ABSTRACT INTERPRETATION: THEORY AND APPLICATIONS

P. Cousot

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

Second International Summer School in Computational Logic, ISCL 2002

25th—30th August 2002, Acquafredda di Maratea (Basilicata, Italy)

LESSON 1: An Introductive Overview

Content

Motivations for Formal Methods

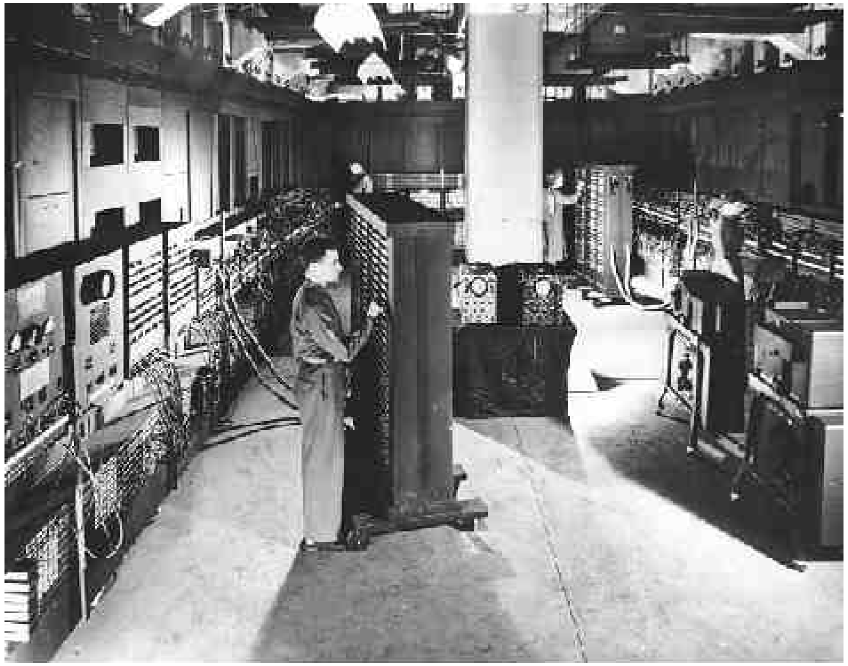
What is (or should be) the essential preoccupation of computer scientists?

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

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

Computer hardware change of scale

The 25 last years, computer hardware has seen its performances multiplied by $\mathbf{10^4}$ to $\mathbf{10^6}$;



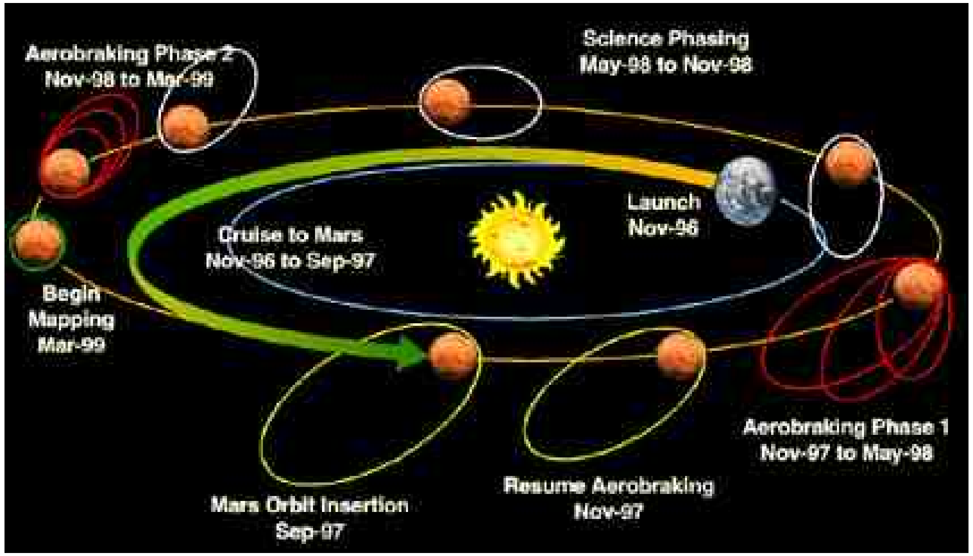
ENIAC (5000 flops)

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

The information processing revolution

A scale of $\mathbf{10^6}$ is typical of a significant **revolution**:

- Energy: nuclear power station / Roman slave;
- Transportation: distance Earth — Mars / Denmark height

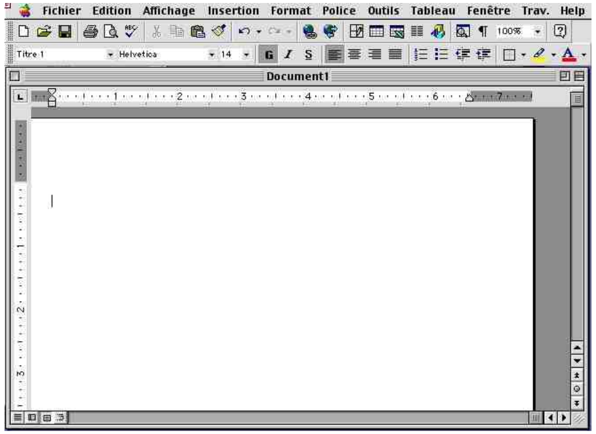


Computer software change of scale

- 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 [1];
  - 20 000 procedures;
  - 400 files;
  - $>$ 15 years of development.

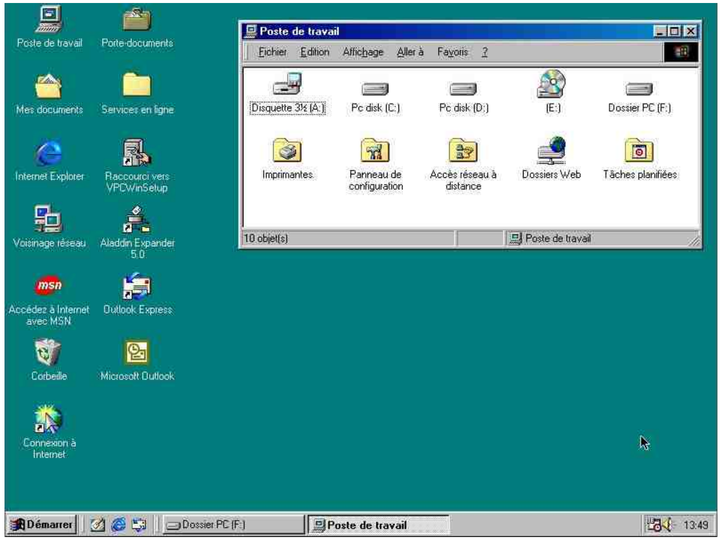


---

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

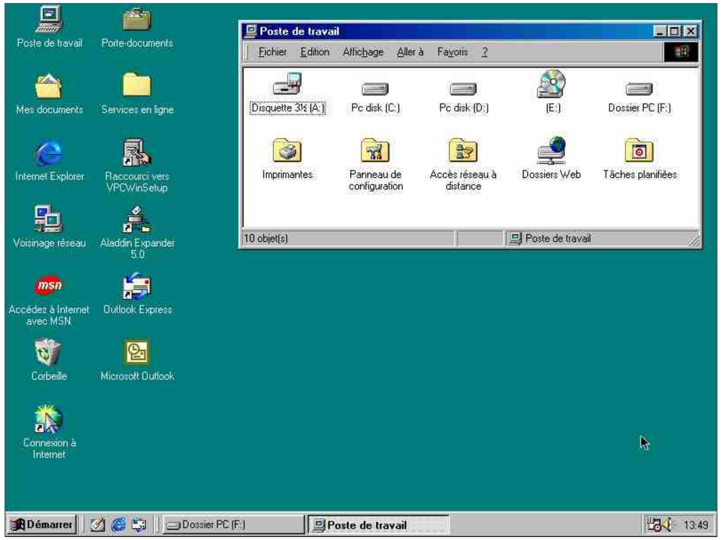
Computer software change of scale (cont'd)

- **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)

