Soutenance de thèse

Analysis of Mobile Systems by Abstract Interpretation

Jérôme Feret
École Normale Supérieure
http://www.di.ens.fr/~feret

February 2005

Introduction I

We propose a unifying framework to design

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

semantics to abstract the properties of mobile systems.

Our framework is model-independent:

$\Longrightarrow$ we use a META-language to encode mobility models,
$\Longrightarrow$ we design analyses at the META-language level.

We use the Abstract Interpretation theory.

Introduction II

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

We design three families of analyses:

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

Overview

1. Mobile systems
2. The π-calculus
3. Abstract Interpretation
4. Environment analyses
5. Occurrence counting analysis
6. Thread partitioning
7. Conclusion

Mobile system

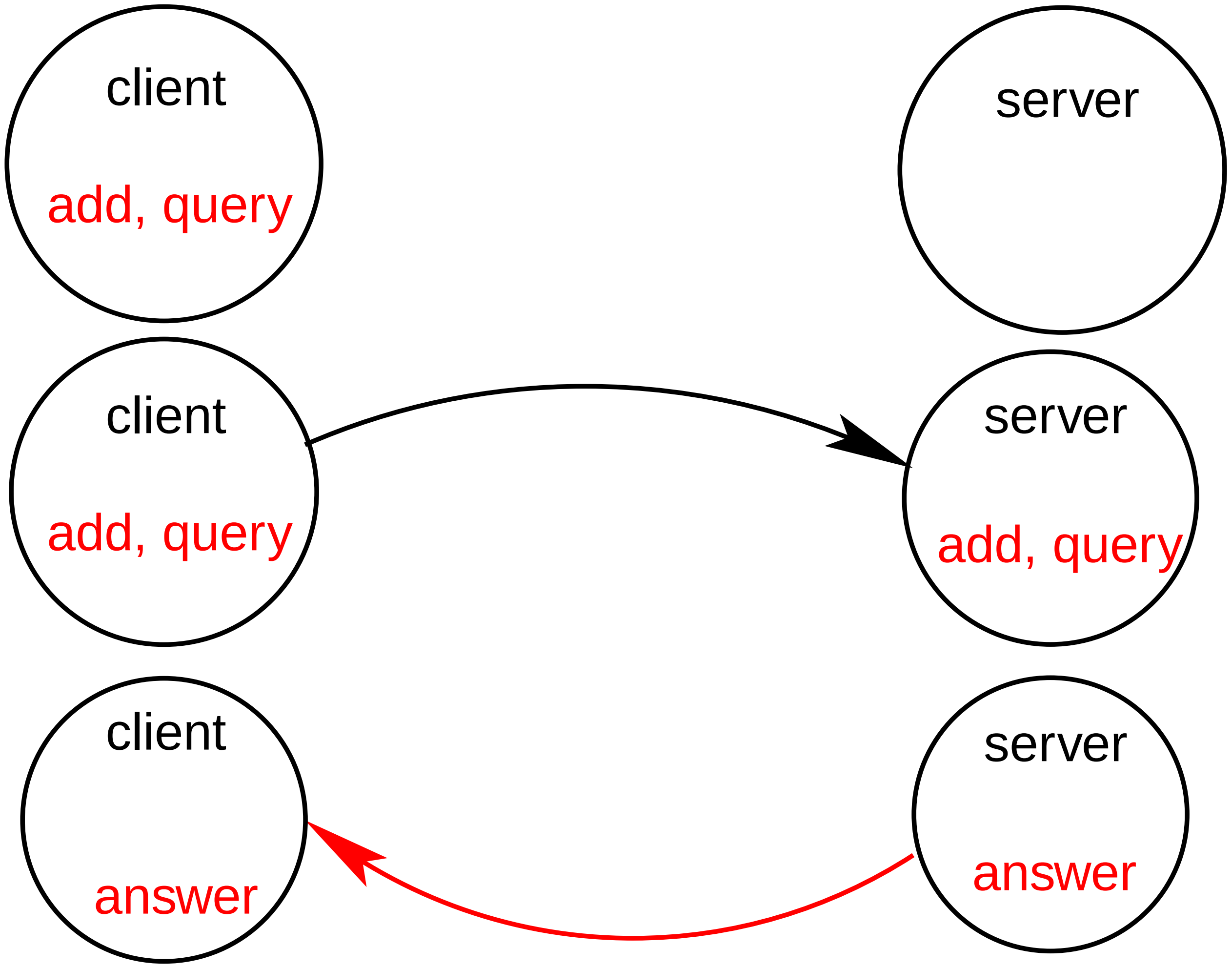
A pool of processes which interact and communicate:

Interactions control:

