On Abstraction in Software Verification

Patrick COUSOT

École Normale Supérieure
45 rue d'Ulm, 75230 Paris cedex 05, France
Patrick.Cousot@ens.fr
www.di.ens.fr/~cousot

CAV 2002 Invited Tutorial
Copenhagen, Denmark July 27-31, 2002

Abstract

Our objective in this talk is to give an intuitive account of abstract interpretation theory and to present and discuss its application to formal methods and in particular to static abstract software checking.

We start with a discussion of formal methods and computer-aided verification and motivate their formalization by abstract interpretation. Then we informally introduce abstract interpretation and present a few basic elements of the theory.

Abstract interpretation theory formalizes the conservative approximation of the semantics of hardware or software computer systems. The *semantics* provides a formal model describing all possible behaviors of a computer system in interaction with any possible environment. By *approximation* we mean the observation of the semantics at some level of abstraction, ignoring irrelevant details. *Conservative* means that the approximation can never lead to an erroneous conclusion.

Abstract interpretation theory provides *thinking tools* since the idea of abstraction by conservative approximation is central to reasoning (in particular on computer systems) and *mechanical tools* since the idea of an effectively computable approximation leads to a systematic and constructive formal design methodology of automatic semantics-based program manipulation algorithms and tools.

We will present various applications of abstract interpretation theory to the design of hierarchies of semantics, program transformations, typing, model-checking and in more details static program analysis.

We show that their always exists an abstraction into a small finite boolean domain to prove any safety property of a single program by fixpoint/model checking. However the design of the model and its soundness proof is logically equivalent to a formal proof. This shows that model-checking is always feasible, that the only difficulty is to design a model and that this model can always be designed by abstraction of the operational semantics of the program to be checked.

The whole problematics of static analysis is to automate the design of this abstract model/semantics. For the static analysis of a full programming language, no such *finite* abstraction exists, so that infinite abstract domains and widenings are needed and more powerful than finite abstractions. We finally discuss the design of program static analyzers and report on an ongoing experience with the design of a parametric specializable program static analyzer for safety-critical real-time embedded software.

Content

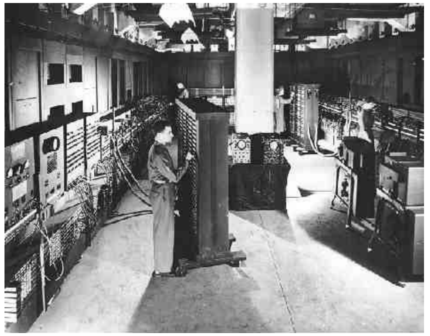
Motivations for Formal Methods

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 $\mathbf{10^6}$;



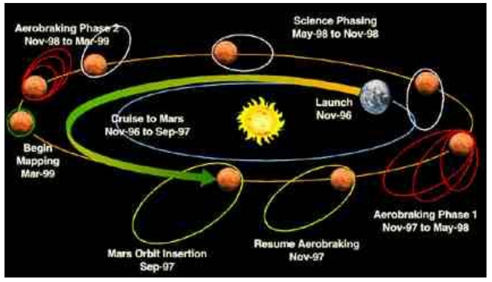
ENIAC (5000 flops)

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

The information processing revolution

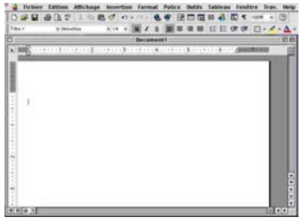
A scale of $\mathbf{10^6}$ is typical of a significant **revolution**:

- 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;
- Example 1 (modern text editor for the

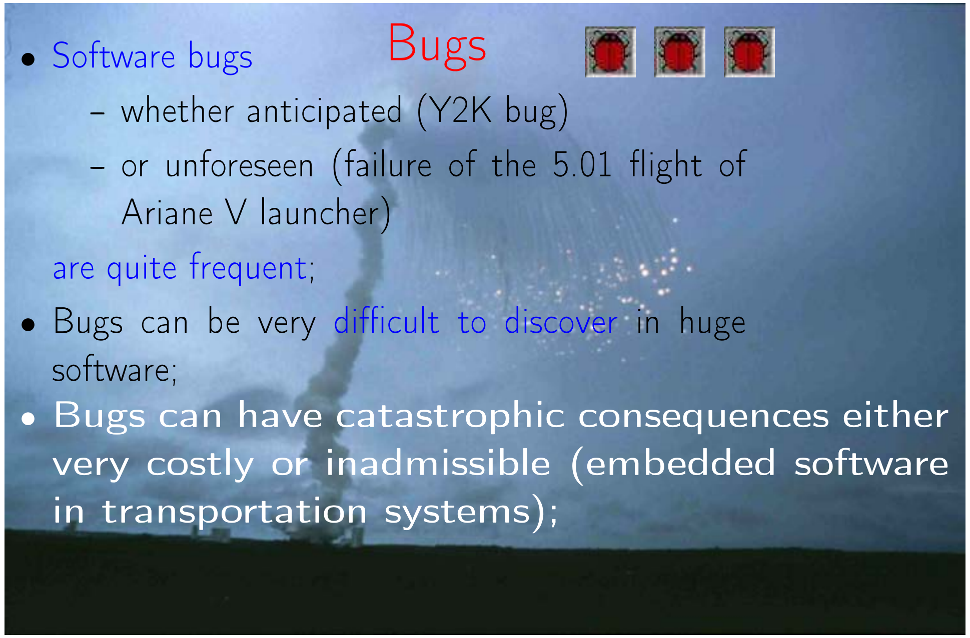


  - $>$ 1 700 000 lines of C[1];
  - 20 000 procedures;
  - 400 files;
  - $>$ 15 years of development.

[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;
  - 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 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

- 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);
- Computer software bugs can become an important societal problem (collective fears and reactions? new legislation?);

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

Capability of computer scientists

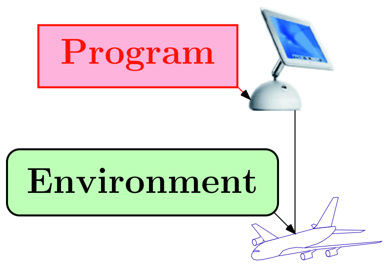
- The intellectual capability of computer scientists remains essentially unchanged year after year;
- The size of programmer teams in charge of software design and maintenance cannot evolve in such huge proportions;
- Classical manual software verification methods (code reviews, simulations, debugging) do not scale up;
- So we should use computers to reason about computers!