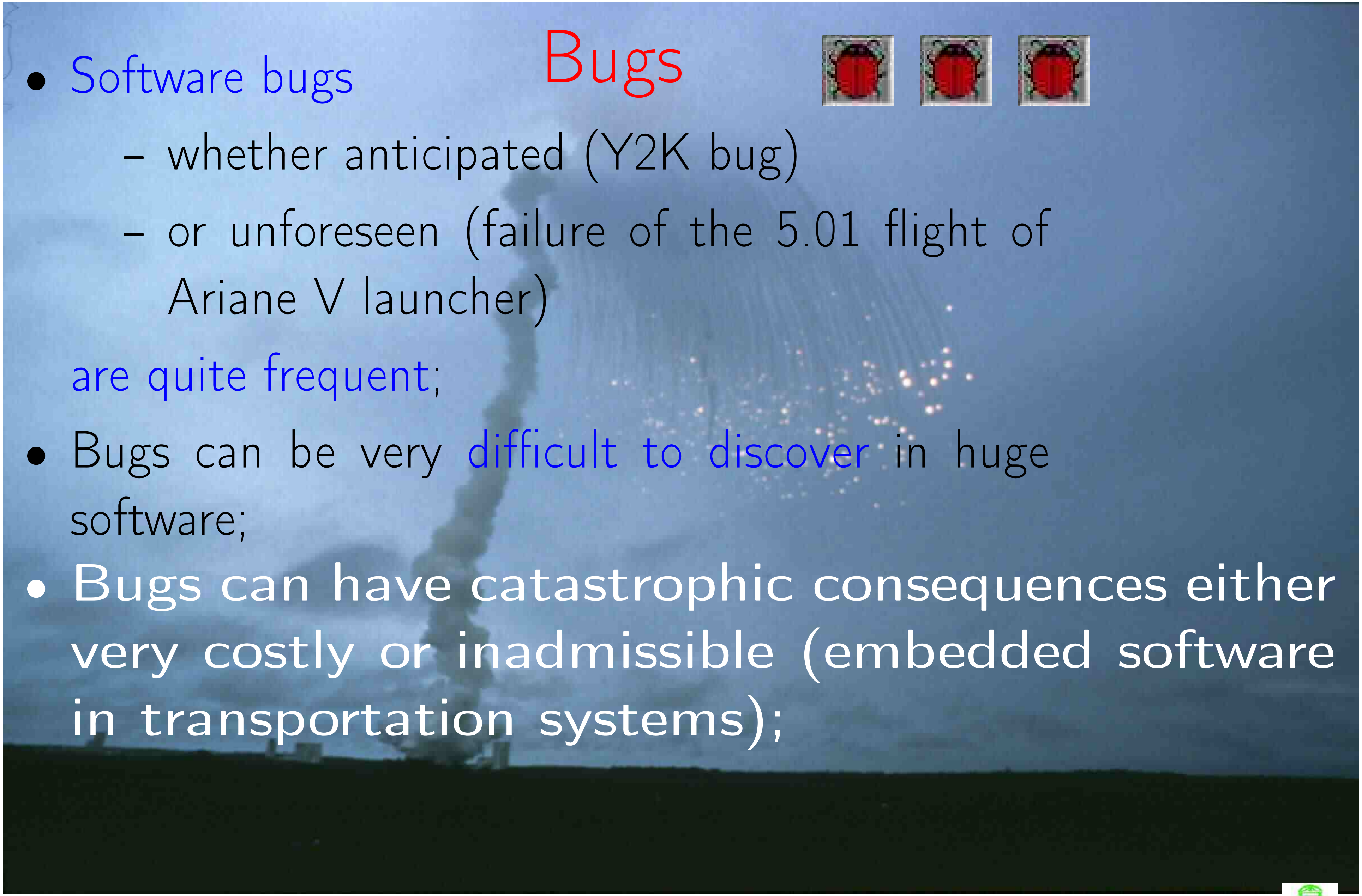
  are quite frequent;

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 quite 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 €;

The estimated cost of an overflow

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

  2 000 000 000 €;

- The financial results of Arianespace were **negative** in 2000, for the first time since 20 years.

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?);

⟹ It is absolutely necessary to widen the full set of methods and tools used to eliminate software bugs.

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 essentially unchanged 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;

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;
- Classical manual software verification methods (code reviews, simulations, debugging) do not scale up;
- So we should use computers to reason about computers!

Capability of computers

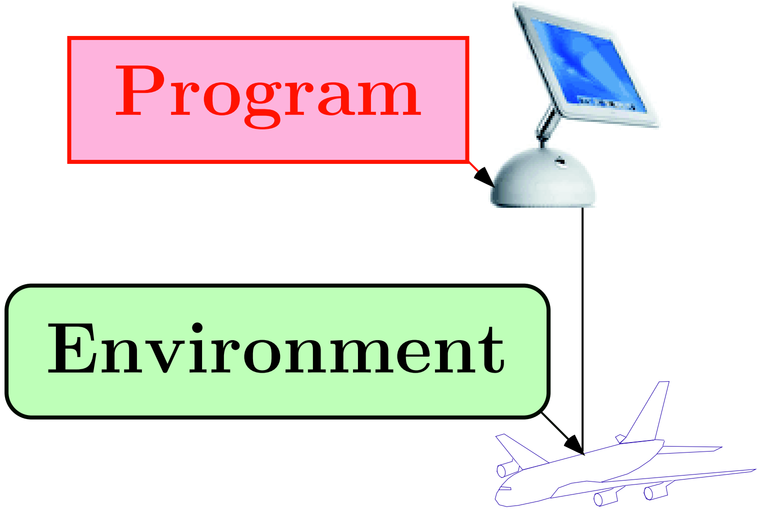
- The computing power and memory size of computers double every 18 months;
- So computer aided verification will scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, . . . ;

Capability of computers

- The computing power and memory size of computers double every 18 months;
- So computer aided verification will scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, . . . ;
- But the size of programs grows proportionally;
- And correctness proofs are exponential in the program size;
- So computers power growth is ultimately **not significant**.

