An Overview of Abstract Interpretation and Program Static Analysis

Patrick COUSOT
École Normale Supérieure
45 rue d'Ulm
75230 Paris cedex 05, France

mailto:Patrick.Cousot@ens.fr, http://www.di.ens.fr/~cousot

1st Int. Advisory Board Workshop, EECS Dept., KAIST,
Taejon, Korea June 14, 2000, 16:20–17:20

Motivations

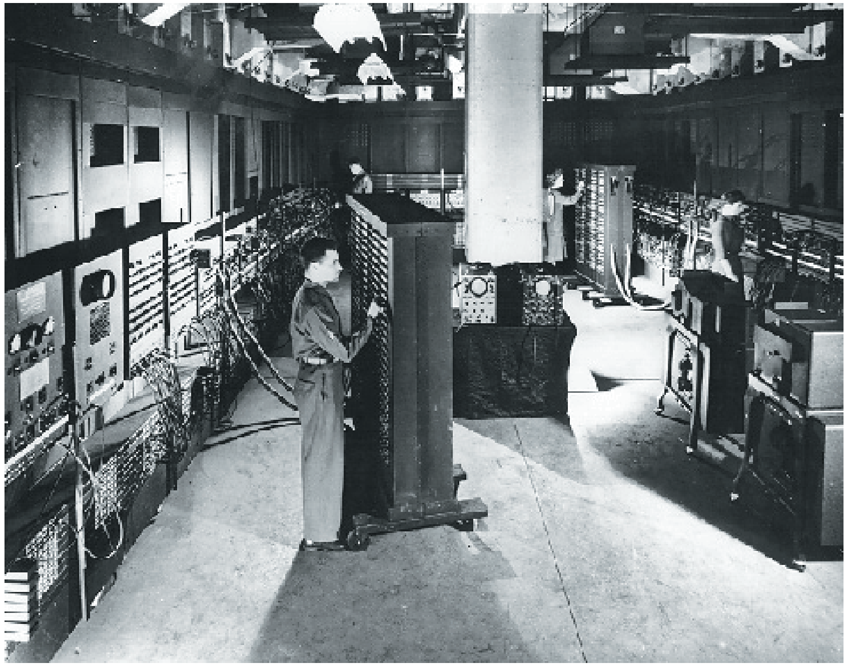
What is (or should be) the main preoccupation of computer scientists?

What is (or should be) the main preoccupation of computer scientists?

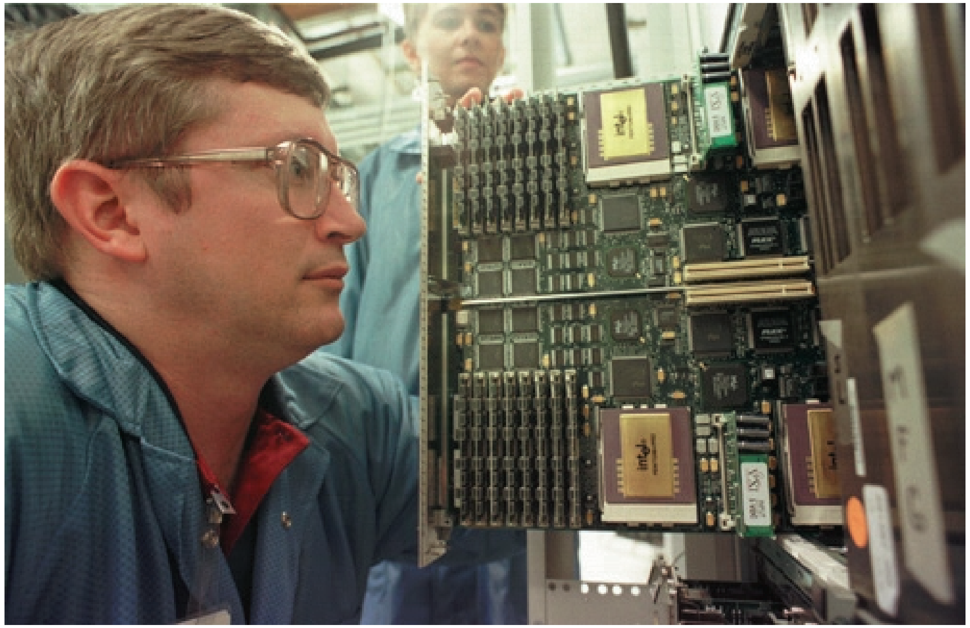
The production of reliable software, its maintenance and safe evolution year after year (up to 20 to 30 years).

Computer hardware change of scale

The 25 last years, computer hardware has seen its performances multiplied by 10^4 to 10^6;



ENIAC (5000 flops)



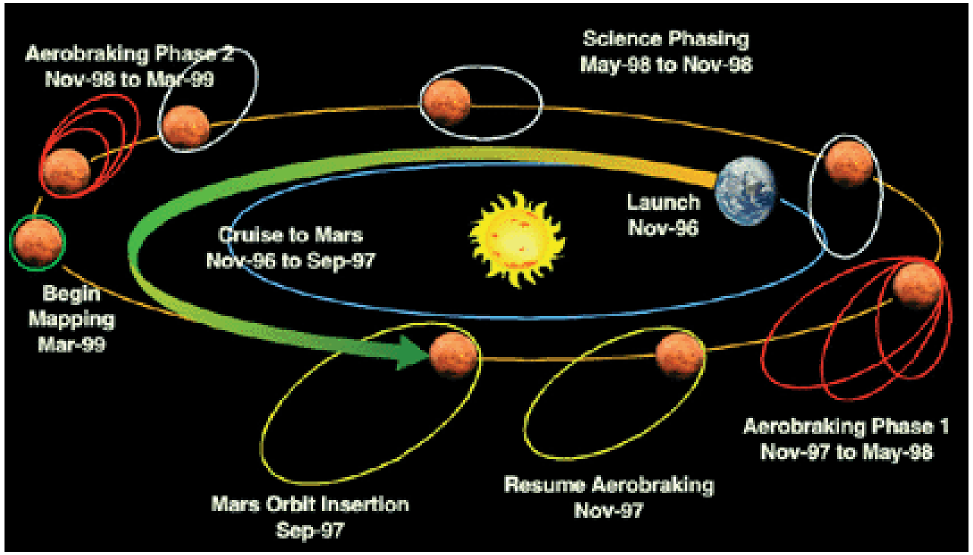
Intel/Sandia Teraflops System (10^{12} flops)

The information processing revolution

A scale of 10^6 is typical of a significant **revolution**:

– Energy: nuclear power station / Roman slave;

– Transportation: distance Earth — Mars / height of Korea

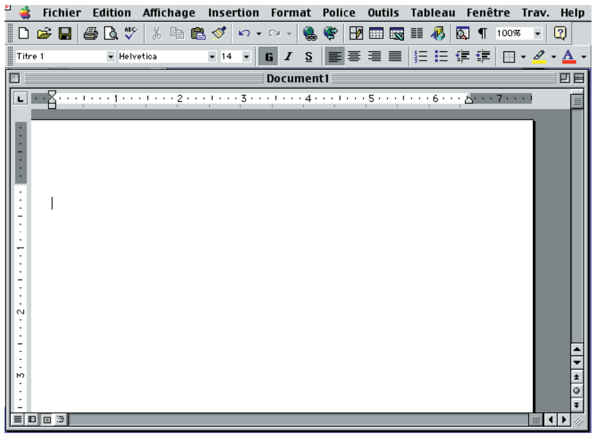


Computer software change of scale

- The size of the programs executed by these computers has grown up in similar proportions;

Computer software change of scale

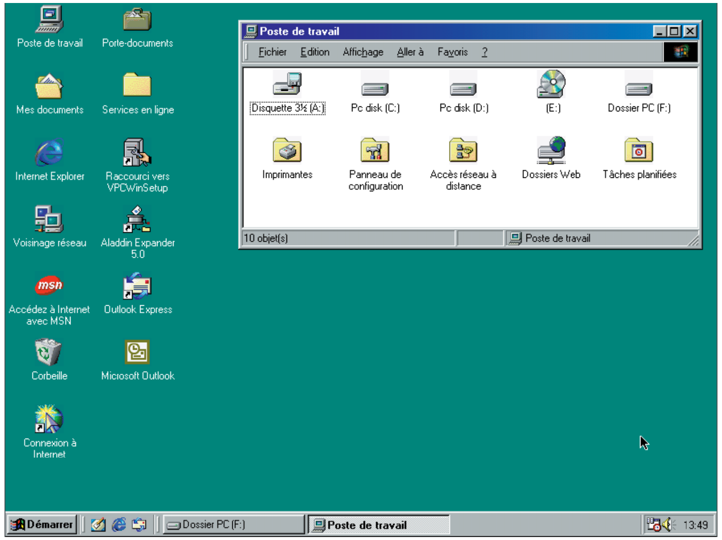
- The size of the programs executed by these computers has grown up in similar proportions;
- **Example 1** (modern text editor for the general public):
  - $>$ 1 700 000 lines of C [2];
  - 20 000 procedures;
  - 400 files;
  - $>$ 15 years of development.



2 full-time reading of the code (35 hours/week) would take at least 3 months!

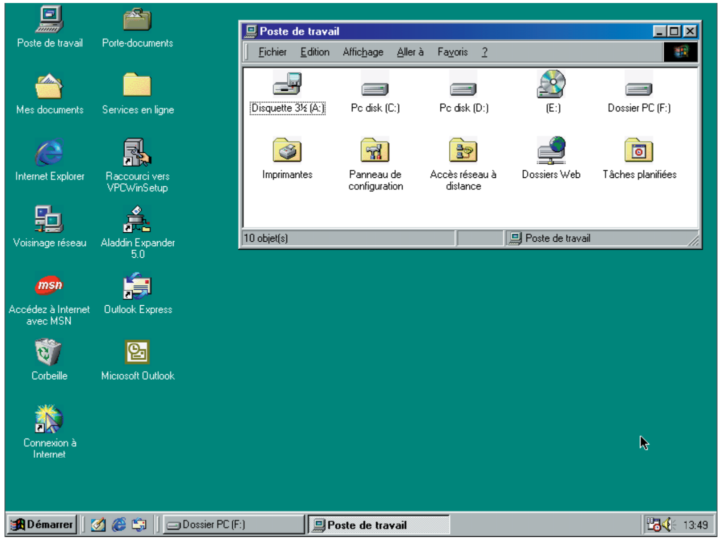
Computer software change of scale (cont'd)

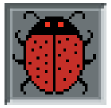
- **Example 2** (professional computer system):
  - 30 000 000 lines of code;



Computer software change of scale (cont'd)

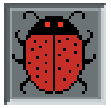
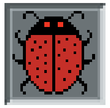
- **Example 2** (professional computer system):
  - 30 000 000 lines of code;
  - 30 000 (known) bugs!



Bugs 

- Software bugs
  - whether anticipated (Y2K bug)
  - or unforeseen (failure of the 5.01 flight of Ariane V launcher)

  are quite frequent;

Bugs  

- Software bugs
  - whether anticipated (Y2K bug)
  - or unforeseen (failure of the 5.01 flight of Ariane V launcher)

  are quite frequent;
- Bugs can be very difficult to discover in huge software;

Bugs

- Software bugs
  - whether anticipated (Y2K bug)
  - or unforeseen (failure of the 5.01 flight of Ariane V launcher)

  are frequent;
- Bugs can be very difficult to discover in huge software;
- **Bugs can have catastrophic consequences either very costly or inadmissible (embedded software in transportation systems);**

The estimated cost of an overflow

The estimated cost of an overflow

- **\$ 500 000 000**

The estimated cost of an overflow

- **$ 500 000 000**
- Including indirect costs (delays, lost markets, etc):

$ 2 000 000 000

Capability of computer scientists

- The intellectual capability of computer scientists remains essentially unchanged year after year;

Capability of computer scientists

- The intellectual capability of computer scientists remains essentially unchanged year after year;

- The size of programmer teams in charge of software design and maintenance cannot evolve in such huge proportions;

Capability of computer scientists

- The intellectual capability of computer scientists remains essentiallyunchanged year after year;

- The size of programmer teams in charge of software design and maintenance cannot evolve in such huge proportions;

- Classical manual software verification methods (code reviews, simulations, debugging) do not scale up.

Responsibility of computer scientists

- The paradox is that the computer scientists do not assume any responsibility for software bugs (compare to the automotive or avionic industry);

Responsibility of computer scientists

- The paradox is that the computer scientists do not assume any responsibility for software bugs (compare to the automotive or avionic industry);
- Computer software bugs can become an important societal problem (collective fears and reactions? new legislation?);

Responsibility of computer scientists

- The paradox is that the computer scientists do not assume any responsibility for software bugs (compare to the automotive or avionic industry);
- Computer software bugs can become an important societal problem (collective fears and reactions? new legislation?);
- The combat against software bugs might even be the next worldwide war;

Responsibility of computer scientists

- The paradox is that the computer scientists do not assume any responsibility for software bugs (compare to the automotive or avionic industry);
- Computer software bugs can become an important societal problem (collective fears and reactions? new legislation?);
- The combat against software bugs might even be the next worldwide war;

$\Longrightarrow$ It is absolutely necessary to widen the full set of methods and tools used to fight against software bugs.

Idea

Use the computer to find programming errors.

(Extremely difficult) question

How can computers be programmed so as to analyze the work they are given to do <u>before</u> effectively doing it?

A simplistic example: a cooking recipe

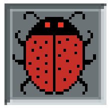
The soft-boiled egg recipe:

- Take a fresh egg out of the refrigerator;
- Plunged it into salted boiling water;
- Pull it out of the water after 4 mn.

A simplistic example: a cooking recipe

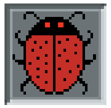
The soft-boiled egg recipe:

- Take a fresh egg out of the refrigerator;
- Plunged it into salted boiling water;
- Pull it out of the water after 4 h.



A simplistic example: a cooking recipe

The soft-boiled egg recipe:

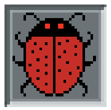
- Take a fresh egg out of the refrigerator;
- Plunged it into salted boiling water;
- Pull it out of the water after 4 h. 

Any cook can find the bug before carrying out the recipe!

A simplistic example: a cooking recipe

The soft-boiled egg recipe:

- Take a fresh egg out of the refrigerator;
- Plunged it into salted boiling water;
- Pull it out of the water after 4 h.



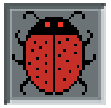
Any cook can find the bug before carrying out the recipe!

Why not computers?

A simplistic example: a cooking recipe

The soft-boiled egg recipe:

- Take a fresh egg out of the refrigerator;
- Plunged it into salted boiling water;
- Pull it out of the water after 4 h.



Any cook can find the bug before carrying out the recipe!

Why not computers?

What can we do about it?

Considered approaches for program verification

Considered approaches for program verification

Deductive methods

Considered approaches for program verification

Deductive methods: The proof size is exponential in the program size!

Considered approaches for program verification

Deductive methods: The proof size is exponential in the program size!

Model-checking

Considered approaches for program verification

Deductive methods: The proof size is exponential in the program size!

Model-checking: Gained only a factor of 100 in 10 years and the limit seems to be reached!

Considered approaches for program verification

Deductive methods: The proof size is exponential in the program size!

Model-checking: Gained only a factor of 100 in 10 years and the limit seems to be reached!

What else?

Abstract Interpretation

Introductory Talk

- Four notions to be introduced:
  - Semantics ,
  - Undecidability ,
  - Abstract interpretation ,
  - Program static analysis;

Informal Introductory Talk

- Four notions to be introduced:
  - Semantics ,
  - Undecidability ,
  - Abstract interpretation ,
  - Program static analysis;
- Completely informal explanation avoiding any formalism;

Informal Introductory Talk

- Four notions to be introduced:
  - Semantics ,
  - Undecidability ,
  - Abstract interpretation ,
  - Program static analysis;
- Completely informal explanation avoiding any formalism;
- Illustrated by the work done in my research team and the theses that I directed since 10 years.

