Groupe de travail

Analysis of Mobile Systems by Abstract Interpretation

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

16/06/2005

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, . . .).

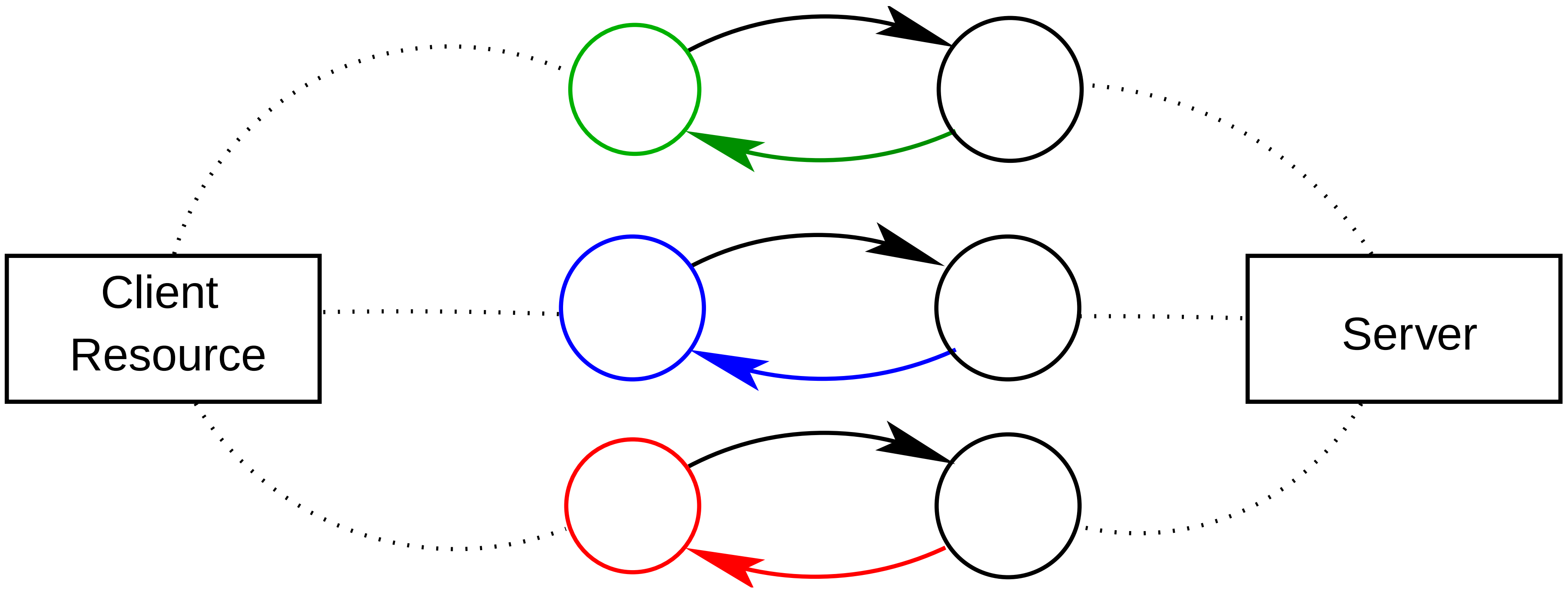
Example: non-standard configuration

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

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

Overview

A network



Generic environment analysis

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

For any finite subset $V \subseteq \mathcal{V}$,

$$\wp(\mathit{Id} \times (V \to (\mathit{Label} \times \mathit{Id}))) \xleftarrow{\gamma_V} \mathit{Atom}^{\sharp}_{V}.$$

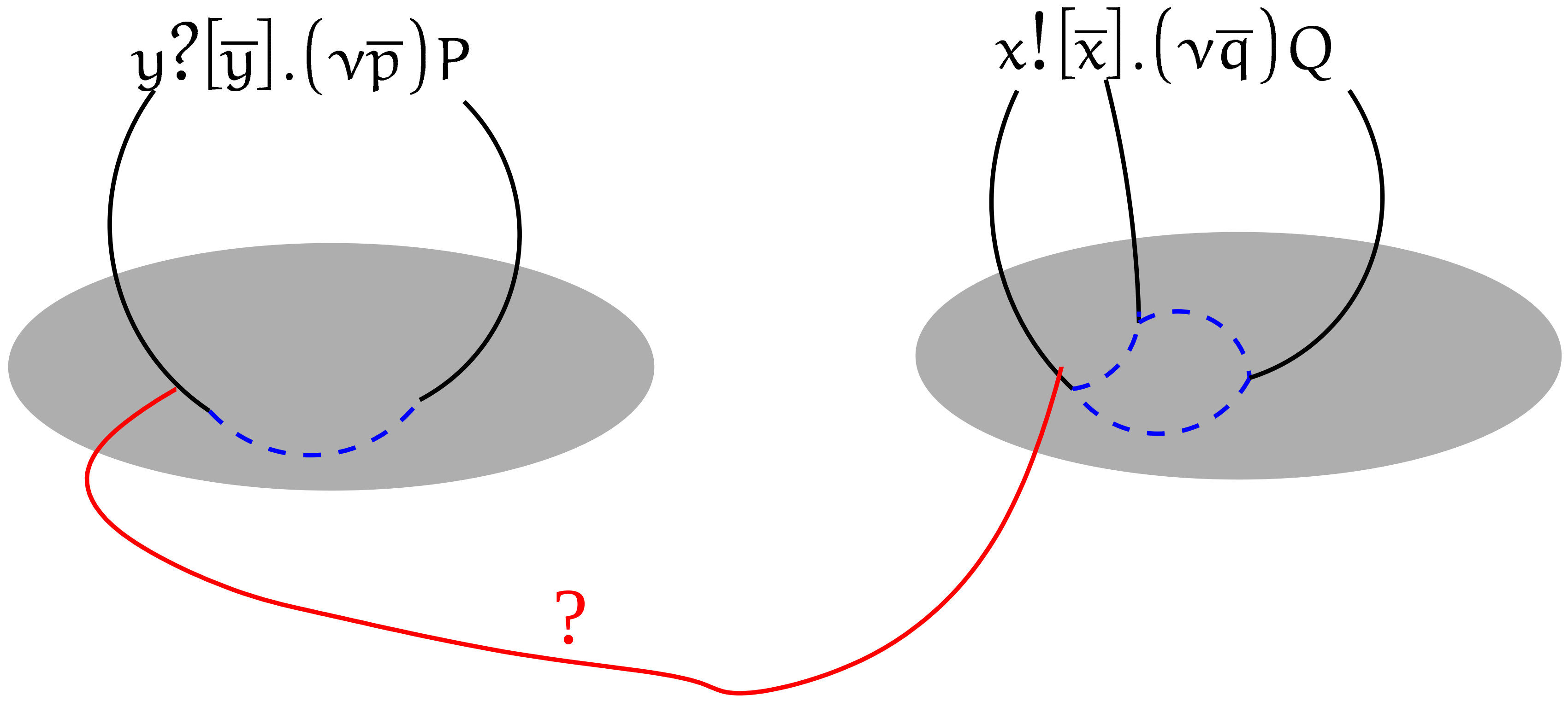
The abstract domain $C^{\sharp}$ is then the set:

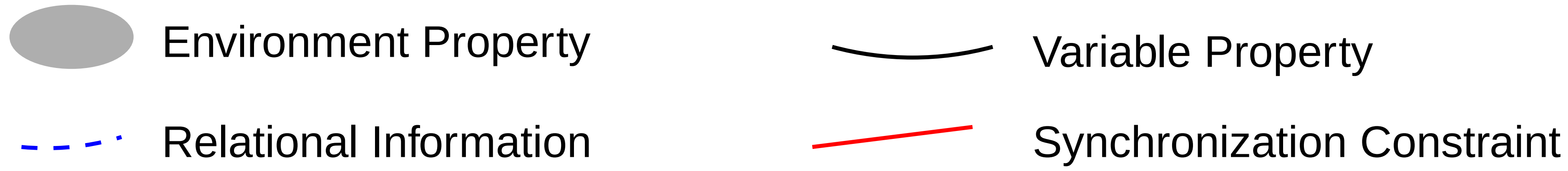
$$C^{\sharp} = \prod_{p \in \mathcal{P}} \mathit{Atom}^{\sharp}_{I(p)}$$

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

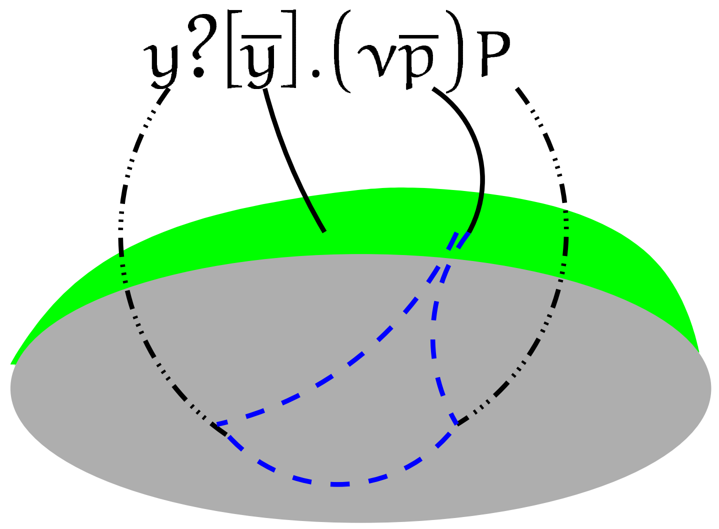
$$\gamma(f) = \{C \mid (p, \mathit{id}, E) \in C \implies (\mathit{id}, E) \in \gamma_{I(p)}(f_p)\}.$$