On Formal Methods and Computer-Aided Verification

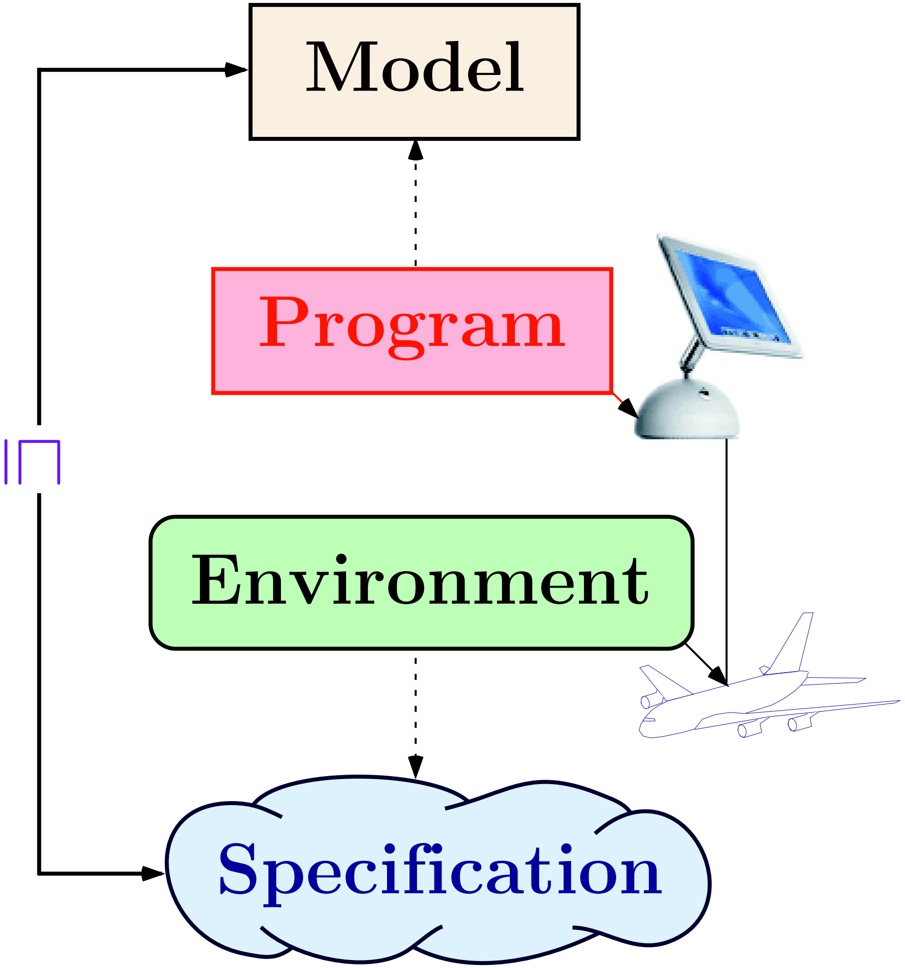
Computer Systems



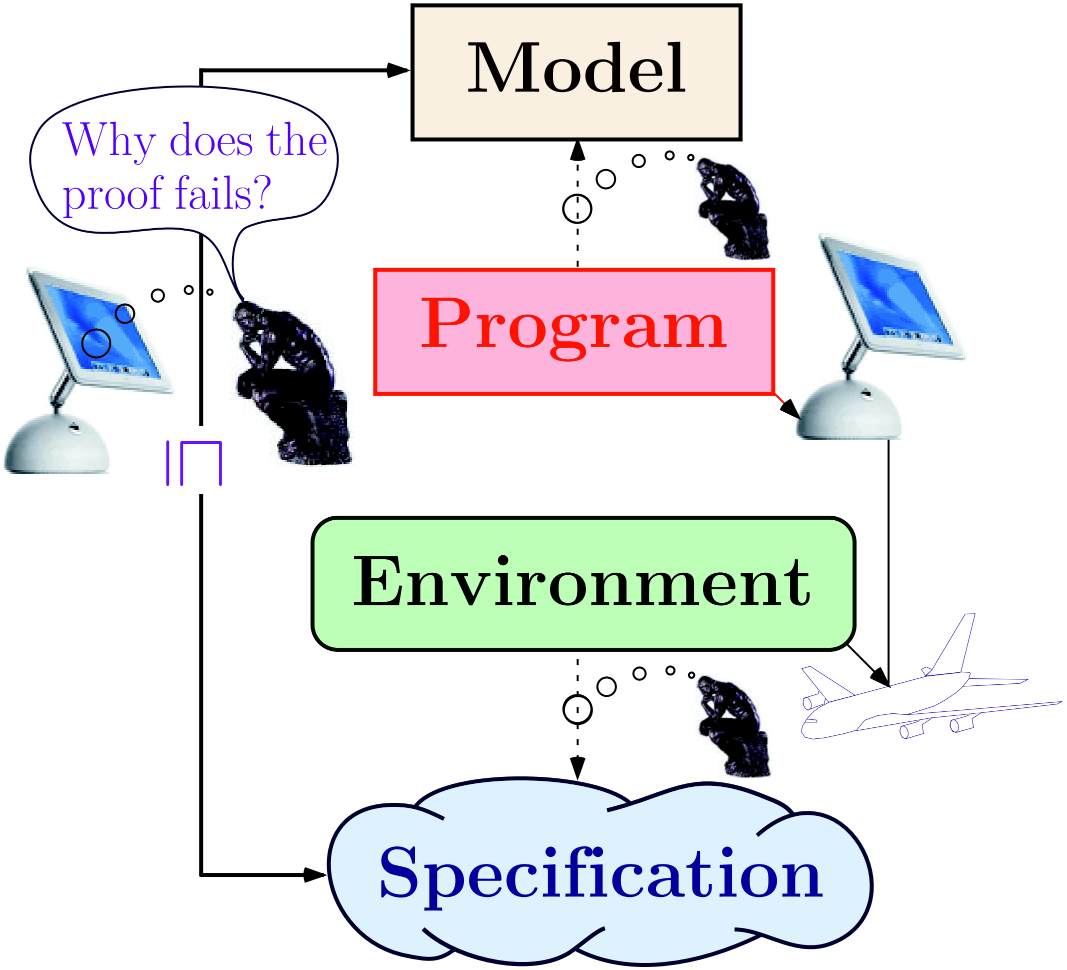
Program

Environment

Formal Methods



Deductive methods



Model

Why does the proof fails?

Program

Environment

Specification

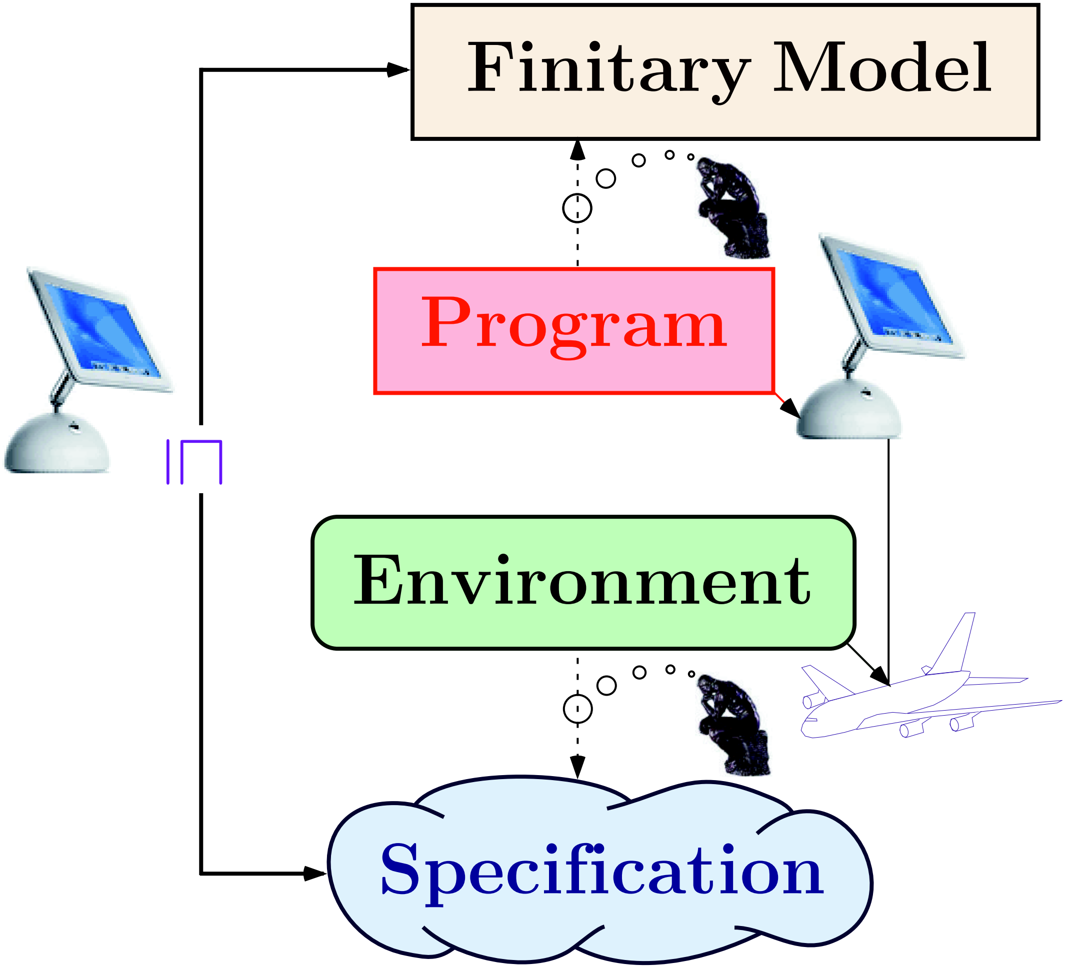
Deductive methods, criticism

- How to apply when lacking formal specifications (e.g. legacy software modification)? for large programs?
- Cost of proof is higher than the cost of the software development and testing[2];
- Only critical parts of the software can be checked formally so errors appear elsewhere (e.g. at interfaces);
- Both the program and its proof have to be maintained (e.g. during ten to twenty years for embedded software).

---

[2] Figures of 600 person-years for $80,000$ lines of C code have been reported for the Meteór metro line 14 in Paris developed with the B-method.