Capability of Computers

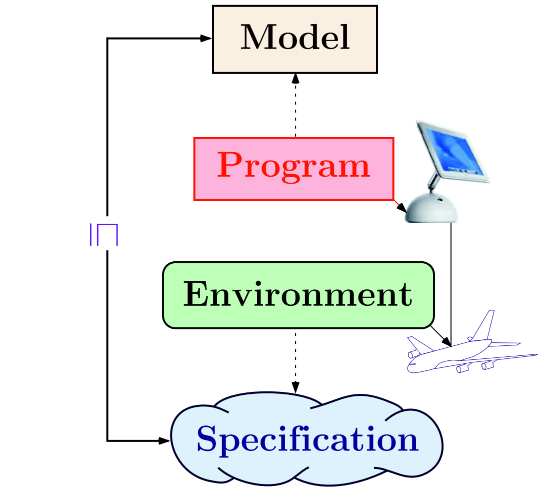
- The computing power and memory size of computers double every 18 months;
- So computer aided verification will scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, scale up, . . . ;
- But the size of programs grows proportionally;
- And correctness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

On Formal Methods and Computer-Aided Verification

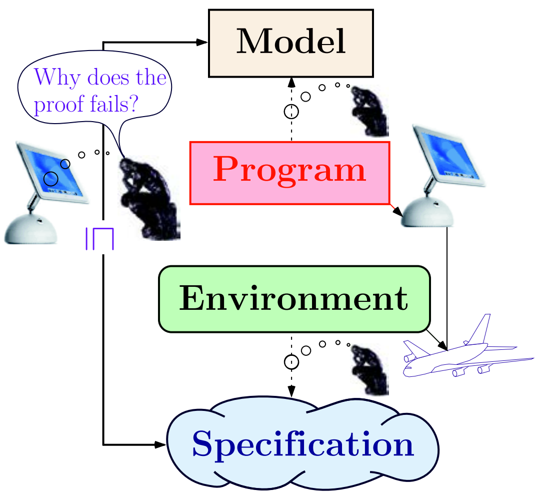
Computer Systems



Formal Methods



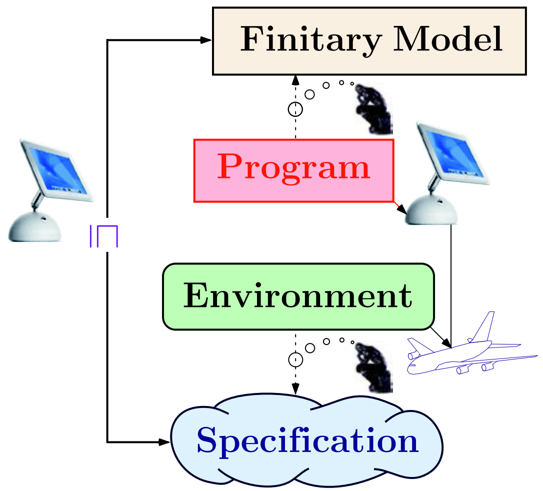
Deductive methods



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).

Software Model Checking



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

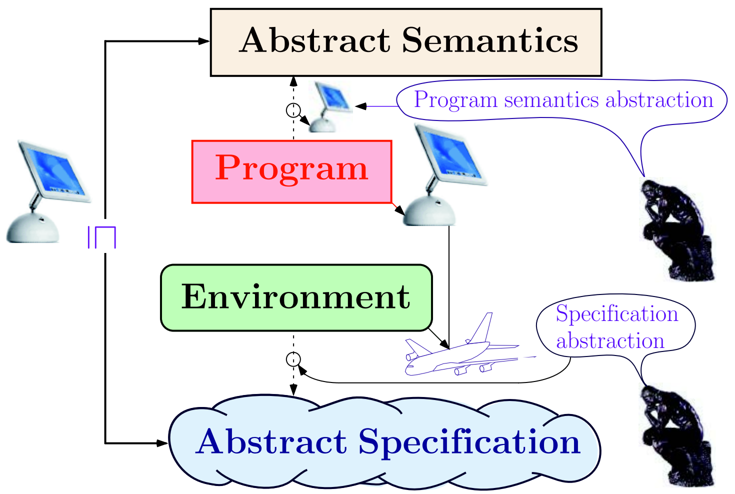
Software Model Checking, Criticism

- 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

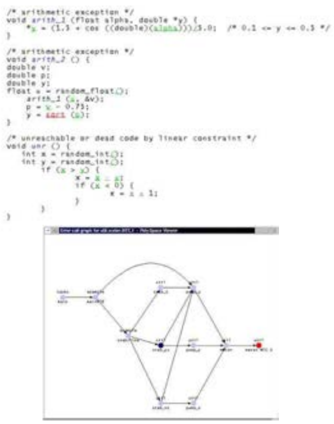


General-PurposeStatic Program Analyzers

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

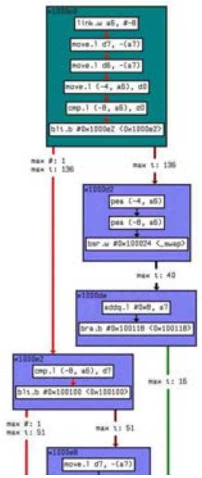
Based on Abstract Interpretation, PolySpace Technologies provides the earliest run-time errors detection solution to dramatically reduce testing and debugging costs with :

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



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]

