

\[
C \cup \left\{ (*y?^i \overline{y}, P, id?, E?); (x!^j \overline{x}, Q, id!, E_1) \right\} \xrightarrow{(i,j)} C \cup \left\{ (*y?^i \overline{y}, P, id?, E?); (P, N((i, j), id?, id!), E?[\overline{y} \rightarrow E_1(\overline{x})]|fn(P)); (Q, id!, E_1|fn(Q)) \right\}
\]
Choice rules

\[ C \cup \{ P + Q, id, E \} \rightarrow C \cup \{ (P, id, E_{\text{fn}(P)} \} \]

\[ C \cup \{ P + Q, id, E \} \rightarrow C \cup \{ (Q, id, E_{\text{fn}(Q)} \} \]
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 $\text{father}(P)$ the father of $P$, when it exists, in the syntactic tree of $S$.

1. the thread $(S, \varepsilon, \emptyset)$ can only be created at the start of the system computation;
2. a thread $(P, id, \_)$ such that $\text{father}(P)$ is not a resource, can only be created by making a thread $(\text{father}(P), id, \_)$ react;
3. a thread $(P, N((i, j), id?, id!), \_)$ can only be created by making a thread $(P_j, id!, \_)$ 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:

\[
\begin{align*}
\phi_1(N((i, j), id_1, id_2)) &= \phi_1(id_2).(i, j) \\
\phi_1(\varepsilon) &= \varepsilon
\end{align*}
\]

2. replacing pairs by their second component:

\[
\begin{align*}
\phi_2(N((i, j), id_1, id_2)) &= \phi_2(id_2).j \\
\phi_2(\varepsilon) &= \varepsilon
\end{align*}
\]

Those simplifications can be seen as an abstraction, they do not lose 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.

⇒ We design a new semantics with only active rules.

(Structural rules are included inside active rules)
Definition

Structural rules:

\[ C \cup \{(P \mid Q, id, E)\} \xrightarrow{\varepsilon} C \cup \{(P, id, E_{fn(P)}); (Q, id, E_{fn(Q)})\} \]

\[ C \cup \{((\nu x)P, id, E)\} \xrightarrow{\varepsilon} C \cup \{(P, id, E[x \rightarrow (x, id)]_{fn(P)})\} \]

\[ C \cup \{(\emptyset, id, E)\} \xrightarrow{\varepsilon} C \]

are a confluent and well-founded transition system, we denote by \(\Rightarrow\) its limit:

\[ a \Rightarrow b \text{ ssi } \begin{cases} a \rightarrow^* b \\ \forall c, b \not\rightarrow c. \end{cases} \]

and we define our new transition system by \(\overset{\lambda}{\Rightarrow} = \overset{\lambda}{\Rightarrow}\).
An extraction function calculates the set of the thread instances spawned at the beginning of the system execution or after a computation step.

\[
\begin{align*}
\beta((\nu n)P, id, E) &= \beta(P, id, (E[n \mapsto (n, id)])) \\
\beta(\emptyset, id, E) &= \emptyset \\
\beta(P | Q, id, E) &= \beta(P, id, E) \cup \beta(Q, id, E) \\
\beta(P + Q, id, E) &= \{(P + Q, id, E|_{fn(P+Q)})\} \\
\beta(y?i[y].P, id, E) &= \{(y?i[y].P, id, E|_{fn(y?i[y].P)})\} \\
\beta(*y?i[y].P, id, E) &= \{(*y?i[y].P, id, E|_{fn(*y?i[y].P)})\} \\
\beta(x!j[x].P, id, E) &= \{(x!j[x].P, id, E|_{fn(x!j[x].P)})\}
\end{align*}
\]
Transition system

\[ C_0(S) = \beta(S, \varepsilon, \emptyset) \]