Software Model Checking



Finitary Model

Program

Environment

Specification

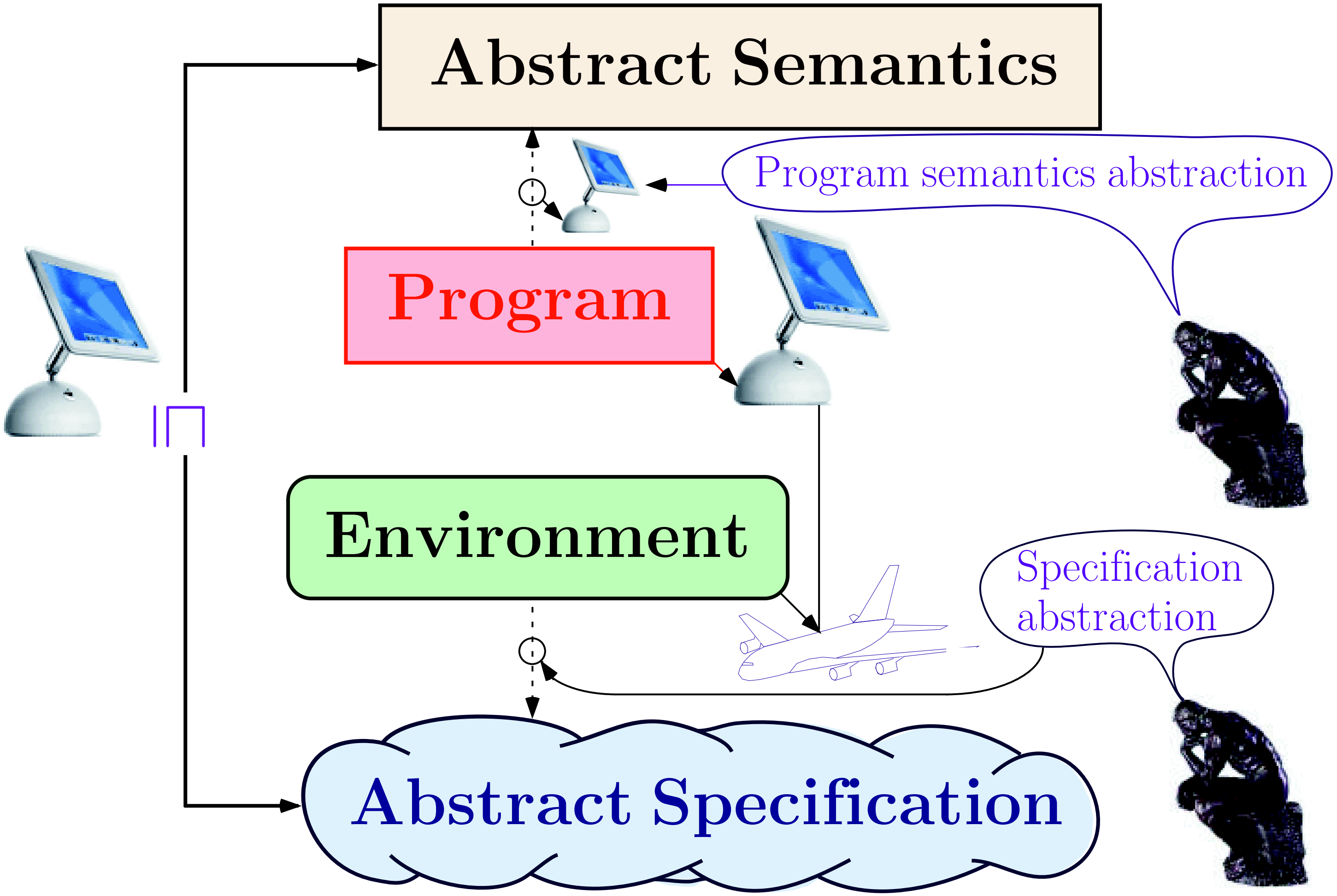
Software Model Checking, criticism

- How to apply when lacking temporal formal specifications? for large programs?
- Ultimately finite models, state explosion;
- Proof of correctness of the model?

  yes: back to deductive methods!

  no: debugging aid, not formal verification;
- Both the program and its model have to be maintained;
- Abstraction is required so software model checking essentially boils down to static program analysis.

Static Program Analysis



Abstract Semantics

Program semantics abstraction

Program

$\sqsubseteq$

Environment

Specification abstraction

Abstract Specification

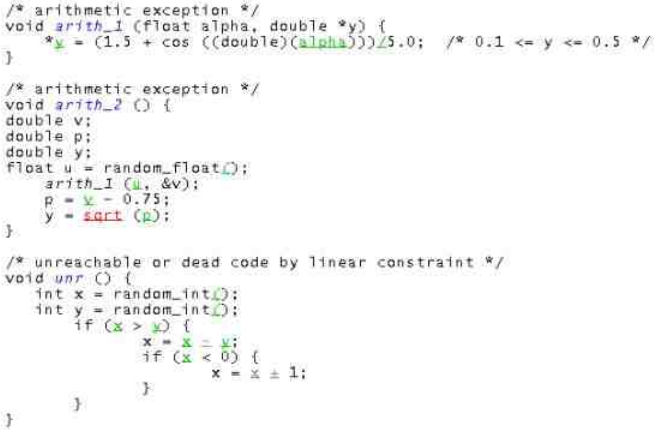
General-PurposeStatic Program Analyzers

“The first product to automatically detect 100% of run-time errors at Compilation Time

---

Based on Abstract Interpretation, PolySpace Technologies provides the earliest run-time errors detection solution to dramatically reduce testing and debugging costs with :

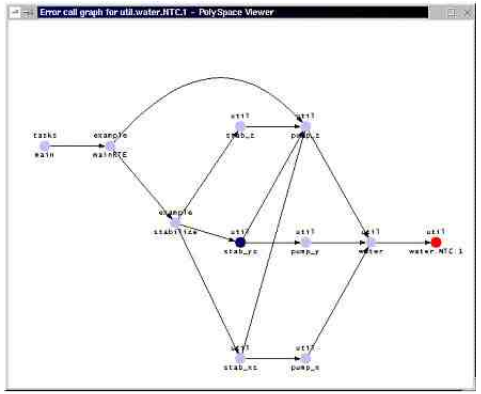
- No Test Case to Write
- No Code Instrumentation
- No Change to your Development Process
- No Execution of your Application”[3]



```
/* arithmetic exception */
void arith_1 (float alpha, double *y) {
    *y = (1.5 + cos ((double)(alpha)))/5.0;  /* 0.1 <= y <= 0.5 */
}

/* arithmetic exception */
void arith_2 () {
double v;
double p;
double y;
float u = random_float();
    arith_1 (u, &v);
    p = v - 0.75;
    y = sqrt (p);
}

/* unreachable or dead code by linear constraint */
void unr () {
   int x = random_int();
   int y = random_int();
       if (x > y) {
              x = x - y;
              if (x < 0) {
                     x = x + 1;
              }
       }
}
```

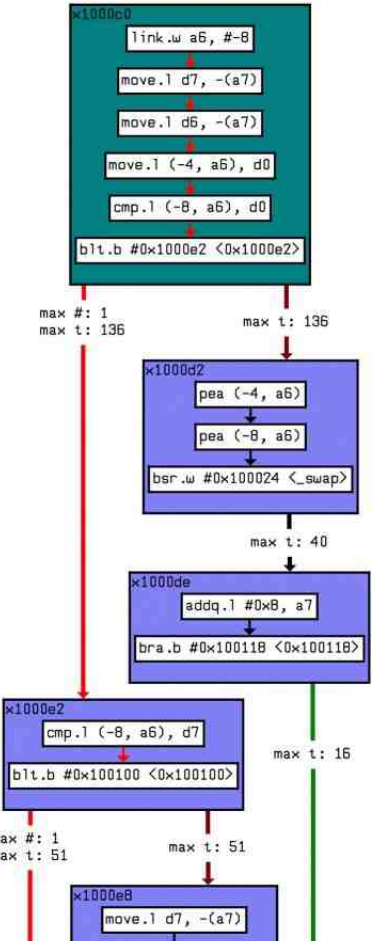


---

[3] http://www.polyspace.com/

Special-Purpose Static Program Analyzers

“The underlying theory of abstract interpretation provides the relation to the programming language semantics, thus enabling the systematic derivation of provably correct and terminating analyses.”[4]