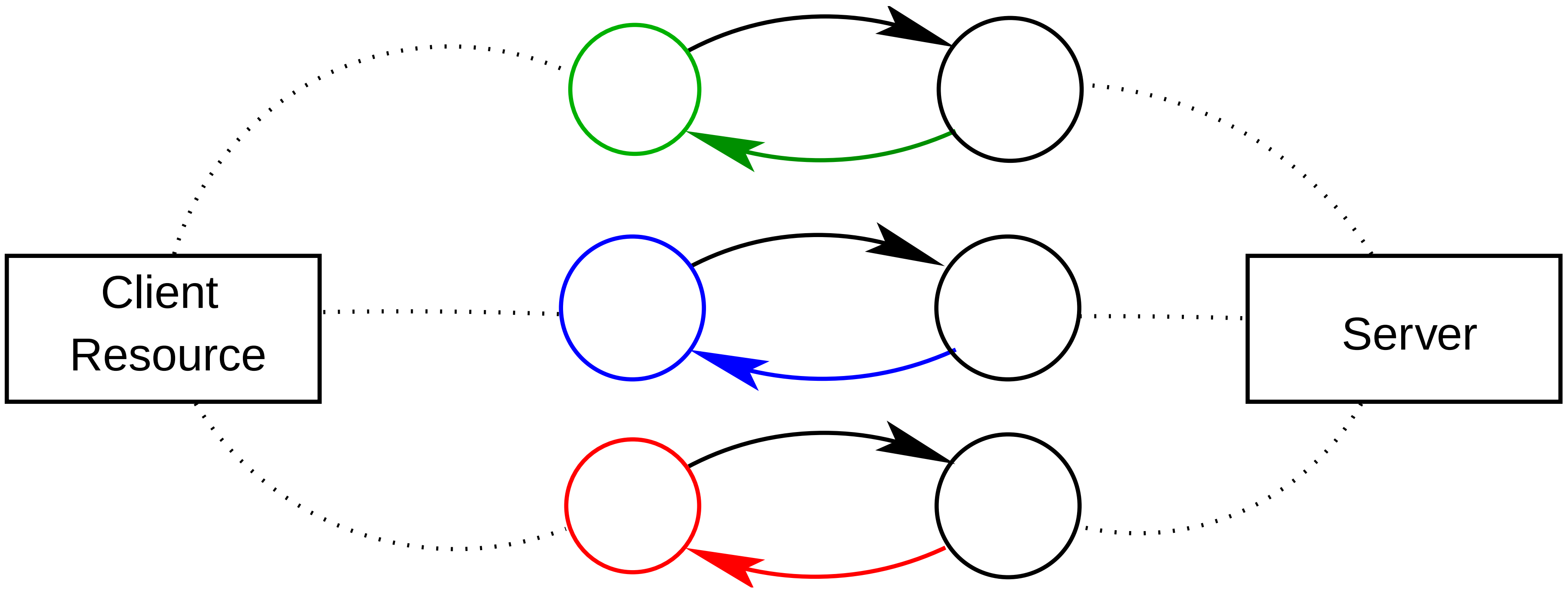
- process synchronization;
- update of link between processes (communication, migration);
- process creation.

The number of processes may be unbounded !

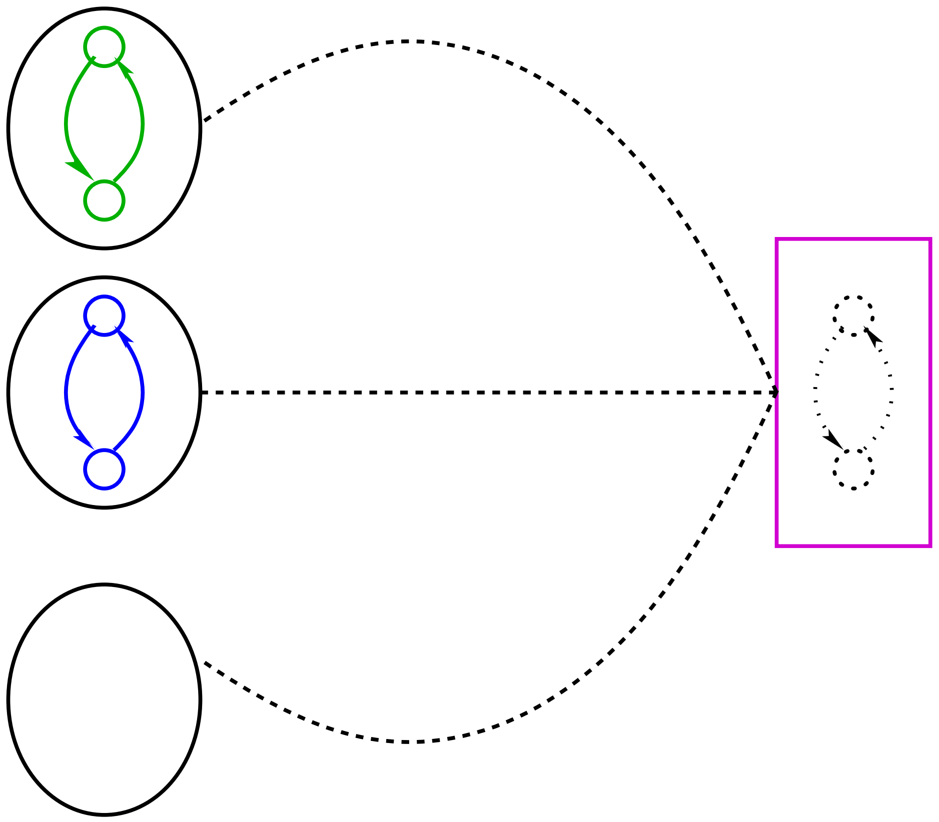
A connection



A network



Example: a 3-port server



Example: a shared-memory

We consider the implementation of a shared-memory where:

- agents may allocate new cells,
- authorized agents may read the content of a cell,
- authorized agents may write inside a cell, overwriting the former content.

The content of a cell is encoded by an output on a channel. We want to prove that the value of each cell never becomes ambiguous (i.e. there never are two outputs over the same channel).