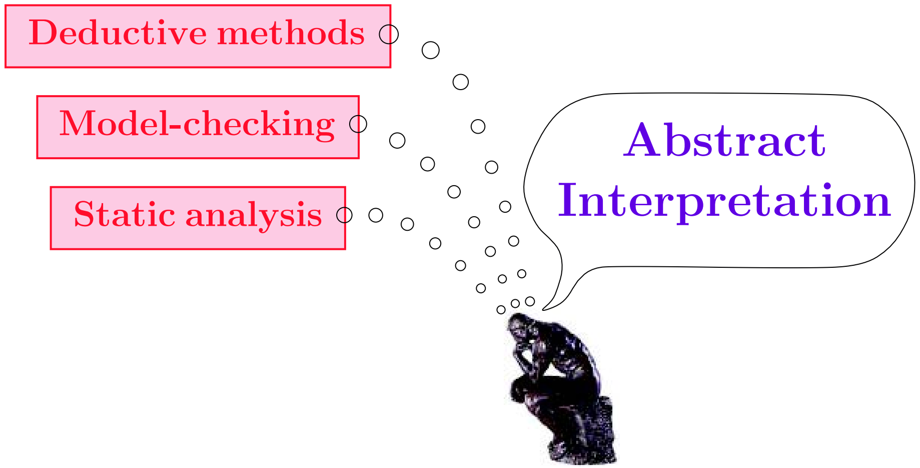


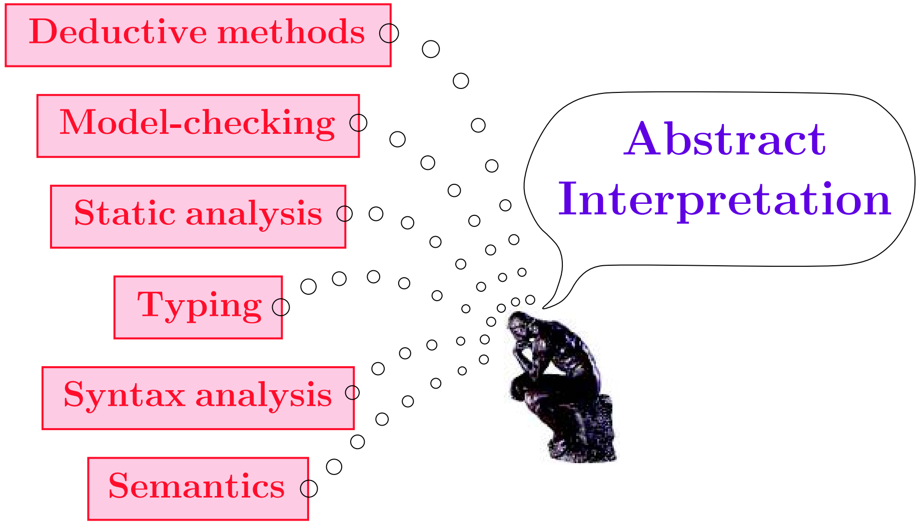
[3] http://www.polyspace.com/

[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).





Motivations for Abstract Interpretation

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);
- 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.

  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.

Informal Introduction to Abstract Interpretation

[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.

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]);
- Otherwise, weaker abstraction/ concretization correspondences are available [7];

6 P. Cousot & R. Cousot. *Systematic design of program analysis frameworks.* ACM POPL'79, pp. 269–282, 1979.
7 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 [8]);
- 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 [9];

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 [10].

8 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.
9 P. Cousot & R. Cousot. *Inductive definitions, semantics and abstract interpretation.* POPL, 83–94, 1992.
10 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[11]

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

$\stackrel{\text{def}}{=}$

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

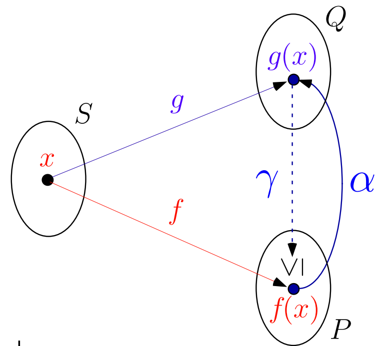
[11] The original Galois correspondence is semi-dual ($\sqsupseteq$ instead of $\sqsubseteq$).

Composing Galois Connections

- If $\langle P, \leq\rangle \xrightleftharpoons[\alpha_1]{\gamma_1} \langle Q, \sqsubseteq\rangle$ and $\langle Q, \sqsubseteq\rangle \xrightleftharpoons[\alpha_2]{\gamma_2} \langle R, \preceq\rangle$ then

$$\langle P, \leq\rangle \xrightleftharpoons[\alpha_2 \circ \alpha_1]{\gamma_1 \circ \gamma_2} \langle R, \preceq\rangle$$ [12]

Function Abstraction (1)

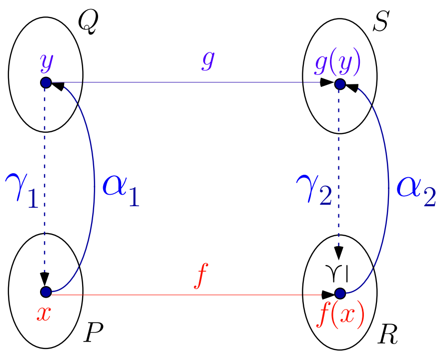


- If $\langle P, \leq\rangle \xrightleftharpoons[\alpha]{\gamma} \langle Q, \sqsubseteq\rangle$ then

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

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

Function Abstraction (2)



- If $\langle P, \leq\rangle \xrightleftharpoons[\alpha_1]{\gamma_1} \langle Q, \subseteq\rangle$ and $\langle R, \preceq\rangle \xrightleftharpoons[\alpha_2]{\gamma_2} \langle S, \sqsubseteq\rangle$ then

$$\langle P \xrightarrow{m} R, \dot{\subseteq}\rangle \xrightleftharpoons[\boldsymbol{\lambda} f \cdot \alpha_2 \circ f \circ \gamma_1]{\boldsymbol{\lambda} g \cdot \gamma_2 \circ g \circ \alpha_1} \langle Q \xrightarrow{m} S, \dot{\sqsubseteq}\rangle$$

Fixpoint Approximation

Let $F \in L \xrightarrow{m} L$ and $\overline{F} \in \overline{L} \xrightarrow{m} \overline{L}$ be respective monotone maps on the cpos $\langle L, \perp, \sqsubseteq\rangle$ and $\langle \overline{L}, \overline{\perp}, \overline{\sqsubseteq}\rangle$ and $\langle L, \sqsubseteq\rangle \xrightleftharpoons[\alpha]{\gamma} \langle \overline{L}, \overline{\sqsubseteq}\rangle$ such that $\alpha \circ F \circ \gamma \;\dot{\overline{\sqsubseteq}}\; \overline{F}$. Then[13]:

- $\forall \delta \in \mathbb{O}$: $\alpha(F^\delta) \mathrel{\overline{\sqsubseteq}} \overline{F}^\delta$ (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: $\mathrm{lfp}^{\overline{\sqsubseteq}} \overline{F} \mathrel{\overline{\sqsubseteq}} \overline{P} \Rightarrow \mathrm{lfp}^{\sqsubseteq} F \sqsubseteq \gamma(\overline{P})$.