---

[4] http://www.absint.com/pag/

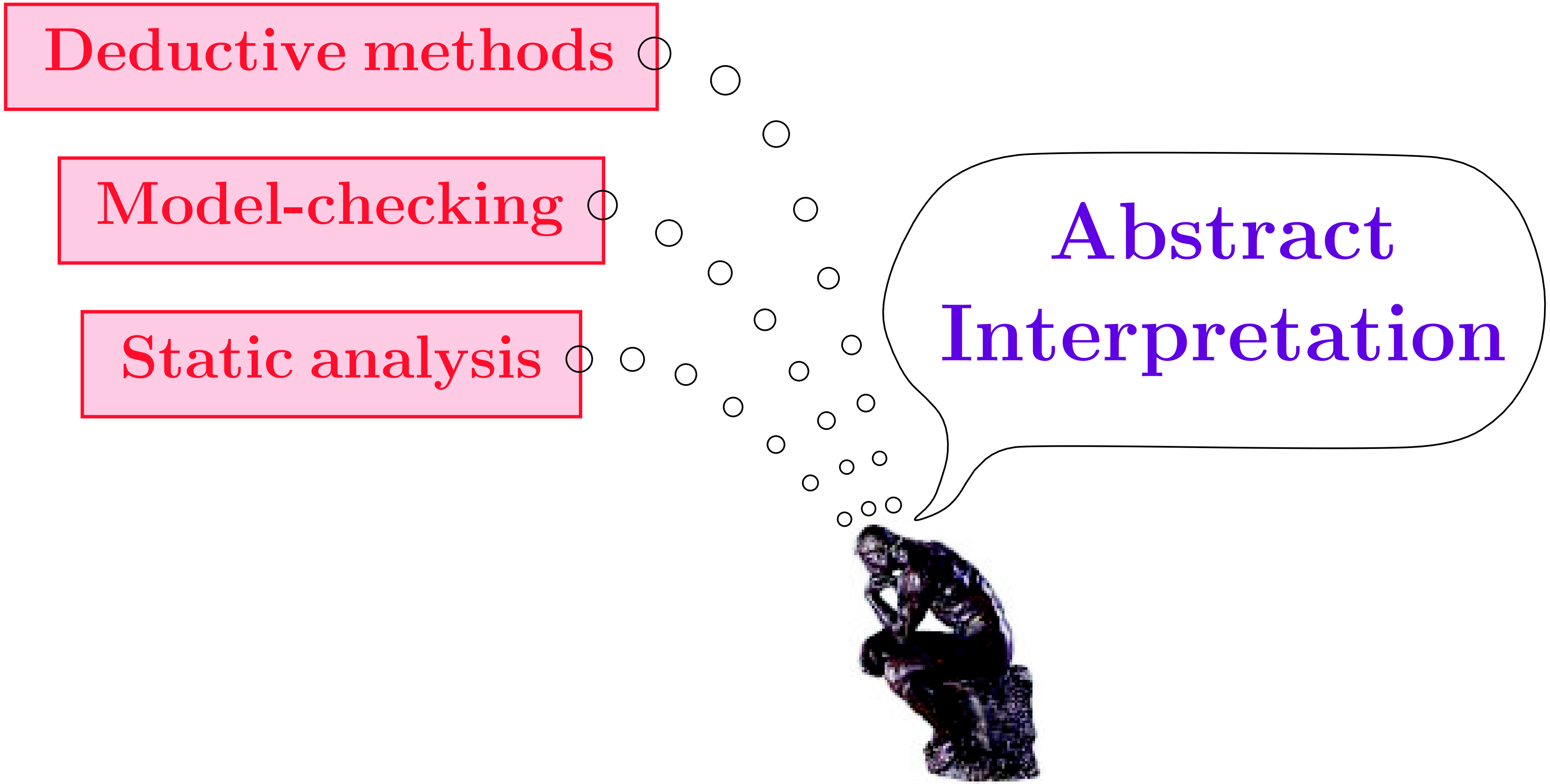
Static Program Analysis, criticism

- Full programming languages (ADA, C), weak specifications (e.g. absence of run-time errors);
- Can handle very large programs, prohibitive time and space costs or unprecise;
- No user specification but residual false alarms;
- Inherent approximations wired in the analyzer, no easy refinement (e.g. assertions).

Deductive methods

Model-checking

Static analysis

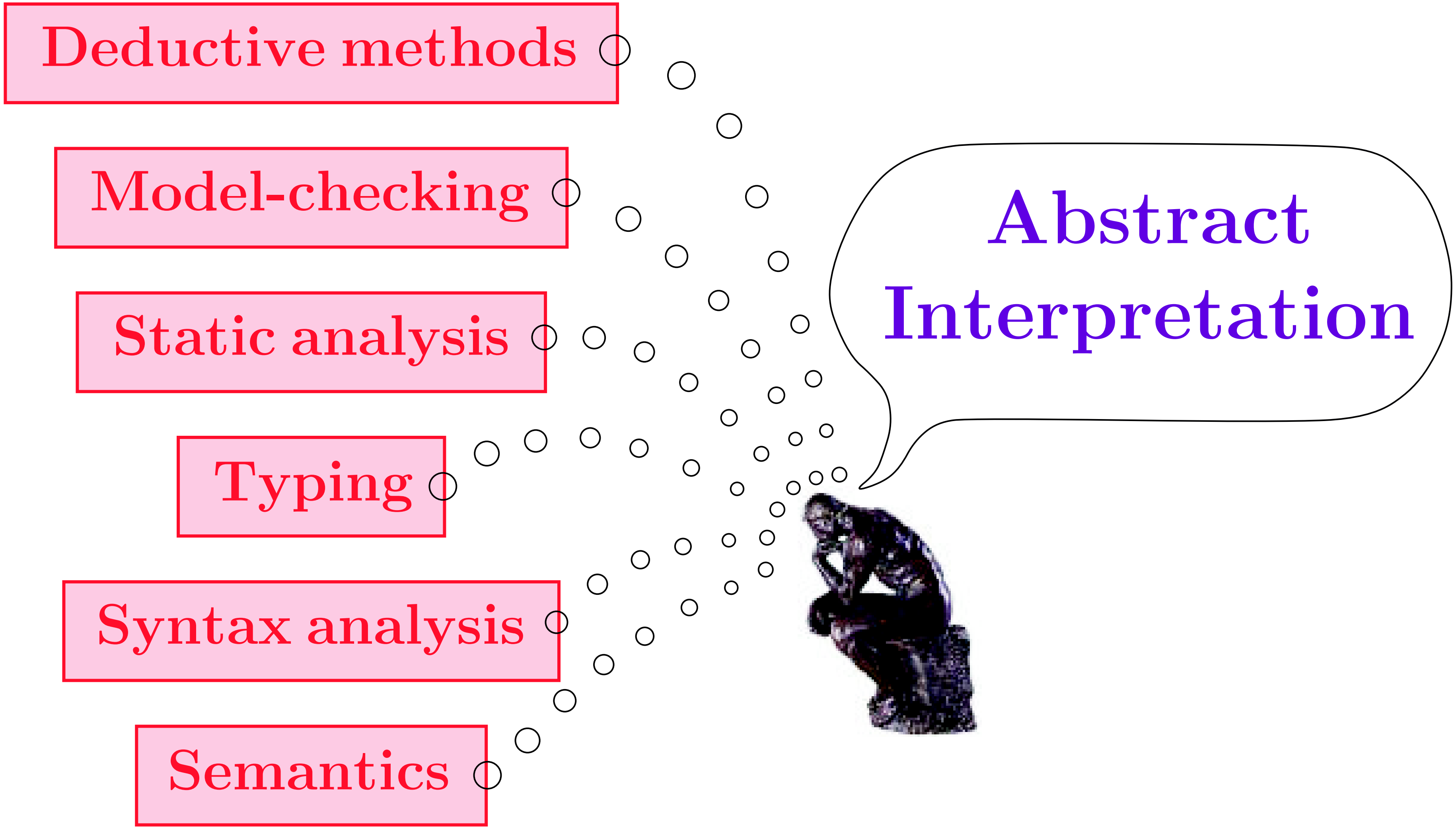


Deductive methods

Model-checking

Static analysis

Abstract Interpretation



Deductive methods

Model-checking

Static analysis

Typing

Syntax analysis

Semantics

Abstract Interpretation

Motivations for Abstract Interpretation

Abstract Interpretation

- **Thinking tool**: the idea of abstraction is central to reasoning (in particular on computer systems);

Abstract Interpretation

- **Thinking tool**: the idea of abstraction is central to reasoning (in particular on computer systems);
- A framework for designing **mechanical tools**: the idea of effective approximation leads to automatic semantics-based formal systems/program manipulation tools.

Abstract Interpretation

- **Thinking tool**: the idea of abstraction is central to reasoning (in particular on computer systems);
- A framework for designing **mechanical tools**: the idea of effective approximation leads to automatic semantics-based formal systems/program manipulation tools.

Reasonings about computer systems and their verification should ideally rely on a few principles rather than on a myriad of techniques and (semi-)algorithms.

Coping With Undecidability When Computing on the Program Semantics

- Ask the programmer to help (e.g. proof assistants);

Coping With Undecidability When Computing on the Program Semantics

- Ask the programmer to help (e.g. proof assistants);
- Consider decidable questions only or semi-algorithms (e.g. model-checking/model-debugging);

Coping With Undecidability When Computing on the Program Semantics

- Ask the programmer to help (e.g. proof assistants);
- Consider decidable questions only or semi-algorithms (e.g. model-checking/model-debugging);
- Consider effective approximations to handle practical complexity limitations;

Coping With Undecidability When Computing on the Program Semantics

- Ask the programmer to help (e.g. proof assistants);
- Consider decidable questions only or semi-algorithms (e.g. model-checking/model-debugging);
- Consider effective approximations to handle practical complexity limitations;

The above approaches can all be formalized within the abstract interpretation framework.