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

\[ C \cup \{(P+Q, id, E)\} \xrightarrow{\varepsilon} (C \cup \beta(Q, id, E)) \]
Communication rules

\[ E_?(y) = E_1(x) \]

\[ C \cup \left\{ \begin{array}{l}
(y?^i[y]P, id?, E_?) \\
(x!^j[x]Q, id_1, E_1)
\end{array} \right\}^{(i,j)} \rightarrow (C \cup \beta(P, id?, E_?[y_i \mapsto E_1(x_i)]) \cup \beta(Q, id_1, E_1)) \]

\[ E_*(y) = E_!(x) \]

\[ C' \cup \left\{ \begin{array}{l}
(*y?^i[y]P, id_*, E_*) \\
(x!^j[x]Q, id_1, E_1)
\end{array} \right\}^{(i,j)} \rightarrow \left(\begin{array}{l}
C \cup \{(*y?^i[y]P, id_*, E_*)\} \\
\cup \beta(P, N((i, j), id_*, id_1), E_*[y_i \mapsto E_!(x_i)])
\cup \beta(Q, id_1, E_1)
\end{array} \right) \]
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 $\Rightarrow$ its non-deterministic limit:

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

and we define our new transition system by $\Rightarrow^\prime = \Rightarrow \circ \Rightarrow$. 
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.

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

$$C_0(S) = \beta(S, \varepsilon, \emptyset)$$

$$E?(y) = E!(x), \ Cont_P \in \beta(P, id?, E?[y_i \mapsto E!(x_i)]), \ Cont_Q \in \beta(Q, id!, E!)$$

$$C \cup \{(y^i[y]P, id?, E?), (x^j[x]Q, id!, E!}\}^{(i,j)} \rightarrow (C \cup Cont_P \cup Cont_Q)$$

$$E_*(y) = E!(x), \ Cont_P \in \beta(P, N((i,j), id_*, id!), E_*[y_i \mapsto E!(x_i)]), \ Cont_Q \in \beta(Q, id!, E!)$$

$$C \cup \left\{ (\ast y^i[y]P, id_*, E_*), (x^j[x]Q, id!, E!) \right\}^{(i,j)} \rightarrow (C \cup \{(\ast y^i[y]P, id_*, E_*)\} \cup Cont_P \cup Cont_Q)$$
META-language: intuition

In the $\pi$-calculus:

- each program point $a?[y]P$ is associated with a partial interaction:
  \[(in, [a], [y], label(P))\]

- each program point $b![x]Q$ is associated with a partial interaction:
  \[(out, [b, x], [], label(Q))\]

- The generic transition rule:
  \[((in, out), [X_1^1 = X_2^2], [Y_1^1 \leftarrow X_2^2])\].

  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 $\alpha$-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

\[
\text{Context} := (\nu \text{ unsafe}) (\text{new} \\
\quad \mid \text{spy}_0 \mid \ldots \mid \text{spy}_n \\
\quad \mid \text{spoil}_0 \mid \ldots \mid \text{spoil}_n )
\]

where

\[
\text{new} := (\ast (\nu \text{ channel}) \ast \text{unsafe}! [\text{channel}])
\]

\[
\text{spoil}_k := (\ast \text{unsafe}? [c] \ast \text{unsafe}? [x_1] \ldots \ast \text{unsafe}? [x_k] c! [x_1, \ldots, x_k])
\]

\[
\text{spy}_k := (\ast \text{unsafe}? [c] c? [x_1, \ldots, x_k] ((\ast \text{unsafe}! [x_1]) \mid \ldots \mid (\ast \text{unsafe}! [x_k])))
\]
Non-Standard Configuration

We flatly represent system configurations:

\[
\begin{align*}
(p^{12}[\bullet], id_0, (top, \varepsilon), [p \mapsto (p, id_0)]) \\
(p^{12}[\bullet], id_1, (top, \varepsilon), [p \mapsto (p, id_1)]) \\
(\text{answer}^8[\bullet], id'_0, (12, id_0), \emptyset) \\
(\text{answer}^8[\bullet], id'_1, (12, id_1), \emptyset) \\
(\langle rep \rangle^9, id'_0, (8, id_0), [rep \mapsto (data, id_0)]) \\
(\langle rep \rangle^9, id'_1, (8, id_1), [rep \mapsto (data, id_1)])
\end{align*}
\]
In migration

\[
\begin{align*}
\lambda &= (n^i[\bullet], id_1, loc_1, E_1), \\
\mu &= (m^j[\bullet], id_2, loc_2, E_2), \\
\psi &= (in^k o.P, id_3, loc_3, E_3), \\
loc_1 &= loc_2, \quad loc_3 = (i, id_1), \quad E_2(m) = E_3(o), \quad \lambda \neq \mu.
\end{align*}
\]