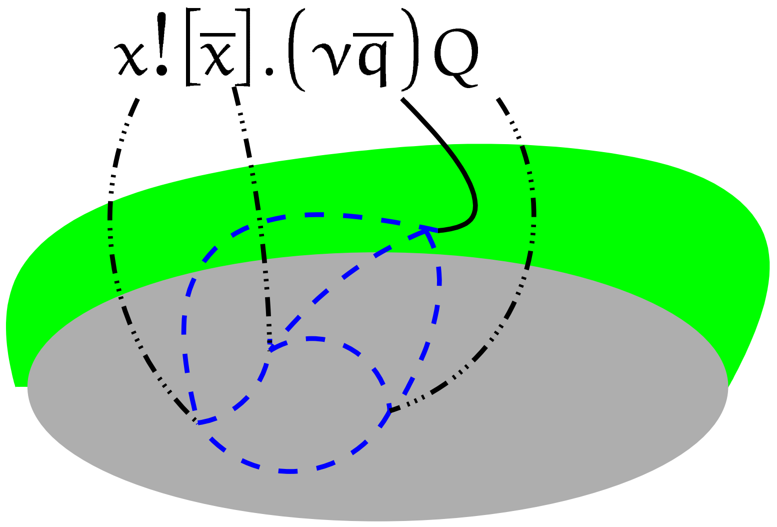
Abstract communication


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



Extending environments





⬬ Environment Property

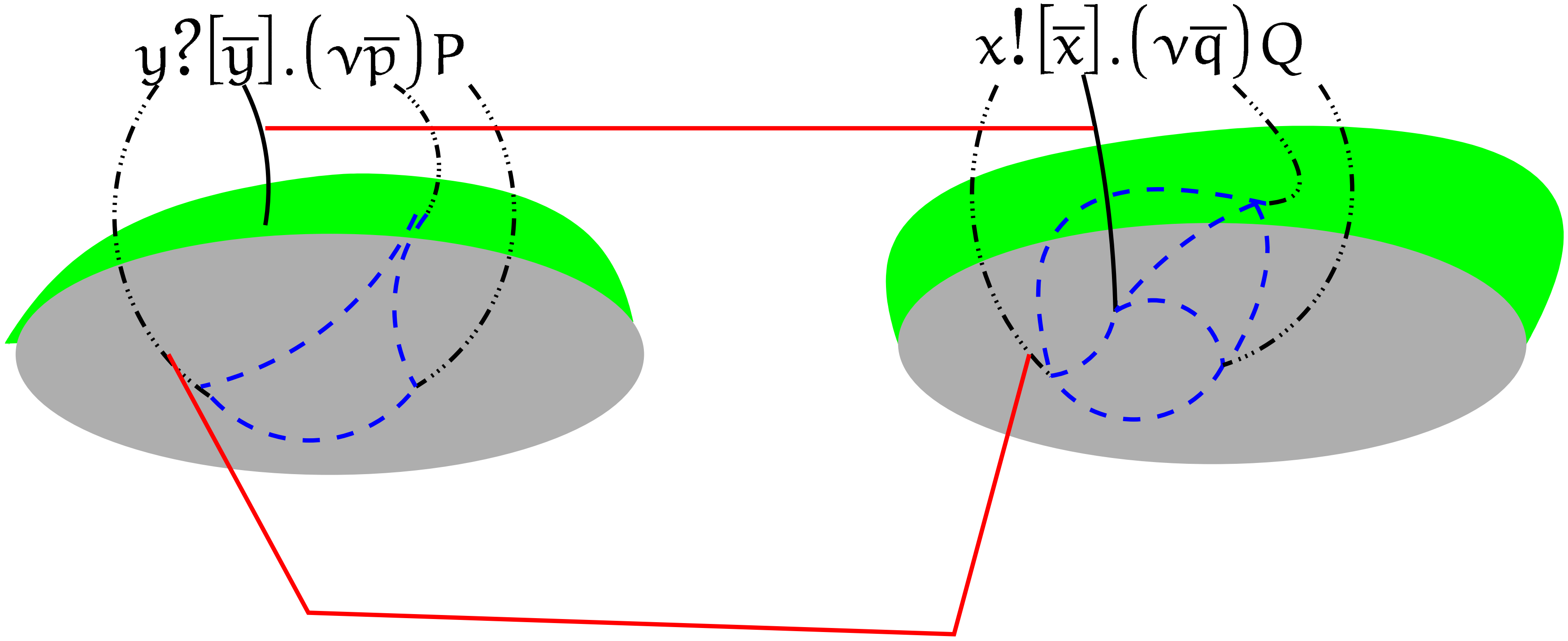
🟩 Environment Extension

- - - Relational Information

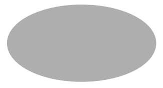
— Variable Property

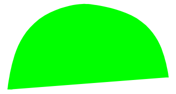
— Synchronization Constraint

Synchronizing environments



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

 Environment Property

 Environment Extension

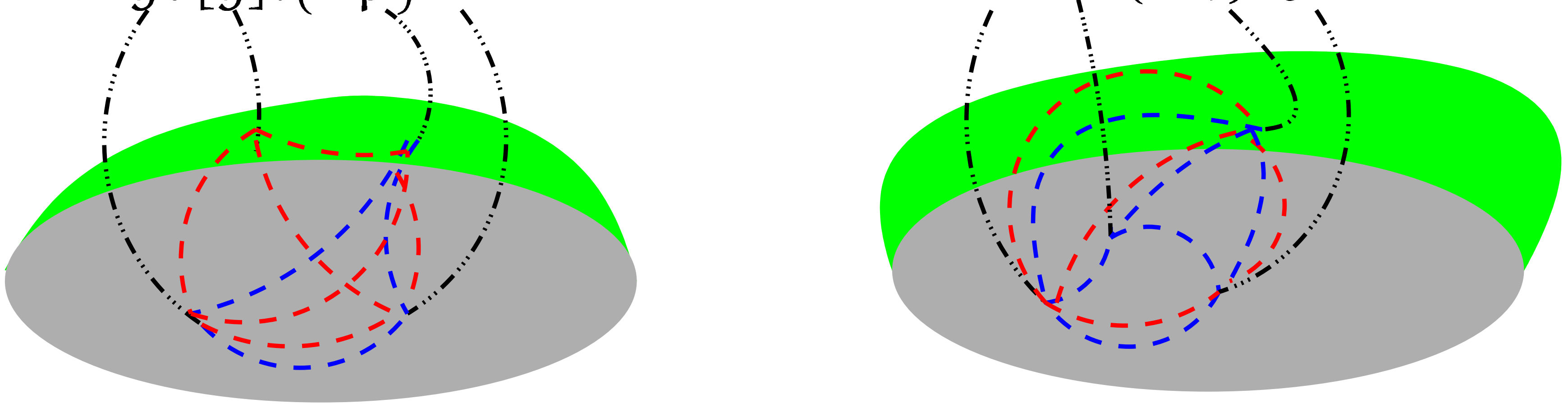
Relational Information

Variable Property

Synchronization Constrai

Propagating information

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



- ⬤ 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 syntactic agents,
3. synchronize variables,
4. compute information closure,
5. separate the 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 $Id^\sharp$ be an abstract domain of properties about marker pairs.

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

$$Atom^\sharp_{\mathcal{V}} = \mathcal{V} \times Label \rightarrow Id^\sharp$$

$\gamma_{\mathcal{V}}(a^\sharp)$ is the set of marker/environment pairs (id, E) such that:

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

Abstract molecules

Given:

$$\gamma_{Id}^{V} : Id_V^\sharp \to \wp(V \to Id),$$

The domain $Molecule^\sharp_{(V_i)_{i\in[\![1;n]\!]}}$ is the set of all 5-tuples (f, S, C, E, r) where:

- $f \in [\![1;n]\!] \to Atom^\sharp_{V_i}$;
- $S \subseteq \{(a,k) \mid 1 \leq k \leq n,\ a \in V_k \cup \{I\}\}$
  (constrained variables);
- $P \in \wp(\wp(S))$ is a partition of S
  (equality constraints);
- $E \in P \times P$
  (disequality constraints);
- the map $r \in (P \to Label) \to Id^\sharp_P$
  (marker relations).

Several trade-offs

1. 0-cfa (0-CFA): $\mathit{Id}^\sharp = \{\bot, \top\}$,
   [Nielson *et al.*:CONCUR'98], [Hennessy and Riely:HLCL'98].
2. Confinement (CONF): $\mathit{Id}^\sharp = \{\bot, =, \top\}$,
   [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 ($i = 1$).

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:

- $\gamma_1(I) = \{L \mid \forall u \in L,\ |u| > 0 \Rightarrow u_1 \in I\}$,
- $\gamma_2(F) = \{L \mid \forall u \in L,\ |u| > 0 \Rightarrow u_{|u|} \in F\}$,
- $\gamma_3(T) = \{L \mid \forall u, v \in \Sigma^*, \lambda, \mu \in \Sigma,\ u.\lambda.\mu.v \in L \Rightarrow (\lambda, \mu) \in T\}$,
- $\gamma_4(b) = \{L \mid \varepsilon \in L \Rightarrow b\}$.