The Theory of Abstract Interpretation

- Abstract interpretation[5] is a theory of conservative approximation of the semantics/models of computer systems.

---

[5] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques. Grenoble, 21 Mar. 1978.

The Theory of Abstract Interpretation

- Abstract interpretation [5] is a theory of conservative approximation of the semantics/models of computer systems.

  Approximation: observation of the behavior of a computer system at some level of abstraction, ignoring irrelevant details;

---

[5] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques. Grenoble, 21 Mar. 1978.

The Theory of Abstract Interpretation

- Abstract interpretation[5] is a theory of conservative approximation of the semantics/models of computer systems.

  Approximation: observation of the behavior of a computer system at some level of abstraction, ignoring irrelevant details;

  Conservative: the approximation cannot lead to any erroneous conclusion.

---

[5] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* Thèse d'État ès sciences mathématiques. Grenoble, 21 Mar. 1978.

Informal Introduction to Abstract Interpretation

1 – Abstract Domains

- Program concrete properties are specified by the semantics of programming languages;

1 – Abstract Domains

- Program concrete properties are specified by the semantics of programming languages;
- Program abstract properties are elements of abstract domains (posets/lattices/. . .);

1 – Abstract Domains

- Program concrete properties are specified by the semantics of programming languages;
- Program abstract properties are elements of abstract domains (posets/lattices/. . .);
- Program property abstraction is performed by (effective) conservative approximation of concrete properties;

1 – Abstract Domains

- Program concrete properties are specified by the semantics of programming languages;
- Program abstract properties are elements of abstract domains (posets/lattices/. . .);
- Program property abstraction is performed by (effective) conservative approximation of concrete properties;
- The abstract properties (hence abstract semantics) are sound but may be incomplete with respect to the concrete properties (semantics);

2 – Correspondence between Concrete and Abstract Properties

- If any concrete property has a best approximation, approximation is formalized by Galois connections (or equivalently closure operators, Moore families, etc. [6], [7]);

---

[6] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques. Grenoble, 21 Mar. 1978.

[7] P. Cousot & R. Cousot. *Systematic design of program analysis frameworks*. ACM POPL'79, pp. 269–282, 1979.

2 – Correspondence between Concrete and Abstract Properties

- If any concrete property has a best approximation, approximation is formalized by Galois connections (or equivalently closure operators, Moore families, etc. [6], [7]);
- Otherwise, weaker abstraction/ concretization correspondences are available [8];

---

[6] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes.* Thèse d'État ès sciences mathématiques. Grenoble, 21 Mar. 1978.

[7] P. Cousot & R. Cousot. *Systematic design of program analysis frameworks.* ACM POPL'79, pp. 269–282, 1979.

[8] P. Cousot & R. Cousot. *Abstract interpretation frameworks.* JLC 2(4):511–547, 1992.

3 – Semantics Abstraction

- Program concrete semantics and specifications are defined by syntactic induction and composition of fixpoints (or using equivalent presentations[9]);

---

[9] P. Cousot & R. Cousot. *Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game theoretic form.* CAV '95, LNCS 939, pp. 293–308, 1995.

3 – Semantics Abstraction

- Program concrete semantics and specifications are defined by syntactic induction and composition of fixpoints (or using equivalent presentations[9]);
- The property abstraction is extended compositionally to all constructions of the concrete/abstract semantics, including fixpoints;

---

[9] P. Cousot & R. Cousot. *Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game theoretic form.* CAV '95, LNCS 939, pp. 293–308, 1995.

3 – Semantics Abstraction

- Program concrete semantics and specifications are defined by syntactic induction and composition of fixpoints (or using equivalent presentations[9]);
- The property abstraction is extended compositionally to all constructions of the concrete/abstract semantics, including fixpoints;
- This leads to a constructive design of the abstract semantics by approximation of the concrete semantics[10];

---

[9] P. Cousot & R. Cousot. *Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game theoretic form*. CAV '95, LNCS 939, pp. 293–308, 1995.

[10] P. Cousot & R. Cousot. *Inductive definitions, semantics and abstract interpretation*. POPL, 83–94, 1992.

4 — Effective Analysis/Checking/ Verification Algorithms

- Computable abstract semantics lead to effective program analysis/checking/verification algorithms;

4 — Effective Analysis/Checking/ Verification Algorithms

- Computable abstract semantics lead to effective program analysis/checking/verification algorithms;
- Furthermore fixpoints can be approximated iteratively by convergence acceleration through widening/narrowing that is non-standard induction [11].

---

[11] P. Cousot & R. Cousot. *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*. ACM POPL, pp. 238–252, 1977.

Elements of Abstract Interpretation

- P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques. Grenoble, 21 Mar. 1978.

Galois Connections[12]

$$\langle P, \leq\rangle \mathrel{\overset{\gamma}{\underset{\alpha}{\leftrightarrows}}} \langle Q, \sqsubseteq\rangle$$

$\overset{\text{def}}{=}$

- $\langle P, \leq\rangle$ is a poset
- $\langle Q, \sqsubseteq\rangle$ is a poset
- $\forall x \in P : \forall y \in Q : \alpha(x) \sqsubseteq y \iff x \leq \gamma(y)$

---

[12] The original Galois correspondence is semi-dual ($\sqsupseteq$ instead of $\sqsubseteq$).

Composing Galois Connections

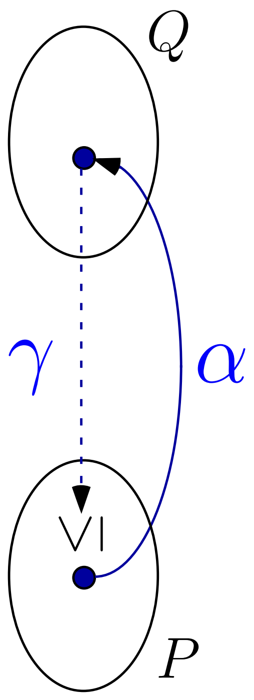
- If $\langle P, \leq\rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle Q, \sqsubseteq\rangle$ and $\langle Q, \sqsubseteq\rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle R, \preceq\rangle$ then

$$\langle P, \leq\rangle \underset{\alpha_2 \circ \alpha_1}{\overset{\gamma_1 \circ \gamma_2}{\leftrightarrows}} \langle R, \preceq\rangle$$ [13]

---

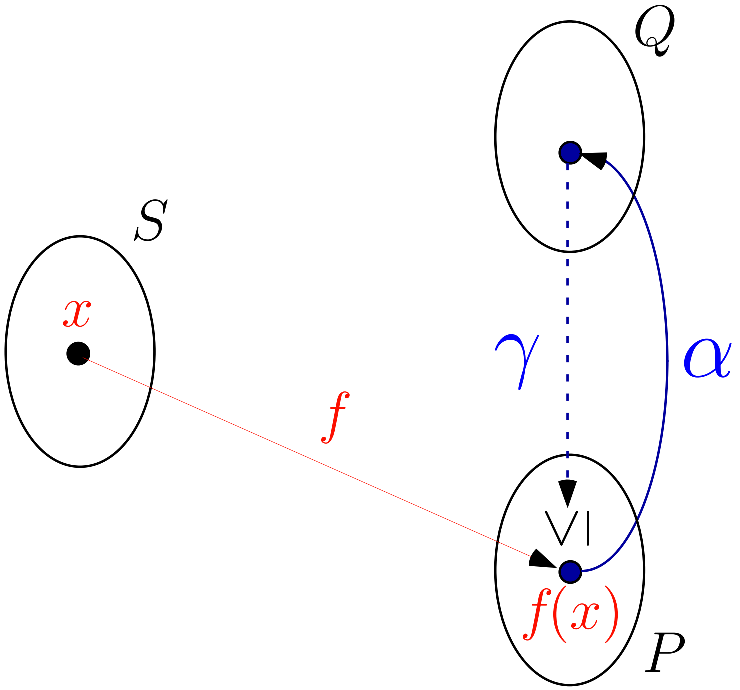
[13] This would not be true with the original definition of Galois correspondences.