Toward a unifying framework

Several models depending on the application field:

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

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

Overview

π-calculus: syntax

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

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

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

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

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

Transition semantics

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

- the reduction relation specifies the result of computations:

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

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

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

Congruence relation

- the congruence relation reveals redexes:

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

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

Example: syntax

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

where

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

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

Example: computation

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

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

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

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

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

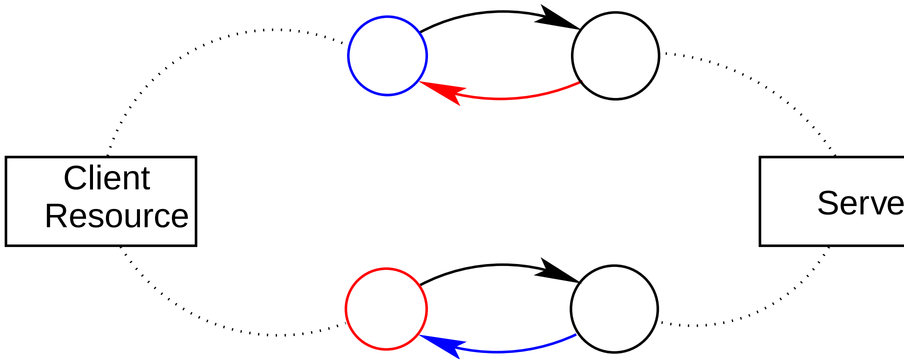
α-conversion

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

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

$\sim_\alpha$

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



Non-standard semantics

A refined semantics where:

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

Example: non-standard configuration

$(\textbf{Server} \mid \textbf{Client} \mid \mathrm{gen}!^5[] \mid email_1!^2[data_1] \mid email_2!^2[data_2])$

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

Marker properties

1. Marker allocation must be consistent:

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

2. Marker allocation should be robust:

   Marker allocation should not depend on the interleaving order.

Extraction function

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

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

Transition system

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

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

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

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

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 contain useful information:

   This allows the inference of:

   - more complex properties;
   - some simple properties the proof of which uses complex properties.

Overview

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 configurations which satisfy the property A.

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:

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

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

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

Approximate reduced product

The Abstract Interpretation framework provides tools for making the product of several abstractions.

Abstract properties may refine each other to get better results.

An abstract computation step is enabled if and only if it is enabled in all abstractions.

Overview

Generic environment analysis

$\Longrightarrow$ Abstract the relations among the marker and the names of threads at each program point.

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:

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

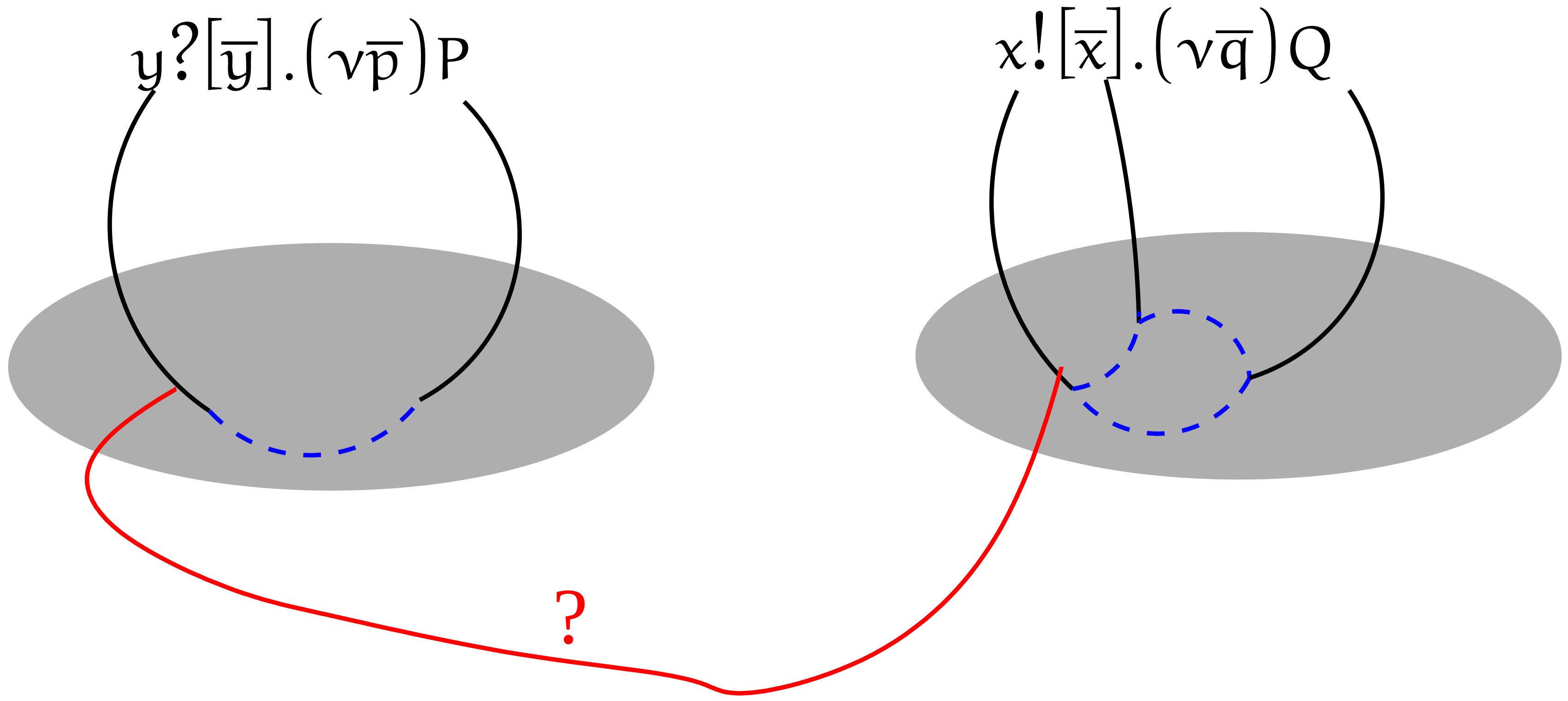
The abstract domain $C^\sharp$ is then the set:

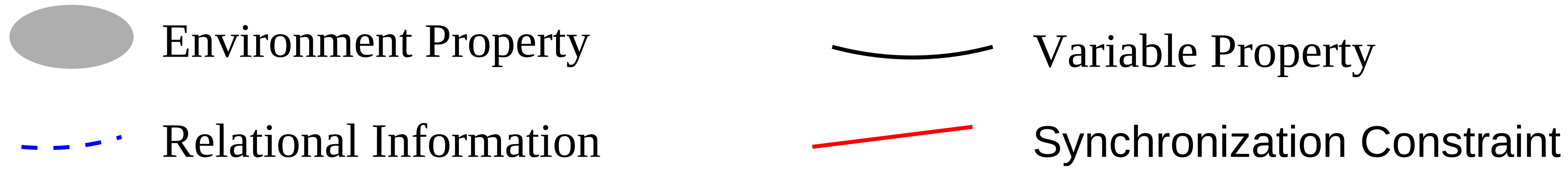
$$C^\sharp = \prod_{p \in \mathcal{P}} \mathcal{G}_{fv(p)}$$

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

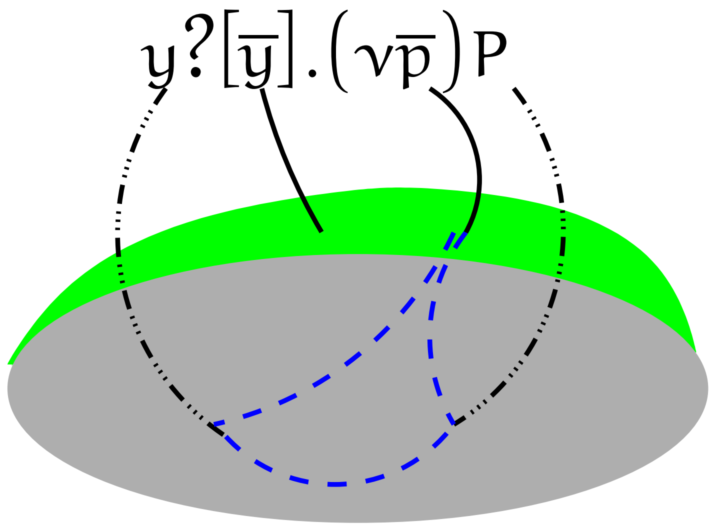
$$\gamma(f) = \{C \mid (p, \mathit{id}, E) \in C \implies (\mathit{id}, E) \in \gamma_V(f_p)\}.$$

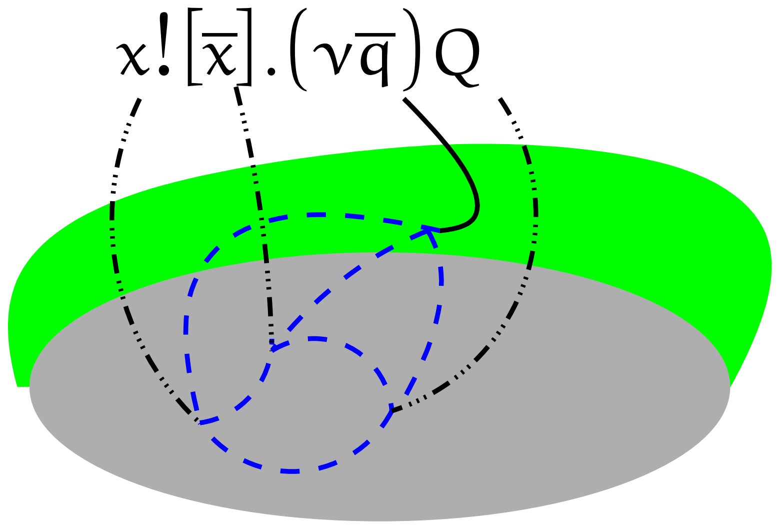
Abstract communication


$y?[\overline{y}].(\nu\overline{p})P$
$x![\overline{x}].(\nu\overline{q})Q$
?