[13] 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[14]:

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

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

Systematic Design of an Abstract Semantics

By structural induction on the language syntax, for each language construct:

- Define the concrete semantics $\mathrm{lfp}^{\sqsubseteq} F$;
- Choose the abstraction $\alpha = \kappa(\alpha_1, \ldots, \alpha_n)$ and check $\langle L, \sqsubseteq\rangle \xrightleftharpoons[\alpha]{\gamma} \langle \overline{L}, \overline{\sqsubseteq}\rangle$;
- Calculate $\overline{F} \stackrel{\text{def}}{=} \alpha \circ F \circ \gamma$ and check that $\overline{F} \circ \alpha = \alpha \circ F$;
- It follows, by construction, that $\alpha(\mathrm{lfp}^{\sqsubseteq} F) = \mathrm{lfp}^{\overline{\sqsubseteq}} \overline{F}$.

(and similarly in case of approximation).

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

Abstract Domains

An abstraction α is a specification of an abstract domain, including:

- the representation of the abstract properties;
- the approximation ordering lattice structure $(\leq, 0, 1, \vee, \wedge, \ldots)$;
- the computational ordering cpo structure $(\sqsubseteq, \perp, \sqcup, \ldots)$;
- the abstract operators, e.g. *non-relational abstract multiplication*:
  - $P \otimes Q \stackrel{\text{def}}{=} \alpha(\{x \times y \mid x \in \gamma(P) \wedge y \in \gamma(Q)\})$ *postcondition*
  - $\otimes^{-1}(R) \stackrel{\text{def}}{=} \alpha(\{\langle x, y \rangle \mid x \times y \in \gamma(R)\})$ *precondition*

Combinations of Abstract Domains[15]

Operation	$\kappa(\alpha_1, \ldots, \alpha_n)$	*Intuition*
Composition	$\alpha_n \circ \ldots \circ \alpha_1$	*Successive abstractions*
Duality	$\neg\kappa(\neg\alpha_1, \ldots, \neg\alpha_n)$	*Contraposition*[16]
Reduced product	$\alpha_1 \sqcap \ldots \sqcap \alpha_n$	*Conjunction*
Reduced power	$\alpha_1 \mapsto \ldots \mapsto \alpha_n$	*Case analysis*

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

16 P. Cousot. *Semantic Foundations of Program Analysis.* In *Program Flow Analysis: Theory and Applications*, Prentice-Hall, pp. 303–342, 1981.

A Potpourri of Applications of Abstract Interpretation

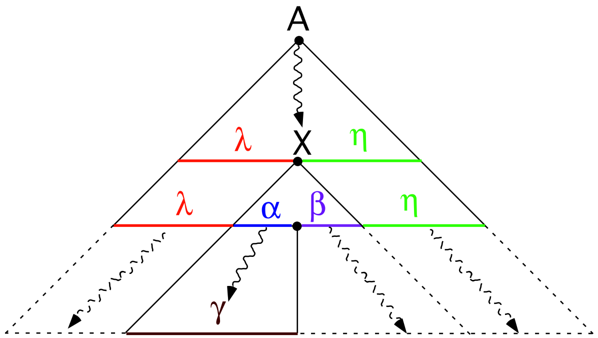
Content of the Potpourri of Applications of Abstract Interpretation

Application to Syntax

• P. Cousot & R. Cousot. *Parsing as Abstract Interpretation of Grammar Semantics*, TCS, 2002, in press.

The Semantics of Syntax

• The semantics of a grammar $G = \langle N, T, P, A\rangle$ is the set of items $[\lambda, X := \alpha/\gamma \bullet \beta]$ such that $\exists \eta : \exists X := \alpha\beta \in P$:



The Fixpoint Semantics of Syntax

$$S = \operatorname{lfp}^{\subseteq} F$$

$$\begin{aligned}
F(I) \stackrel{\text{def}}{=} & \{[\epsilon, A := \epsilon/\epsilon \bullet \beta] \mid A := \beta \in P\} \\
& \cup \{[\lambda, X := \alpha Y/\gamma\delta \bullet \beta] \mid [\lambda, X := \alpha/\gamma \bullet Y\beta] \in I \wedge \\
& \qquad Y := \delta \in P\} \\
& \cup \{[\lambda, X := \alpha Y/\gamma\xi \bullet \beta] \mid [\lambda, X := \alpha/\gamma \bullet Y\beta] \in I \wedge \\
& \qquad [\lambda\gamma, Y := \delta/\xi \bullet \epsilon] \in I\} \\
& \cup \{[\lambda, X := \alpha a/\gamma a \bullet \beta] \mid [\lambda, X := \alpha/\gamma \bullet a\beta] \in I\}\ .
\end{aligned}$$

Syntactic Abstractions

• $\alpha_\ell(I) \stackrel{\text{def}}{=} \{\gamma \in T^\star \mid [\epsilon, A := \alpha/\gamma \bullet \epsilon] \in I\}$

Language of the grammar $G = \langle N, T, P, A\rangle$

• $\omega = \omega_1 \ldots \omega_i \omega_{i+1} \ldots \omega_j \ldots \omega_n$ input string

$$\begin{aligned}
\alpha_\omega(I) \stackrel{\text{def}}{=} & \{\langle X := \alpha \bullet \beta, i, j\rangle \mid 0 \leq i \leq j \leq n \wedge \\
& [\omega_1 \ldots \omega_i, X := \alpha/\omega_{i+1} \ldots \omega_j \bullet \beta] \in I\}
\end{aligned}$$