Function Abstraction (1)



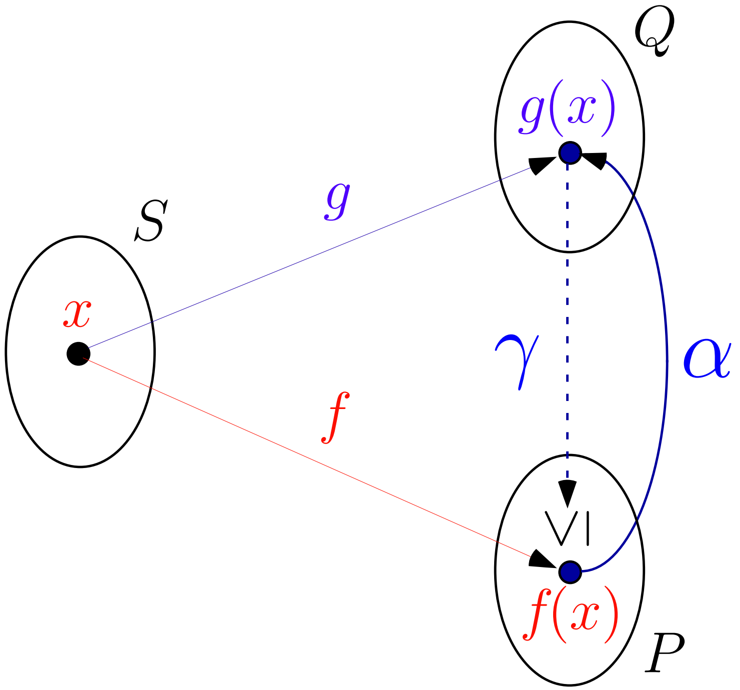
$$\langle P, \leq \rangle \xleftarrow[\alpha]{\gamma} \langle Q, \sqsubseteq \rangle$$

Function Abstraction (1)



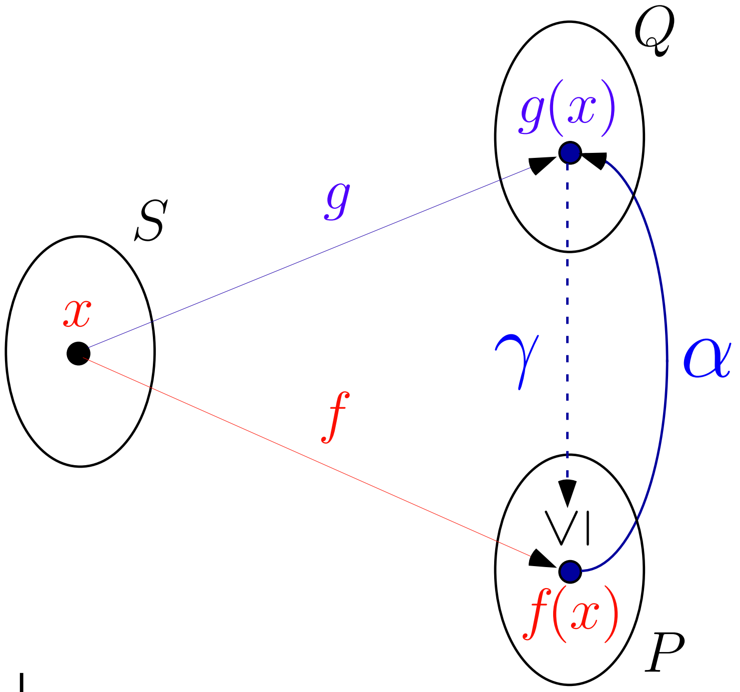
$$\langle P, \leq\rangle \xleftrightarrows[\alpha]{\gamma} \langle Q, \sqsubseteq\rangle$$

Function Abstraction (1)



$$\langle P, \leq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle Q, \sqsubseteq \rangle$$

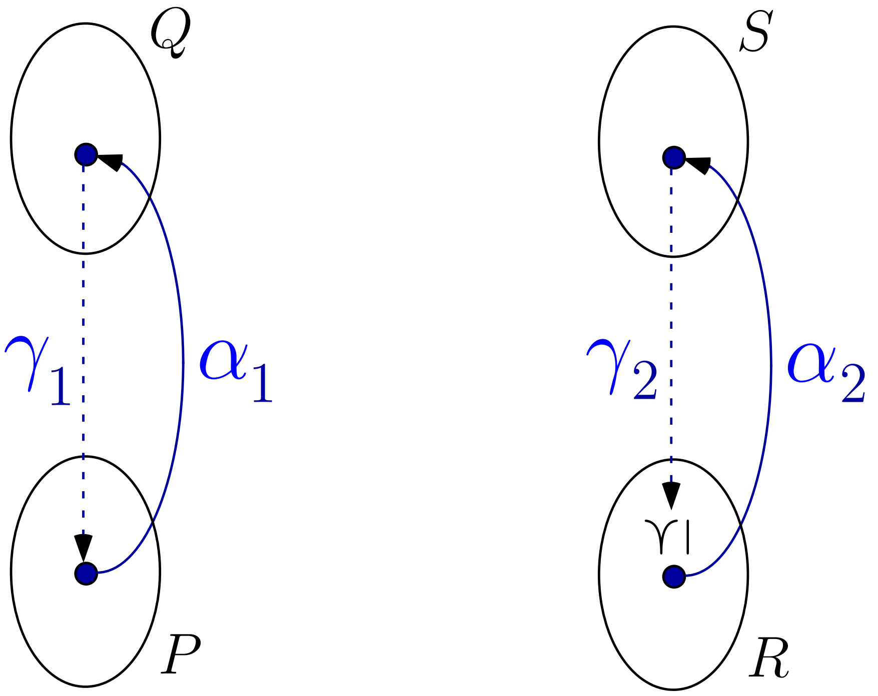
Function Abstraction (1)



- If $\langle P, \leq \rangle \xleftarrow[\alpha]{\gamma} \xrightarrow{} \langle Q, \sqsubseteq \rangle$ then

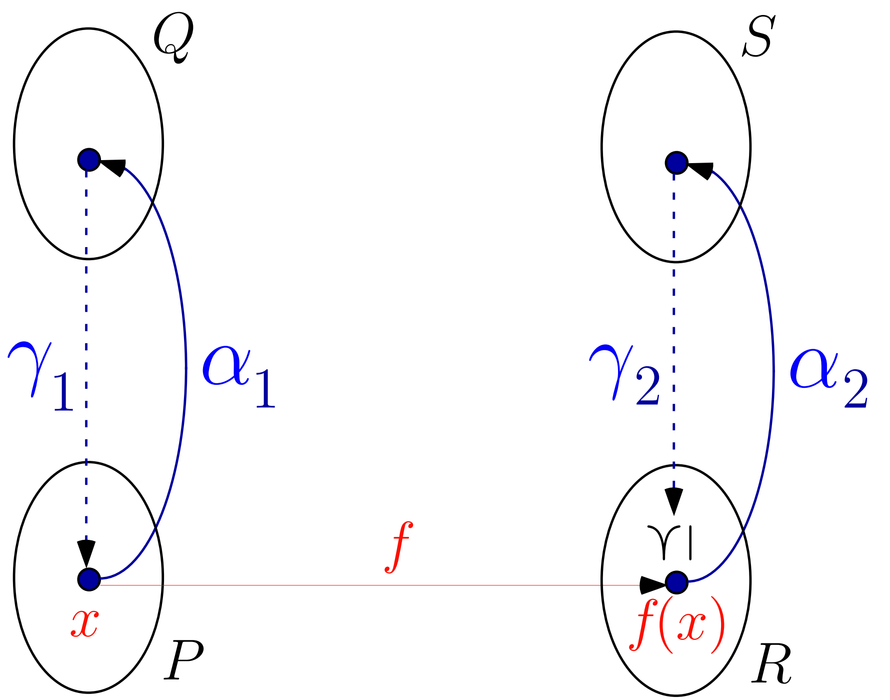
$$\langle S \mapsto P, \dot{\leq} \rangle \xleftarrow[\boldsymbol{\lambda} f \cdot \boldsymbol{\lambda} x \cdot \alpha(f(x))]{\boldsymbol{\lambda} g \cdot \boldsymbol{\lambda} x \cdot \gamma(g(x))} \xrightarrow{} \langle S \mapsto Q, \dot{\sqsubseteq} \rangle$$

Function Abstraction (2)