Semantics & Undecidability

Hence we must first explain semantics, for example:

Syntax:

$$
\begin{array}{rcl}
\mathtt{x}, \mathtt{f} \in \mathbb{X} & : & \text{variables} \\
e \in \mathbb{E} & : & \text{expressions} \\
e & ::= & \mathtt{x} \mid \boldsymbol{\lambda}\mathtt{x} \cdot e \mid e_1(e_2) \mid \\
 & & \boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e \mid e_1 - e_2 \mid \\
 & & \mathbf{1} \mid (e_1 \mathbin{\mathbf{?}} e_2 \mathbin{\mathbf{:}} e_3)
\end{array}
$$

Semantic domains:

$$
\begin{array}{rcll}
\mathbb{W} & \stackrel{\triangle}{=} & \{\omega\} & \text{error} \\
\mathsf{z} \in \mathbb{Z} & & & \text{integers} \\
\mathsf{u}, \mathsf{f}, \varphi \in \mathbb{U} & \cong & \mathbb{W}_\bot \oplus \mathbb{Z}_\bot \oplus [\mathbb{U} \mapsto \mathbb{U}]_\bot & \text{values} \\
\mathsf{R} \in \mathbb{R} & \stackrel{\triangle}{=} & \mathbb{X} \mapsto \mathbb{U} & \text{environments} \\
\phi \in \mathbb{S} & \stackrel{\triangle}{=} & \mathbb{R} \mapsto \mathbb{U} & \text{semantic domain}
\end{array}
$$

Semantics:

$$
\begin{array}{rcl}
\mathbf{S}[\![\mathtt{x}]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R} \cdot \mathsf{R}(\mathtt{x}) \\
\mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R} \cdot \uparrow(\Lambda\mathsf{u} \cdot (\mathsf{u} = \bot \vee \mathsf{u} = \Omega \mathbin{?} \mathsf{u} \mid \\
 & & \qquad \mathbf{S}[\![e]\!]\mathsf{R}[\mathtt{x} \leftarrow \mathsf{u}])) :: [\mathbb{U} \mapsto \mathbb{U}]_\bot \\
\mathbf{S}[\![e_1(e_2)]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R} \cdot (\mathbf{S}[\![e_1]\!]\mathsf{R} = \bot \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \bot \mathbin{?} \bot \mid \\
 & & \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{f} :: [\mathbb{U} \mapsto \mathbb{U}]_\bot \mathbin{?} \downarrow(\mathsf{f})(\mathbf{S}[\![e_2]\!]\mathsf{R}) \mid \Omega) \\
\mathbf{S}[\![\boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R} \cdot \mathbf{lfp}^{\sqsubseteq}_{\uparrow(\Lambda\mathsf{u} \cdot \bot)::[\mathbb{U} \mapsto \mathbb{U}]_\bot} \; \Lambda\varphi \cdot \mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!]\mathsf{R}[\mathtt{f} \leftarrow \varphi] \\
\mathbf{S}[\![\mathbf{1}]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R} \cdot \uparrow(1) :: \mathbb{Z}_\bot \\
\mathbf{S}[\![e_1 - e_2]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R} \cdot (\mathbf{S}[\![e_1]\!]\mathsf{R} = \bot \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \bot \mathbin{?} \bot \mid \\
 & & \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z}_1 :: \mathbb{Z}_\bot \wedge \mathbf{S}[\![e_2]\!]\mathsf{R} = \mathsf{z}_2 :: \mathbb{Z}_\bot \mathbin{?} \\
 & & \uparrow(\downarrow(\mathsf{z}_1) - \downarrow(\mathsf{z}_2)) :: \mathbb{Z}_\bot \mid \Omega) \\
\mathbf{S}[\![(e_1 \mathbin{\mathbf{?}} e_2 \mathbin{\mathbf{:}} e_3)]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R} \cdot (\mathbf{S}[\![e_1]\!]\mathsf{R} = \bot \mathbin{?} \bot \mid \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z} :: \mathbb{Z}_\bot \mathbin{?} \\
 & & (\downarrow(\mathsf{z}) = 0 \mathbin{?} \mathbf{S}[\![e_2]\!]\mathsf{R} \mid \mathbf{S}[\![e_3]\!]\mathsf{R}) \mid \Omega)
\end{array}
$$

Hence we must first explain semantics, for example:

Syntax:

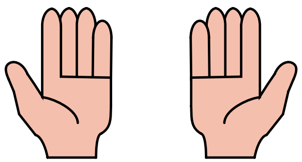
$$
\begin{array}{rcl}
\mathtt{x}, \mathtt{f} \in \mathbb{X} & : & \text{variables} \\
e \in \mathbb{E} & : & \text{expressions} \\
e & ::= & \mathtt{x} \mid \boldsymbol{\lambda}\mathtt{x} \cdot e \mid e_1(e_2) \mid \\
 & & \boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e \mid e_1 - e_2 \mid \\
 & & \mathbf{1} \mid (e_1 \mathbf{?}\ e_2 \mathbf{:}\ e_3)
\end{array}
$$

Semantic domains:

$$
\begin{array}{rcll}
\mathbb{W} & \stackrel{\triangle}{=} & \{\omega\} & \text{error} \\
\mathsf{z} \in \mathbb{Z} & & & \text{integers} \\
\mathsf{u}, \mathsf{f}, \varphi \in \mathbb{U} & \cong & \mathbb{W}_\perp \oplus \mathbb{Z}_\perp \oplus [\mathbb{U} \mapsto \mathbb{U}]_\perp & \text{values} \\
\mathsf{R} \in \mathbb{R} & \stackrel{\triangle}{=} & \mathbb{X} \mapsto \mathbb{U} & \text{environments} \\
\phi \in \mathbb{S} & \stackrel{\triangle}{=} & \mathbb{R} \mapsto \mathbb{U} & \text{semantic domain}
\end{array}
$$

Semantics:

$$
\begin{array}{rcl}
\mathbf{S}[\![\mathtt{x}]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet \mathsf{R}(\mathtt{x}) \\
\mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet \uparrow(\Lambda\mathsf{u}\bullet(\mathsf{u} = \perp \vee \mathsf{u} = \Omega\ ?\ \mathsf{u} \mid \\
 & & \qquad \mathbf{S}[\![e]\!]\mathsf{R}[\mathtt{x} \leftarrow \mathsf{u}])) :: [\mathbb{U} \mapsto \mathbb{U}]_\perp \\
\mathbf{S}[\![e_1(e_2)]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet(\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \perp\ ?\ \perp \mid \\
 & & \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{f} :: [\mathbb{U} \mapsto \mathbb{U}]_\perp\ ?\ \downarrow(\mathsf{f})(\mathbf{S}[\![e_2]\!]\mathsf{R}) \mid \Omega) \\
\mathbf{S}[\![\boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet \mathbf{lfp}^{\sqsubseteq}_{\uparrow(\Lambda\mathsf{u}\bullet\perp)::[\mathbb{U}\mapsto\mathbb{U}]_\perp}\ \Lambda\varphi\bullet \mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!]\mathsf{R}[\mathtt{f} \leftarrow \varphi] \\
\mathbf{S}[\![\mathbf{1}]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet \uparrow(1) :: \mathbb{Z}_\perp \\
\mathbf{S}[\![e_1 - e_2]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet(\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \perp\ ?\ \perp \mid \\
 & & \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z}_1 :: \mathbb{Z}_\perp \wedge \mathbf{S}[\![e_2]\!]\mathsf{R} = \mathsf{z}_2 :: \mathbb{Z}_\perp\ ? \\
 & & \uparrow(\downarrow(\mathsf{z}_1) - \downarrow(\mathsf{z}_2)) :: \mathbb{Z}_\perp \mid \Omega) \\
\mathbf{S}[\![(e_1 \mathbf{?}\ e_2 \mathbf{:}\ e_3)]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet(\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp\ ?\ \perp \mid \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z} :: \mathbb{Z}_\perp\ ? \\
 & & (\downarrow(\mathsf{z}) = 0\ ?\ \mathbf{S}[\![e_2]\!]\mathsf{R} \mid \mathbf{S}[\![e_3]\!]\mathsf{R}) \mid \Omega)
\end{array}
$$

with this 

Hence we must first explain semantics, for example:

Syntax:

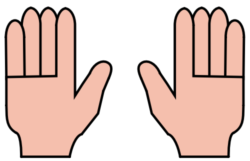
$$
\begin{array}{rcl}
\mathtt{x}, \mathtt{f} \in \mathbb{X} & : & \text{variables} \\
e \in \mathbb{E} & : & \text{expressions} \\
e & ::= & \mathtt{x} \mid \boldsymbol{\lambda}\mathtt{x} \cdot e \mid e_1(e_2) \mid \\
 & & \boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e \mid e_1 - e_2 \mid \\
 & & \mathbf{1} \mid (e_1 \mathbin{\mathbf{?}} e_2 \mathbin{\mathbf{:}} e_3)
\end{array}
$$

Semantic domains:

$$
\begin{array}{rcll}
\mathbb{W} & \stackrel{\triangle}{=} & \{\omega\} & \text{error} \\
\mathsf{z} \in \mathbb{Z} & & & \text{integers} \\
\mathsf{u}, \mathsf{f}, \varphi \in \mathbb{U} & \cong & \mathbb{W}_\perp \oplus \mathbb{Z}_\perp \oplus [\mathbb{U} \mapsto \mathbb{U}]_\perp & \text{values} \\
\mathsf{R} \in \mathbb{R} & \stackrel{\triangle}{=} & \mathbb{X} \mapsto \mathbb{U} & \text{environments} \\
\phi \in \mathbb{S} & \stackrel{\triangle}{=} & \mathbb{R} \mapsto \mathbb{U} & \text{semantic domain}
\end{array}
$$

Semantics:

$$
\begin{array}{rcl}
\mathbf{S}[\![\mathtt{x}]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet \mathsf{R}(\mathtt{x}) \\
\mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet \uparrow(\Lambda\mathsf{u}\bullet(\mathsf{u} = \perp \vee \mathsf{u} = \Omega \mathbin{?} \mathsf{u} \mid \\
 & & \qquad \mathbf{S}[\![e]\!]\mathsf{R}[\mathtt{x} \leftarrow \mathsf{u}])) :: [\mathbb{U} \mapsto \mathbb{U}]_\perp \\
\mathbf{S}[\![e_1(e_2)]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet(\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \perp \mathbin{?} \perp \mid \\
 & & \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{f} :: [\mathbb{U} \mapsto \mathbb{U}]_\perp \mathbin{?} \downarrow(\mathsf{f})(\mathbf{S}[\![e_2]\!]\mathsf{R}) \mid \Omega) \\
\mathbf{S}[\![\boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet \mathbf{lfp}^{\sqsubseteq}_{\uparrow(\Lambda\mathsf{u}\bullet \perp) :: [\mathbb{U} \mapsto \mathbb{U}]_\perp} \Lambda\varphi\bullet \mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!]\mathsf{R}[\mathtt{f} \leftarrow \varphi] \\
\mathbf{S}[\![\mathbf{1}]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet \uparrow(1) :: \mathbb{Z}_\perp \\
\mathbf{S}[\![e_1 - e_2]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet(\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \perp \mathbin{?} \perp \mid \\
 & & \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z}_1 :: \mathbb{Z}_\perp \wedge \mathbf{S}[\![e_2]\!]\mathsf{R} = \mathsf{z}_2 :: \mathbb{Z}_\perp \mathbin{?} \\
 & & \uparrow(\downarrow(\mathsf{z}_1) - \downarrow(\mathsf{z}_2)) :: \mathbb{Z}_\perp \mid \Omega) \\
\mathbf{S}[\![(e_1 \mathbin{\mathbf{?}} e_2 \mathbin{\mathbf{:}} e_3)]\!] & \stackrel{\triangle}{=} & \Lambda\mathsf{R}\bullet(\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \mathbin{?} \perp \mid \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z} :: \mathbb{Z}_\perp \mathbin{?} \\
 & & (\downarrow(\mathsf{z}) = 0 \mathbin{?} \mathbf{S}[\![e_2]\!]\mathsf{R} \mid \mathbf{S}[\![e_3]\!]\mathsf{R}) \mid \Omega)
\end{array}
$$

with this 

Hence we must first explain semantics, for example:

Syntax:

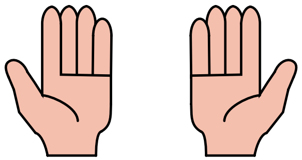
$$
\begin{array}{rcl}
\mathtt{x}, \mathtt{f} \in \mathbb{X} & : & \text{variables} \\
e \in \mathbb{E} & : & \text{expressions} \\
e & ::= & \mathtt{x} \mid \boldsymbol{\lambda}\mathtt{x} \cdot e \mid e_1(e_2) \mid \\
 & & \boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e \mid e_1 - e_2 \mid \\
 & & \mathbf{1} \mid (e_1 \mathbin{\mathbf{?}} e_2 \mathbin{\mathbf{:}} e_3)
\end{array}
$$

Semantic domains:

$$
\begin{array}{rcll}
\mathbb{W} & \triangleq & \{\omega\} & \text{error} \\
\mathsf{z} \in \mathbb{Z} & & & \text{integers} \\
\mathsf{u}, \mathsf{f}, \varphi \in \mathbb{U} & \cong & \mathbb{W}_\perp \oplus \mathbb{Z}_\perp \oplus [\mathbb{U} \mapsto \mathbb{U}]_\perp & \text{values} \\
\mathsf{R} \in \mathbb{R} & \triangleq & \mathbb{X} \mapsto \mathbb{U} & \text{environments} \\
\phi \in \mathbb{S} & \triangleq & \mathbb{R} \mapsto \mathbb{U} & \text{semantic domain}
\end{array}
$$

Semantics:

$$
\begin{array}{rcl}
\mathbf{S}[\![\mathtt{x}]\!] & \triangleq & \Lambda\mathsf{R}\boldsymbol{\cdot} \mathsf{R}(\mathtt{x}) \\
\mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!] & \triangleq & \Lambda\mathsf{R}\boldsymbol{\cdot} \uparrow\!(\Lambda\mathsf{u}\boldsymbol{\cdot}(\mathsf{u} = \perp \vee \mathsf{u} = \Omega \mathbin{?} \mathsf{u} \mid \\
 & & \qquad \mathbf{S}[\![e]\!]\mathsf{R}[\mathtt{x} \leftarrow \mathsf{u}])) :: [\mathbb{U} \mapsto \mathbb{U}]_\perp \\
\mathbf{S}[\![e_1(e_2)]\!] & \triangleq & \Lambda\mathsf{R}\boldsymbol{\cdot}(\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \perp \mathbin{?} \perp \mid \\
 & & \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{f} :: [\mathbb{U} \mapsto \mathbb{U}]_\perp \mathbin{?} \downarrow\!(\mathsf{f})(\mathbf{S}[\![e_2]\!]\mathsf{R}) \mid \Omega) \\
\mathbf{S}[\![\boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e]\!] & \triangleq & \Lambda\mathsf{R}\boldsymbol{\cdot} \mathbf{lfp}^{\sqsubseteq}_{\uparrow(\Lambda\mathsf{u}\boldsymbol{\cdot} \perp)::[\mathbb{U} \mapsto \mathbb{U}]_\perp} \Lambda\varphi\boldsymbol{\cdot} \mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!]\mathsf{R}[\mathtt{f} \leftarrow \varphi] \\
\mathbf{S}[\![\mathbf{1}]\!] & \triangleq & \Lambda\mathsf{R}\boldsymbol{\cdot} \uparrow\!(1) :: \mathbb{Z}_\perp \\
\mathbf{S}[\![e_1 - e_2]\!] & \triangleq & \Lambda\mathsf{R}\boldsymbol{\cdot}(\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \perp \mathbin{?} \perp \mid \\
 & & \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z}_1 :: \mathbb{Z}_\perp \wedge \mathbf{S}[\![e_2]\!]\mathsf{R} = \mathsf{z}_2 :: \mathbb{Z}_\perp \mathbin{?} \\
 & & \uparrow\!(\downarrow\!(\mathsf{z}_1) - \downarrow\!(\mathsf{z}_2)) :: \mathbb{Z}_\perp \mid \Omega) \\
\mathbf{S}[\![(e_1 \mathbin{\mathbf{?}} e_2 \mathbin{\mathbf{:}} e_3)]\!] & \triangleq & \Lambda\mathsf{R}\boldsymbol{\cdot}(\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \mathbin{?} \perp \mid \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z} :: \mathbb{Z}_\perp \mathbin{?} \\
 & & (\downarrow\!(\mathsf{z}) = 0 \mathbin{?} \mathbf{S}[\![e_2]\!]\mathsf{R} \mid \mathbf{S}[\![e_3]\!]\mathsf{R}) \mid \Omega)
\end{array}
$$

with this 

Semantics

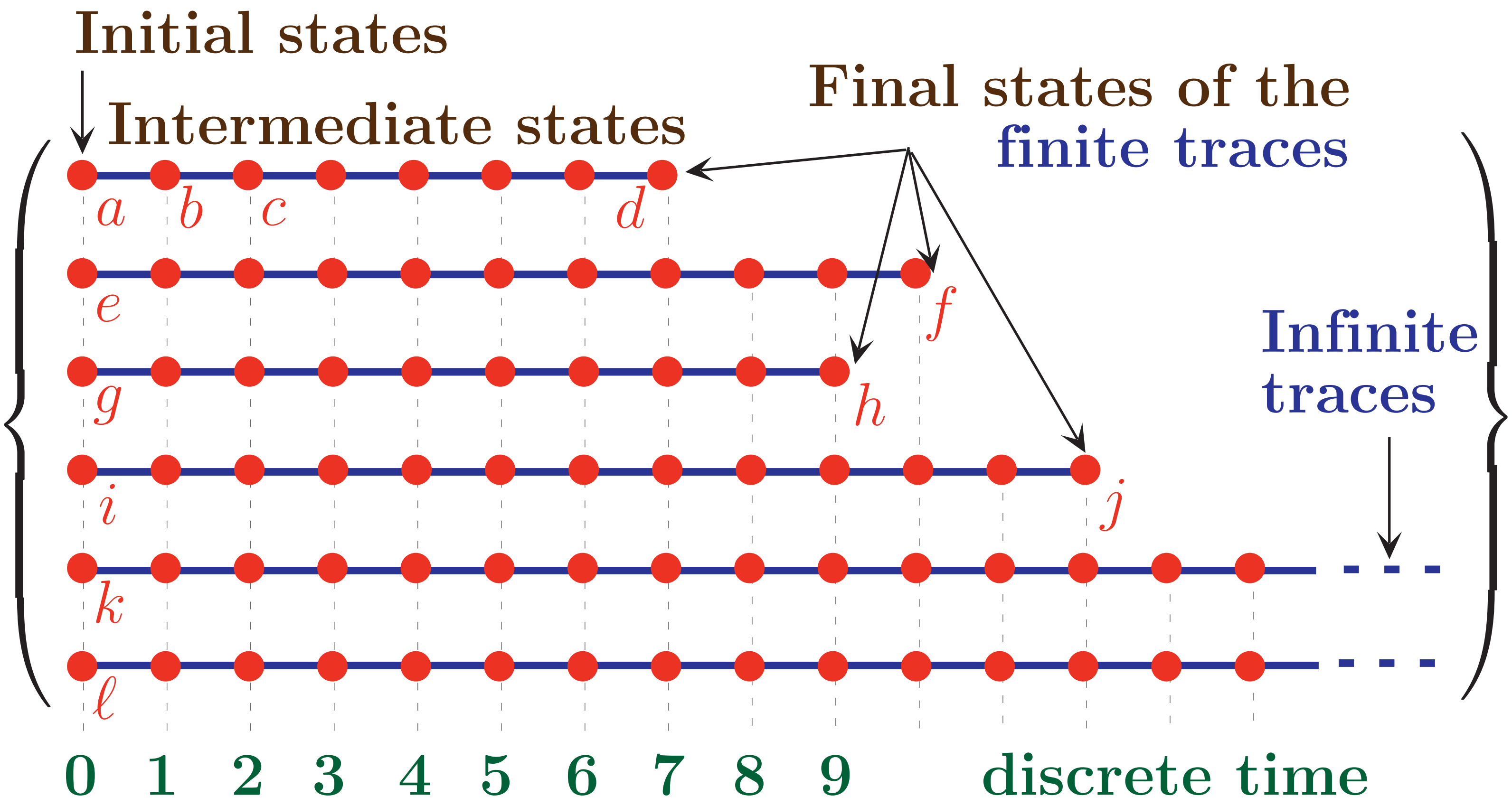
- The semantics of a program provides a formal mathematical model of all possible behaviors of a computer system executing this program (interacting with any possible environment);

Semantics

- The semantics of a program provides a formal mathematical model of all possible behaviors of a computer system executing this program (interacting with any possible environment);

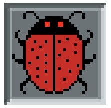
- The semantics of a language defines the semantics of any program written in this language.