Comparison between channel and agent markers

We capture affine relationship among the occurrence number of litterals inside markers.
Two trade-offs:

1. Component-wise (LOC_i):

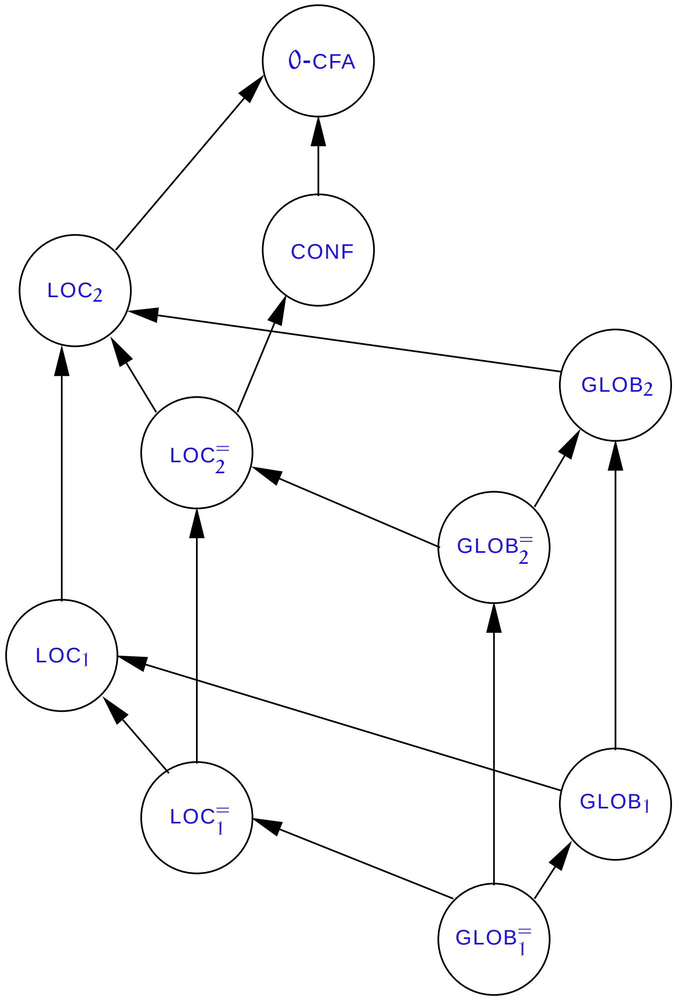
   For each litteral, we compute an affine system that describes number of occurrence of this litteral in each marker.

   ex: $Id^\sharp = (\Sigma \to (\mathbb{N} \cup \{\top\})) \cup \{\bot\}$

2. Global (GLOB_i):

   we associate a variable for each litteral and each component and compute affine relationships among these variable.

Abstract semantics hierarchy



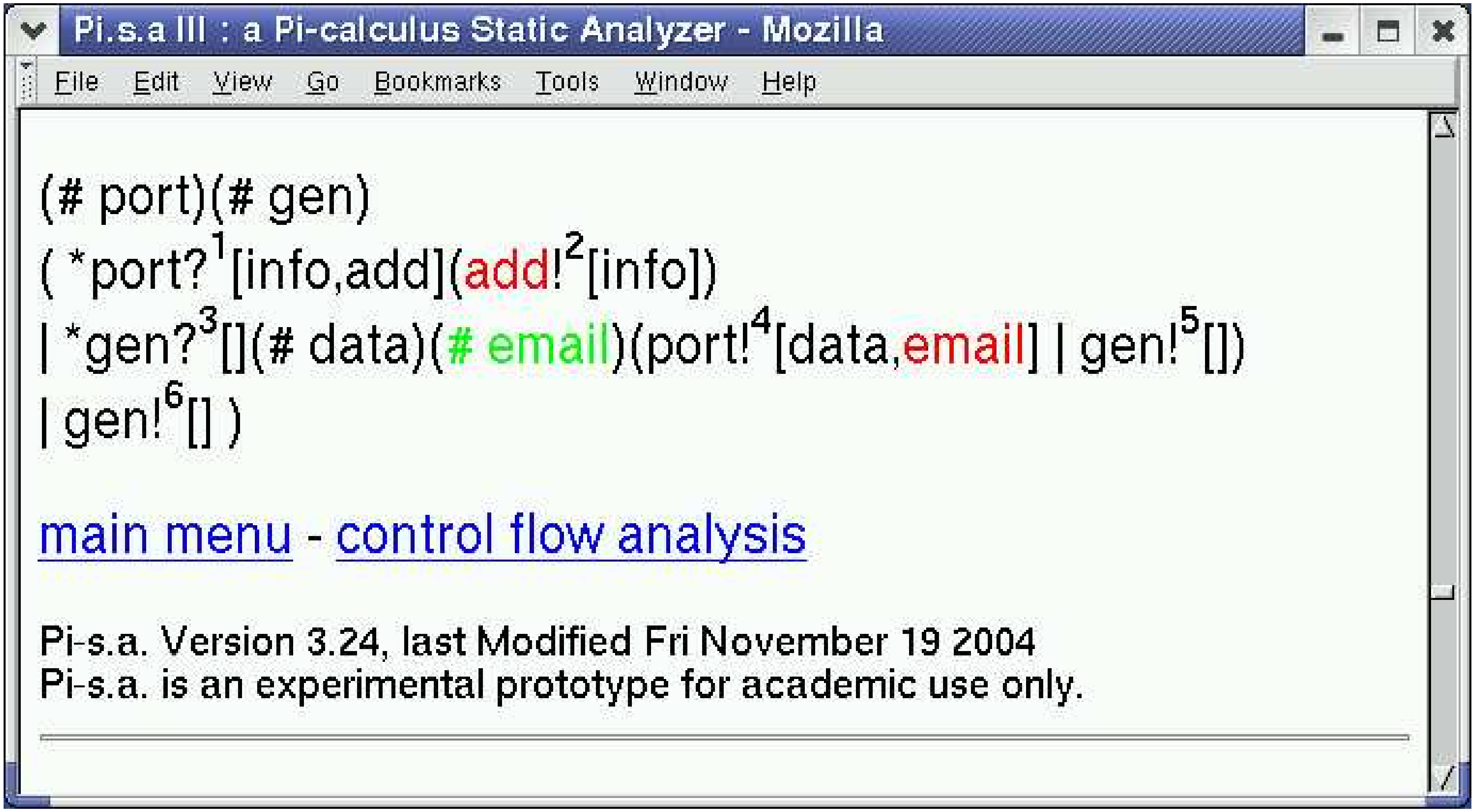
where

$$A \longrightarrow B$$

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

$$\alpha(\llbracket S \rrbracket_A^\sharp) \sqsubseteq_B \llbracket S \rrbracket_B^\sharp.$$

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

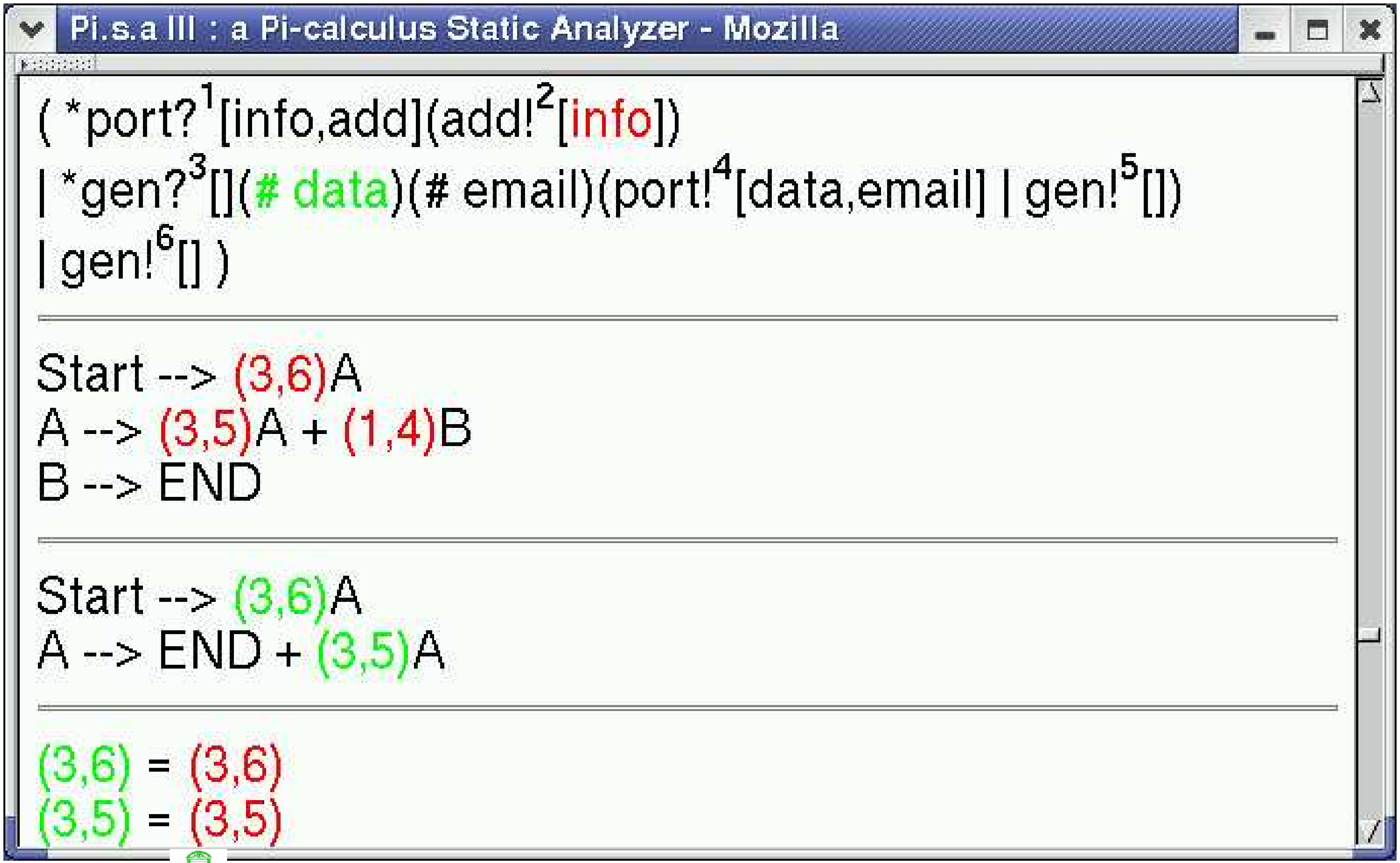
Pi-s.a. Version 3.24, last Modified Fri November 19 2004
Pi-s.a. is an experimental prototype for academic use only.