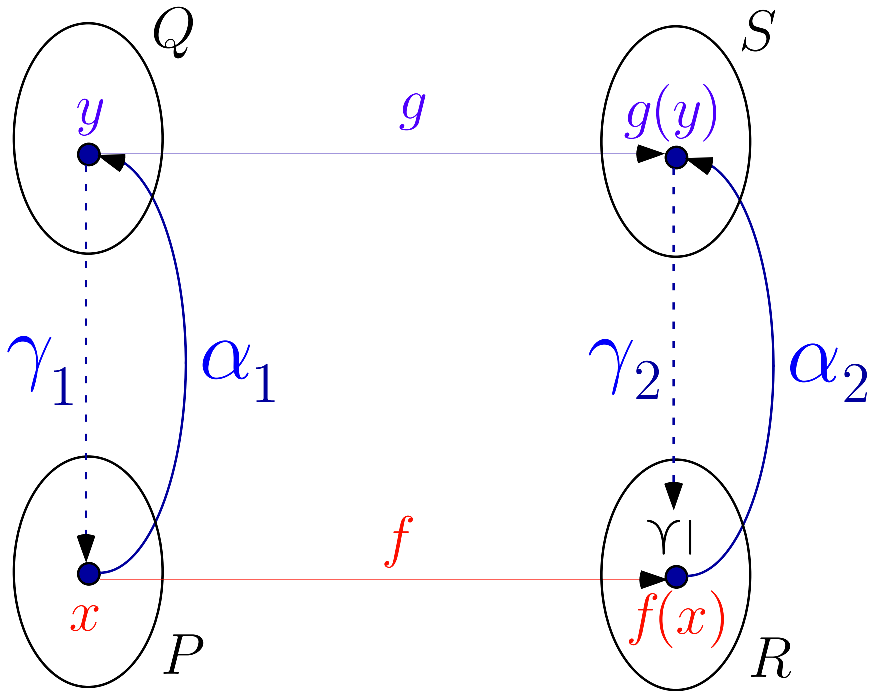
$$\langle P, \leq\rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle Q, \subseteq\rangle \qquad \langle R, \preceq\rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle S, \sqsubseteq\rangle$$

Function Abstraction (2)



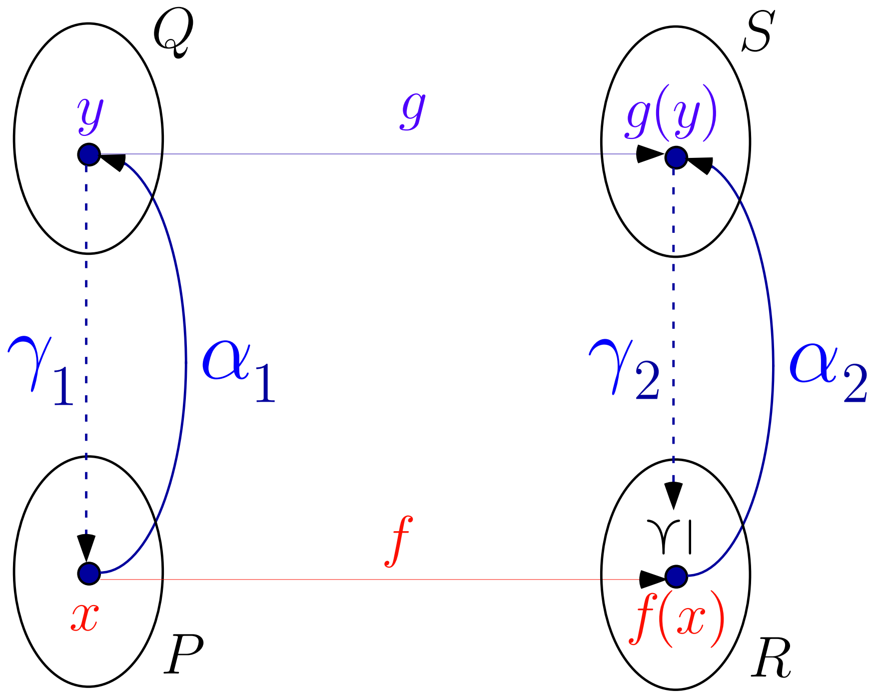
$$\langle P, \leq\rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle Q, \subseteq\rangle \qquad \langle R, \preceq\rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle S, \sqsubseteq\rangle$$

Function Abstraction (2)



$$\langle P, \leq\rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle Q, \subseteq\rangle \qquad \langle R, \preceq\rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle S, \sqsubseteq\rangle$$

Function Abstraction (2)



- If $\langle P, \leq\rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle Q, \subseteq\rangle$ and $\langle R, \preceq\rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle S, \sqsubseteq\rangle$ then

$$\langle P \xmapsto{m} R, \dot{\subseteq}\rangle \underset{\boldsymbol{\lambda} f \cdot \alpha_2 \circ f \circ \gamma_1}{\overset{\boldsymbol{\lambda} g \cdot \gamma_2 \circ g \circ \alpha_1}{\leftrightarrows}} \langle Q \xmapsto{m} S, \dot{\sqsubseteq}\rangle$$

Fixpoint Approximation

Let $F \in L \overset{m}{\longmapsto} L$ and $\overline{F} \in \overline{L} \overset{m}{\longmapsto} \overline{L}$ be respective monotone maps on the cpos $\langle L, \bot, \sqsubseteq\rangle$ and $\langle \overline{L}, \overline{\bot}, \overline{\sqsubseteq}\rangle$ and $\langle L, \sqsubseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \overline{L}, \overline{\sqsubseteq}\rangle$ such that $\alpha \circ F \circ \gamma \mathrel{\dot{\overline{\sqsubseteq}}} \overline{F}$. Then[14]:

- $\forall \delta \in \mathbb{O}\colon\ \alpha(F^\delta) \mathrel{\overline{\sqsubseteq}} \overline{F}^\delta$ (iterates from the infimum);
- The iteration order of $\overline{F}$ is $\leq$ to that of F;
- $\alpha(\mathrm{lfp}^{\sqsubseteq} F) \mathrel{\overline{\sqsubseteq}} \mathrm{lfp}^{\overline{\sqsubseteq}} \overline{F}$;

---

[14] P. Cousot & R. Cousot. *Systematic design of program analysis frameworks*. ACM POPL'79, pp. 269–282, 1979. Numerous variants!

Fixpoint Approximation

Let $F \in L \overset{m}{\longmapsto} L$ and $\overline{F} \in \overline{L} \overset{m}{\longmapsto} \overline{L}$ be respective monotone maps on the cpos $\langle L, \bot, \sqsubseteq\rangle$ and $\langle \overline{L}, \overline{\bot}, \overline{\sqsubseteq}\rangle$ and $\langle L, \sqsubseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \overline{L}, \overline{\sqsubseteq}\rangle$ such that $\alpha \circ F \circ \gamma \mathrel{\dot{\overline{\sqsubseteq}}} \overline{F}$. Then [14]:

- $\forall \delta \in \mathbb{O}$: $\alpha(F^{\delta}) \mathrel{\overline{\sqsubseteq}} \overline{F}^{\delta}$ (iterates from the infimum);
- The iteration order of $\overline{F}$ is $\leq$ to that of F;
- $\alpha(\mathrm{lfp}^{\sqsubseteq} F) \mathrel{\overline{\sqsubseteq}} \mathrm{lfp}^{\overline{\sqsubseteq}} \overline{F}$;

Soundness: $\mathrm{lfp}^{\overline{\sqsubseteq}} \overline{F} \mathrel{\overline{\sqsubseteq}} \overline{P} \Rightarrow \mathrm{lfp}^{\sqsubseteq} F \sqsubseteq \gamma(\overline{P})$.

---

[14] P. Cousot & R. Cousot. *Systematic design of program analysis frameworks*. ACM POPL'79, pp. 269–282, 1979. Numerous variants!

Fixpoint Abstraction

Moreover, the *commutation condition* $\overline{F} \circ \alpha = \alpha \circ F$ implies[15]:

- $\overline{F} = \alpha \circ F \circ \gamma$, and
- $\alpha(\mathrm{lfp}^{\sqsubseteq} F) = \mathrm{lfp}^{\overline{\sqsubseteq}} \overline{F}$;

---

[15] P. Cousot & R. Cousot. *Systematic design of program analysis frameworks*. ACM POPL'79, pp. 269–282, 1979. Numerous variants!

Fixpoint Abstraction

Moreover, the *commutation condition* $\overline{F} \circ \alpha = \alpha \circ F$ implies[15]:

- $\overline{F} = \alpha \circ F \circ \gamma$, and
- $\alpha(\mathrm{lfp}^{\sqsubseteq} F) = \mathrm{lfp}^{\overline{\sqsubseteq}} \overline{F}$;

Completeness: $\mathrm{lfp}^{\sqsubseteq} F \sqsubseteq \gamma(\overline{P}) \Rightarrow \mathrm{lfp}^{\overline{\sqsubseteq}} \overline{F} \mathrel{\overline{\sqsubseteq}} \overline{P}$.

---

[15] P. Cousot & R. Cousot. *Systematic design of program analysis frameworks.* ACM POPL'79, pp. 269–282, 1979. Numerous variants!