Earley's algorithm

• $\alpha_f(I) \stackrel{\text{def}}{=} \{a \in T \mid [\lambda, X := \alpha/a\gamma \bullet \beta] \in I\}$

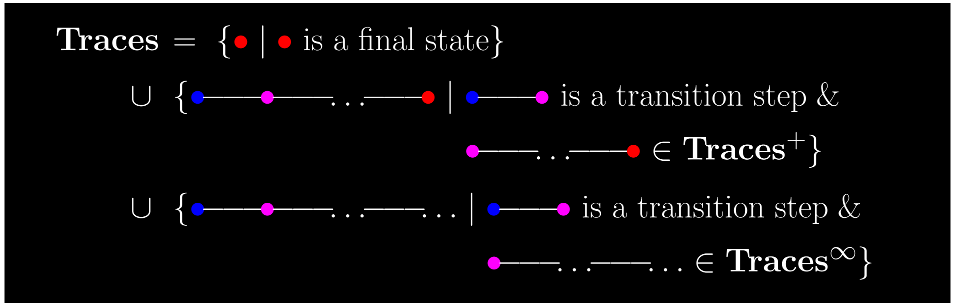
$$\cup \{\epsilon \mid [\lambda, X := \alpha\beta/\epsilon \bullet \epsilon] \in I\}$$

FIRST algorithm

Application to Semantics

- P. Cousot, *Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation*. MFPS XIII, ENTCS 6, 1997. http://www.elsevier.nl/locate/entcs/volume6.html, 25 p.
- P. Cousot, *Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation*, TCS 277(1-2):47–103, 2002.

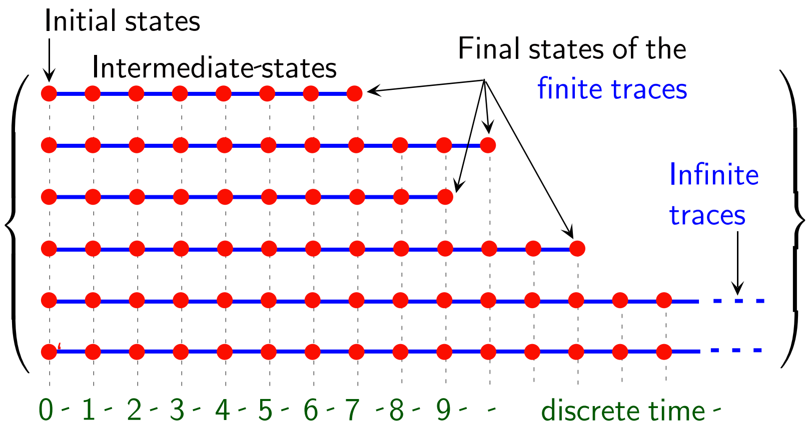
Least Fixpoint Trace Semantics



- In general, the equation has multiple solutions;
- Choose the least one for the computational ordering:

"more finite traces & less infinite traces".

Trace Semantics, intuition



Trace Semantics, Formally

Trace semantics of a transition system $\langle \Sigma, \tau \rangle$:

- $\Sigma^+ \stackrel{\text{def}}{=} \bigcup_{n>0} [0, n[\longmapsto \Sigma$ finite traces
- $\Sigma^w \stackrel{\text{def}}{=} [0, \omega[\longmapsto \Sigma$ infinite traces
- $S = \operatorname{lfp}^{\sqsubseteq} F \in \Sigma^+ \cup \Sigma^\omega$ trace semantics
- $F(X) = \{s \in \Sigma^+ \mid s \in \Sigma \wedge \forall s' \in \Sigma : \langle s, s' \rangle \notin \tau\}$
  $\cup \{ss'\sigma \mid \langle s, s' \rangle \in \tau \wedge s'\sigma \in X\}$ trace transformer
- $X \sqsubseteq Y \stackrel{\text{def}}{=} (X \cap \Sigma^+) \subseteq (Y \cap \Sigma^+) \wedge (X \cap \Sigma^\omega) \supseteq (Y \cap \Sigma^\omega)$
  computational ordering

Semantics Abstractions

1 — Relational Semantics Abstractions

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \subseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \wp(\Sigma \times (\Sigma \cup \{\bot\})), \subseteq \rangle$$

1 — Relational Semantics Abstractions (Cont'd)

- $\alpha^\natural(X) = \{\langle s, s'\rangle \mid s\sigma s' \in X \cap \Sigma^+\}$
  $\cup \{\langle s, \bot\rangle \mid s\sigma \in X \cap \Sigma^\omega\}$
  trace to natural relational semantics
- $\alpha^\flat(X) = \{\langle s, s'\rangle \mid s\sigma s' \in X \cap \Sigma^+\}$
  trace to angelic relational semantics
- $\alpha^\sharp(X) = \{\langle s, s'\rangle \mid s\sigma s' \in X \cap \Sigma^+\}$
  $\cup \{\langle s, s'\rangle \mid s\sigma \in X \cap \Sigma^\omega \wedge s' \in \Sigma \cup \{\bot\}\}$
  trace to demoniac relational semantics

2 — Functional/Denotational Semantics Abstractions

$$\langle \wp(\Sigma \times (\Sigma \cup \{\bot\})), \subseteq \rangle \xleftrightarrow[\alpha^\varphi]{\gamma^\varphi} \langle \Sigma \longmapsto \wp(\Sigma \cup \{\bot\}), \dot{\subseteq} \rangle$$

- $\alpha^\varphi(X) = \lambda s.\{s' \in \Sigma \cup \{\bot\} \mid \langle s, s'\rangle \in X\}$
  relational to denotational semantics

3 — Predicate Transformer Semantics Abstractions

$$\langle \Sigma \longmapsto \wp(\Sigma \cup \{\bot\}), \dot{\subseteq} \rangle \xleftrightarrow[\alpha^\pi]{\gamma^\pi} \langle \wp(\Sigma) \overset{\cup}{\longmapsto} \wp(\Sigma \cup \{\bot\}), \dot{\subseteq} \rangle$$