Example 1: trace semantics

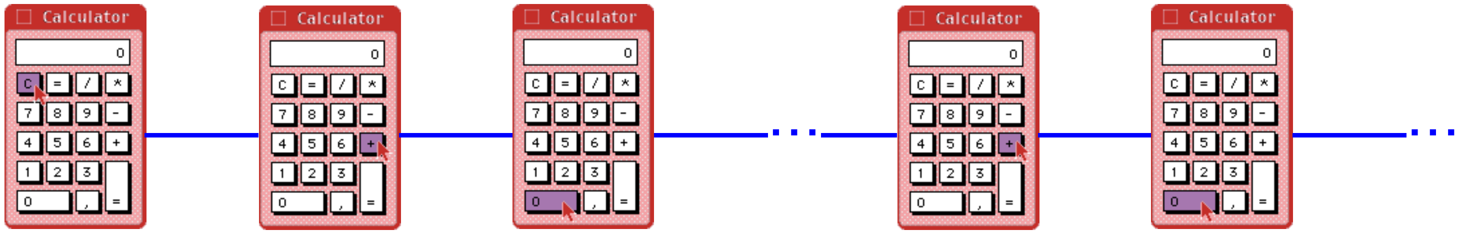


Examples of computation traces

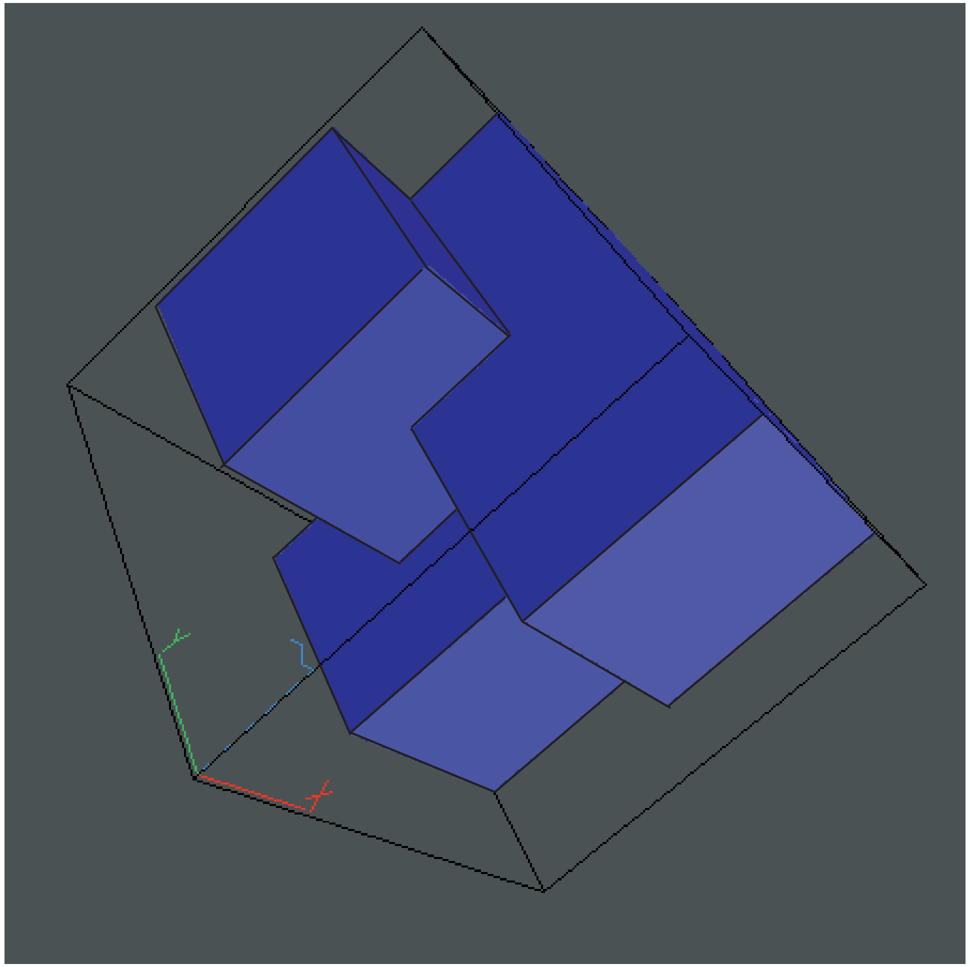
- Finite (C1+1=):

- Erroneous  (C1+1+1+1...):

- Infinite (C+0+0+0...):



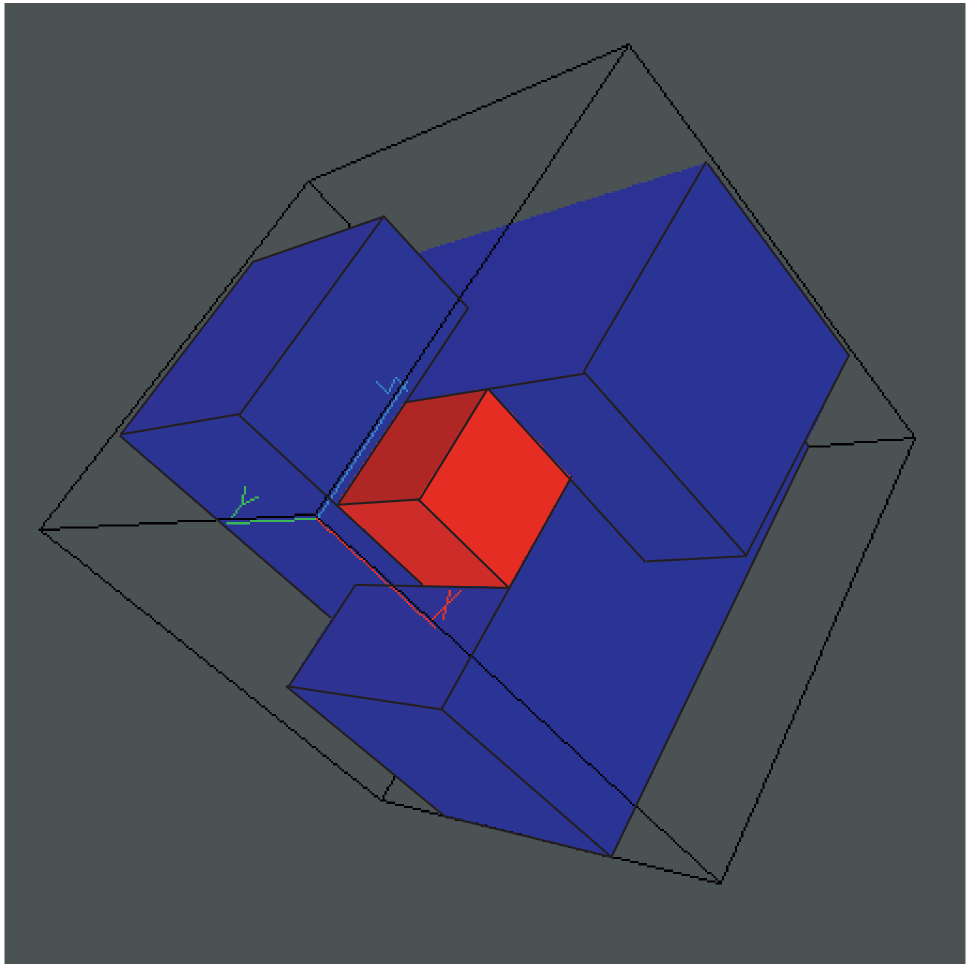
Example 2: geometric semantics



⟦ `Pa.Pb.Va.Vb`
‖ `Pb.Pc.Vb.Vc`
‖ `Pc.Pa.Vc.Va` ⟧

É. Goubault thesis, 1995

Example 2: geometric semantics (deadlock)



$[\![$ `Pa.Pb.Va.Vb`
$\|$ `Pb.Pc.Vb.Vc`
$\|$ `Pc.Pa.Vc.Va` $]\!]$

deadlock

É. Goubault thesis, 1995

Undecidability

Undecidability

- All interesting questions relative to the semantics of non trivial programs are undecidable;

Undecidability

- All interesting questions relative to the semantics of non trivial programs are undecidable:

  $\Rightarrow$ no computer can always exactly answer such questions in finite time;

Undecidability

- All interesting questions relative to the semantics of non trivial programs are undecidable:

  $\Rightarrow$ no computer can always exactly answer such questions in finite time;

- One can mathematically define the semantics of a program as the solution of a fixpoint equation;

Undecidability

- All interesting questions relative to the semantics of non trivial programs are undecidable:

  $\Rightarrow$ no computer can always exactly answer such questions in finite time;

- One can mathematically define the semantics of a program as the solution of a fixpoint equation:

  $\Rightarrow$ but no computer can exactly solve these equations in finite time.

Semantics and fixpoints

Syntax:

$$\begin{array}{rcl}
\mathtt{x}, \mathtt{f} \in \mathbb{X} & : & \text{variables} \\
e \in \mathbb{E} & : & \text{expressions} \\
e & ::= & \mathtt{x} \mid \boldsymbol{\lambda}\mathtt{x} \cdot e \mid e_1(e_2) \mid \\
& & \boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e \mid e_1 - e_2 \mid \\
& & \mathbf{1} \mid (e_1 \,\textbf{?}\, e_2 \,\textbf{:}\, e_3)
\end{array}$$

Semantic domains:

$$\begin{array}{rcll}
\mathbb{W} & \stackrel{\triangle}{=} & \{\omega\} & \text{error} \\
\mathsf{z} \in \mathbb{Z} & & & \text{integers} \\
\mathsf{u}, \mathsf{f}, \varphi \in \mathbb{U} & \cong & \mathbb{W}_\perp \oplus \mathbb{Z}_\perp \oplus [\mathbb{U} \mapsto \mathbb{U}]_\perp & \text{values} \\
\mathsf{R} \in \mathbb{R} & \stackrel{\triangle}{=} & \mathbb{X} \mapsto \mathbb{U} & \text{environments} \\
\phi \in \mathbb{S} & \stackrel{\triangle}{=} & \mathbb{R} \mapsto \mathbb{U} & \text{semantic domain}
\end{array}$$

Semantics:

$$\begin{array}{rcl}
\mathbf{S}[\![\mathtt{x}]\!] & \stackrel{\triangle}{=} & \Lambda \mathsf{R} \bullet \mathsf{R}(\mathtt{x}) \\
\mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!] & \stackrel{\triangle}{=} & \Lambda \mathsf{R} \bullet \uparrow(\Lambda \mathsf{u} \bullet (\mathsf{u} = \perp \vee \mathsf{u} = \Omega \mathrel{?} \mathsf{u} \mid \\
& & \qquad \mathbf{S}[\![e]\!]\mathsf{R}[\mathtt{x} \leftarrow \mathsf{u}])) :: [\mathbb{U} \mapsto \mathbb{U}]_\perp \\
\mathbf{S}[\![e_1(e_2)]\!] & \stackrel{\triangle}{=} & \Lambda \mathsf{R} \bullet (\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \perp \mathrel{?} \perp \mid \\
& & \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{f} :: [\mathbb{U} \mapsto \mathbb{U}]_\perp \mathrel{?} \downarrow(\mathsf{f})(\mathbf{S}[\![e_2]\!]\mathsf{R}) \mid \Omega) \\
\mathbf{S}[\![\boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathtt{x} \cdot e]\!] & \stackrel{\triangle}{=} & \boxed{\Lambda \mathsf{R} \bullet \mathbf{lfp}^{\sqsubseteq}_{\uparrow(\Lambda \mathsf{u} \bullet \perp) :: [\mathbb{U} \mapsto \mathbb{U}]_\perp} \Lambda \varphi \bullet \mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!]\mathsf{R}[\mathtt{f} \leftarrow \varphi]} \\
\mathbf{S}[\![\mathbf{1}]\!] & \stackrel{\triangle}{=} & \Lambda \mathsf{R} \bullet \uparrow(1) :: \mathbb{Z}_\perp \\
\mathbf{S}[\![e_1 - e_2]\!] & \stackrel{\triangle}{=} & \Lambda \mathsf{R} \bullet (\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \perp \mathrel{?} \perp \mid \\
& & \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z}_1 :: \mathbb{Z}_\perp \wedge \mathbf{S}[\![e_2]\!]\mathsf{R} = \mathsf{z}_2 :: \mathbb{Z}_\perp \mathrel{?} \\
& & \uparrow(\downarrow(\mathsf{z}_1) - \downarrow(\mathsf{z}_2)) :: \mathbb{Z}_\perp \mid \Omega) \\
\mathbf{S}[\![(e_1 \,\textbf{?}\, e_2 \,\textbf{:}\, e_3)]\!] & \stackrel{\triangle}{=} & \Lambda \mathsf{R} \bullet (\mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \mathrel{?} \perp \mid \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z} :: \mathbb{Z}_\perp \mathrel{?} \\
& & (\downarrow(\mathsf{z}) = 0 \mathrel{?} \mathbf{S}[\![e_2]\!]\mathsf{R} \mid \mathbf{S}[\![e_3]\!]\mathsf{R}) \mid \Omega)
\end{array}$$

Fixpoints: Intuition

Behaviors $=$

Fixpoints: Intuition

$\mathbf{Behaviors} = \{\bullet \mid \bullet \text{ is a final state}\}$

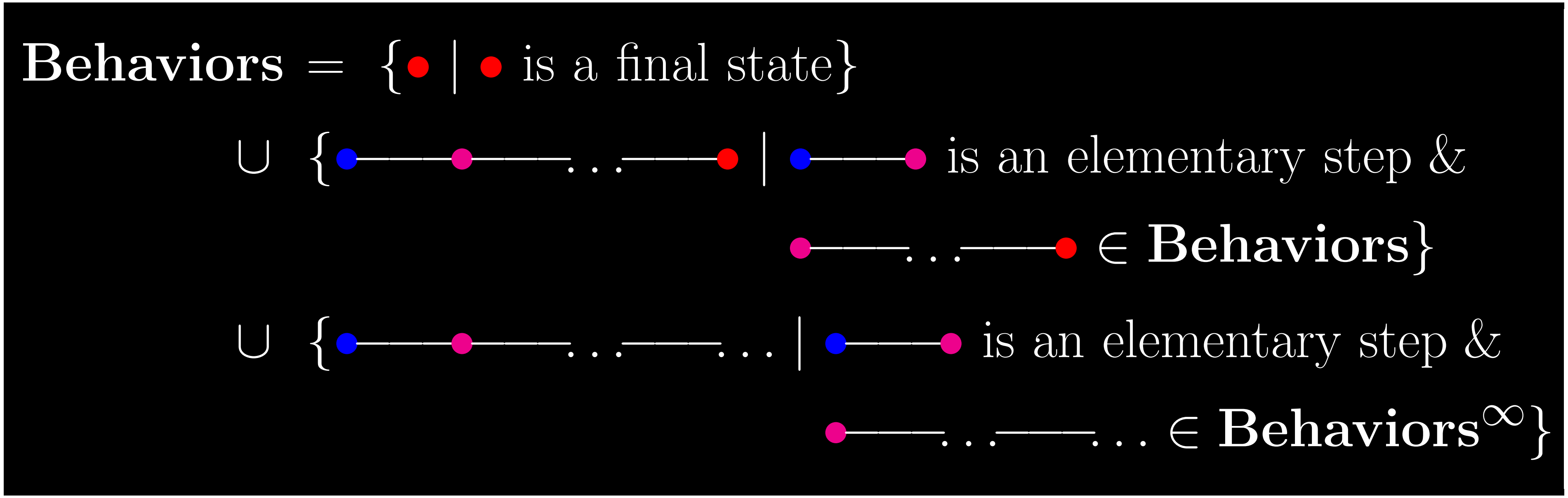
Fixpoints: Intuition

$$
\begin{aligned}
\textbf{Behaviors} = \; & \{\bullet \mid \bullet \text{ is a final state}\} \\
& \cup \; \{\bullet\!\!\longrightarrow\!\!\bullet\!\!\longrightarrow\!\!\ldots\!\!\longrightarrow\!\!\bullet \mid \bullet\!\!\longrightarrow\!\!\bullet \text{ is an elementary step } \& \\
& \qquad \bullet\!\!\longrightarrow\!\!\ldots\!\!\longrightarrow\!\!\bullet \in \textbf{Behaviors}\}
\end{aligned}
$$

Fixpoints: Intuition

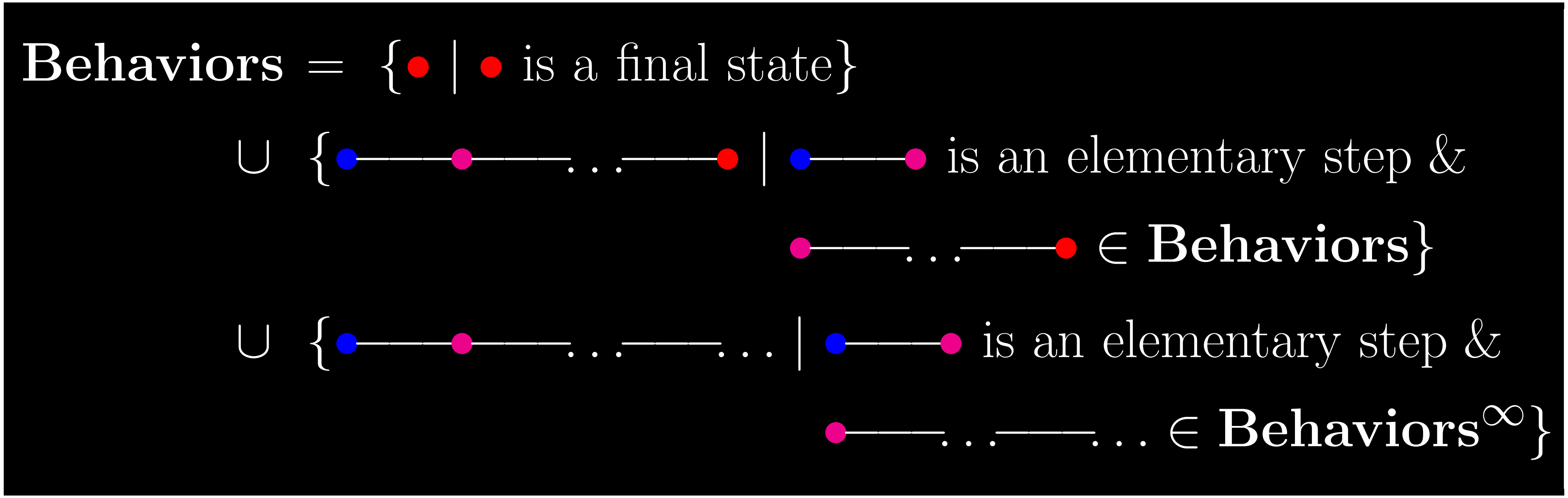
$$
\begin{aligned}
\textbf{Behaviors} = \ & \{\bullet \mid \bullet \text{ is a final state}\} \\
\cup \ & \{\bullet \!-\!\!-\!\!-\! \bullet \!-\!\!-\!\!-\! \ldots \!-\!\!-\!\!-\! \bullet \mid \bullet \!-\!\!-\!\!-\! \bullet \text{ is an elementary step } \& \\
& \qquad \bullet \!-\!\!-\!\!-\! \ldots \!-\!\!-\!\!-\! \bullet \in \textbf{Behaviors}\} \\
\cup \ & \{\bullet \!-\!\!-\!\!-\! \bullet \!-\!\!-\!\!-\! \ldots \!-\!\!-\!\!-\! \ldots \mid \bullet \!-\!\!-\!\!-\! \bullet \text{ is an elementary step } \& \\
& \qquad \bullet \!-\!\!-\!\!-\! \ldots \!-\!\!-\!\!-\! \ldots \in \textbf{Behaviors}^{\infty}\}
\end{aligned}
$$

Fixpoints: Intuition



In general, the equation has multiple solutions.