\[
C \cup \{\lambda; \mu; \psi\} \xrightarrow{in(i,j,k)} (C \cup \{\mu\}) \cup (n^i[\bullet], id_1, (j, id_2), E_1) \cup \beta (P, id_3, loc_3, E_3|_{fn(P)}).
\]
out migration

\[
\begin{align*}
\lambda &= (m^i[\bullet], id_1, loc_1, E_1), \\
\mu &= (n^j[\bullet], id_2, loc_2, E_2), \\
\psi &= (\text{out}^k o.P, id_3, loc_3, E_3), \\
loc_2 &= (i, id_1), \quad loc_3 = (j, id_2), \quad E_1(m) = E_3(o)
\end{align*}
\]

\[C \cup \{\lambda; \mu; \psi\} \xrightarrow{\text{out}(i,j,k)} (C \cup \{\lambda\}) \cup (n^j[\bullet], id_2, loc_1, E_2) \cup \beta \left( P, id_3, loc_3, E_3|_{fn(P)} \right).\]
Dissolution

\[
\begin{align*}
\lambda &= \left( \text{open}^j_m.P, \text{id}_1, \text{loc}_1, E_1 \right) \\
\mu &= \left( n^j[\bullet], \text{id}_2, \text{loc}_2, E_2 \right), \\
\text{loc}_1 &= \text{loc}_2, \ E_1(m) = E_2(n),
\end{align*}
\]

\[
C \cup \{ \lambda; \mu \} \xrightarrow{\text{open}(i,j)} (C \setminus A) \cup A' \cup \beta \left( P, \text{id}_1, \text{loc}_1, E_1|_{fn(P)} \right)
\]

where

\[
\begin{align*}
A &= \{(a, \text{id}, \text{loc}, E) \in C \mid \text{loc} = (j, \text{id}_2)\} \\
A' &= \{(a, \text{id}, \text{loc}_2, E) \mid (a, \text{id}, (j, \text{id}_2), E) \in C\}.
\end{align*}
\]
Overview

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

\[ S = \{ C \mid \exists i \in C_0, \ i \rightarrow^* C \} \]

It is also given by the least fixpoint of the following \( \cup \)-complete endomorphism \( F \):

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

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: \((\mathcal{D}^\#, \subseteq, \cup, \bot, \cap, \top)\) and is related to the concrete domain \(\wp(\mathcal{C})\) by a monotonic concretization function \(\gamma\).

\[ \forall A \in \mathcal{D}^\#, \gamma(A) \text{ is the set of the elements which satisfy the property } A.\]
Numerical domains

- sign approximation;
- interval approximation;
- octagonal approximation;
- polyhedra approximation;
- concrete domain.
Abstract transition system

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

Then, $S \subseteq \bigcup_{n \in \mathbb{N}} \gamma(\mathbb{F}^n(C_0^\#))$, where $\mathbb{F}(C^\#) = C_0^\# \sqcup C^\# \sqcup \left( \bigcup_{\text{finite}} \{ \overline{C^\#} \mid C^\# \sim \overline{C^\#} \} \right)$. 

Jérôme Feret 76 Wednesday, April the 20th
Widening operator

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

\[ \nabla : D^\# \times D^\# \rightarrow D^\# \]

such that:

- \( \forall X^\#_1, X^\#_2 \in D^\#, \quad X^\#_1 \sqcup X^\#_2 \subseteq X^\#_1 \nabla X^\#_2 \)

- for all increasing sequence \( (X^\#_n) \in (D^\#)^\mathbb{N} \), the sequence \( (X^\nabla_n) \) defined as

\[
\begin{align*}
X^\nabla_0 &= X^\#_0 \\
X^\nabla_{n+1} &= X^\nabla_n \nabla X^\#_{n+1}
\end{align*}
\]

is ultimately stationary.
The abstract iteration \((C_n^\nabla)\) of \(\mathcal{F}^\#\) defined as follows

\[
\begin{align*}
C_0^\nabla &= C_0^\# \\
C_{n+1}^\nabla &= \begin{cases} 
C_n^\nabla & \text{if } \mathcal{F}^\#(C_n^\nabla) \subseteq C_n^\nabla \\
C_n^\nabla \nabla \mathcal{F}^\#(C_n^\nabla) & \text{otherwise}
\end{cases}
\end{align*}
\]

is ultimately stationary and its limit \(C^\nabla\) satisfies \(lfp_0\mathcal{F} \subseteq \gamma(C^\nabla)\).
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 $\mathcal{I}$:

\[
\begin{cases}
[a; b] \triangledown [c; d] &= [\min\{a; c\}; \infty]\quad \text{if } d > \max\{n; b\} \\
I \triangledown J &= I \sqcup J \quad \text{otherwise}
\end{cases}
\]
Composing two abstractions