Non uniform property

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

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

Example: non-uniform result



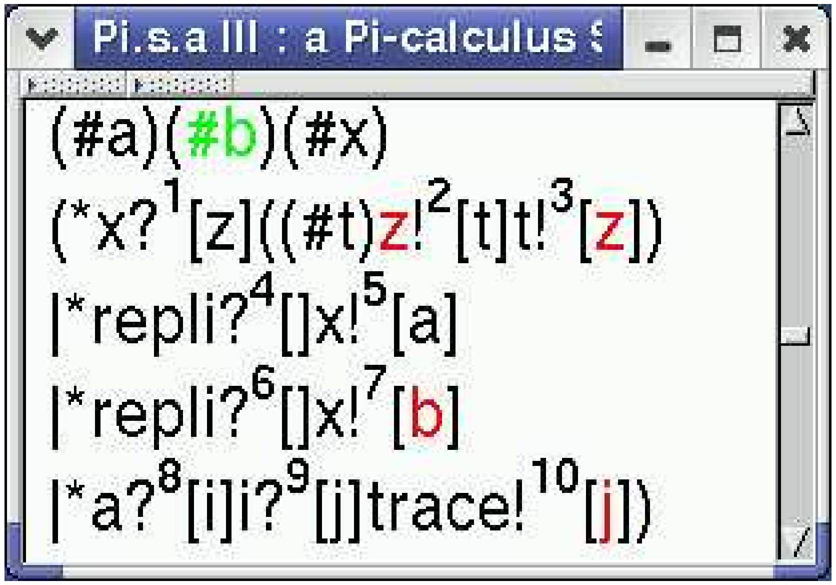
```
( *port?¹[info,add](add!²[info])
| *gen?³[](# data)(# email)(port!⁴[data,email] | gen!⁵[])
| gen!⁶[] )
```

```
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)
```

uniform analysis VS non uniform analysis



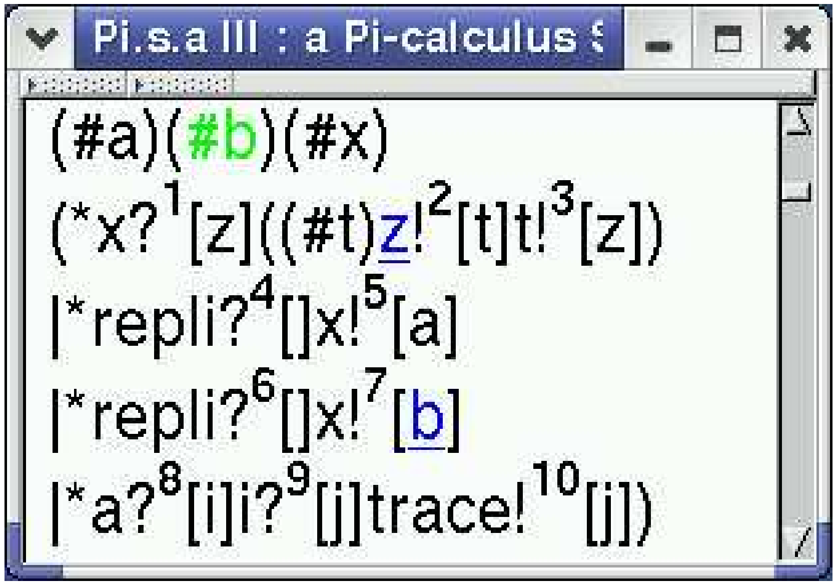
$(\#a)(\#b)(\#x)$

$(^{*}x?^{1}[z]((\#t)z!^{2}[t]t!^{3}[z])$

$|^{*}repli?^{4}[]x!^{5}[a]$

$|^{*}repli?^{6}[]x!^{7}[b]$

$|^{*}a?^{8}[i]i?^{9}[j]trace!^{10}[j])$



$(\#a)(\#b)(\#x)$

$(^{*}x?^{1}[z]((\#t)\underline{z}!^{2}[t]t!^{3}[z])$

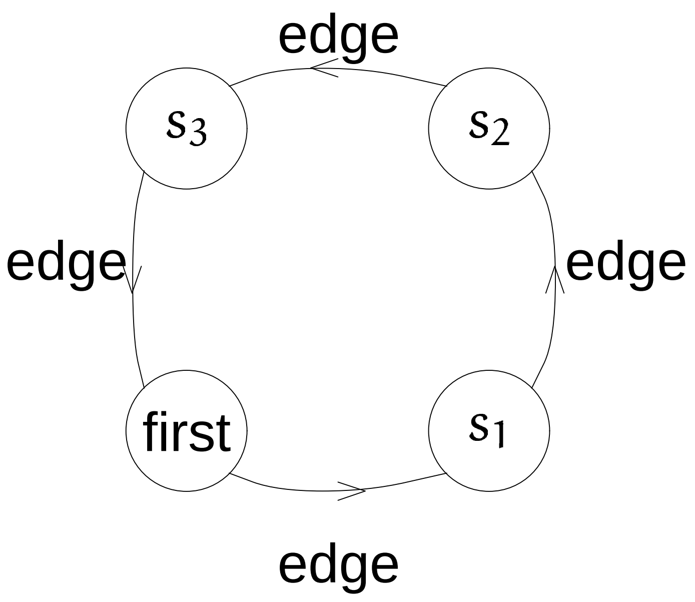
$|^{*}repli?^{4}[]x!^{5}[a]$

$|^{*}repli?^{6}[]x!^{7}[\underline{b}]$

$|^{*}a?^{8}[i]i?^{9}[j]trace!^{10}[j])$

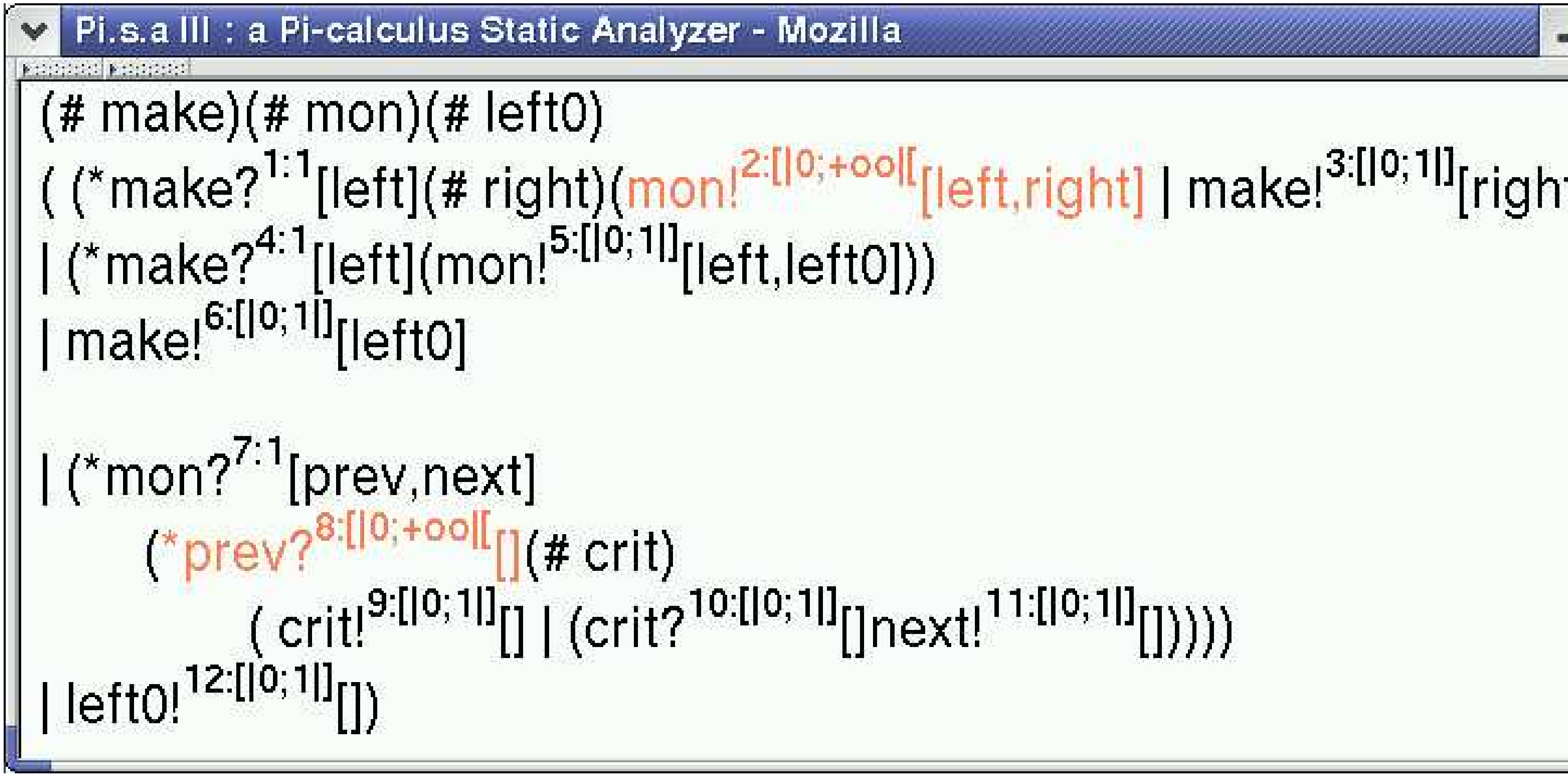
Example: the ring of processes

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



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

Example: Algebraic properties


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|]}[righ
| (*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|]}[])
```

Example

We detect that:

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

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

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

answer

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

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

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

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:

```
$(\nu\ \text{make})(\nu\ \text{edge})(\nu\ \text{first})(\nu\ \text{first})$
    $(*\text{make}?^{1}[\textit{last}](\nu \textit{next})$
                        $(\text{edge}!^{2}[\textit{last},\textit{next}]$
                        $\mid\ \text{make}!^{3}[\textit{next}])$
    $\mid\ *\text{make}?^{4}[\textit{last}](\text{edge}!^{5}[\textit{last},\text{first}])$
    $\mid\ \text{make}!^{6}[\text{first}])$
    $\mid\ \text{edge}?^{7}[x,y][x =^{8} y][x \neq^{9} \text{first}]\text{Ok}!^{10}[]$
```

we cannot infer that we can never satisfy both guards at the program points 8 and 9.

Dependency analysis

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

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

$\mathit{Atom}^{\sharp}_{V}$ is related to $\wp(\mathit{Id} \times (V \rightarrow (\mathit{Label} \times \mathit{Id})))$ 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.

Pair-wise numerical analysis

We compare pair-wisely markers, while partitioning channels in accordance with the name restrictions having opened them.
Let Φ be a linear form defined on $\mathbb{R}^{\Sigma}$, for each $V \subseteq \mathcal{V}$, the domain $\mathit{Atom}^{\sharp}_{V}$ is a pair of function (f, g):

$$\begin{array}{rcl} f &:& V \times \mathit{Label} \rightarrow \{\,\mathit{Affine\ subspace\ of}\ \mathbb{R}^2\}, \\ g &:& (V \times \mathit{Label})^2 \rightarrow \{\,\mathit{Affine\ subspace\ of}\ \mathbb{R}^2\}, \end{array}$$

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

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

Global numerical analysis

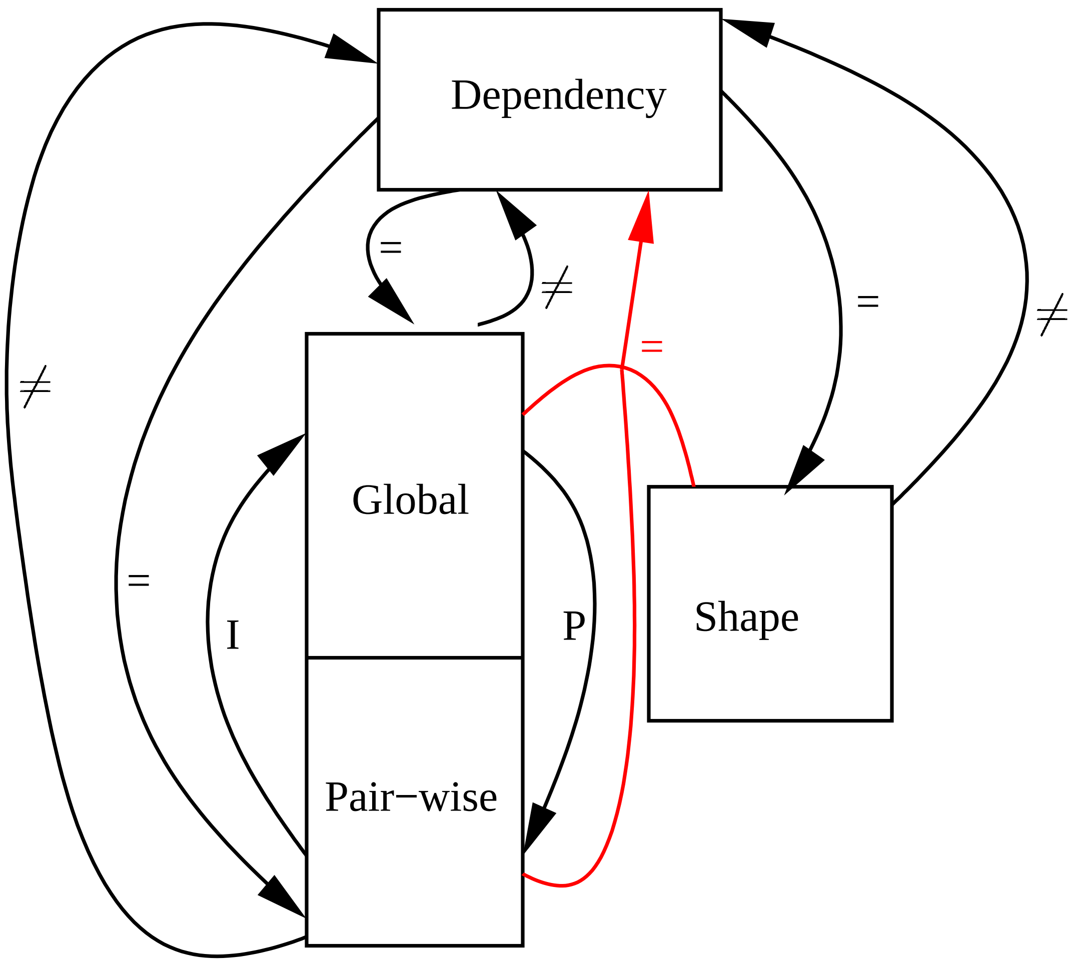
We abstract relations between all name markers and all names linked to variables, and the thread markers:
For each $V \subseteq \mathcal{V}$, we introduce the set of variables:

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

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

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

Reduction



Example

$(\nu$ make$)(\nu$ edge$)(\nu$ first$)$
$(*\text{make?}^{1}[\mathit{last}](\nu\ \mathit{next})\ (\text{edge!}^{2}[\mathit{last},\mathit{next}] \mid \text{make!}^{3}[\mathit{next}])$
$\mid\ *\text{make?}^{4}[\mathit{last}](\text{edge!}^{5}[\mathit{last},\text{first}])$
$\mid\ \text{make!}^{6}[\text{first}])$
$\mid\ \text{edge?}^{7}[x,y][x=^{8}y][x \neq^{9}\text{first}]\text{Ok!}^{10}[]$

we first prove in global abstraction that:

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