Least Fixpoints: Intuition



In general, the equation has multiple solutions. Choose the least one for the partial ordering:

« *more finite traces & less infinite traces* ».

Abstract Interpretation

Abstract interpretation

- Abstract interpretation is a theory of the approximation of the behavior of discrete systems, including the semantics of (programming or specification) languages;

Abstract interpretation

- Abstract interpretation is a theory of the approximation of the behavior of discrete systems, including the semantics of (programming or specification) languages;

- Abstract interpretation formalizes the intuitive idea that a semantics is more or less precise according to the considered observation level.

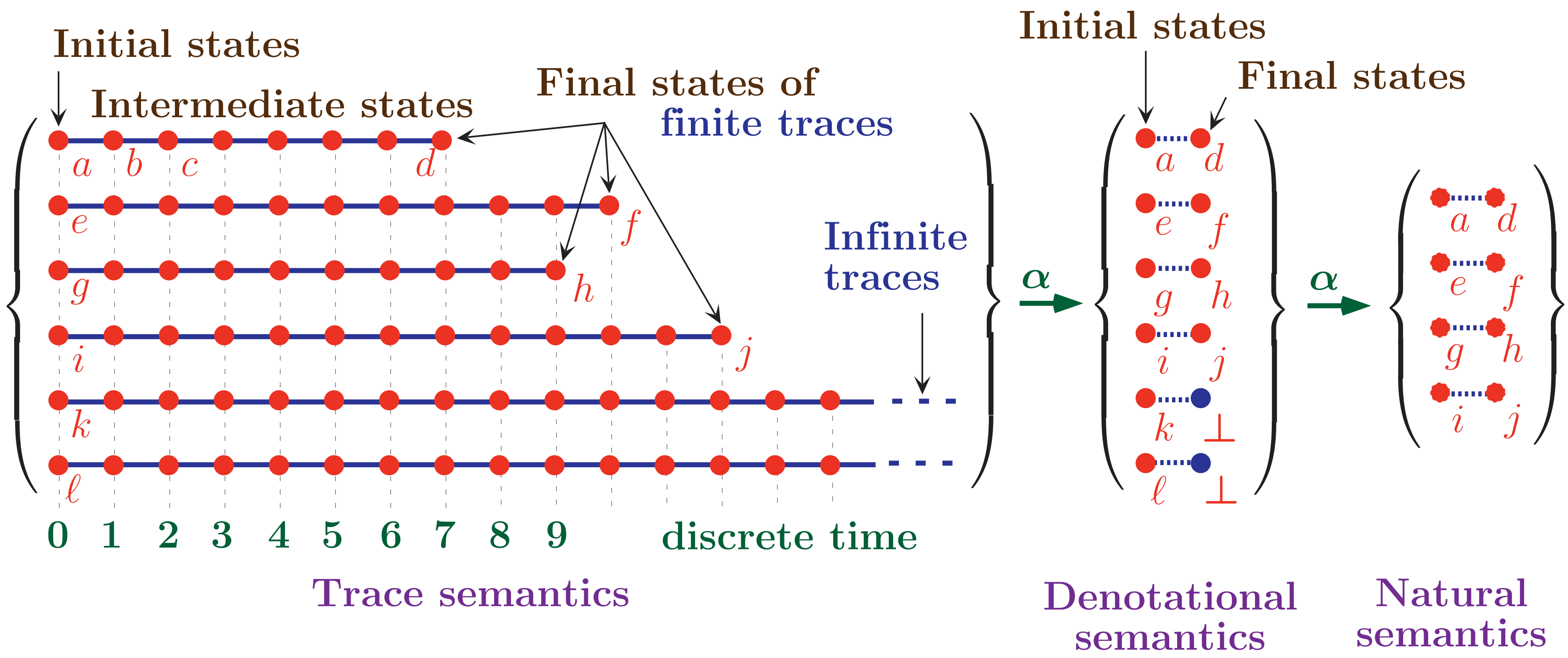
Familiar abstraction examples

concrete	abstract
citizen	
road network	
film	
car	
scientific article	
scientific article	
number	

Familiar abstraction examples

concrete	abstract
citizen	ID card
road network	road map
film	bill
car	trade mark
scientific article	abstract
scientific article	keywords
number	sign and/or parity

Examples of approximate semantics [3]



3 P. Cousot. *Constructive design of a hierarchy of semantics of a transition system by abstract interpretation*. To appear in TCS (2000).

Information loss

- Because of the information loss, not all questions can be definitely answered;

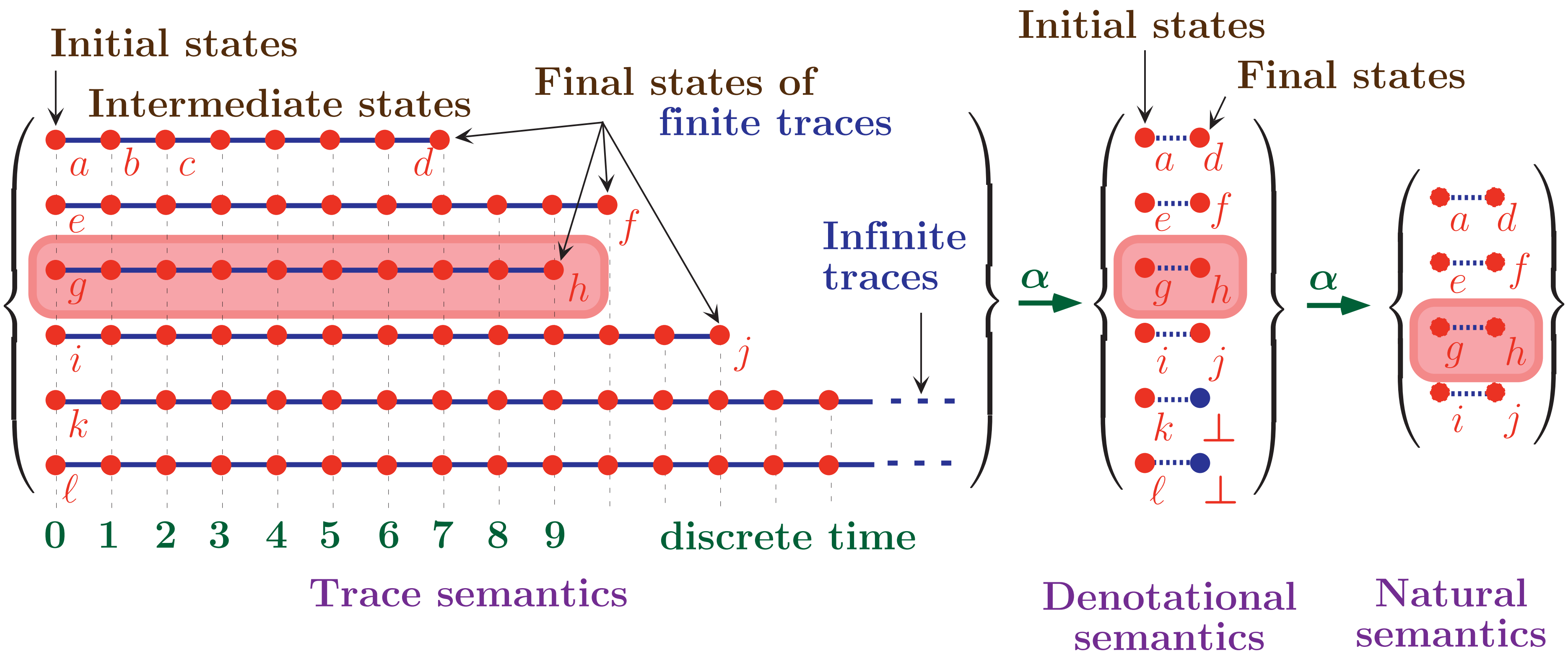
Information loss

- Because of the information loss, not all questions can be definitely answered;
- All answers given by the abstract semantics are always correct with respect to the concrete semantics.

Example of information loss

	Concrete ←		→ Abstract
Question	trace semantics	denotational semantics	natural semantics
Starting from state g can execution terminate in state h?	—	—	—

Semantics



Initial states
Intermediate states
Final states of
finite traces
Infinite
traces
a
b
c
d
e
f
g
h
i
j
k
ℓ
0 1 2 3 4 5 6 7 8 9
discrete time
Trace semantics
α
Initial states
Final states
a d
e f
g h
i j
k ⊥
ℓ ⊥
Denotational
semantics
α
a d
e f
g h
i j
Natural
semantics

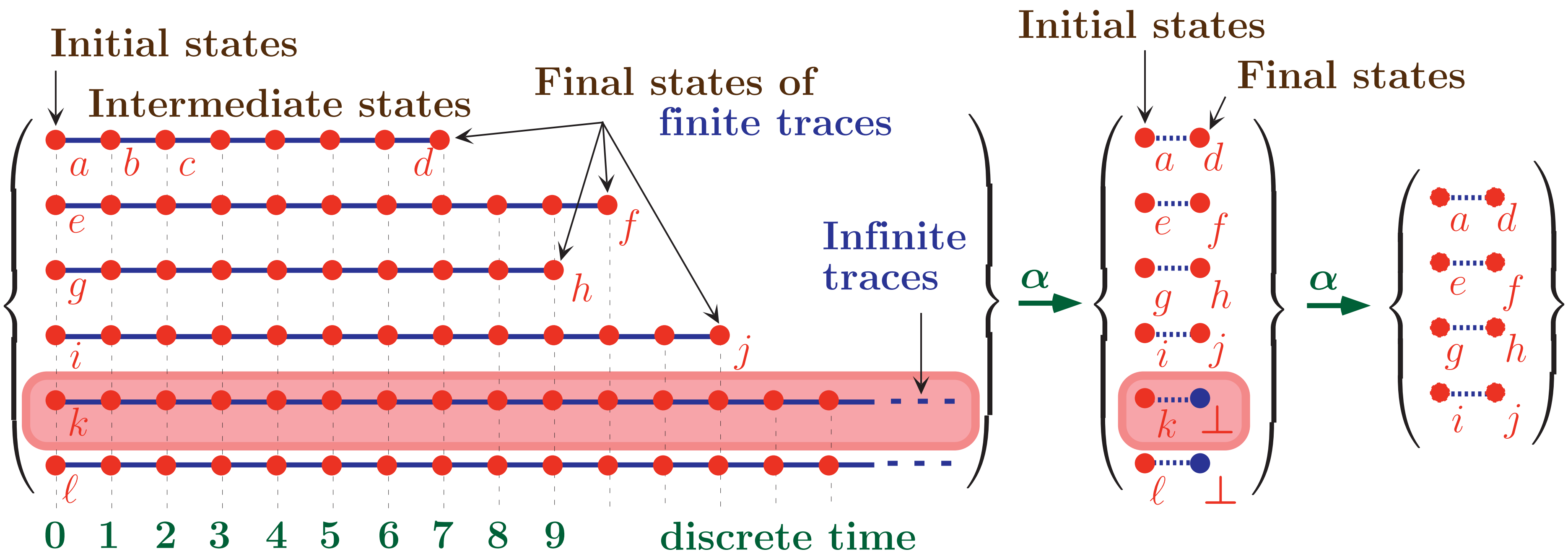
Example of information loss

	Concrete $\longleftarrow$		$\longrightarrow$Abstract
Question	trace semantics	denotational semantics	natural semantics
Starting from state g can execution terminate in state h?	yes	yes	yes

Example of information loss

	Concrete ⟵		⟶Abstract
Question	trace semantics	denotational semantics	natural semantics
Starting from state g can execution terminate in state h?	yes	yes	yes
Does execution starting from state k always terminate?	—	—	—

Semantics



Trace semantics

Denotational semantics

Natural semantics

Example of information loss

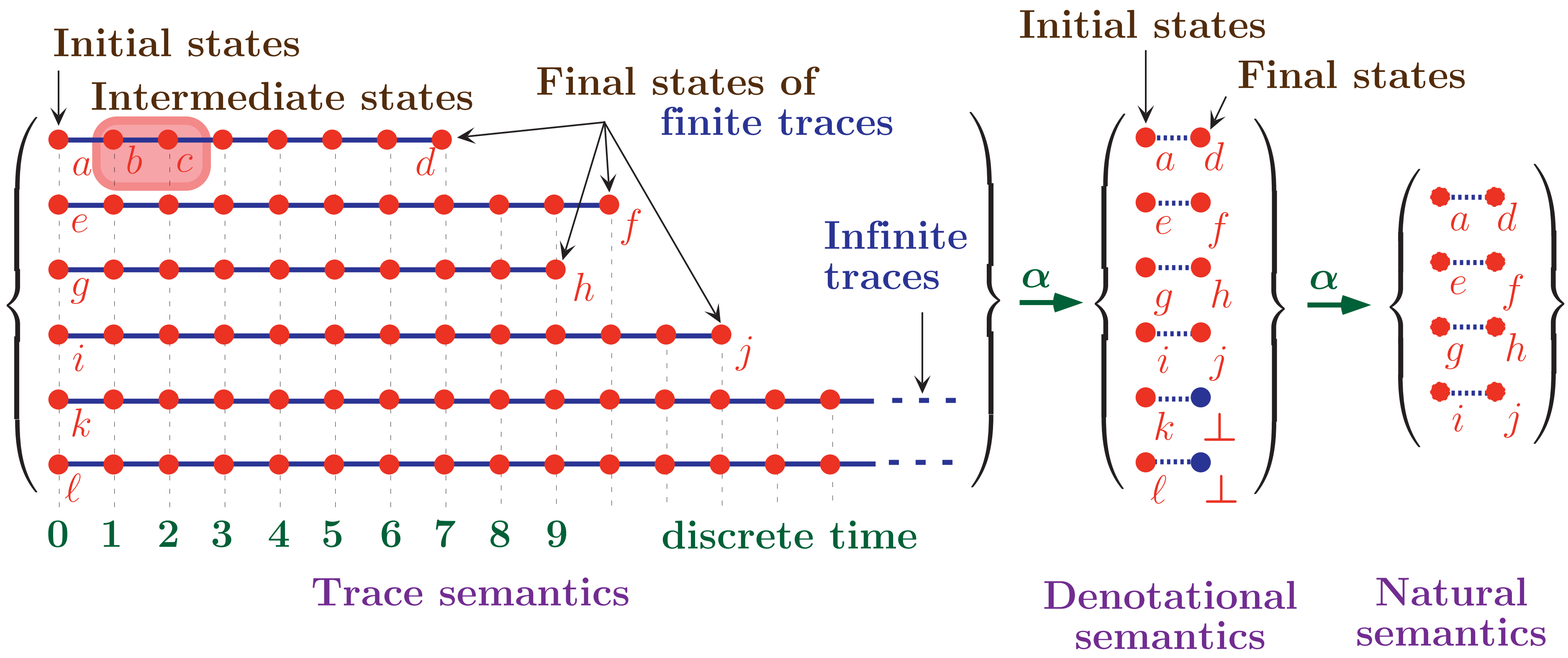
	Concrete $\longleftarrow$		$\longrightarrow$Abstract
Question	trace semantics	denotational semantics	natural semantics
Starting from state g can execution terminate in state h?	yes	yes	yes
Does execution starting from state k always terminate?	no	no	???