Given two abstractions \((D^\#, \gamma, C_0^\#, \leadsto, \nabla)\) and \((D^\#, \gamma, C_0^\#, \leadsto, \nabla)\), and a reduction \(\rho : D^\# \times D^\# \rightarrow D^\# \times D^\#\) which satisfy:

\[
\forall (A, A) \in D^\# \times D^\#, \ \gamma(A) \cap \gamma(A) \subseteq \gamma(a) \cap \gamma(a) \quad \text{where} \quad (a, a) = \rho(A, A).
\]

Then \((D^\#, \gamma, C_0^\#, \leadsto, \nabla)\) where:

- \(D^\# = D^\# \times D^\#\);
- \(\nabla\) is pair-wisely defined;
- \(\gamma(A, A) = \gamma(A) \cap \gamma(A)\);
- \(C_0^\# = \rho(C_0^\#, C_0^\#)\);
- \((A, A) \leadsto \rho(C, C)\)
  - if \(B \leadsto C\) and \(B \leadsto C\) and \((B, B) = \rho(A, A)\)

is also an abstraction.

Jérôme Feret 80 Wednesday, April the 20th
Overview

1. Overview
2. Mobile systems
3. Non standards 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 $G_V$ to describe the markers and the environments which may be associated to a syntactic component the free name of which is $V$:

$$\mathcal{A}(Id \times (V \rightarrow (Name \times Id))) \overset{\gamma_V}{\longleftarrow} G_V.$$  

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

$$C^\# = \prod_{p \in P} G_{fn(p)}$$

related to $\mathcal{A}(C)$ by the concretization $\gamma$:

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

\[ y? \overline{y}. (\nu \overline{p}) P \]

\[ x! \overline{x}. (\nu \overline{q}) Q \]

Environment Property

Variable Property

Relational Information

Synchronization Constraint

Jérôme Feret

Wednesday, April the 20th
Extending environments

\[ y?\overline{y}.(\nu p)P \]

\[ x!\overline{x}.(\nu q)Q \]
Synchronizing environments

\[ y[y].(\nu p)P \]

\[ x[x].(\nu q)Q \]

Environment Property

Environment Extension

Relational Information

Variable Property

Synchronization Constraint
Propagating information

\[ y ? [y].(\nu p)P \]

\[ x ! [x].(\nu q)Q \]
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
Control flow analyses

We abstract for each variable $x$ and each name restriction $\nu y$ the set of marker pair $(id_x, id_y)$ 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.

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

$$G_V = V \times Name \rightarrow Id^\sharp$$

$\gamma_V(a^\sharp)$ 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:
\[ \wp(\Sigma) \times \wp(\Sigma) \times \wp(\Sigma \times \Sigma) \times \{\text{true};\text{false}\} \].

\( \gamma(I, F, T, b) \) 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 \( O(n.\vert \Sigma \vert) \) and maximum iteration number is \( O(n^4.\vert \Sigma \vert) \).
Comparison between channel and agent markers

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

\[ \text{Id}^2 = (\Sigma 
\rightarrow (\mathbb{Z} \cup \{T\})) \cup \{\bot\} \]

\(\gamma_{\text{Id}^2}\) is defined as follows:

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

Domain complexity is \(O(|\Sigma|)\) and maximum iteration number is \(O(n^3.|\Sigma|).\)
Several trade-offs

1. 0-cfa (0-CFA): \( ld^\# = \{ \bot; \top \} \),
   Cf [Nielson et al.:CONCUR’98], [Hennessy and Riely:HLCL’98].
2. Confinement (CONF): \( ld^\# = \{ \bot, =, \top \} \),
   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

\[ A \rightarrow B \]

means that there exists \( \alpha : A \rightarrow B \), such that for any system \( S \),

\[ \alpha(\llbracket S \rrbracket_A^\#) \preceq_B \llbracket S \rrbracket_B^\#. \]
Example: 0-CFA