$$f(5)\ \textit{satisfies}\ \begin{cases} c^{\mathit{next},\mathit{last}} + c^{\text{first},\mathit{last}} = 1 \\ c^{\text{first},\text{first}} = 1 \end{cases}$$

Example

We then prove in pair-wise analysis that at program point 8, x and y are respectively linked to names created by some instance of the restrictions :

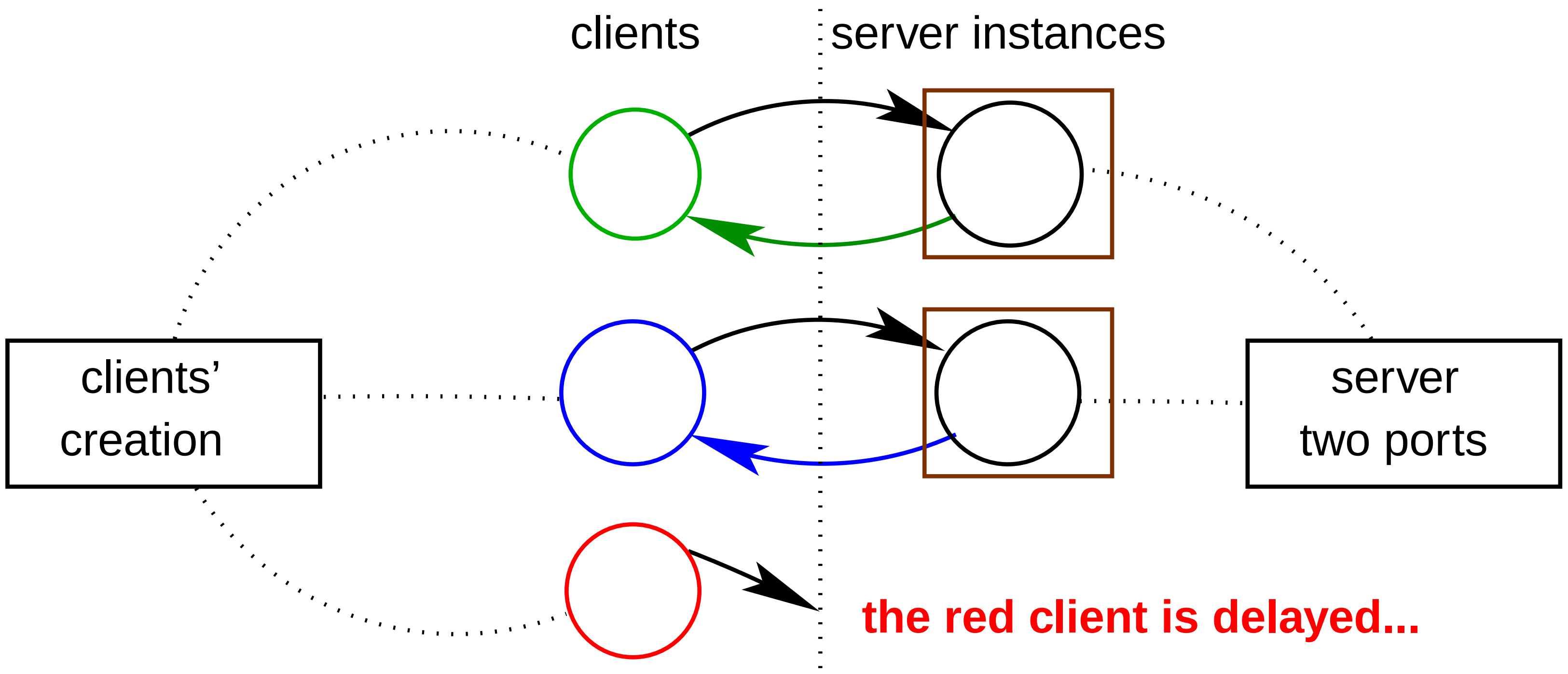
1. (ν first) and (ν first),
2. (ν first) and (ν *next*),
3. (ν *next*) and (ν *next*) but distinct instances,
4. (ν *next*) and (ν first).

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

Overview

1. Abstract Interpretation
2. Environment analysis
3. Occurrence counting analysis
4. Thread partitioning
5. Conclusion

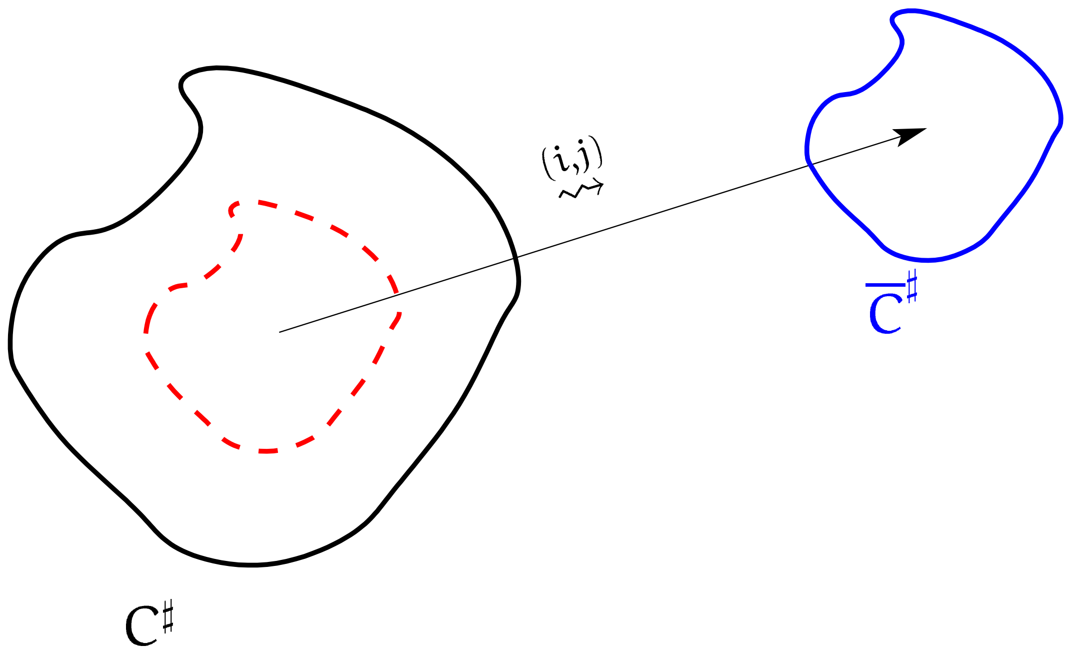
Example: a 3-port server



Occurrences counting analysis

$$\left\{\begin{array}{l}
\left(1, \varepsilon, \left\{\mathrm{port} \mapsto (\mathrm{port}, \varepsilon)\right.\right) \\
\left(3, \varepsilon, \left\{\begin{array}{ll} \mathrm{gen} & \mapsto (\mathrm{gen}, \varepsilon) \\ \mathrm{port} & \mapsto (\mathrm{port}, \varepsilon) \end{array}\right.\right) \\
\left(2, id'_1, \left\{\begin{array}{ll} \mathit{add} & \mapsto (\mathit{email}, id_1) \\ \mathit{info} & \mapsto (\mathit{data}, id_1) \end{array}\right.\right) \\
\left(2, id'_2, \left\{\begin{array}{ll} \mathit{add} & \mapsto (\mathit{email}, id_2) \\ \mathit{info} & \mapsto (\mathit{data}, id_2) \end{array}\right.\right) \\
\left(5, id_2, \left\{\mathrm{gen} \mapsto (\mathrm{gen}, \varepsilon)\right.\right)
\end{array}\right\}$$

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.

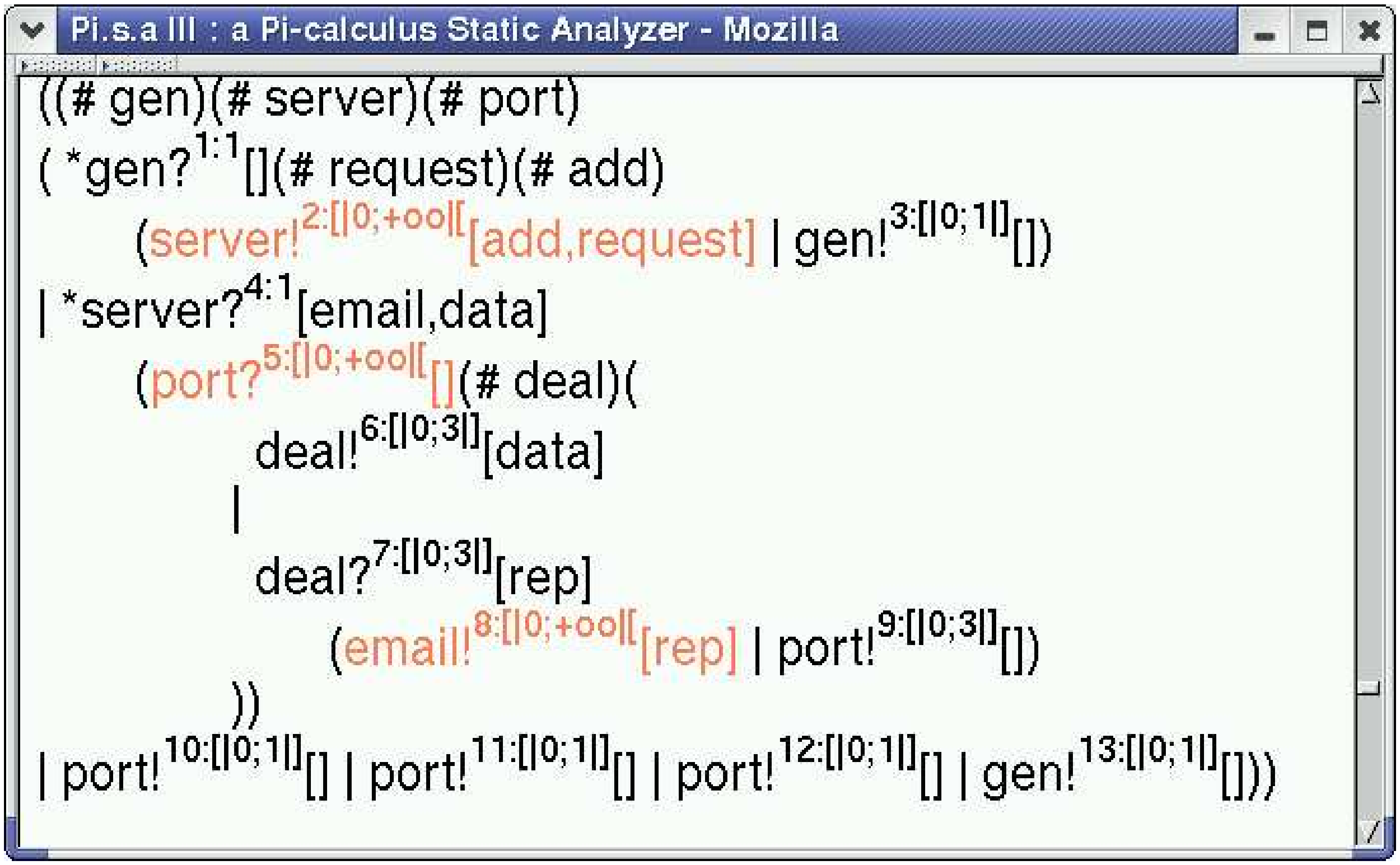
Interval narrowing

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: $$\begin{cases} x+y-z=3 \\ x \in [\![1;2]\!] \end{cases} \Longrightarrow \begin{cases} x+y-z=3 \\ y-z \in [\![1;2]\!] \\ x \in [\![1;2]\!] \end{cases}$$

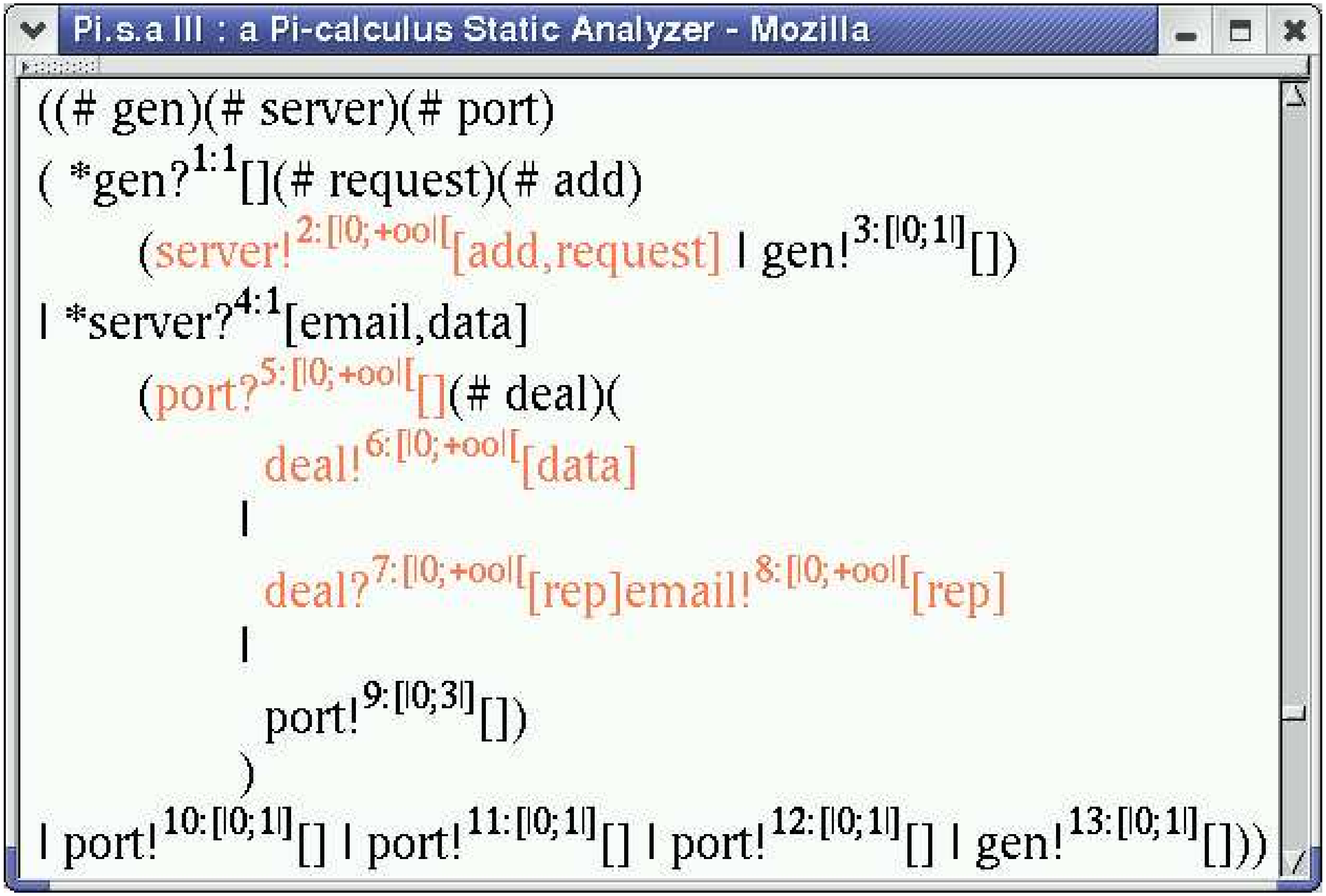
to get a cubic approximated reduction.

Example: non-exhaustion of resources


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

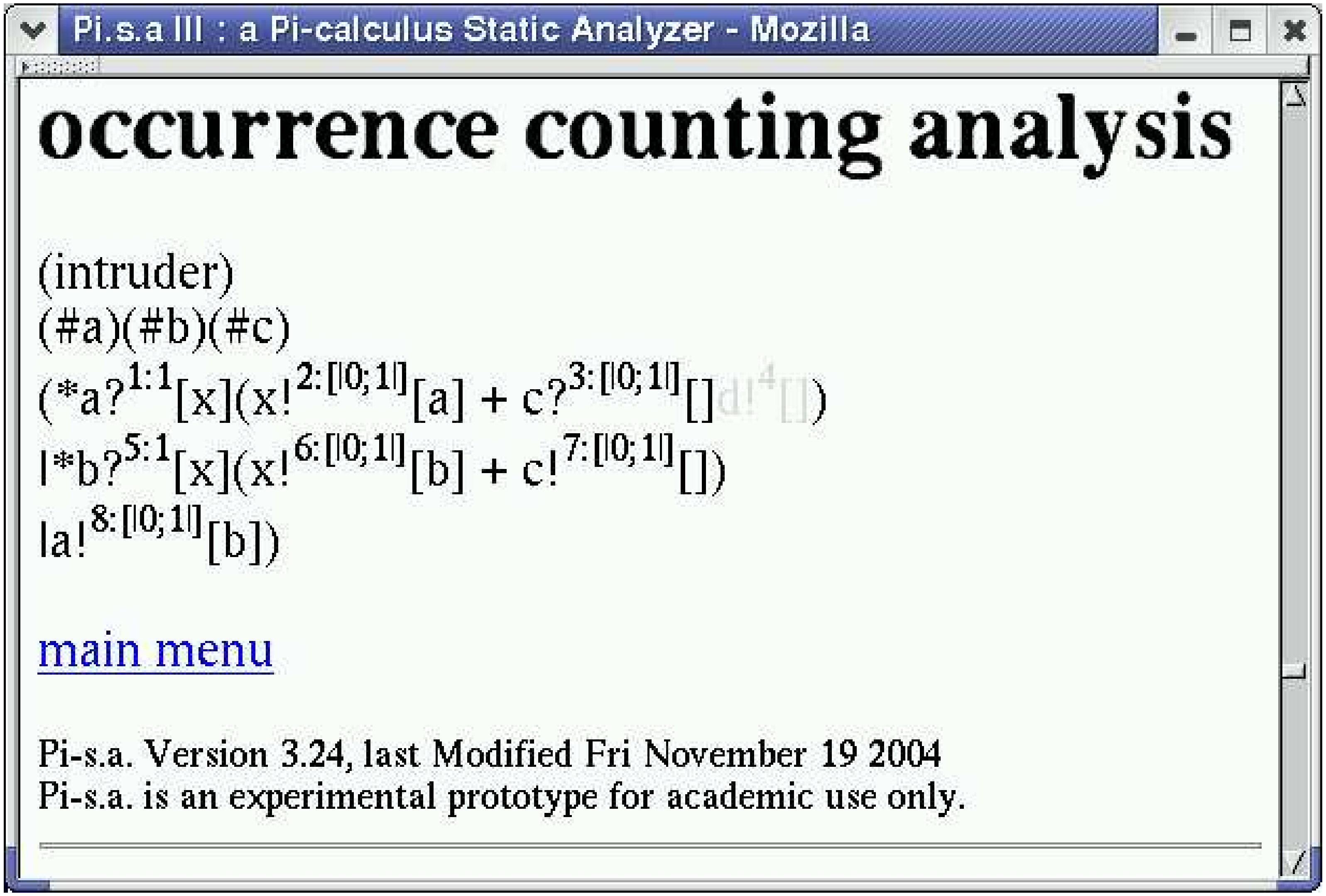
```
((# gen)(# server)(# port)
( *gen?^{1:1}[](# request)(# add)
      (server!^{2:[|0;+oo|[}[add,request] | gen!^{3:[|0;1|]}[])
| *server?^{4:1}[email,data]
      (port?^{5:[|0;+oo|[}[](# deal)(
            deal!^{6:[|0;3|]}[data]
            |
            deal?^{7:[|0;3|]}[rep]
                  (email!^{8:[|0;+oo|[}[rep]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