Extending environments





⬤ Environment Property

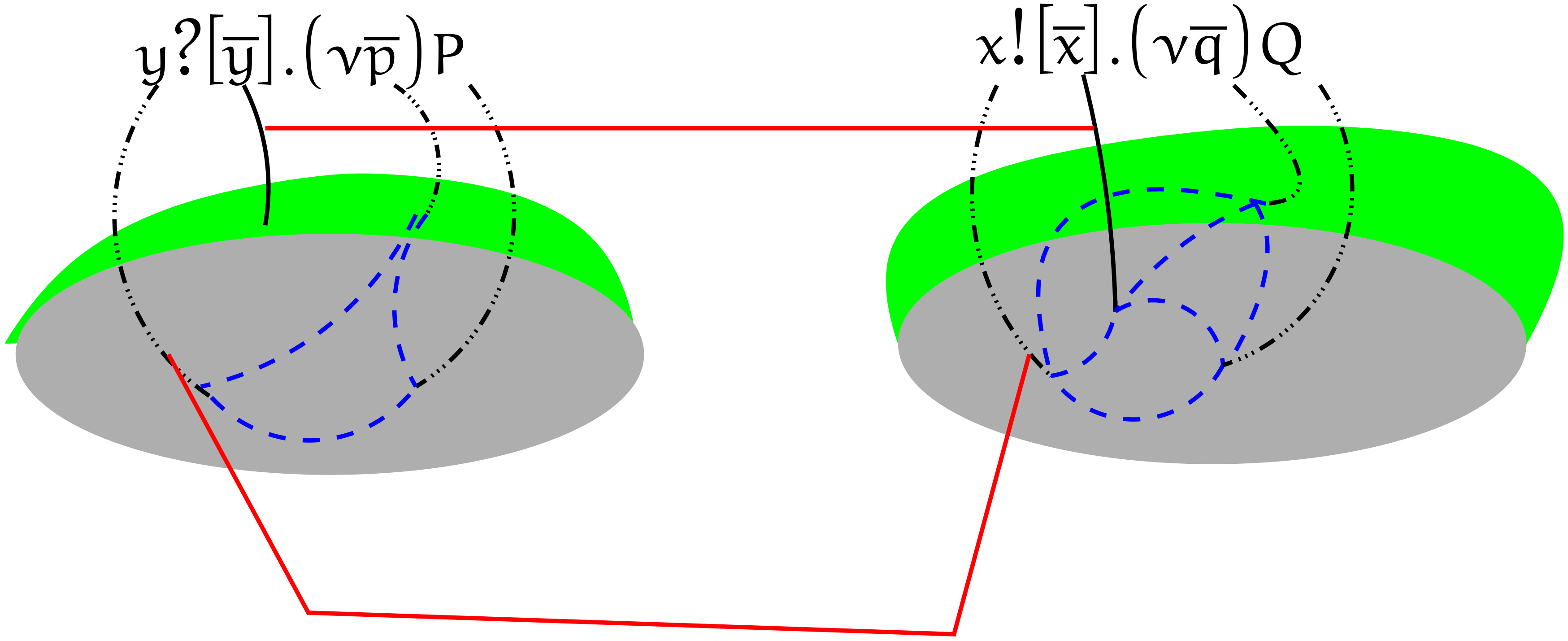
🟩 Environment Extension

╌ Relational Information

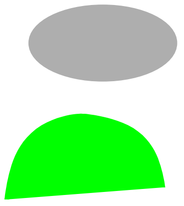
— Variable Property

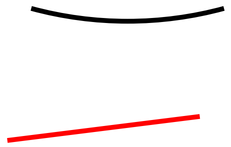
— Synchronization Constraint

Synchronizing environments

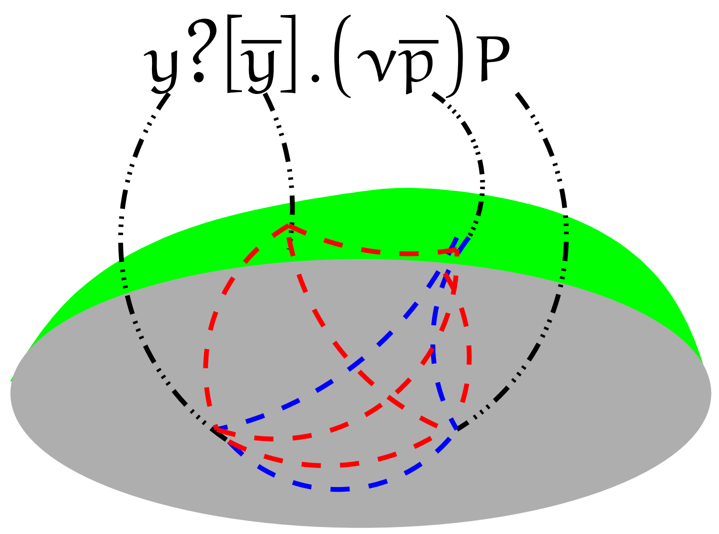


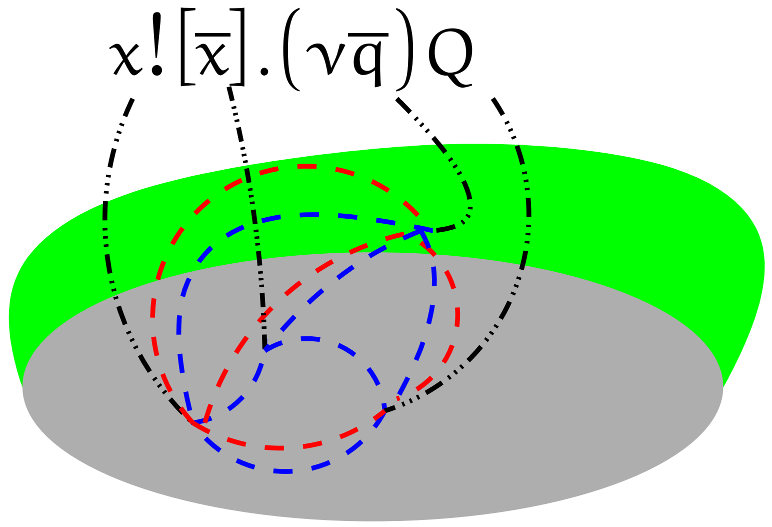
$y?[\overline{y}].(\nu\overline{p})P$
$x![\overline{x}].(\nu\overline{q})Q$





Propagating information





⬮	Environment Property
🟩	Environment Extension
- - -	Relational Information
—	Variable Property
- - -	Information closure

Generic primitives

We only require abstract primitives to:

1. extend the domain of the environments,
2. gather the description of the linkage of the two syntactic agents,
3. synchronize variables,
4. compute information closure,
5. separate the two descriptions,
6. restrict the domain of the environments.

Control flow analysis

Detect the origin of the channels that are communicated to variables.
Abstract relationship between the history of threads that open channels and the history of threads that receive these channels.