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

\[
\left( 2, (3, 6)(3, 5)^n (1, 4), \begin{cases} 
\text{add} & \mapsto (email, (3, 6)(3, 5)^n) \\
\text{info} & \mapsto (data, (3, 6)(3, 5)^n) 
\end{cases} \right)
\]
Example: non-uniform result

( *port?^[info,add](add!^[info])
| *gen?^[#/data](#/email)(port!^[data,email] | gen!^[[])
| gen!^[[] )

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)
Example: the ring of processes

\[
(\nu \text{ make})(\nu \text{ edge})(\nu \text{ first}) \\
(*\text{make}^1[\text{last}](\nu \text{next}) \\
(\text{edge}^2[\text{last},\text{next}] \\
| \text{make}^3[\text{next}]) \\
| *\text{make}^4[\text{last}](\text{edge}^5[\text{last},\text{first}]) \\
| \text{make}^6[\text{first}])
\]

\[\#(1, 3) + 1 = \#(1, 3)\]
Example: Algebraic properties

```
((# make)(# mon)(# left0)
 ((# make?1[left])(# right)(mon?2[left,right]make?3[right]))
 ((# make?6[left])(mon?5[left,left0]))
 (make?6[left0])))
```

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

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

(1,6) = (1,6)
(1,3) = (1,3) + 1

main menu – control flow analysis – (# right)
We detect that:

\[
\begin{cases}
(p^{12}[\bullet], (11, 20)^m.(11, 21), _, [p \mapsto (p, (11, 20)^m.(11, 21))]) \\
(answer^8[\bullet], (3, 19).(11, 20)^n.(11, 21), (12, (11, 20)^n.(11, 21), _)
\end{cases}
\]

\[
\begin{cases}
(\langle rep\rangle^9, _, (8, (3, 19).(11, 20)^p.(11, 21), [rep \mapsto (data, (11, 20)^p.(11, 21))])
\end{cases}
\]

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

\[
(p.(11, 20)^n.(11, 21))
\]

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

\[
(\nu \text{ make})(\nu \text{ edge})(\nu \text{ first})(\nu \text{ first})
\]

\[
(*\text{make}\!^1[last])(\nu \text{next})
\]

\[
(\text{edge}\!^2[last,next])
\]

\[
| (*\text{make}\!^6[last](\text{edge}\!^7[last,first])
\]

\[
| \text{make}\!^8[\text{first}]
\]

\[
| \text{edge}?[x,y][x =^9 y][x \neq^{10}\text{first}]\text{Ok}\!^{11}[]
\]

we cannot infer that 11 is unreachable.
Dependency analysis between names

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

\[ G_V = \left\{ (A, R) \mid A \text{ is a partition of } V \right\} \]

\[ R \text{ is a symmetric anti-reflexive relation on } A \]

\( G_V \) is related to \( \wp(\text{Id} \times (V \rightarrow (\text{Name} \times \text{Id}))) \) by the following concretization function:

\[ \gamma_V((A, R)) = \left\{ (\text{id}, E) \mid \forall X \in A, \{x, y\} \subseteq X \implies E(x) = E(y) \right\} \]

\[ (X, Y) \in R \implies \forall x \in X, y \in Y, E(x) \neq E(y) \]

\[ \implies \text{implicit closure of relations and information propagation.} \]


**Dependency analysis between markers**

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

\[
G_V = \left\{ (A, R) \mid \begin{array}{l}
A \text{ is a partition of } V \cup \{id_p\} \\
R \text{ is a symmetric anti-reflexive relation on } A
\end{array} \right\}.
\]

\(G_V\) is related to \(\wp(Id \times (V \to (\text{Name} \times Id)))\) by the following concretization function:

\[
\gamma_V((A, R)) = \left\{ (id, E) \mid \begin{array}{l}
\forall \mathcal{X} \in A,\ x \in V,\ \{id_p, x\} \subseteq \mathcal{X} \implies id = \text{snd}(E(x)) \\
\forall \mathcal{X} \in A,\ x, y \in V,\ \{x, y\} \subseteq \mathcal{X} \implies \text{snd}(E(x)) = \text{snd}(E(y)) \\
\forall (\mathcal{X}, \mathcal{Y}) \in R,\ y \in V,\id_p \in \mathcal{X} \text{ and } y \in \mathcal{Y} \implies id \neq \text{snd}(E(y)) \\
\forall (\mathcal{X}, \mathcal{Y}) \in R,\ x, y \in V,\ x \in \mathcal{X} \text{ and } y \in \mathcal{Y} \implies \text{snd}(E(x)) \neq \text{snd}(E(y))
\end{array} \right\}
\]