- $\alpha^\pi(\phi) = \lambda P.\{s' \in \Sigma \cup \{\bot\} \mid \exists s \in P : s' \in \phi(s)\}$
  denotational to predicate transformer semantics

4 — Predicate Transformer Semantics Abstractions (Cont'd)

$$\begin{array}{ccc}
\langle \wp(\Sigma) \overset{\cup}{\mapsto} \wp(\Sigma \cup \{\bot\}), \dot{\subseteq}\rangle & \underset{\alpha^{\sim}}{\overset{\gamma^{\sim}}{\leftrightarrows}} & \langle \wp(\Sigma) \overset{\cap}{\mapsto} \wp(\Sigma \cup \{\bot\}), \dot{\supseteq}\rangle \\
\alpha^{\cup} \downarrow\uparrow \gamma^{\cup} & & \alpha^{\cap} \downarrow\uparrow \gamma^{\cap} \\
\langle \wp(\Sigma \cup \{\bot\}) \overset{\cup}{\mapsto} \wp(\Sigma), \dot{\subseteq}\rangle & \underset{\alpha^{\sim}}{\overset{\gamma^{\sim}}{\leftrightarrows}} & \langle \wp(\Sigma \cup \{\bot\}) \overset{\cap}{\mapsto} \wp(\Sigma), \dot{\supseteq}\rangle
\end{array}$$

- $\alpha^{\sim}(\Phi) = \lambda P . \neg(\Phi(\neg P))$ dual
- $\alpha^{\cup}(\Phi) = \lambda Q . \{s \in \Sigma \mid \Phi(\{s\}) \cap Q \neq \emptyset\}$ $\cup$-inversion
- $\alpha^{\cap}(\Phi) = \lambda Q . \{s \in \Sigma \mid \Phi(\neg\{s\}) \cup Q = \Sigma \cup \{\bot\}\}$ $\cap$-inversion

5 — Hoare Logic Semantics Abstractions

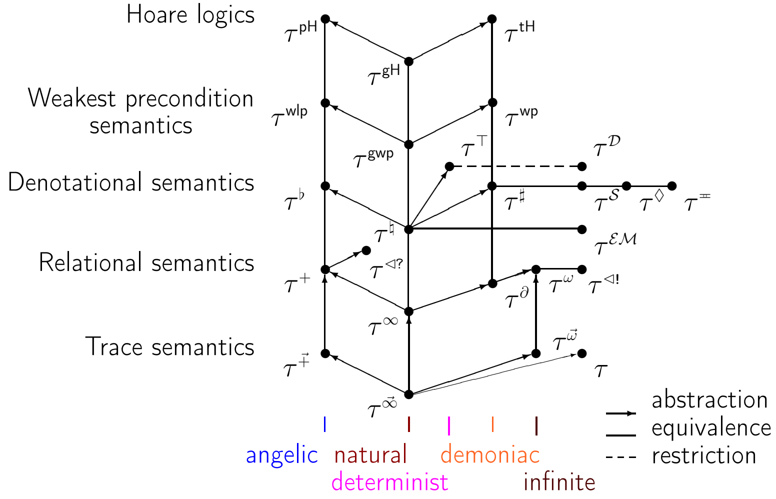
$$\langle \wp(\Sigma) \overset{\cap}{\longmapsto} \wp(\Sigma \cup \{\bot\}), \dot{\supseteq}\rangle \underset{\alpha^{H}}{\overset{\gamma^{H}}{\leftrightarrows}} \wp(\Sigma) \otimes^{17} \wp(\Sigma \cup \{\bot\}), \dot{\supseteq}\rangle$$

- $\alpha^{H}(\Phi) = \{\langle P, Q\rangle \mid P \subseteq \Phi(Q)\}$

predicate transformer to Hoare logic semantics

[17] Semi-dual Shmuely tensor product.

Lattice of Semantics



Application to Typing

- P. Cousot, *Types as Abstract Interpretations*, ACM 24th POPL, 1997, pp. 316-331.

Syntax of the Eager Lambda Calculus

$$\begin{array}{rll} \mathrm{x}, \mathrm{f}, \ldots \in \mathbb{X} & : & \text{variables} \\ e \in \mathbb{E} & : & \text{expressions} \\ e ::= & \mathrm{x} & \text{variable} \\ \mid & \boldsymbol{\lambda}\mathrm{x} \cdot e & \text{abstraction} \\ \mid & e_1(e_2) & \text{application} \\ \mid & \boldsymbol{\mu}\mathrm{f} \cdot \boldsymbol{\lambda}\mathrm{x} \cdot e & \text{recursion} \\ \mid & \mathbf{1} & \text{one} \\ \mid & e_1 - e_2 & \text{difference} \\ \mid & (e_1 \mathbin{?} e_2 : e_3) & \text{conditional} \end{array}$$

Semantic Domains

$$\begin{array}{ll} \Omega & \text{wrong/runtime error value} \\ \bot & \text{non-termination} \\ \mathbb{W} \stackrel{\text{def}}{=} \{\Omega\} & \text{wrong} \\ \mathsf{z} \in \mathbb{Z} & \text{integers} \\ \mathsf{u}, \mathsf{f}, \varphi \in \mathbb{U} \cong \mathbb{W}_\bot \oplus \mathbb{Z}_\bot \oplus [\mathbb{U} \mapsto \mathbb{U}]^{18}{}_\bot & \text{values} \\ \mathsf{R} \in \mathbb{R} \stackrel{\text{def}}{=} \mathbb{X} \mapsto \mathbb{U} & \text{environments} \\ \phi \in \mathbb{S} \stackrel{\text{def}}{=} \mathbb{R} \mapsto \mathbb{U} & \text{semantic domain} \end{array}$$

18 $[\mathbb{U} \mapsto \mathbb{U}]$: continuous, $\bot$-strict, Ω-strict functions from values $\mathbb{U}$ to values $\mathbb{U}$.