Example of information loss

Concrete $\longleftarrow$ $\longrightarrow$Abstract

Question	trace semantics	denotational semantics	natural semantics
Starting from state g can execution terminate in state h?	yes	yes	yes
Does execution starting from state k always terminate?	no	no	???
Can state b be immediately followed by state c?	—	—	—

Semantics



Initial states
Intermediate states
Final states of
finite traces
Infinite
traces
a
b
c
d
e
f
g
h
i
j
k
ℓ
0 1 2 3 4 5 6 7 8 9
discrete time
Trace semantics
α
Initial states
Final states
⊥
α
Denotational
semantics
Natural
semantics

Example of information loss

Concrete $\longleftarrow$ $\longrightarrow$Abstract

Question	trace semantics	denotational semantics	natural semantics
Starting from state g can execution terminate in state h?	yes	yes	yes
Does execution starting from state k always terminate?	no	no	???
Can state b be immediately followed by state c?	yes	???	???

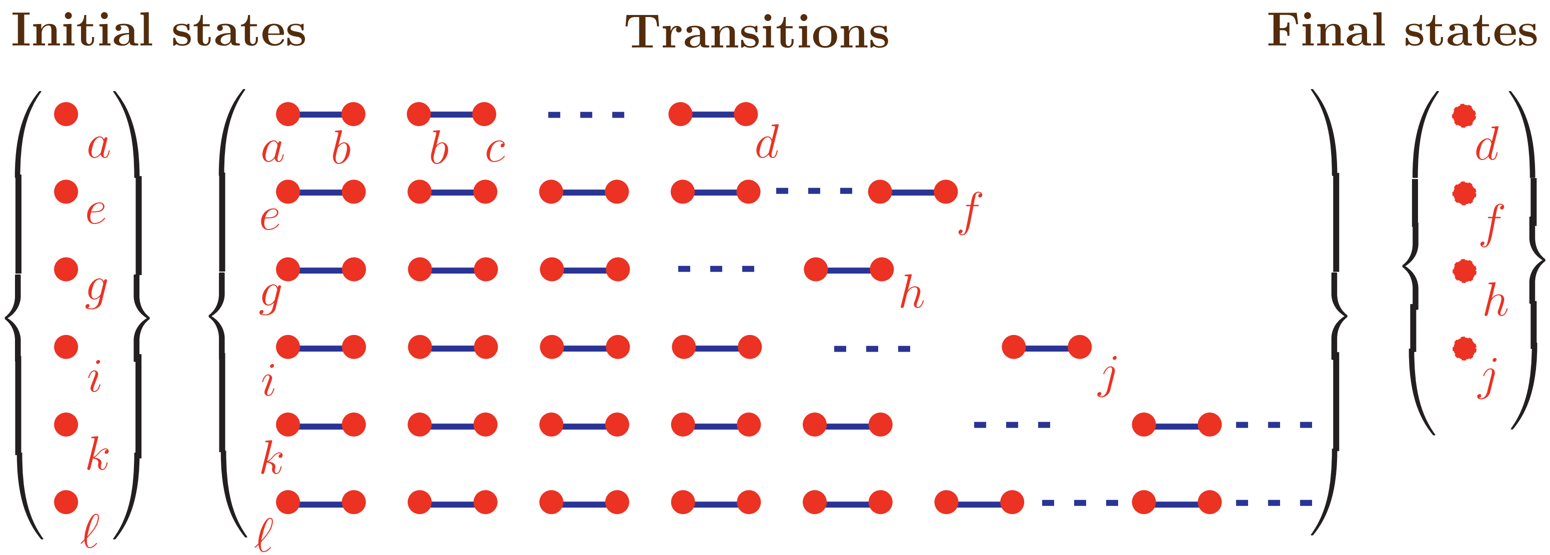
Example of information loss

Concrete $\longleftarrow$ $\longrightarrow$ Abstract

Question	trace semantics	denotational semantics	natural semantics
Starting from state g can execution terminate in state h?	yes	yes	yes
Does execution starting from state k always terminate?	no	no	???
Can state b be immediately followed by state c?	yes	???	???

The more concrete semantics can answer more questions. The more abstract semantics are more simple.

Example of non comparable approximated semantics [4]



Operational semantics

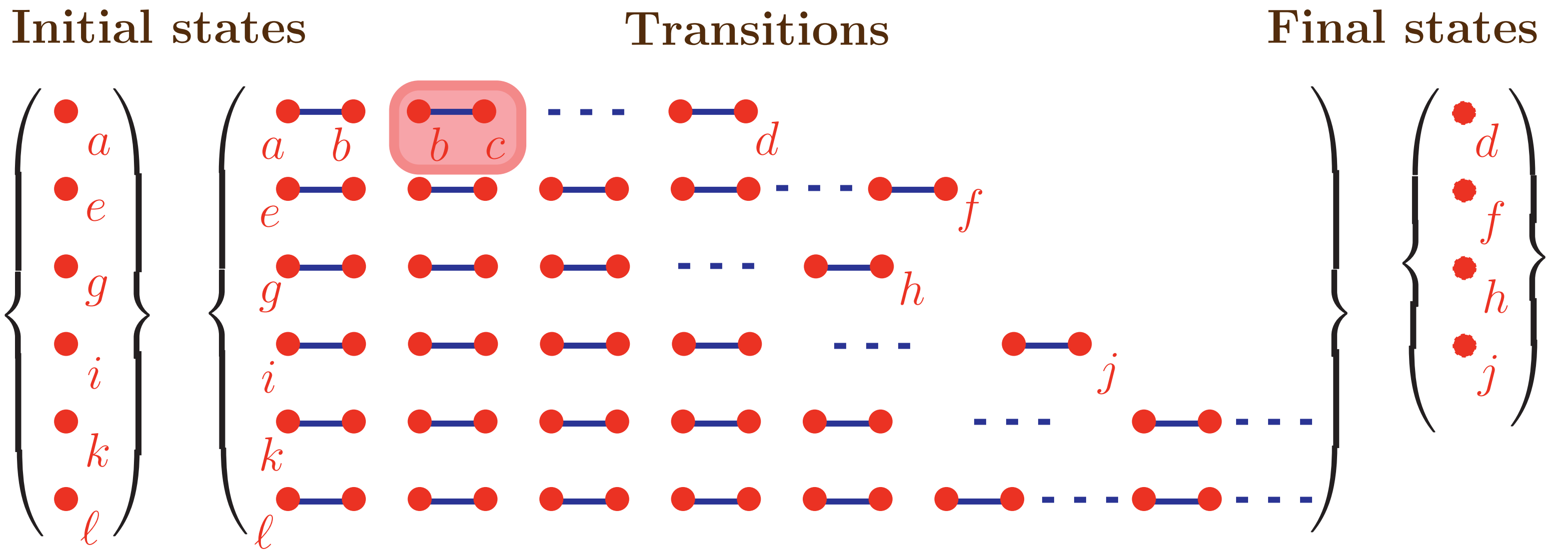
[4] P. Cousot. *Constructive design of a hierarchy of semantics of a transition system by abstract interpretation*. To appear in TCS (2000).

What is the information loss?

Concrete⟵ ⟶Abstract

Question	trace semantics	denotational semantics	natural semantics	operational semantics
Starting from state g can execution terminate in state h?	yes	yes	yes	—
Does execution starting from state k always terminate?	no	no	???	—
Can state b be immediately followed by state c?	yes	???	???	—

Operational semantics



Operational semantics

The information loss is incomparable

	Concrete$\longleftarrow$		$\longrightarrow$Abstract	Incomparable
Question	trace semantics	denotational semantics	natural semantics	operational semantics
Starting from state g can execution terminate in state h?	yes	yes	yes	???
Does execution starting from state k always terminate?	no	no	???	???
Can state b be immediately followed by state c?	yes	???	???	yes

Computable approximations

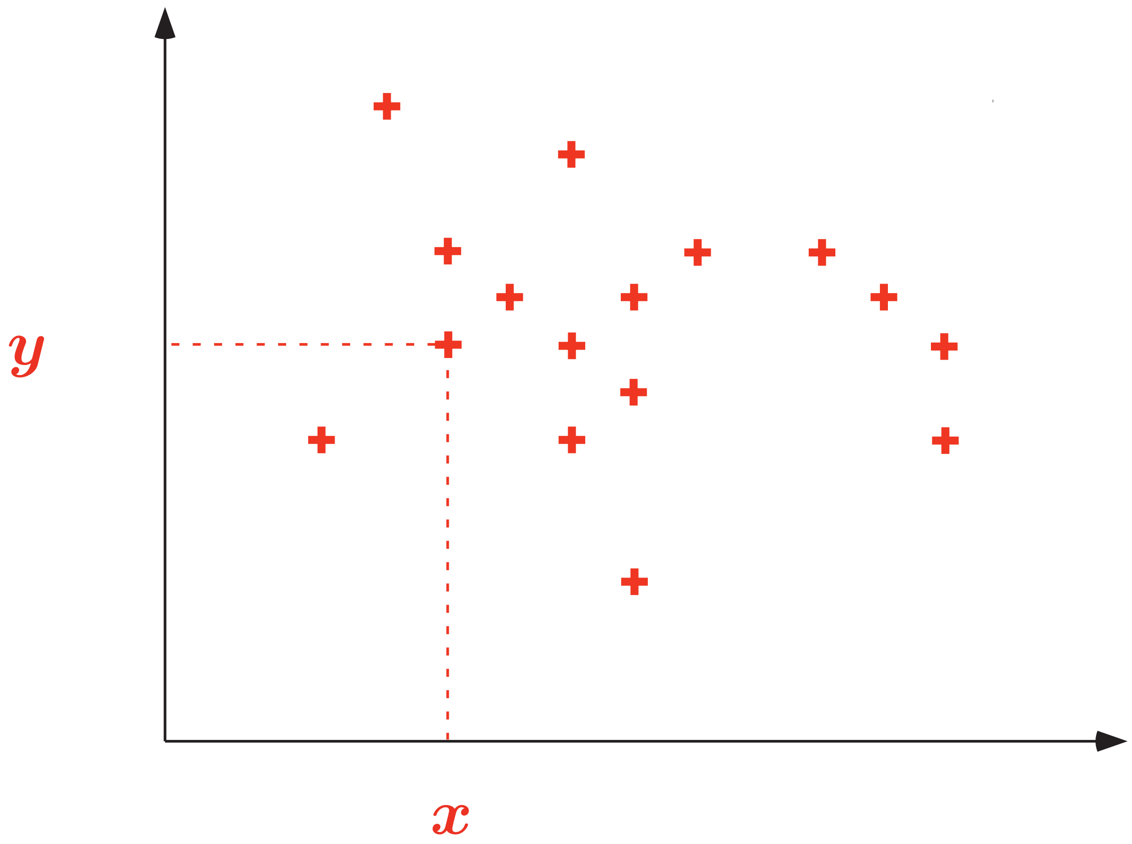
- If the approximation is rough enough, the abstraction of a semantics can lead to a version which is less precise but is effectively computable by a computer;

Computable approximations

- If the approximation is rough enough, the abstraction of a semantics can lead to a version which is less precise but is effectively computable by a computer;

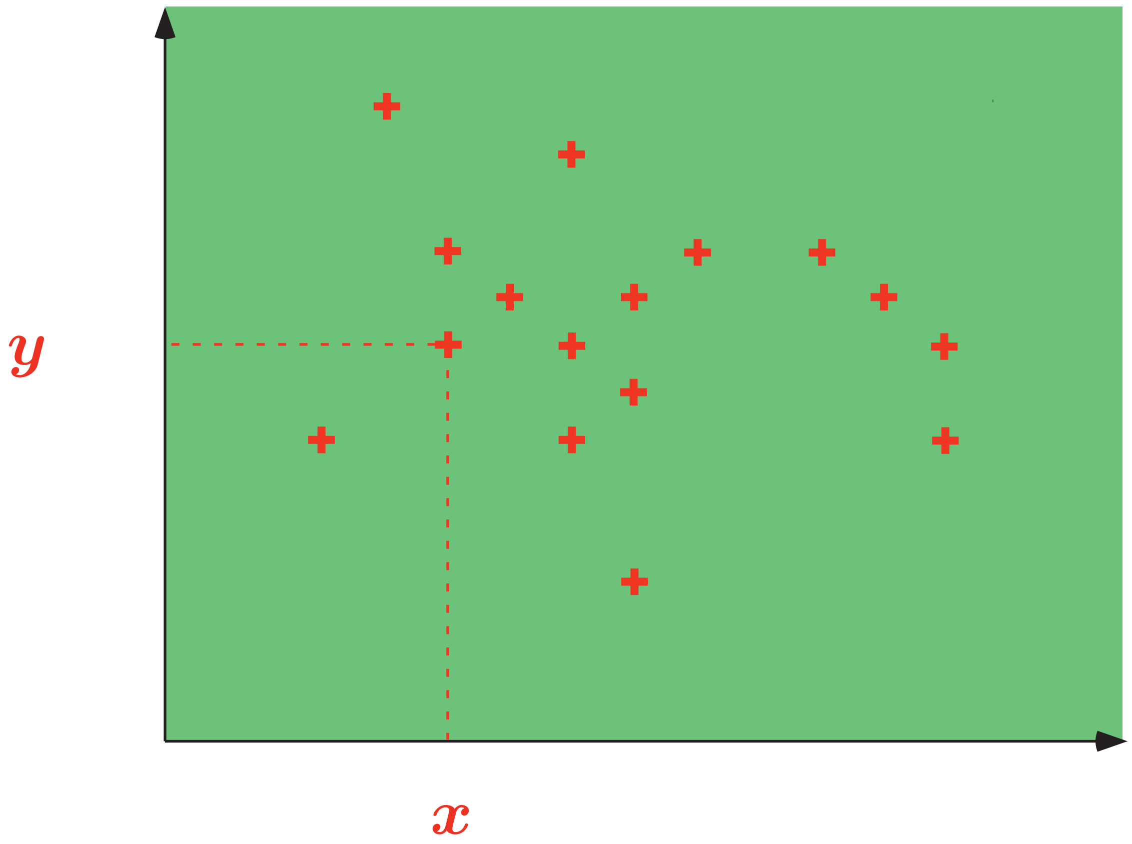
- By effective computation of the abstract semantics , the computer is able to analyze the behavior of programs and of software before and without executing them.

Example of computable approximations of an [in]finite set of points



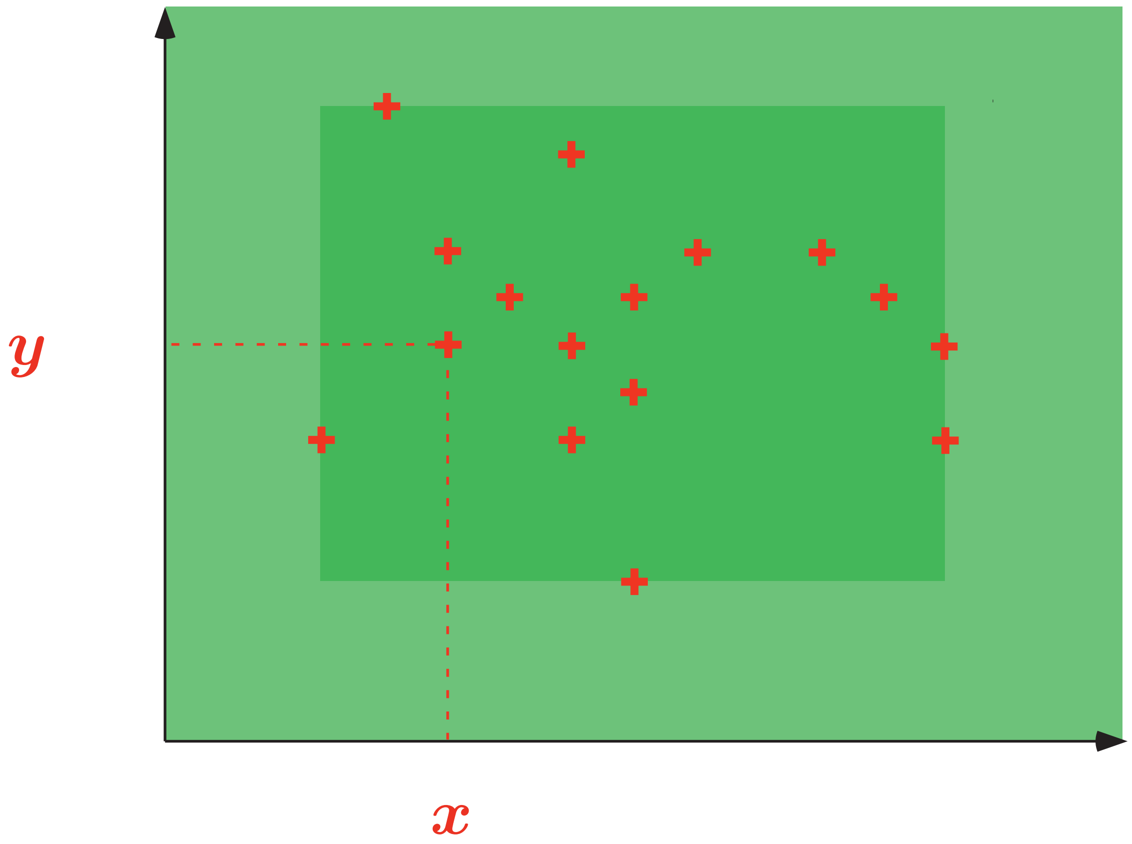
$$\{\ldots, \langle 19,\ 88\rangle, \ldots,\\ \langle 19,\ 99\rangle, \ldots\}$$

Example of computable approximations of an [in]finite set of points (signs)