\(\implies\) 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 $V \subseteq Name$, we introduce the set

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

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

$$\gamma_V(\mathcal{K}) = \left\{ (id, E) \mid \begin{pmatrix} p^\lambda \rightarrow |id|_\lambda \\ x(y,v) \rightarrow (y = first(E(v))) \\ x(\lambda,v) \rightarrow |snd(E(v))|_\lambda \end{pmatrix} \text{ satisfies } \mathcal{K} \right\}.$$
Pair-wise numerical analysis

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

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

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

$$
\begin{align*}
  \left\{ \\
  (id, E) & : E(x) = (y, id_y) \implies (\Phi((|id|_\lambda)_\lambda \in \Sigma), \Phi((|id_y|_\lambda)_\lambda \in \Sigma)) \in f(x, y) \\
  E(x) = (y, id_y) & \implies (\Phi((|id_y|_\lambda)_\lambda \in \Sigma), \Phi((|id'_y|_\lambda)_\lambda \in \Sigma)) \in g((x, y), (x', y')) \\
  E(x') = (y', id'_y) &
\end{align*}
$$
Reduction
Example

$$(\nu \text{ make})(\nu \text{ edge})(\nu \text{ first})$$

$$(\ast \text{ make}^1[\text{last}](\nu \text{ next}) \ (\text{edge}^2[\text{last,next}] \ | \ \text{make}^3[\text{next}]))$$

$$| \ast \text{ make}^6[\text{last}](\text{edge}^7[\text{last,first}])$$

$$| \ \text{make}^8[\text{first}]$$

$$| \ \text{edge}?[x,y][x =^9 y][x \neq^{10} \text{first}] \text{Ok}^1[\]$$

we first prove in global abstraction that:

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

$$f(7) \text{ satisfies } \begin{cases} c^{\text{next, last}} + c^{\text{first, last}} = 1 \\ c^{\text{first, first}} = 1 \end{cases}$$
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. \((\nu \text{ first})\) and \((\nu \text{ first})\),
2. \((\nu \text{ first})\) and \((\nu \text{ next})\),
3. \((\nu \text{ next})\) and \((\nu \text{ next})\) but distinct instances,
4. \((\nu \text{ next})\) and \((\nu \text{ first})\).

So, the matching pattern \([x = y]\) is satisfiable only in the first case !!!
Overview

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

\[
\begin{align*}
\left(1, \varepsilon, \{ \text{port} \mapsto (\text{port}, \varepsilon) \} \right) \\
\left(1, \varepsilon, \{ \text{gen} \mapsto (\text{gen}, \varepsilon) \} \right) \\
\left(3, \varepsilon, \{ \text{port} \mapsto (\text{port}, \varepsilon) \} \right) \\
\left(2, \text{id}_1', \{ \text{add} \mapsto (\text{email}, \text{id}_1), \text{info} \mapsto (\text{data}, \text{id}_1) \} \right) \\
\left(2, \text{id}_2', \{ \text{add} \mapsto (\text{email}, \text{id}_2), \text{info} \mapsto (\text{data}, \text{id}_2) \} \right) \\
\left(5, \text{id}_2', \{ \text{gen} \mapsto (\text{gen}, \varepsilon) \} \right)
\end{align*}
\]
Abstract transition

\[ C^\# \overset{(i,j)}{\sim} J \]

\[ C'^\# \]
Abstract domains

We design a domain for representing numerical constraints between
- the number of occurrences of processes \( \#(i) \);
- the number of performed transitions \( \#(i,j) \).