Denotational Semantics with Run-Time Type Checking

$$\mathsf{S}[\![\mathbf{1}]\!]\mathsf{R} \stackrel{\text{def}}{=} 1$$

$$\begin{array}{rl} \mathsf{S}[\![e_1 - e_2]\!]\mathsf{R} \stackrel{\text{def}}{=} & (\, \mathsf{S}[\![e_1]\!]\mathsf{R} = \bot \vee \mathsf{S}[\![e_2]\!]\mathsf{R} = \bot \mathbin{?} \bot \\ & \mid \mathsf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z}_1 \wedge \mathsf{S}[\![e_2]\!]\mathsf{R} = \mathsf{z}_2 \mathbin{?} \mathsf{z}_1 - \mathsf{z}_2 \\ & \mid \Omega\,) \end{array}$$

$$\begin{array}{rl} \mathsf{S}[\![(e_1 \mathbin{?} e_2 : e_3)]\!]\mathsf{R} \stackrel{\text{def}}{=} & (\, \mathsf{S}[\![e_1]\!]\mathsf{R} = \bot \mathbin{?} \bot \\ & \mid \mathsf{S}[\![e_1]\!]\mathsf{R} = 0 \mathbin{?} \mathsf{S}[\![e_2]\!]\mathsf{R} \\ & \mid \mathsf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z} \neq 0 \mathbin{?} \mathsf{S}[\![e_3]\!]\mathsf{R} \\ & \mid \Omega\,) \end{array}$$

$$\mathsf{S}[\![\mathrm{x}]\!]\mathsf{R} \stackrel{\text{def}}{=} \mathsf{R}(\mathrm{x})$$

$$\begin{array}{rl} \mathsf{S}[\![\boldsymbol{\lambda}\mathrm{x} \cdot e]\!]\mathsf{R} \stackrel{\text{def}}{=} \boldsymbol{\lambda}\,\mathsf{u} \cdot & (\, \mathsf{u} = \bot \mathbin{?} \bot \\ & \mid \mathsf{u} = \Omega \mathbin{?} \Omega \\ & \mid \mathsf{S}[\![e]\!]\mathsf{R}[\mathrm{x} \leftarrow \mathsf{u}]\,) \end{array}$$

$$\begin{array}{rl} \mathsf{S}[\![e_1(e_2)]\!]\mathsf{R} \stackrel{\text{def}}{=} & (\, \mathsf{S}[\![e_1]\!]\mathsf{R} = \bot \vee \mathsf{S}[\![e_2]\!]\mathsf{R} = \bot \mathbin{?} \bot \\ & \mid \mathsf{S}[\![e_1]\!]\mathsf{R} = \mathsf{f} \in [\mathbb{U} \mapsto \mathbb{U}] \mathbin{?} \mathsf{f}(\mathsf{S}[\![e_2]\!]\mathsf{R}) \\ & \mid \Omega\,) \end{array}$$

$$\mathsf{S}[\![\boldsymbol{\mu}\mathrm{f} \cdot \boldsymbol{\lambda}\mathrm{x} \cdot e]\!]\mathsf{R} \stackrel{\text{def}}{=} \mathrm{lfp}^{\sqsubseteq}\, \boldsymbol{\lambda}\,\varphi \cdot \mathsf{S}[\![\boldsymbol{\lambda}\mathrm{x} \cdot e]\!]\mathsf{R}[\mathrm{f} \leftarrow \varphi]$$

Standard Denotational & Collecting Semantics

- The denotational semantics is:

$$\mathsf{S}[\![\bullet]\!] \in \mathbb{E} \mapsto \mathbb{S}$$

- A concrete property P of a program is a set of possible program behaviors:

$$P \in \mathbb{P} \stackrel{\text{def}}{=} \wp(\mathbb{S})$$

- The standard collecting semantics is the strongest concrete property:

$$\mathsf{C}[\![\bullet]\!] \in \mathbb{E} \mapsto \mathbb{P} \qquad \mathsf{C}[\![e]\!] \stackrel{\text{def}}{=} \{\mathsf{S}[\![e]\!]\}$$

Church/Curry Monotypes

- Simple types are monomorphic:

$$m \in \mathbb{M}^{\mathsf{c}}, \quad m ::= \mathtt{int} \mid m_1 \mathbin{\texttt{->}} m_2 \qquad \text{monotype}$$

- A type environment associates a type to free program variables:

$$H \in \mathbb{H}^{\mathsf{c}} \stackrel{\text{def}}{=} \mathbb{X} \mapsto \mathbb{M}^{\mathsf{c}} \qquad \text{type environment}$$

Church/Curry Monotypes (continued)

- A typing $\langle H, m\rangle$ specifies a possible result type m in a given type environment H assigning types to free variables:

$$\theta \in \mathbb{I}^{\mathsf{c}} \stackrel{\text{def}}{=} \mathbb{H}^{\mathsf{c}} \times \mathbb{M}^{\mathsf{c}} \qquad \text{typing}$$

- An abstract property or program type is a set of typings;

$$T \in \mathbb{T}^{\mathsf{c}} \stackrel{\text{def}}{=} \wp(\mathbb{I}^{\mathsf{c}}) \qquad \text{program type}$$

Concretization Function

The meaning of types is a program property, as defined by the concretization function γ^{c}: [19]

- Monotypes $\gamma_1^{\mathsf{c}} \in \mathbb{M}^{\mathsf{c}} \mapsto \wp(\mathbb{U})$:

$$\begin{aligned}
\gamma_1^{\mathsf{c}}(\mathtt{int}) &\stackrel{\text{def}}{=} \mathbb{Z} \cup \{\bot\}\\
\gamma_1^{\mathsf{c}}(m_1 \mathbin{\texttt{->}} m_2) &\stackrel{\text{def}}{=} \{\varphi \in [\mathbb{U} \mapsto \mathbb{U}] \mid \\
&\qquad \forall \mathsf{u} \in \gamma_1^{\mathsf{c}}(m_1) : \varphi(\mathsf{u}) \in \gamma_1^{\mathsf{c}}(m_2)\}\\
&\quad \cup \{\bot\}
\end{aligned}$$

[19] For short up/down lifting/injection are omitted.