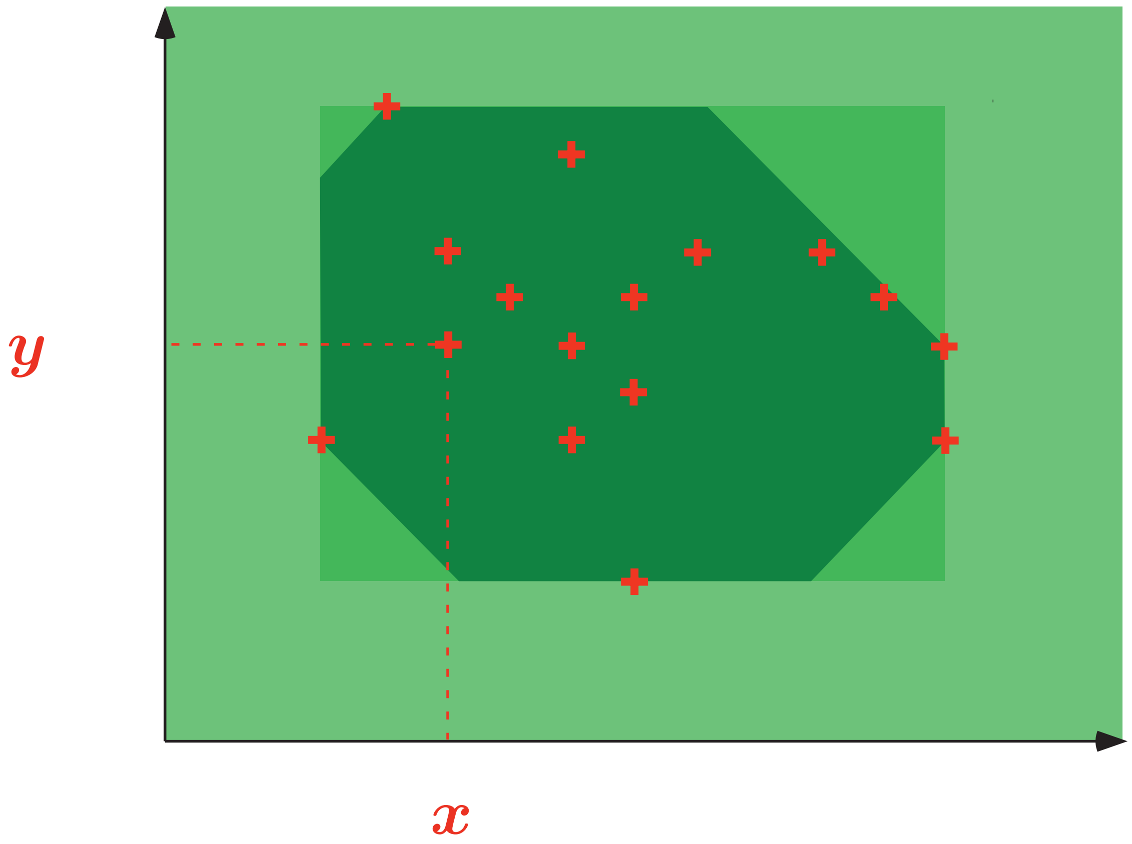
$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

Example of computable approximations of an [in]finite set of points (intervals)



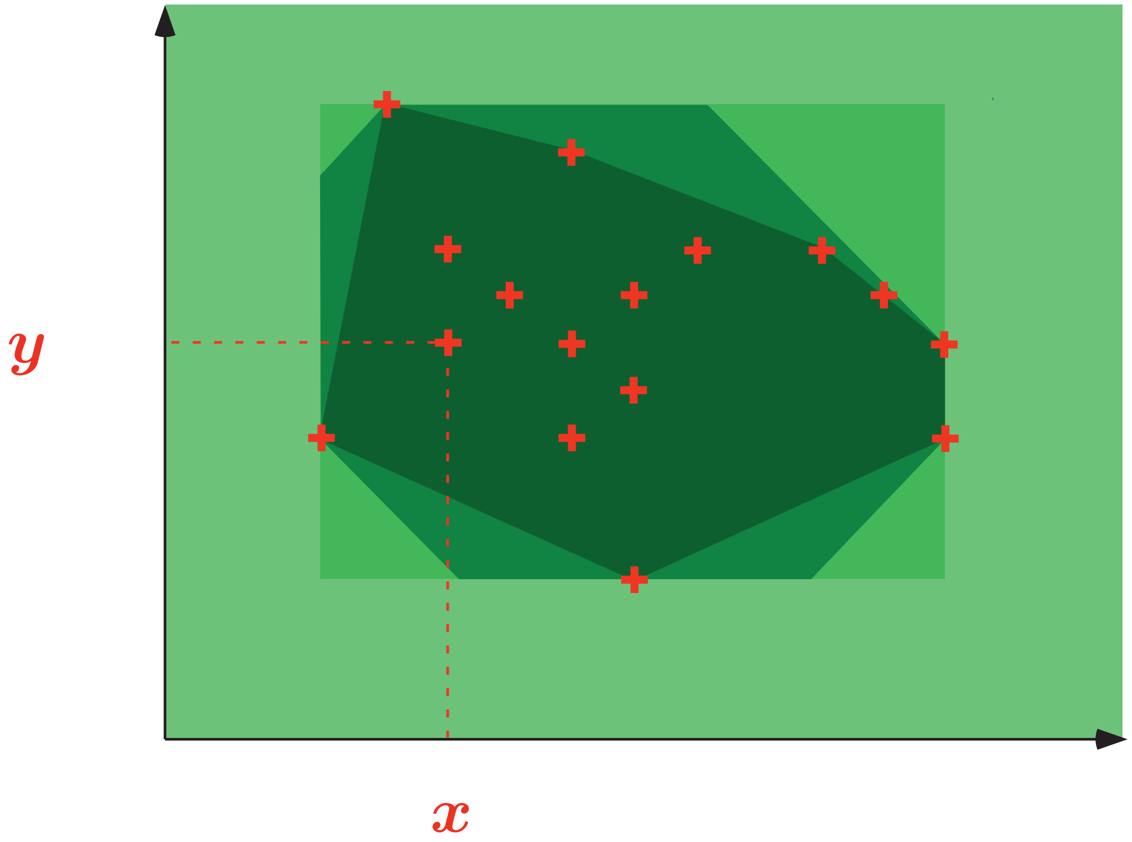
$$\begin{cases} x \in [19,\ 88] \\ y \in [19,\ 99] \end{cases}$$

Example of computable approximations of an [in]finite set of points (octagons)



$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 88 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

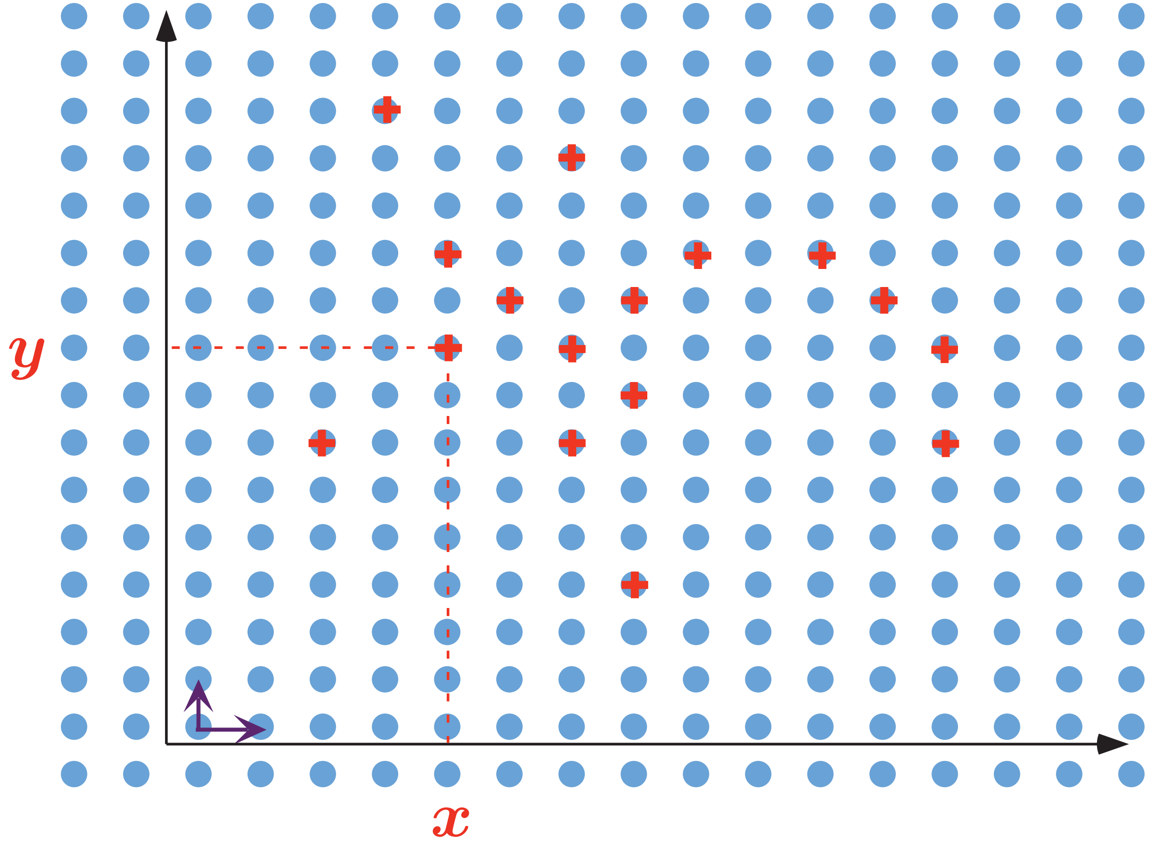
Example of computable approximations of an [in]finite set of points (polyhedra)



$$\begin{cases} 19x + 88y \leq 2000 \\ 19x + 99y \geq 0 \end{cases}$$

P. Cousot & N. Halbwachs, POPL'78

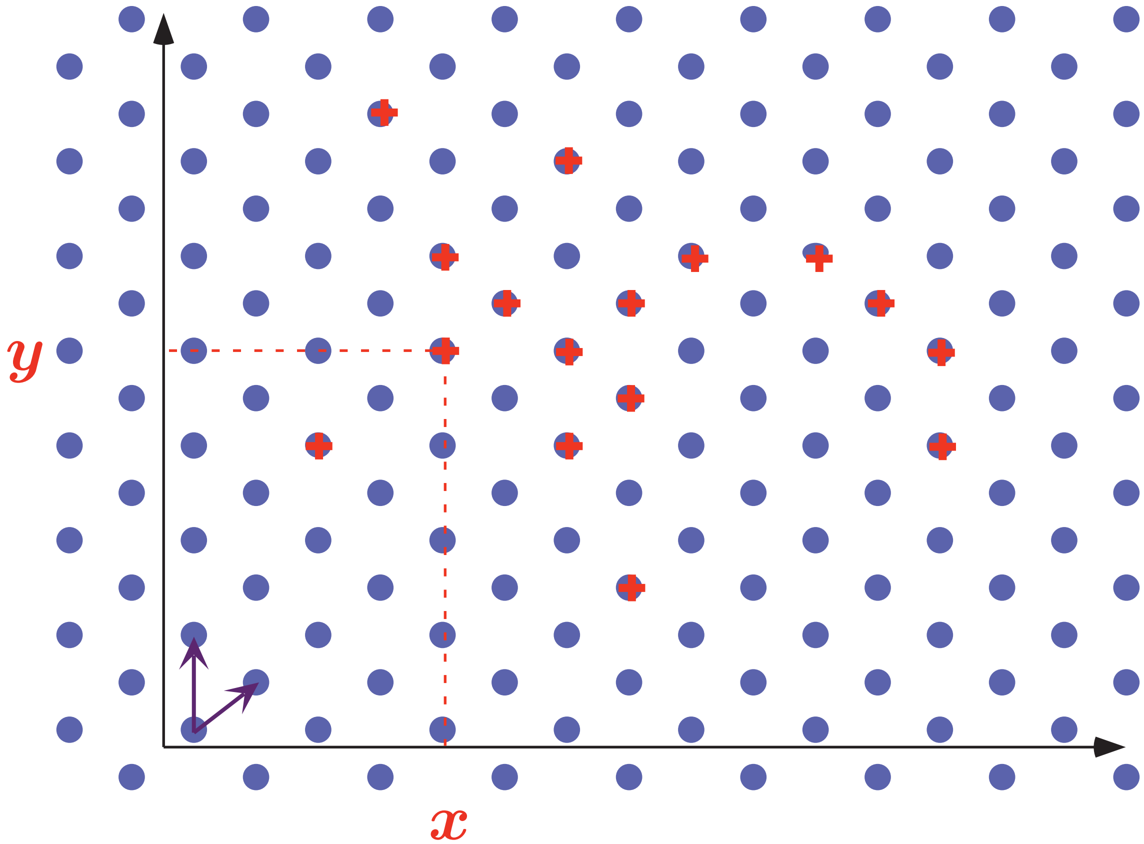
Example of computable approximations of an [in]finite set of points (simple congruences)



$$\begin{cases} x = 19 \bmod 88 \\ y = 19 \bmod 99 \end{cases}$$

thesis P. Granger, 1991

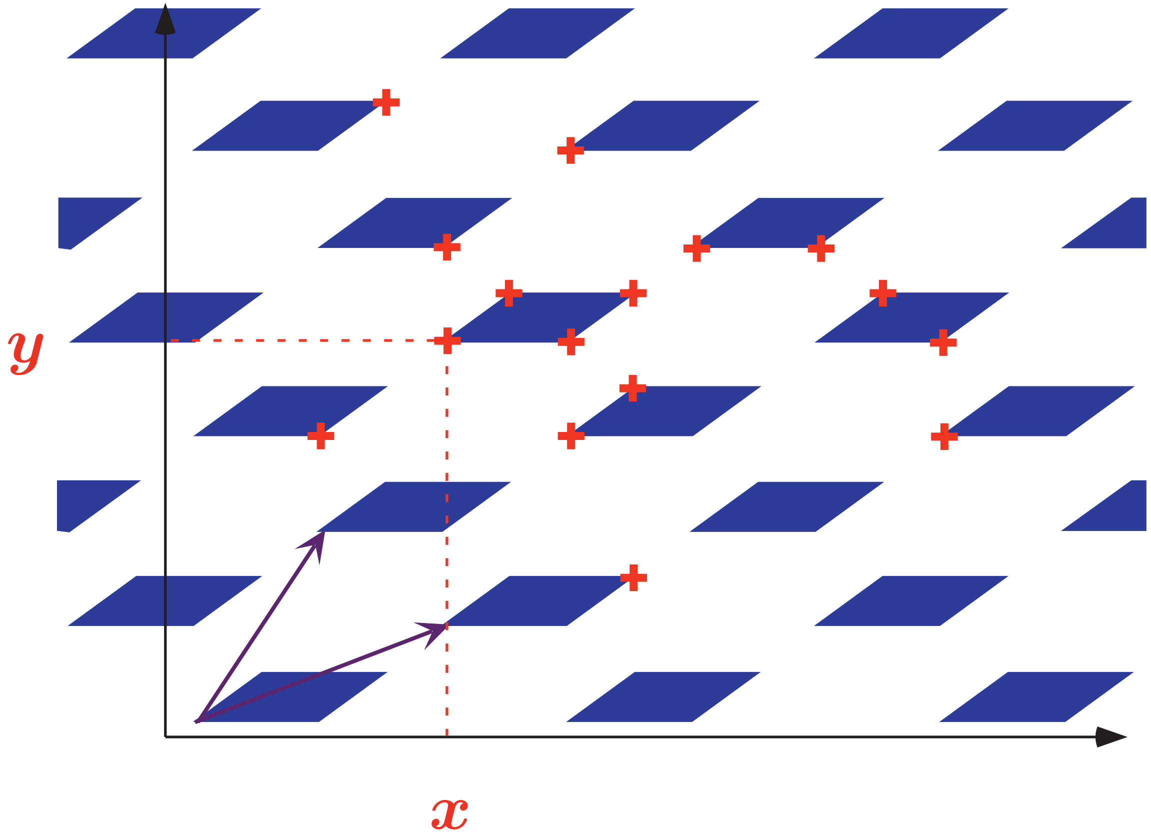
Example of computable approximations of an [in]finite set of points (linear congruences)



$$\begin{cases} 1x + 9y = 8 \bmod 8 \\ 1x - 9y = 9 \bmod 9 \end{cases}$$

thesis P. Granger, 1991

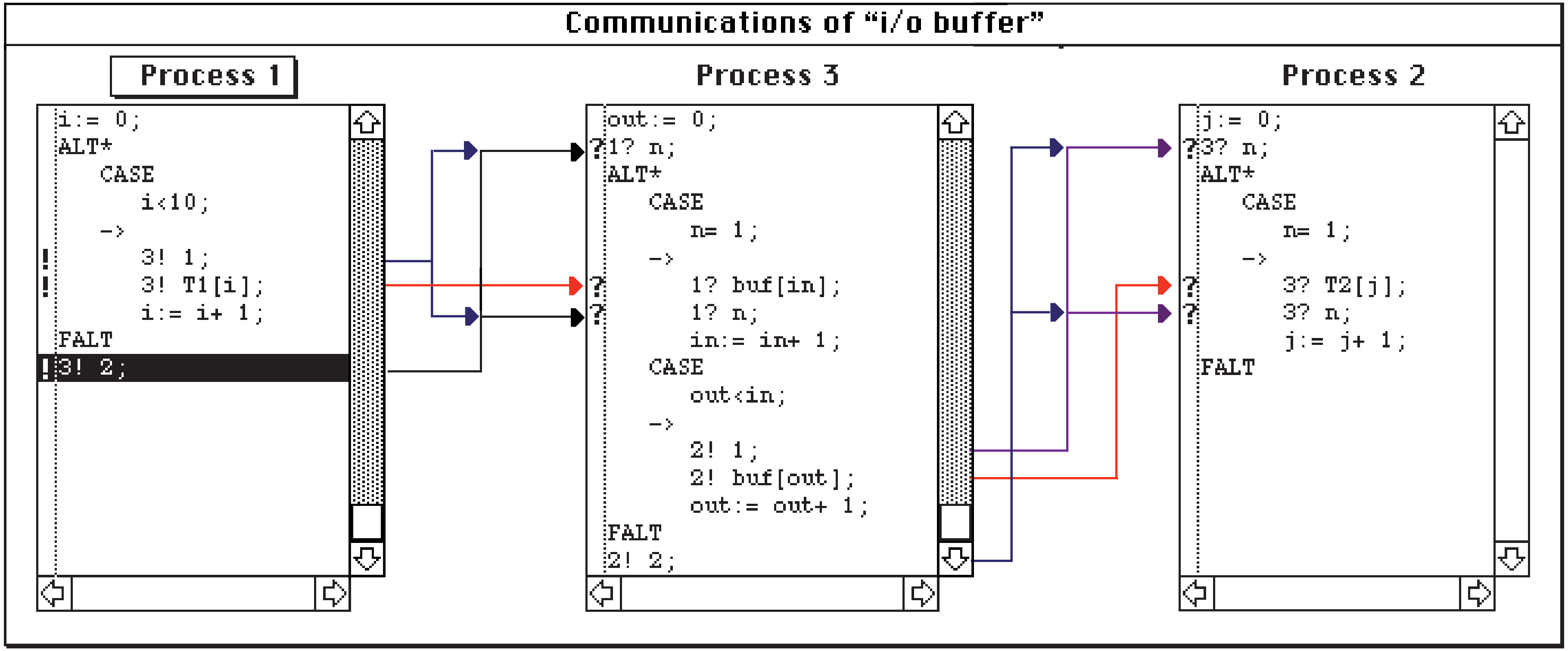
Example of computable approximations of an [in]finite set of points (trapezoidal linear congruences)



$$\begin{cases} 1x + 9y \in [0, 88] \bmod 10 \\ 1x - 9y \in [0, 99] \bmod 11 \end{cases}$$

thesis F. Masdupuy, 1993

Application of the congruence analysis: communications in OCCAM



thesis N. Mercouroff, 1990

More difficult: non numerical structures

More difficult: non numerical structures

- Most structures manipulated by programs are not numerical (so called *symbolic structures*);