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

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

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

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

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

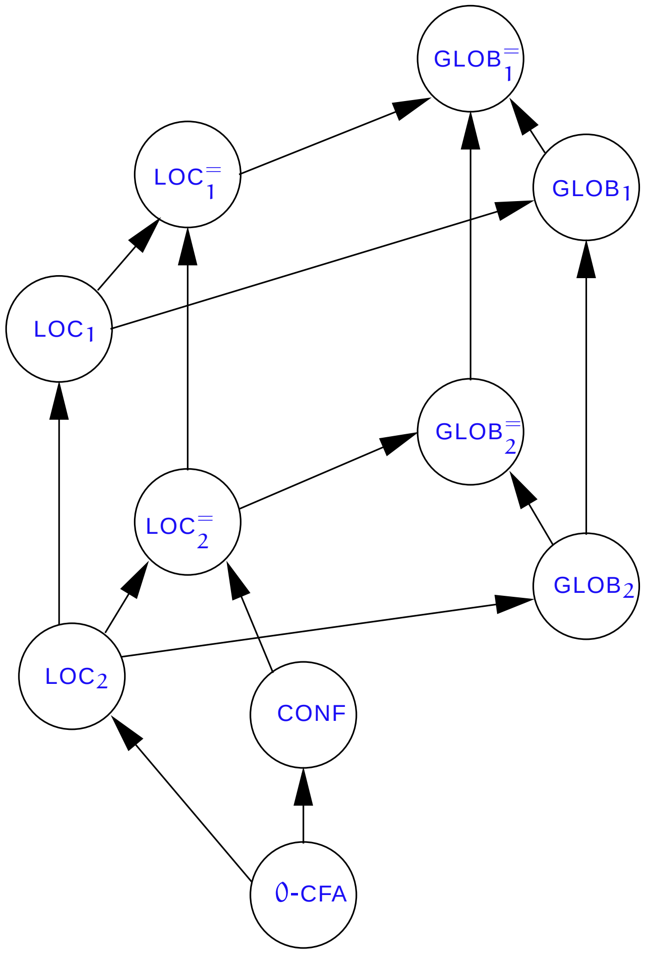
Several trade-offs

1. 0-cfa (0-CFA): *Id* $^\sharp = \{\bot, \top\}$,
   [Nielson *et al.*:CONCUR'98], [Hennessy and Riely:HLCL'98].
2. Confinement (CONF): *Id* $^\sharp = \{\bot, =, \top\}$,
   [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$_i$), where $i \in \{1;2\}$,
   - by choosing the set of literals among *Label* ($i = 2$) or *Label* ^{2} ($i = 1$).

Abstract semantics hierarchy



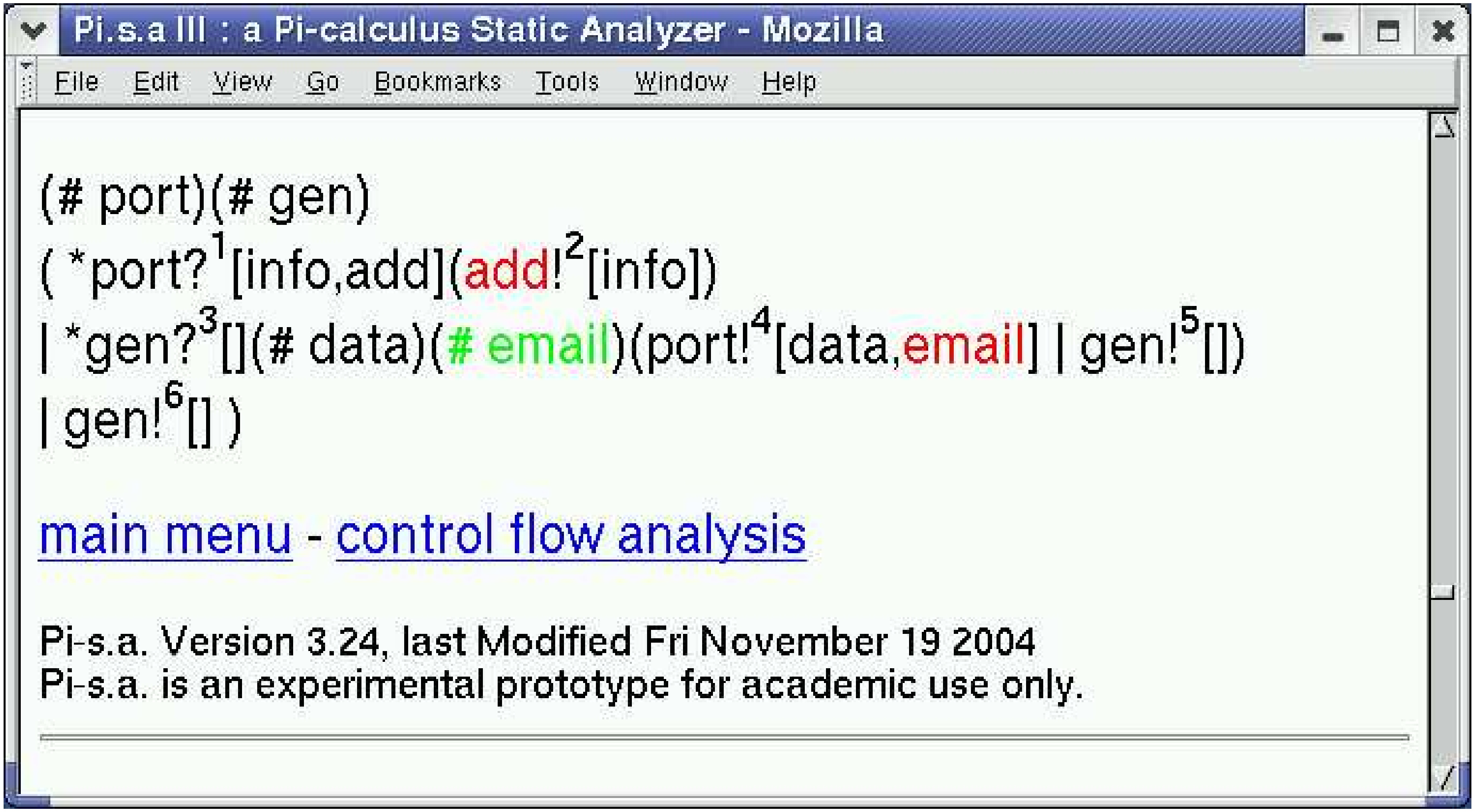
where

$$A \longrightarrow B$$

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

$$\alpha(\llbracket S \rrbracket^\sharp_B) \sqsubseteq_A \llbracket S \rrbracket^\sharp_A.$$