We use the product of
- a non-relational domain:
  \( \implies \) the interval lattice;
- a relational domain:
  \( \implies \) the lattice of affine relationships.
Interval narrowing

An exact reduction is exponential.
We use:

- **Gaus reduction:**
  \[
  \begin{align*}
  x + y + z &= 1 \\
  x + y + t &= 2 \\
  \end{align*}
  \Rightarrow
  \begin{align*}
  x + y + z &= 1 \\
  t - z &= 1 \\
  \end{align*}
  \]

- **Interval propagation:**
  \[
  \begin{align*}
  x + y + z &= 3 \\
  x \in \mathbb{R}_+ \cup \{\infty\} &
  \Rightarrow
  \begin{align*}
  x + y + z &= 3 \\
  x \in \mathbb{R}_+ \cup \{3\} \\
  \end{align*}
  \]

- **Redundancy introduction:**
  \[
  \begin{align*}
  x + y - z &= 3 \\
  x \in \mathbb{R}_+ \cup \{1; 2\} &
  \Rightarrow
  \begin{align*}
  x + y - z &= 3 \\
  y - z \in \{1; 2\} \\
  x \in \{1; 2\} \\
  \end{align*}
  \]

to get a cubic approximated reduction.
Example: non-exhaustion of resources
Example: exhaustion of resources
Example: mutual exclusion

\[(# a)(# b)(# c)
(a ? [0; 1] b ? [0; 1] a ? [0; 1] b ? [0; 1] a ? [0; 1])
\]

main menu
Example: token ring

\[
\begin{align*}
(# \text{make})(# \text{mon})(# \text{left0})
\quad &\left( (*\text{make}\?^{1:1}[\text{left}](# \text{right})(\text{mon}!^{2:\{0;+\infty\}}[\text{left},\text{right}] \mid \text{make}!^{3:\{0;1\}}[\text{right}])
\quad \mid (*\text{make}\?^{4:1}[\text{left}](\text{mon}!^{5:\{0;1\}}[\text{left},\text{left0}])
\quad \mid \text{make}!^{6:\{0;1\}}[\text{left0}]
\quad \mid (*\text{mon}\?^{7:1}[\text{prev},\text{next}]
\quad \left( (*\text{prev}\?^{8:\{0;+\infty\}}[])(# \text{crit}
\quad \left( \text{crit}!^{9:\{0;1\}}[] \mid (*\text{crit}\?^{10:\{0;1\}}[]\text{next}!^{11:\{0;1\}}[]))
\quad \mid \text{left0}!^{12:\{0;1\}}[])\right)\right)\right)\right)\right)
\end{align*}
\]
Comparison

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

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

- Abstract multisets.

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

1. Overview
2. Mobile systems
3. Non standards semantics
4. Abstract Interpretation
5. Environment analyses
6. Occurrence counting analysis
7. Thread partitioning
8. Conclusion
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 $\pi$-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 indices.
We define the set of computation units as:

$$unit \triangleq B \rightarrow Label \times Id.$$ 

give-index maps each program point $p$ to a function $give-index(p) \in B \rightarrow fn(p)$.

Given a thread $t = (p, id, E)$, we define its computation unit $give-unit(t)$ as:

$$give-unit(t) = [b \in B \rightarrow E(give-index(p)(b))].$$
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:

\[
\text{UNIT}^\# \triangleq B \rightarrow \text{Label}.
\]

The abstraction function:

\[
\Pi_{\text{unit}} \in \begin{cases} 
\text{unit} \\
[b \in B \mapsto (l_b, \_)] \\
\end{cases} \rightarrow \text{UNIT}^\# \\
[b \mapsto l_b];
\]

maps each computation unit to an abstract one.
Abstract domain

Our main domain is a Cartesian product:

\[ C^\#_{part} \triangleq (\prod_{p \in L} G_{fn(p)}) \times (\text{UNIT}^\# \rightarrow N_{L_p}). \]

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

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

\[ t \in \{(0) \in N^{L_p} \} \cup (\gamma_{N_{L_p}}(\text{CU}(\Pi_{\text{unit}(u)}))) \]

such that:

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

System := (ν create)(ν null)(*create?[d].Allocate(d))

Allocate(d) :=
(ν cell)(ν write)(ν read)
  init(cell) | read(read,cell) | write(write,cell) | d![read;write]

where

- init(cell) := cell![null]
- read(read,cell) := *read?[port].cell?[u](cell![u] | port![u])
- write(write,cell) := *write?[data,ack].cell?[u].(cell![data] | 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 \((\nu \text{ cell})\) restriction.
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 standards 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 $\pi$),
  $\implies$ analyzing cryptographic protocols with $\oplus$;
- higher order communication;
  $\implies$ agents may communicate running programs;
  $\implies$ agents may duplicate running programs;
- Using our framework to describe and analyze mobility in industrial applications ($\texttt{ERLANG}$).
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.

$\implies$ a reachability analysis is useless.

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

  $\implies$ (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.