More difficult: non numerical structures

- Most structures manipulated by programs are not numerical (so called *symbolic structures*);
- It is the case, for example, of the following structures:
  - control structures (call graphs, recursion trees),

More difficult: non numerical structures

- Most structures manipulated by programs are not numerical (so called *symbolic structures*);
- It is the case, for example, of the following structures:
  - control structures (call graphs, recursion trees),
  - data structures (search trees),

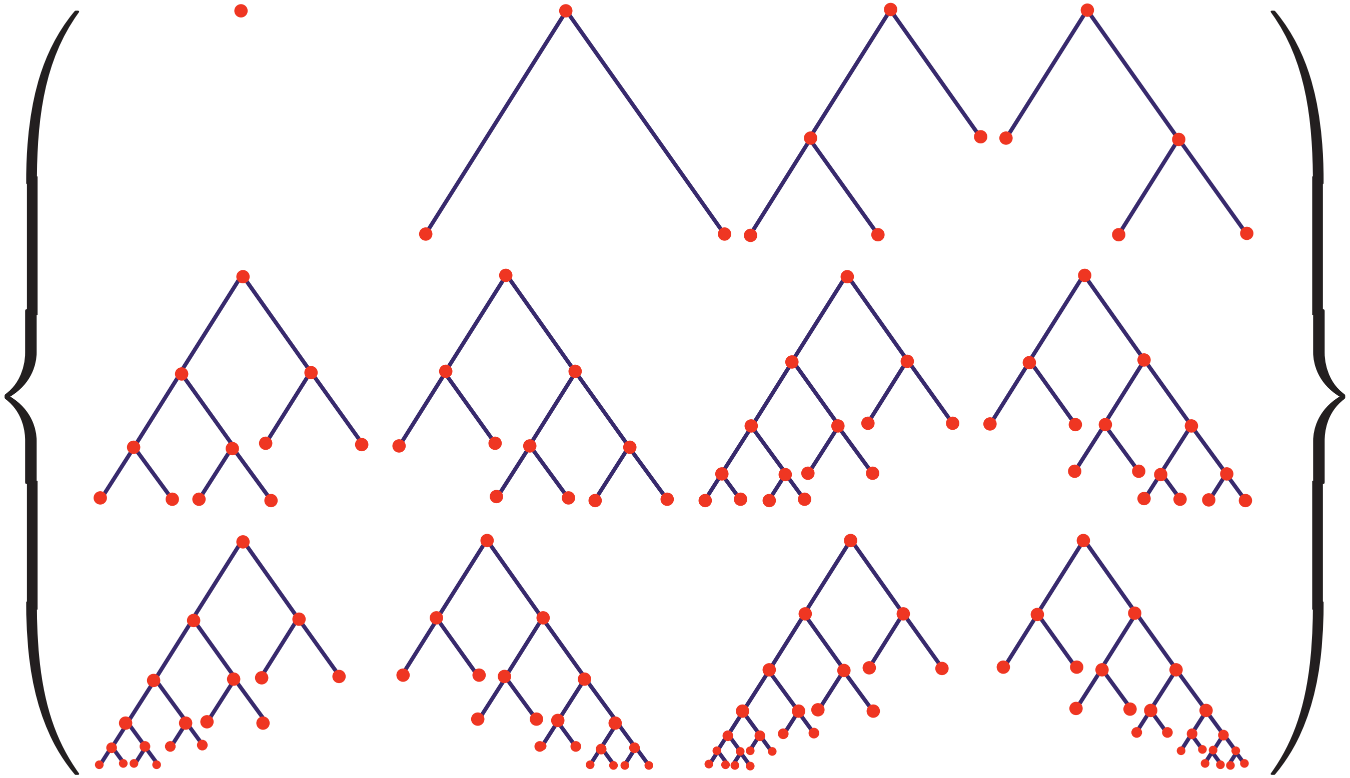
More difficult: non numerical structures

- Most structures manipulated by programs are not numerical (so called *symbolic structures*);
- It is the case, for example, of the following structures:
  - control structures (call graphs, recursion trees),
  - data structures (search trees),
  - communication structures (distributed programs),

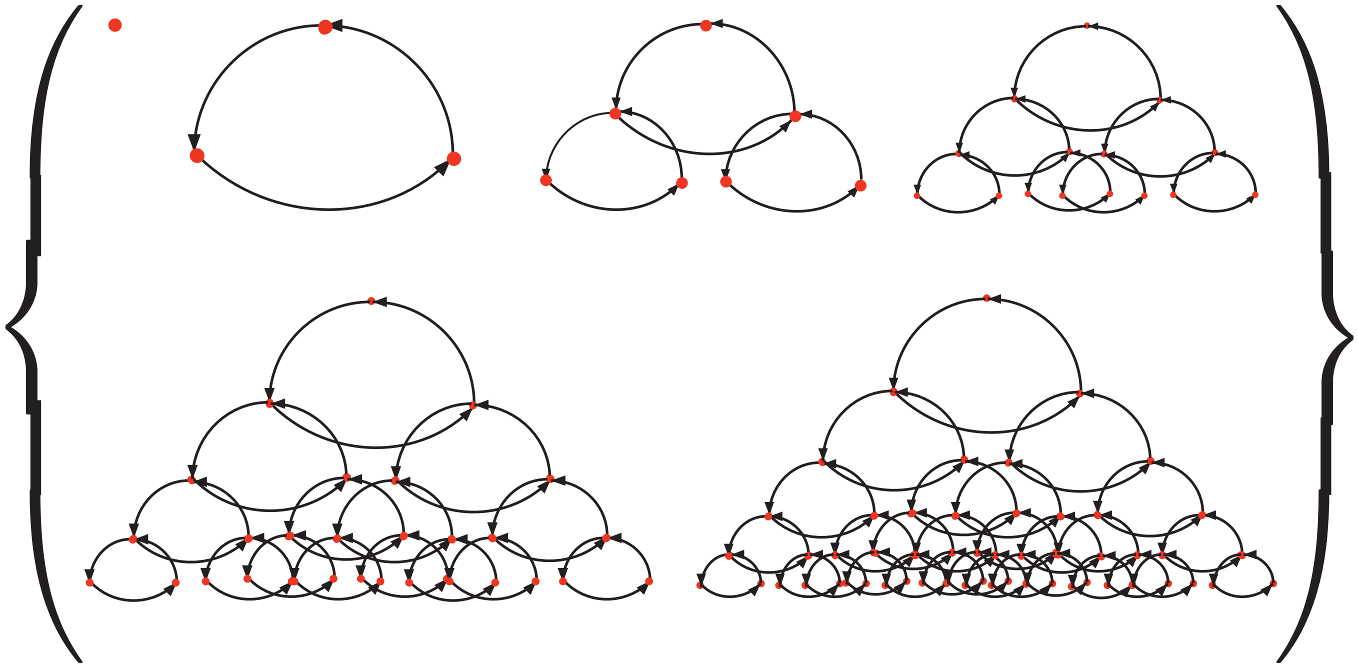
More difficult: non numerical structures

- Most structures manipulated by programs are not numerical (so called *symbolic structures*);
- It is the case, for example, of the following structures:
  - control structures (call graphs, recursion trees),
  - data structures (search trees),
  - communication structures (distributed programs),
  - information transfer structures (mobile programs), etc.

Example 1: (infinite) sets of (infinite) decorated trees



Example 2: (infinite) set of (infinite) decorated graphs



Precise compact approximations

Precise compact approximations

- It is very difficult to find compact and expressive computer representations of such sets of objects (languages, automata, trees, graphs, etc.)

Precise compact approximations

- It is very difficult to find compact and expressive computer representations of such sets of objects (languages, automata, trees, graphs, etc.) such that:
  - the various set-theoretic operations can be efficiently implemented;

Precise compact approximations

- It is very difficult to find compact and expressive computer representations of such sets of objects (languages, automata, trees, graphs, etc.) such that:
  - the various set-theoretic operations can be efficiently implemented;
  - the memory size does not explode combinatorially for complex and/or irregular sets;

Precise compact approximations

- It is very difficult to find compact and expressive computer representations of such sets of objects (languages, automata, trees, graphs, etc.) such that:
  - the various set-theoretic operations can be efficiently implemented;
  - the memory size does not explode combinatorially for complex and/or irregular sets;
  - the approximations remain precise.

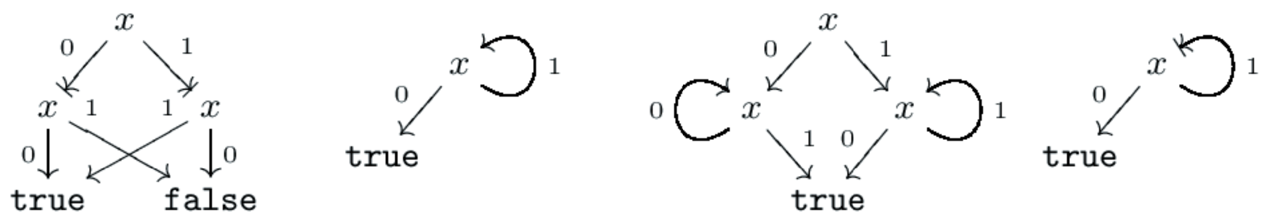
Precise compact approximations

- It is very difficult to find compact and expressive computer representations of such sets of objects (languages, automata, trees, graphs, etc.) such that:
  - the various set-theoretic operations can be efficiently implemented;
  - the memory size does not explode combinatorially for complex and/or irregular sets;
  - the approximations remain precise.

theses I. Stransky, 1988, A. Deutsch, 1992, A. Venet, 1998, L. Mauborgne, 1999, F. Védrine, 2000

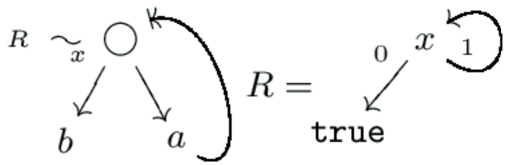
Example of compact approximations of infinite sets of infinite trees

Binary Decision Graphs:

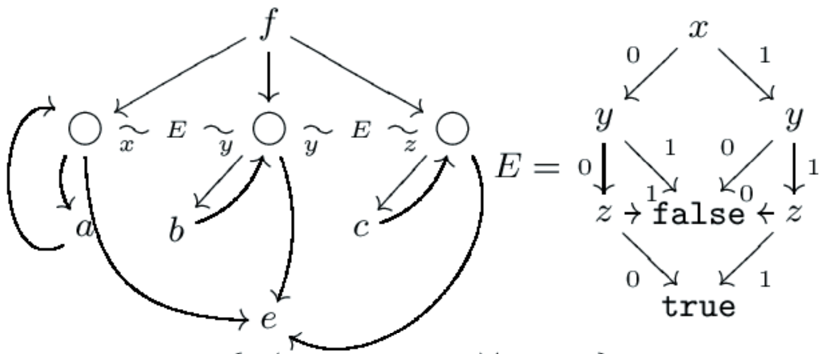


$\{0^\omega, 1^\omega\}$ infinite number of 0's fair vectors ends by 0^ω

Tree schemata:



$\{a^n b | n \in \mathbb{N}\}$



$\{f(a^n e, b^n e, c^n e) | n \in \mathbb{N}\}$

Note that E is the equality relation.

these L. Mauborgne, 1999

Program Static Analysis

Difficulty of programming

- Large scale computer programming is very difficult;

Difficulty of programming

- Large scale computer programming is very difficult;
- Reasoning on large programs is very difficult;

Difficulty of programming

- Large scale computer programming is very difficult;
- Reasoning on large programs is very difficult;
- Errors are quite frequent.

Example 1: first year exam at the École polytechnique

What is the effect of the following PASCAL program:

```
program P (input, output);
 procedure NewLine; begin writeln end;
 procedure P (X : integer; procedure Q);
  procedure R;
   begin write(X); Q; end;
 begin
  if X > 0 then begin R; P(X - 1, R); end;
 end;
begin
 P(5, NewLine);
end.
```

Example 1: first year exam at the École polytechnique

What is the effect of the following PASCAL program:

```
program P (input, output);
 procedure NewLine; begin writeln end;
 procedure P (X : integer; procedure Q);
  procedure R;
   begin write(X); Q; end;
 begin
  if X > 0 then begin R; P(X - 1, R); end;
 end;
begin
 P(5, NewLine);
end.
```

```
5
4    5
3    4    5
2    3    4    5
1    2    3    4    5
```

Less than 5% of the answers are correct!

Example 2: first year exam at the École polytechnique

Prove that the following program prints the value ≥ 91:

```
program MacCarthy  (input,output);
 var x, m : integer;
 function MC(n : integer) : integer;
  begin
   if n > 100 then MC := n - 10
   else MC := MC(MC(n + 11));
  end;
begin
 read(x); m := MC(x); writeln(m);
end.
```

Example 2: first year exam at the École polytechnique

Prove that the following program prints the value $\geq\ 91$:

```
program MacCarthy  (input,output);
 var x, m : integer;
 function MC(n : integer) : integer;
  begin
   if n > 100 then MC := n - 10
   else MC := MC(MC(n + 11));
  end;
begin
 read(x); m := MC(x); writeln(m);
end.
```

Less than 50 % of the proofs given as answers are correct!

Program static analysis

- Objective: discover programming errors before they lead to disastrous catastrophes!

Program static analysis

- Objective: discover programming errors before they lead to disastrous catastrophes!
- Program static analysis uses *abstract interpretation* to derive, from a standard semantics, an approximate and computable semantics;

Program static analysis

- Objective: discover programming errors before they lead to disastrous catastrophes!
- Program static analysis uses *abstract interpretation* to derive, from a standard semantics, an approximate and computable semantics;
- It follows that the computer is able to analyze the behavior of software before and without executing it;

Program static analysis

- Objective: discover programming errors before they lead to disastrous catastrophes!
- Program static analysis uses *abstract interpretation* to derive, from a standard semantics, an approximate and computable semantics;
- It follows that the computer is able to analyze the behavior of software before and without executing it;
- This is essential for computer-based safety-critical systems (for example: planes, trains, launchers, nuclear plants, etc.).

Example: interval analysis (1975) [5]

Program to be analyzed:

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

[5] P. Cousot & R. Cousot, ISOP'76.