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

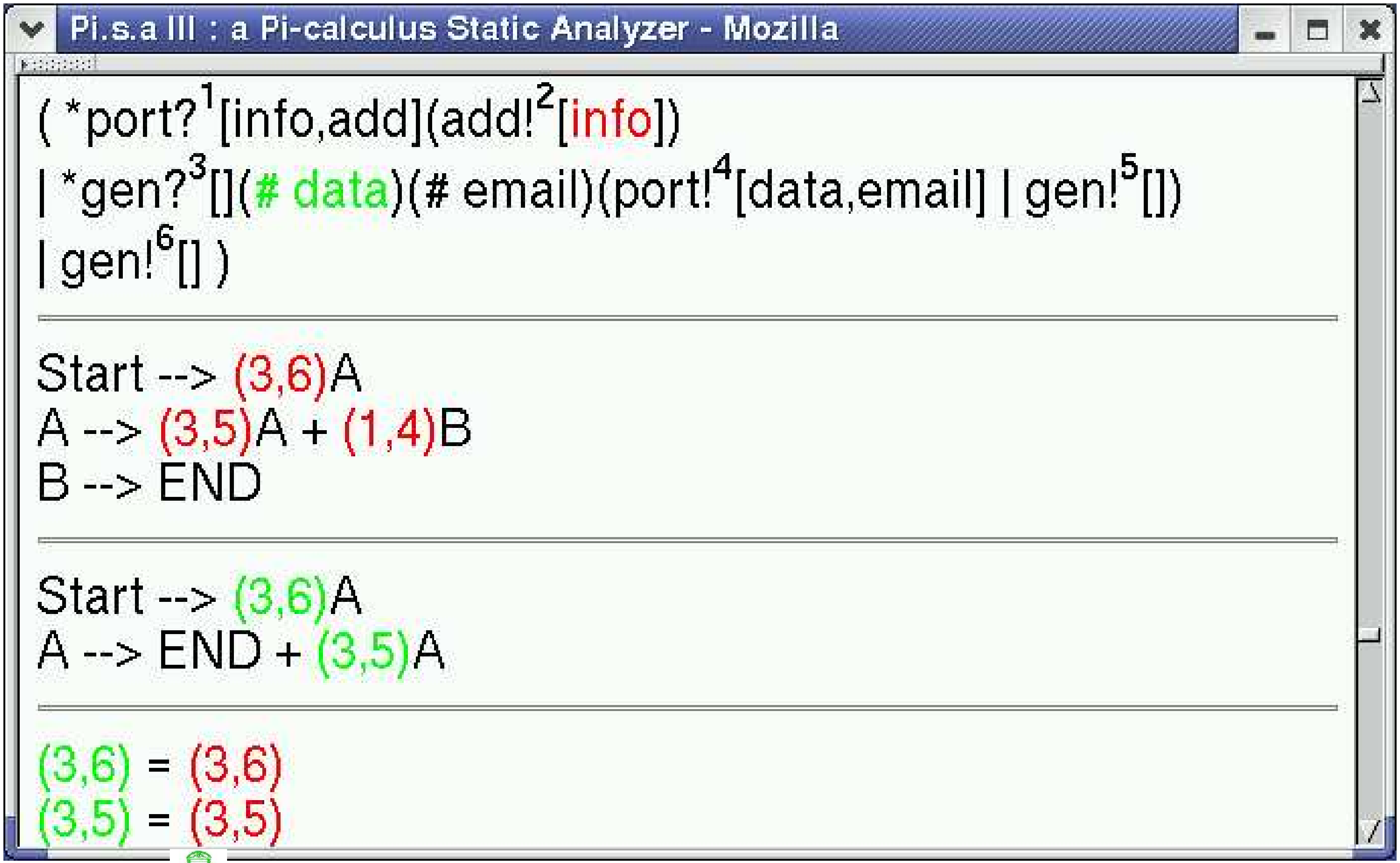
Pi-s.a. Version 3.24, last Modified Fri November 19 2004
Pi-s.a. is an experimental prototype for academic use only.

Non uniform property

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

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

Example: non-uniform result



```
( *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
```

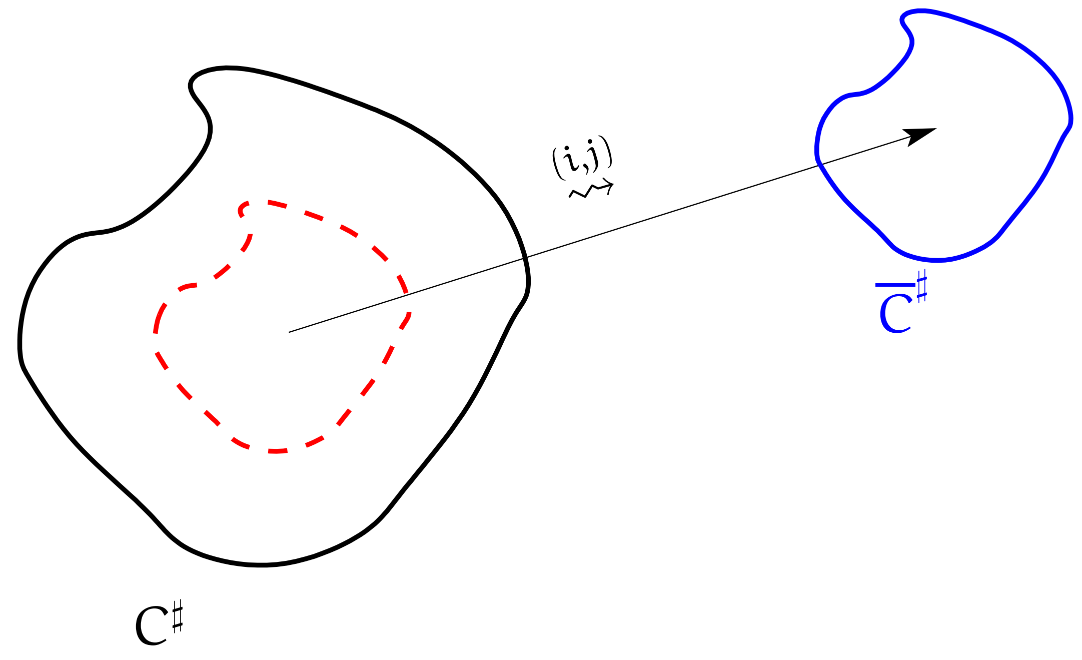
```
(3,6) = (3,6)
(3,5) = (3,5)
```

Overview

Occurrences counting analysis

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

Abstract transition



Abstract domains

We design a domain for representing numerical constraints between

- the number of occurrences of threads $\sharp(i)$;
- 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:

- Gaussian Elimination:

$$\begin{cases} x+y+z=1 \\ x+y+t=2 \end{cases} \Longrightarrow \begin{cases} x+y+z=1 \\ t-z=1 \end{cases}$$

- Interval propagation:

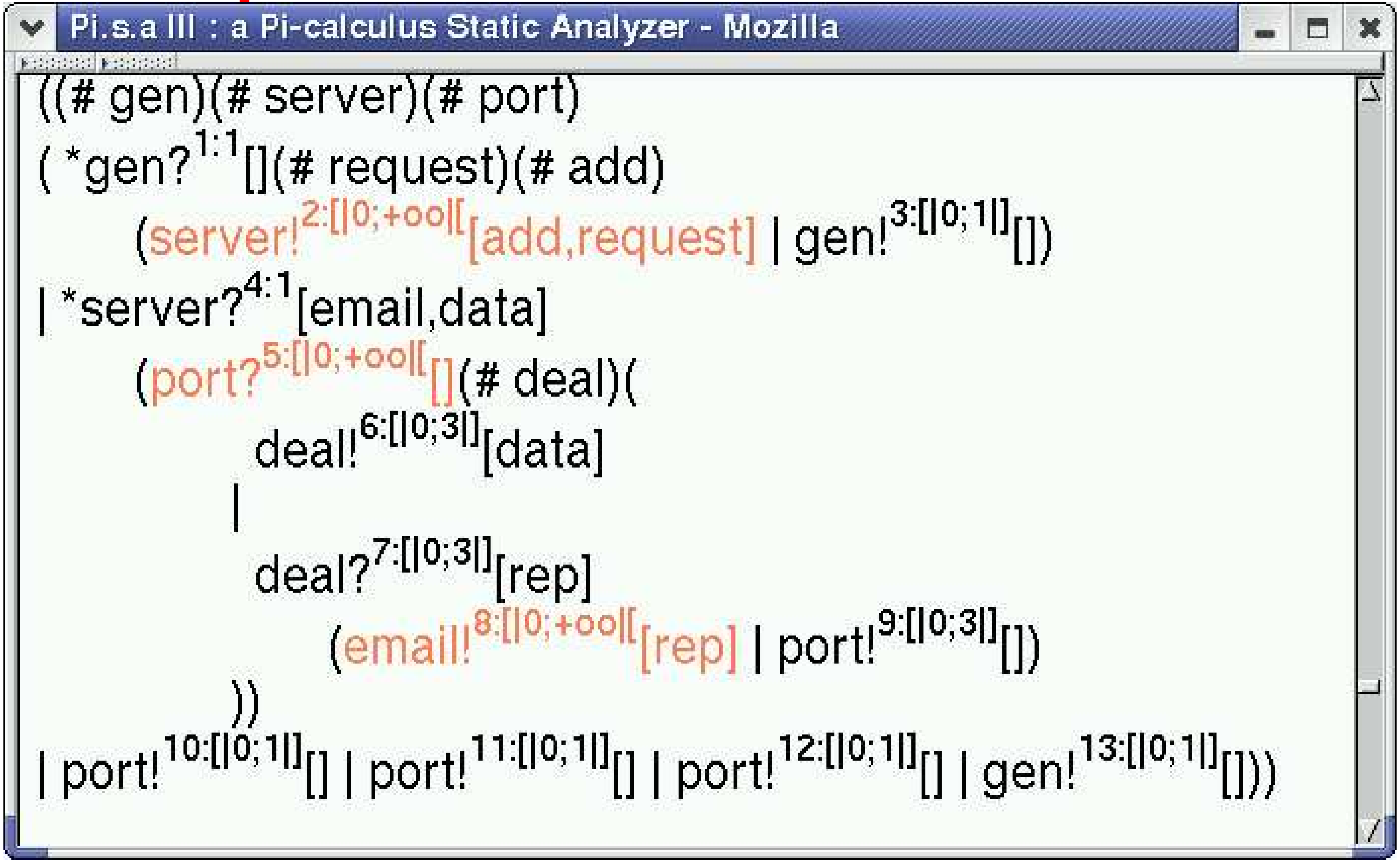
$$\begin{cases} x+y+z=3 \\ x \in [|0;\infty|[\\ y \in [|0;\infty|[\\ z \in [|0;\infty|[\end{cases} \Longrightarrow \begin{cases} x+y+z=3 \\ x \in [|0;3|] \\ y \in [|0;\infty|[\\ z \in [|0;\infty|[\end{cases}$$

- 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



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

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

Related works

- 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

Computation unit

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

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.

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

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

Allocate(d) :=

$\quad$ (ν *cell*)(ν *write*)(ν *read*)

$\quad\quad$ 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 $(\nu\ cell)$ 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

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

- symmetric communication (fusion calculus),
  $\Longrightarrow$ theoretical problem;
- term defined up to an equational theory (applied pi),
  $\Longrightarrow$ analyzing cryptographic protocols with XOR;
- higher order communication;
  $\Longrightarrow$ agents may communicate running programs;
  $\Longrightarrow$ agents may duplicate running programs;
- 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.

THANK YOU FOR YOUR ATTENTION !!!