Example: 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;+oo|[[data]
            |
            deal?7:[|0;+oo|[[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|][]))
```

Example: mutual exclusion



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

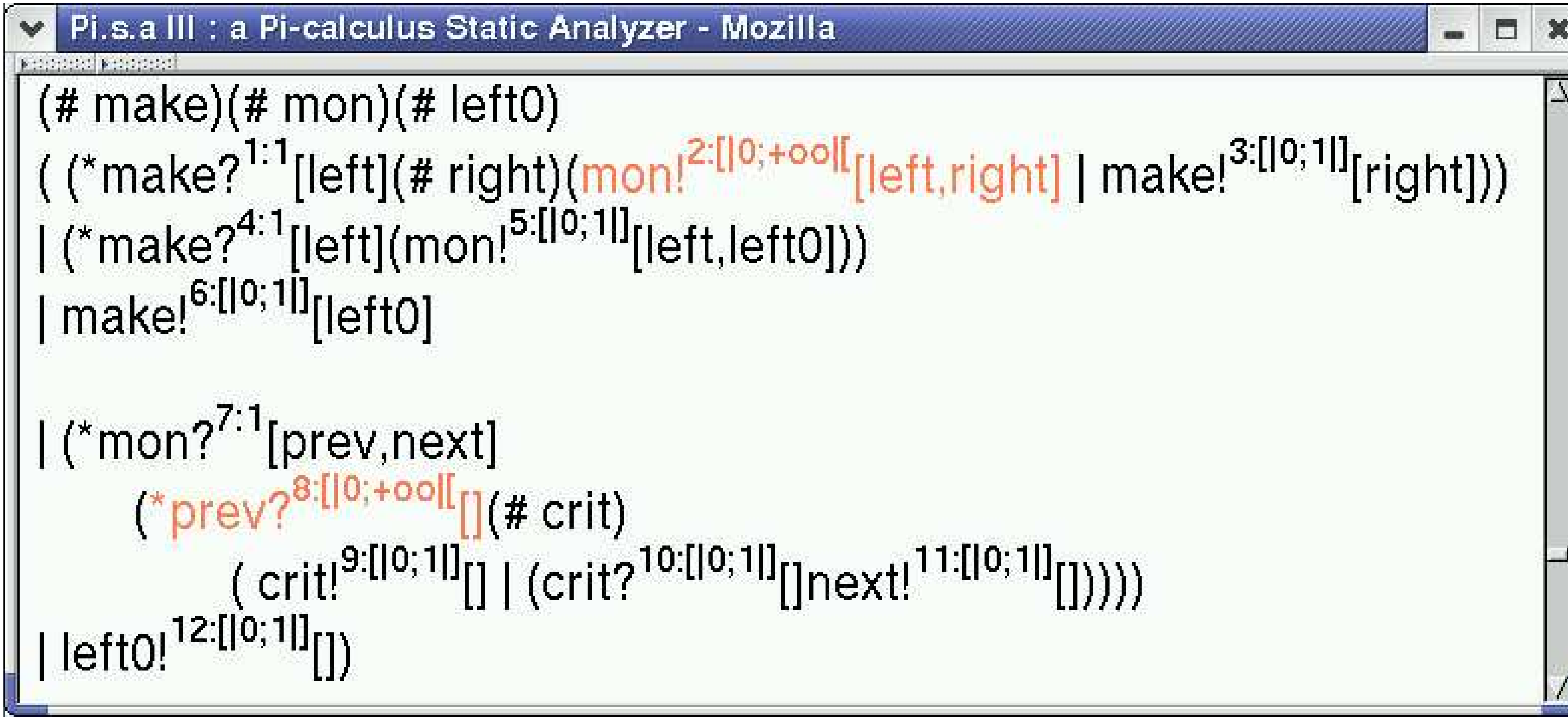
occurrence counting analysis

(intruder)
(#a)(#b)(#c)
$(*a?^{1:1}[x](x!^{2:[|0;1|]}[a] + c?^{3:[|0;1|]}[]\, d!^{4}[])$
$|*b?^{5:1}[x](x!^{6:[|0;1|]}[b] + c!^{7:[|0;1|]}[])$
$|a!^{8:[|0;1|]}[b])$

main menu

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

Example: token ring