Example: interval analysis (1975) [5]

Equations (abstract interpretation of the semantics):

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

5 P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration, initialization:

```
   x := 1;
1:
   while x < 10000 do
2:
         x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = \emptyset \\ X_2 = \emptyset \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = \emptyset \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,1] \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,1] \\ X_3 = [2,2] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration:

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,2] \\ X_3 = [2,2] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration: **convergence?**

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,2] \\ X_3 = [2,3] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration: **convergence??**

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 3] \\ X_3 = [2, 3] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration: **convergence???**

```
   x := 1;
1:
   while x < 10000 do
2:
         x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,3] \\ X_3 = [2,4] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration: **convergence????**

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 4] \\ X_3 = [2, 4] \\ X_4 = \emptyset \end{cases}$$

5 P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration: **convergence?????**

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,4] \\ X_3 = [2,5] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration: **convergence??????**

```
   x := 1;
1:
   while x < 10000 do
2:
         x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,5] \\ X_3 = [2,5] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Increasing chaotic iteration: **convergence???????**

```
   x := 1;
1:
   while x < 10000 do
2:
        x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 5] \\ X_3 = [2, 6] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Convergence speed-up by extrapolation:

```
   x := 1;
1:
   while x < 10000 do
2:
         x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, +\infty] \quad \Leftarrow \text{widening} \\ X_3 = [2,6] \\ X_4 = \emptyset \end{cases}$$

5 P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Decreasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, +\infty] \\ X_3 = [2, +\infty] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Decreasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, 9999] \\ X_3 = [2, +\infty] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Decreasing chaotic iteration:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, 9999] \\ X_3 = [2, +10000] \\ X_4 = \emptyset \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Final solution:

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, 9999] \\ X_3 = [2, +10000] \\ X_4 = [+10000, +10000] \end{cases}$$

---

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

Example: interval analysis (1975) [5]

Result of the interval analysis:

```
   x := 1;
1: {x = 1}
   while x < 10000 do
2: {x ∈ [1, 9999]}
         x := x + 1
3: {x ∈ [2, +10000]}
   od;
4: {x = 10000}
```

$$\begin{cases} X_1 = [1, 1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1, 1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1, 1] \\ X_2 = [1, 9999] \\ X_3 = [2, +10000] \\ X_4 = [+10000, +10000] \end{cases}$$

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

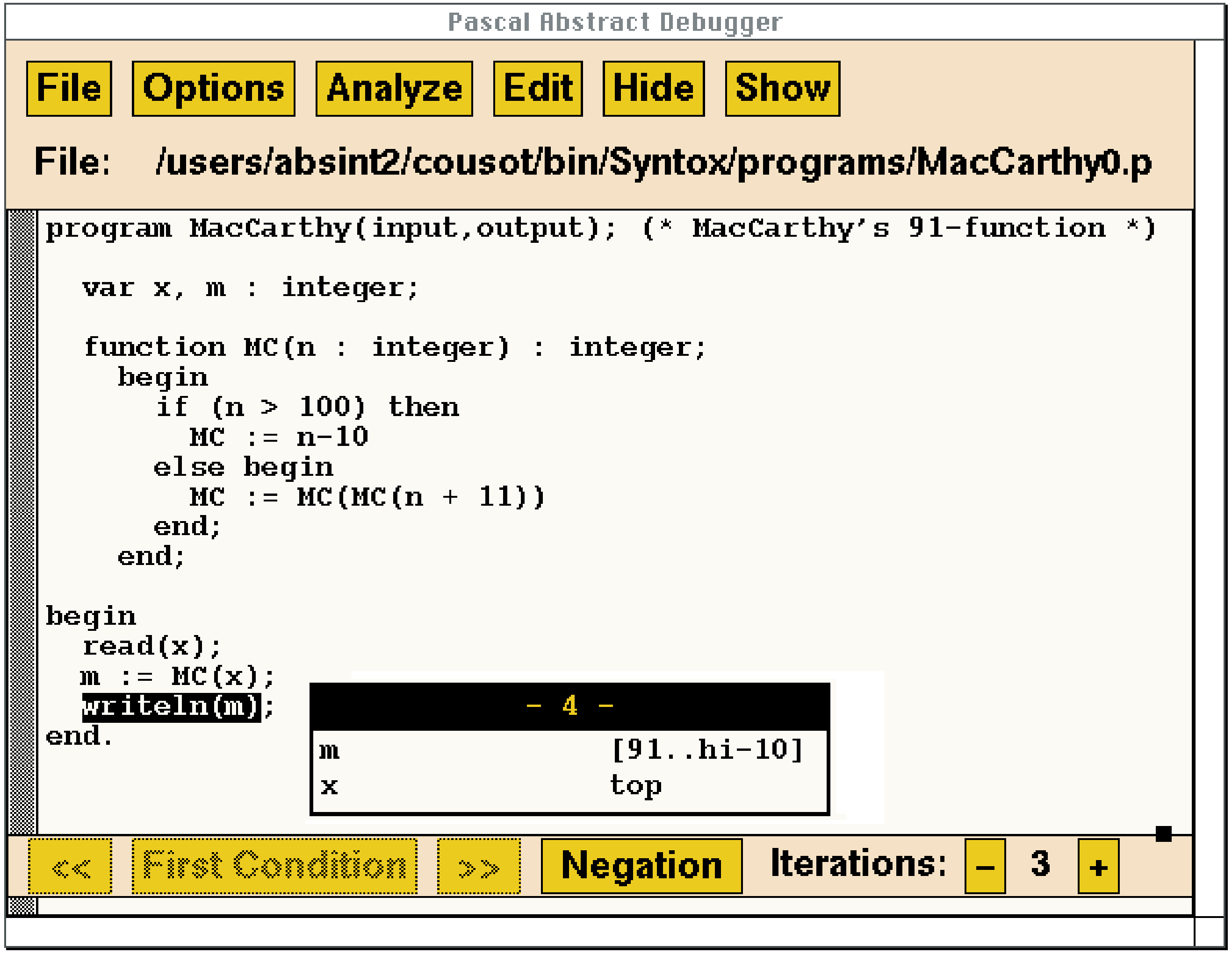
Example: interval analysis (1975) [5]

Exploitation of the result of the interval analysis:

```
    x := 1;
1: {x = 1}
    while x < 10000 do
2: {x ∈ [1, 9999]}
        x := x + 1          ←—— no overflow
3: {x ∈ [2, +10000]}
    od;
4: {x = 10000}
```

[5] P. Cousot & R. Cousot, ISOP'1976, POPL'77.

For imperative languages like PASCAL ...



thesis F. Bourdoncle, 1992

An impressive application (1996/97)

An impressive application (1996/97)

- A. Deutsch uses abstract interpretation (including interval analysis) for the static analysis of the embedded ADA software of the Ariane 5 launcher [6];

[6] Flight software (60,000 lines of Ada code) and Inertial Measurement Unit (30,000 lines of Ada code).

An impressive application (1996/97)

- A. Deutsch uses abstract interpretation (including interval analysis) for the static analysis of the embedded ADA software of the Ariane 5 launcher [6];
- Automatic detection of the definiteness, potentiality, impossibility or inaccessibility of run-time errors [7];

---

6 Flight software (60,000 lines of Ada code) and Inertial Measurement Unit (30,000 lines of Ada code).

7 such as scalar and floating-point overflows, array index errors, divisions by zero and related arithmetic exceptions, uninitialized variables, data races on shared data structures, etc.

An impressive application (1996/97)

- A. Deutsch uses abstract interpretation (including interval analysis) for the static analysis of the embedded ADA software of the Ariane 5 launcher [6];
- Automatic detection of the definiteness, potentiality, impossibility or inaccessibility of run-time errors [7];
- Success for the 502 & 503 flights and the ARD [8].

---

6 Flight software (60,000 lines of Ada code) and Inertial Measurement Unit (30,000 lines of Ada code).

7 such as scalar and floating-point overflows, array index errors, divisions by zero and related arithmetic exceptions, uninitialized variables, data races on shared data structures, etc.

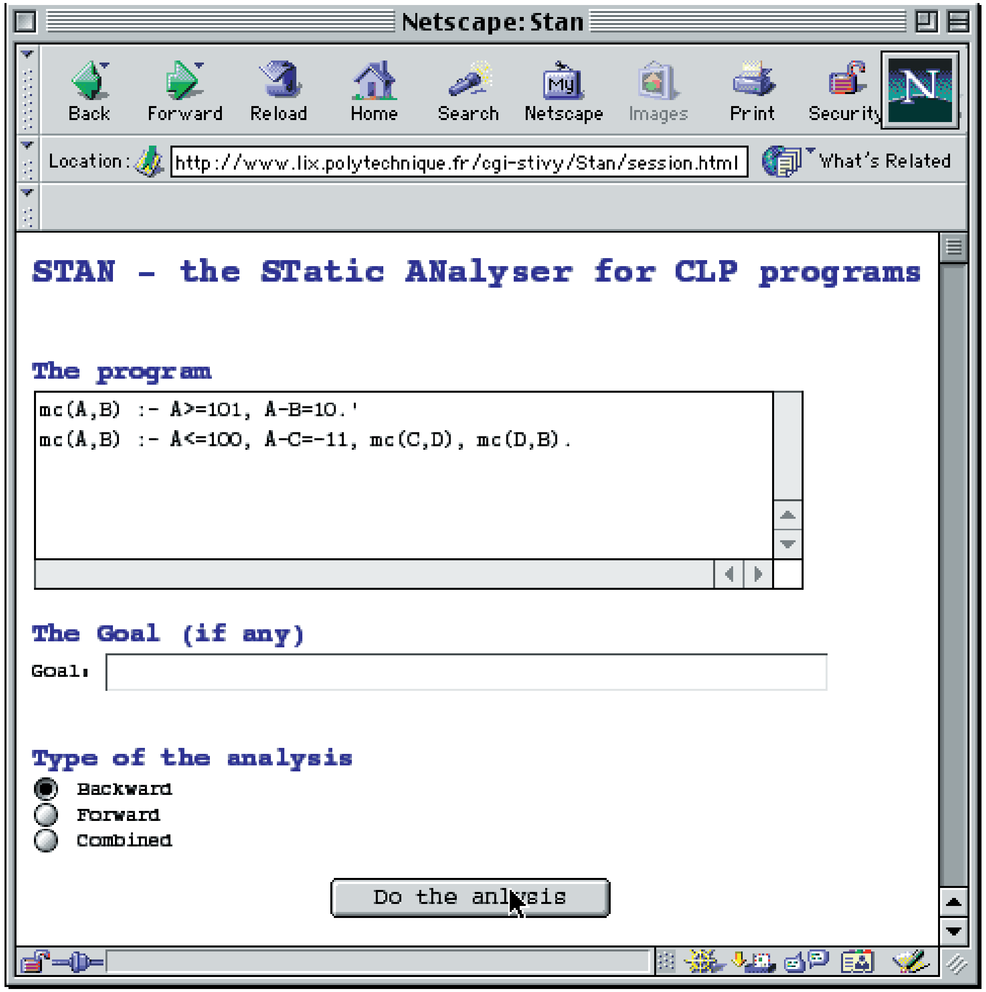
8 Atmospheric Reentry Demonstrator: module coming back to earth.

Some other recent applications of static analysis by abstract interpretation

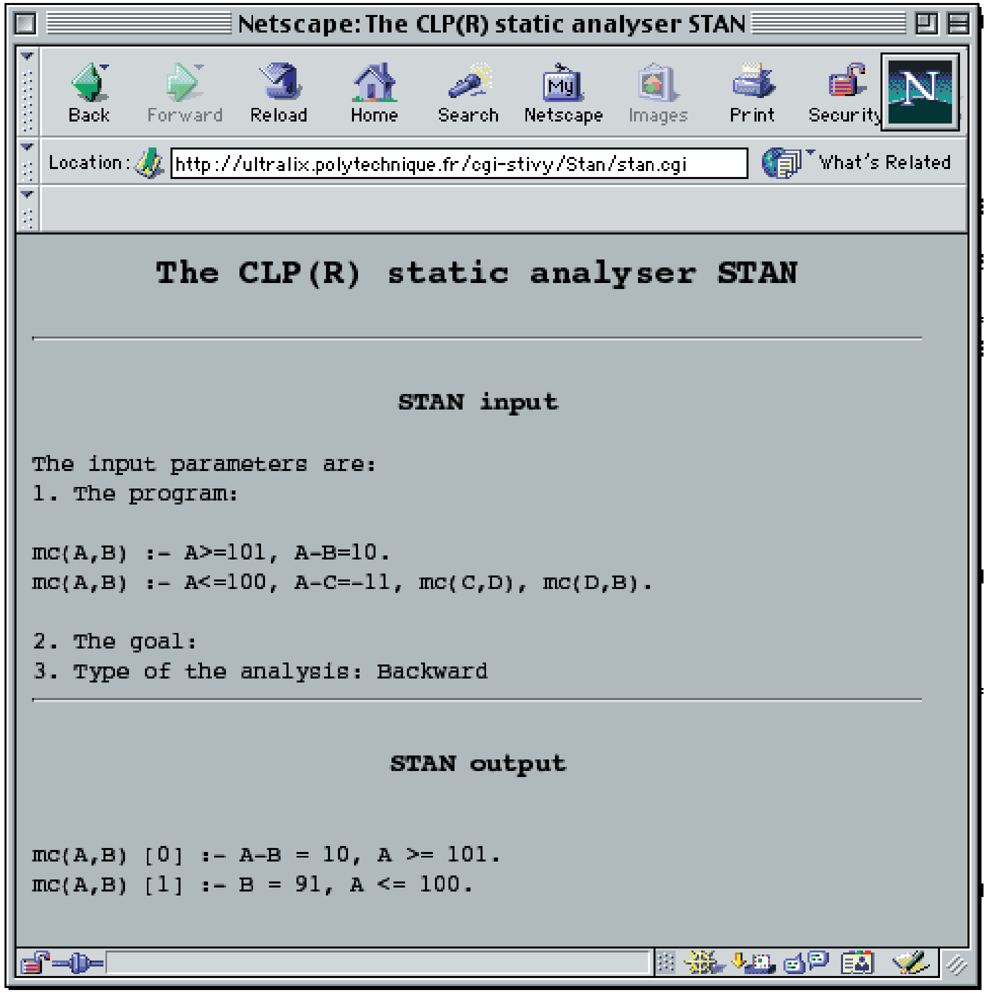
- program transformation & optimization;
- abstract model-checking of infinite systems;
- abstract testing;
- type inference (for undecidable systems);
- mobile code communication topology;
- automatic differentiation;
- ...

theses F. Bourdoncle, 1992, B. Monsuez, 1994,A. Venet, 1998, F. Védrine, 2000, R. Cridlig, 2000

Example of application of static analysis to program transformation & optimization



Example of application of static analysis to program transformation & optimization

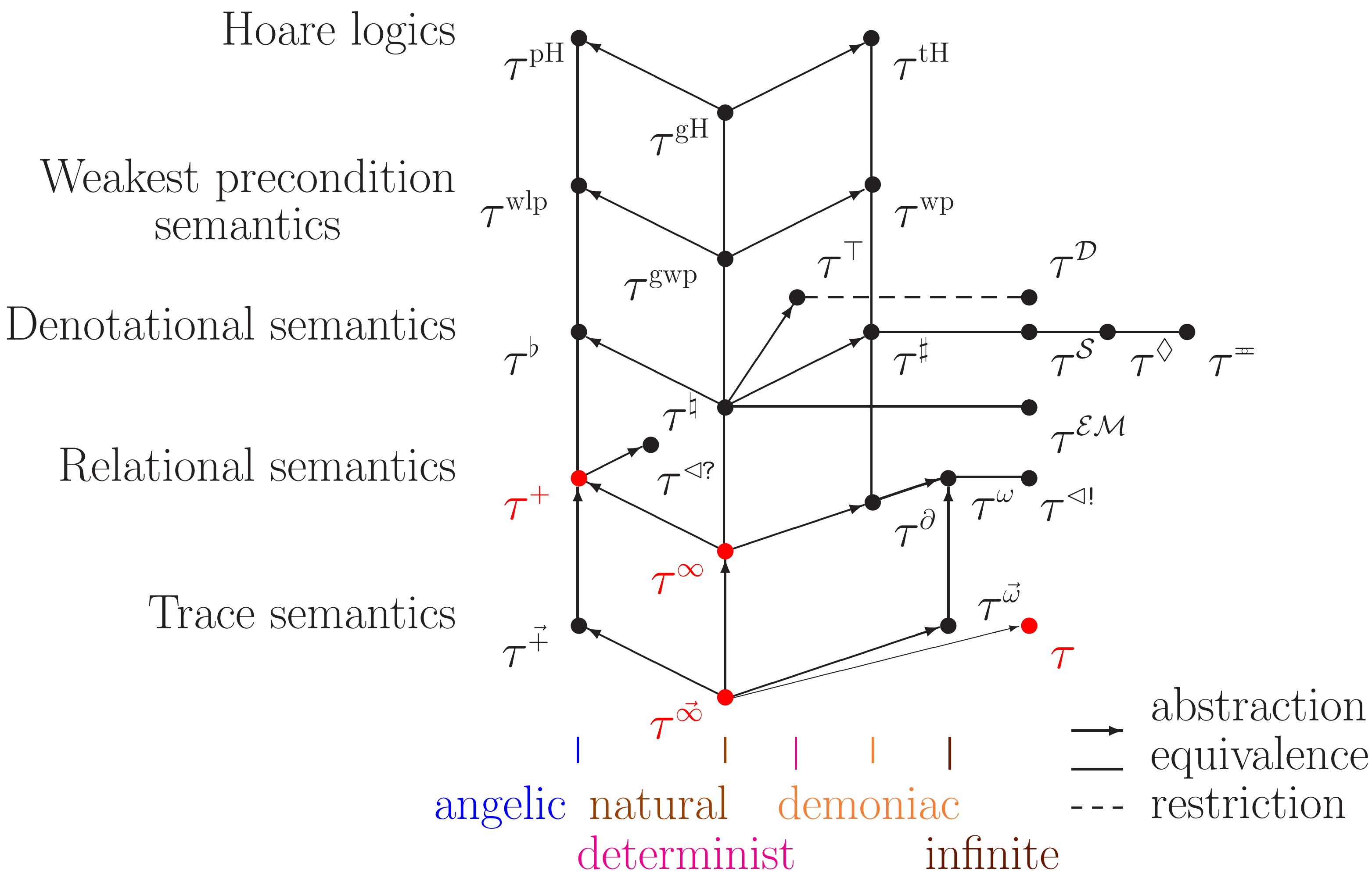


Some other recent applications of abstract interpretation

- Fundamental applications:
  - design of hierarchies of semantics,
  - ...;
- Practical applications:
  - security (analysis of cryptographic protocols, mobile code),
  - semantic tattooing of software,
  - data mining,
  - ....

ongoing theses J. Feret, D. Monniaux

Lattice of semantics



Forthcoming research

A lot of fundamental research remains to be one:

- modularity,
- higher order functions & modules,
- floating point numbers,
- probabilistic analyses,
- liveness properties with fairness,
- ...;

A few references

Starter:

P. Cousot. Abstract interpretation. *ACM Computing Surveys* 28 (2), 1996, 324–328.

On the web:

`http://www.di.ens.fr/~cousot/`

Industrialization of static analysis by abstract interpretation

- First research results: 1975;
- First industrializations:
  - Connected Components Corporation (U.S.A.), L. Harrison, 1993;
  - AbsInt Angewandte Informatik GmbH (Germany), R. Wilhelm, 1998;
  - Polyspace Technologies (France), A. Deutsch & D. Pilaud, 1999.

Prospects

- The fundamental problems of computer science are difficult to explain to non specialists (only applications are well understood);

Prospects

- The fundamental problems of computer science are difficult to explain to non specialists (only applications are well understood);
- In the future, the society will certainly be better aware of these computer software related problems (e.g. through catastrophes);

Prospects

- The fundamental problems of computer science are difficult to explain to non specialists (only applications are well understood);
- In the future, the society will certainly be better aware of these computer software related problems (e.g. through catastrophes);
- Research on fundamental ideas on software design is essential for modern societies;

Prospects

- The fundamental problems of computer science are difficult to explain to non specialists (only applications are well understood);
- In the future, the society will certainly be better aware of these computer software related problems (e.g. through catastrophes);
- Research on fundamental ideas on software design is essential for modern societies;
- The application of such fundamental research can hardly be scheduled in the short term (3 years);

Conclusion

Computer scientists need long term research funding.

Conclusion

Computer scientists need long term research funding.

THANK YOU FOR YOUR ATTENTION