```
(# 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|]}[])
```

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 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

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:

$$\textit{unit} \overset{\Delta}{=} B \rightarrow \textit{Label} \times \textit{Id}.$$

$\textit{give-index}$ maps each program point p to a function $\textit{give-index}(p) \in B \rightarrow I(p)$.

Given a thread $t = (p, \textit{id}, E)$, we define its computation unit $\mathrm{giveunit}(t)$ as:

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

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:

$$\mathrm{UNIT}^\sharp \stackrel{\Delta}{=} \mathcal{B} \rightarrow \mathit{Label}.$$

The abstraction function:

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

maps each computation unit to an abstract one.

Abstract domain

We use:

1. a set of abstract domains:

$$\left(\mathit{Atom}^{\sharp}_{V}\right)_{V\subseteq\mathcal{V}},\ \left(\mathit{Molecule}^{\sharp}_{(V_i)}\right)_{(V_i)\in\wp(\mathcal{V})^*};$$

   We also need an extra primitive:
   given:
   (a) $(V_i) \in \wp(\mathcal{V})^n$, $m \in \mathit{Molecule}^{\sharp}_{(V_i)}$,
   (b) and $S \in \wp(\mathcal{V} \times \mathbb{N} \times \mathit{Label})$;
   the abstract molecule $\mathit{force\text{-}lab}(S, m) \in \mathit{Molecule}^{\sharp}_{(V_i)}$ satisfies:

$$\left\{(\mathit{id}_i, E_i) \in \gamma_{(V_i)}(m) \mid \forall (x, k, l) \in S, \mathit{fst}(E_k(x)) = l\right\} \subseteq \gamma_{(V_i)}(\mathit{force\text{-}lab}(S, m)).$$

2. We assume that we are given a numerical domain $\mathcal{N}_{\mathcal{L}_p}$.

Abstract domain

Our main domain is a Cartesian product:

$$\mathcal{C}^{\sharp}_{part} \triangleq \left(\Pi_{p\in\mathcal{L}_p} Atom^{\sharp}_{I(p)}\right) \times \left(\text{UNIT}^{\sharp} \rightarrow \mathcal{N}_{\mathcal{L}_p}\right).$$

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

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

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

   such that:

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

Partition primitives

We define three primitives to take into account computation unit constraints in the flow analysis.
Given:

1. $n \in \mathbb{N}$,
2. $(p_i)_{1 \le i \le n} \in \mathcal{L}_p^n$,
3. $\mathit{mol} \in \mathit{Molecule}^\sharp_{(I(p_i))_{1 \le i \le n}}$,
4. $a \in \text{UNIT}^\sharp$;

we define:

1. enforcing equality among computation units:

$$\textit{same-unit}(i, j, (p_k), \mathit{mol}) \stackrel{\Delta}{=} \text{SYNC}^\sharp(S, (p_k), \mathit{mol}),$$

where $S = \{((\textit{give-index}(p_i)(b)), i) = ((\textit{give-index}(p_j)(b)), j) \mid b \in B\}$.

2. enforcing disequality among computation units:
   - if $\mathit{Card}(B) = 1$,

$$\mathit{distinct\text{-}unit}(i, j, (p_k), \mathit{mol}) \stackrel{\Delta}{=} \mathrm{SYNC}^{\sharp}(S, (p_k), \mathit{mol}),$$

     where:
     $S = \{((\mathit{give\text{-}index}(p_i)(b)), i) \neq ((\mathit{give\text{-}index}(p_j)(b)), j) \mid b \in B\}$;
   - otherwise,

$$\mathit{distinct\text{-}unit}(i, j, (p_k), \mathit{mol}) \stackrel{\Delta}{=} \begin{cases} \bot_{(I(p_i))} & \text{if } \forall b \in B,\ \mathrm{SYNC}^{\sharp}(S(b), (p_k), \mathit{mol}) = \\ \mathit{mol} & \text{otherwise} \end{cases}$$

     where: $S(b) = \{((\mathit{give\text{-}index}(p_i)(b)), i) \neq ((\mathit{give\text{-}index}(p_j)(b)))\}$;;
3. enforcing an abstract computation unit

$$\mathit{set\text{-}unit}(i, a, (p_k), \mathit{mol}) \stackrel{\Delta}{=} \mathit{force\text{-}lab}(S, \mathit{mol}),$$

   where $S = \{(\mathit{give\text{-}index}(p_i)(b), i, a(b)) \mid b \in B\}$.

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.

Broadcast communication: Concrete level

At the concrete level, broadcast computation apply a substitution τ.

A thread in the computation unit u migrate in the computation unit $\tau \circ u$.

Thus, during broadcast communication,

- several computation units may be merged,
  ex: dissolved ambient;
- several computation units may be renamed,
  ex: the son of a dissolved ambient.

Broadcast communication: Abstract level

For any abstract computation unit $u^\sharp$,
we collect the set of families $(v_j^\sharp, i_j)$,
such that:

1. for any $j \in J$, $v_j^\sharp \in \text{UNIT}^\sharp$ is an abstract computation unit;
2. for any $j \in J$, $i_j \in B \to [\![0;n]\!]$ is a function;
3. there exists:
   (a) a concrete computation unit u such that $\Pi_{\mathit{unit}}(u) = u^\sharp$,
   (b) a family of concrete computation units (v_j) such that $\Pi_{\mathit{unit}}(v_j) = v_j^\sharp$,

   such that for any $b \in B$, $j \in J$:

   (a) $i_j(b) = 0$ implies $v_j(b)$ is not replaced by the substitution;
   (b) $i_j(b) = k$ implies $v_j(b) = (p^k, \mathit{id}^k)$ and $I^k \in \mathit{Dom}(\mathit{broadcast})$;
4. for any $j_1, j_2 \in J$, $[\forall b \in B_s,\ i_{j_1}(b) = i_{j_2}(b)] \implies j_1 = j_2$.

and simulate the corresponding broadcast substitution.

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

$\mathsf{System} := (\nu\ \mathrm{null})(*\mathrm{create}?^{1}[\mathrm{d}].\mathsf{Allocate}(d))$

$\mathsf{Allocate}(\mathrm{d}) :=$

$\quad (\nu\ cell)(\nu\ write)(\nu\ read)$

$\quad\quad \mathsf{init}(cell)\ \mid\ \mathsf{read}(read,cell)\ \mid\ \mathsf{write}(write,cell)\ \mid\ \mathrm{d}![read;write]$

where

- $\mathsf{init}(cell) := cell!^{2}[\mathrm{null}]$
- $\mathsf{read}(read,cell) := *read?^{3}[port].cell?^{4}[u](cell!^{5}[u]\ \mid\ port!^{6}[u])$
- $\mathsf{write}(write,cell) := *write?^{7}[data,ack].cell?^{8}[u].(cell!^{9}[data] \mid ack!^{10}[])$

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 $(\nu\ \mathit{cell})$ 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. Abstract Interpretation
2. Environment analysis
3. Occurrence counting analysis
4. Thread partitioning
5. 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

- 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.