- type environment $\gamma_2^{\mathsf{c}} \in \mathbb{H}^{\mathsf{c}} \mapsto \wp(\mathbb{R})$:
$$\gamma_2^{\mathsf{c}}(H) \stackrel{\text{def}}{=} \{\mathsf{R} \in \mathbb{R} \mid \forall \mathrm{x} \in \mathbb{X} : \mathsf{R}(\mathrm{x}) \in \gamma_1^{\mathsf{c}}(H(\mathrm{x}))\}$$

- typing $\gamma_3^{\mathsf{c}} \in \mathbb{I}^{\mathsf{c}} \mapsto \mathbb{P}$:
$$\gamma_3^{\mathsf{c}}(\langle H, m\rangle) \stackrel{\text{def}}{=} \{\phi \in \mathbb{S} \mid \forall \mathsf{R} \in \gamma_2^{\mathsf{c}}(H) : \phi(\mathsf{R}) \in \gamma_1^{\mathsf{c}}(m)\}$$

- program type $\gamma^{\mathsf{c}} \in \mathbb{T}^{\mathsf{c}} \mapsto \mathbb{P}$:
$$\gamma^{\mathsf{c}}(T) \stackrel{\text{def}}{=} \bigcap_{\theta \in T} \gamma_3^{\mathsf{c}}(\theta)$$
$$\gamma^{\mathsf{c}}(\emptyset) \stackrel{\text{def}}{=} \mathbb{S}$$

Program Types

- Galois connection:
$$\langle \mathbb{P}, \subseteq, \emptyset, \mathbb{S}, \cup, \cap\rangle \underset{\alpha^{\mathsf{c}}}{\overset{\gamma^{\mathsf{c}}}{\longleftrightarrow}} \langle \mathbb{T}^{\mathsf{c}}, \supseteq, \mathbb{I}^{\mathsf{c}}, \emptyset, \cap, \cup\rangle$$

- Types $\mathsf{T}[\![e]\!]$ of an expression e:
$$\mathsf{T}[\![e]\!] \subseteq \alpha^{\mathsf{c}}(\mathsf{C}[\![e]\!]) = \alpha^{\mathsf{c}}(\{\mathsf{S}[\![e]\!]\})$$

Typable Programs Cannot Go Wrong

$$\Omega \in \gamma^{\mathsf{c}}(\mathsf{T}[\![e]\!]) \quad \Longleftrightarrow \quad \mathsf{T}[\![e]\!] = \emptyset$$

Church/Curry Monotype Abstract Semantics

$$\mathsf{T}[\![\mathrm{x}]\!] \stackrel{\text{def}}{=} \{\langle H, H(\mathrm{x})\rangle \mid H \in \mathbb{H}^{\mathsf{c}}\} \qquad \text{(VAR)}$$

$$\mathsf{T}[\![\boldsymbol{\lambda}\mathrm{x} \cdot e]\!] \stackrel{\text{def}}{=} \{\langle H, m_1 \rightarrow m_2\rangle \mid \langle H[\mathrm{x} \leftarrow m_1], m_2\rangle \in \mathsf{T}[\![e]\!]\} \qquad \text{(ABS)}$$

$$\mathsf{T}[\![e_1(e_2)]\!] \stackrel{\text{def}}{=} \{\langle H, m_2\rangle \mid \langle H, m_1 \rightarrow m_2\rangle \in \mathsf{T}[\![e_1]\!] \wedge \langle H, m_1\rangle \in \mathsf{T}[\![e_2]\!]\} \qquad \text{(APP)}$$

$$\mathsf{T}[\![1]\!] \stackrel{\text{def}}{=} \{\langle H, \mathtt{int}\rangle \mid H \in \mathbb{H}^{\mathsf{c}}\} \qquad \text{(CST)}$$

$$\mathsf{T}[\![e_1 - e_2]\!] \stackrel{\text{def}}{=} \{\langle H, \mathtt{int}\rangle \mid \langle H, \mathtt{int}\rangle \in \mathsf{T}[\![e_1]\!] \cap \mathsf{T}[\![e_2]\!]\} \qquad \text{(DIF)}$$

$$\mathsf{T}[\![(e_1\,?\,e_2 : e_3)]\!] \stackrel{\text{def}}{=} \{\langle H, m\rangle \mid \langle H, \mathtt{int}\rangle \in \mathsf{T}[\![e_1]\!] \wedge \langle H, m\rangle \in \mathsf{T}[\![e_2]\!] \cap \mathsf{T}[\![e_3]\!]\} \qquad \text{(CND)}$$

$$\mathsf{T}[\![\boldsymbol{\mu}\mathtt{f} \cdot \boldsymbol{\lambda}\mathrm{x} \cdot e]\!] \stackrel{\text{def}}{=} \{\langle H, m\rangle \mid \langle H[\mathtt{f} \leftarrow m], m\rangle \in \mathsf{T}[\![\boldsymbol{\lambda}\mathrm{x} \cdot e]\!]\} \qquad \text{(REC)}$$ [20]

[20] The abstract fixpoint has been eliminated thanks to fixpoint induction: $\mathrm{lfp} F \sqsubseteq P \Leftrightarrow \exists I : F(I) \sqsubseteq I \wedge I \sqsubseteq P$.

The Herbrand Abstraction to Get Hindley's Unification-Based Type Inference Algorithm

$$\langle \wp(\mathbf{ground}(T)),\ \subseteq,\ \emptyset,\ \mathbf{ground}(T),\ \cup,\ \cap\rangle \underset{\mathbf{lcg}}{\overset{\mathbf{ground}}{\leftrightarrows}} \langle \overset{\emptyset}{T}/_{\equiv},\ \leq,\ \emptyset,\ [\text{'a}]_{\equiv},\ \mathbf{lcg},\ \mathbf{gci}\rangle$$

where:

- T: set of terms with variables 'a, . . . ,
- $\mathbf{lcg}$: least common generalization,
- $\mathbf{ground}$: set of ground instances,
- $\leq$: instance preordering,
- $\mathbf{gci}$: greatest common instance.

Application to Model Checking

- P. Cousot & R. Cousot, *Temporal Abstract Interpretation*, ACM 27th POPL, 2000, pp. 12-25.

Objective of Model Checking

1) Built a model M of the computer system;
2) Check (i.e. prove enumeratively) or semi-check (with semi-algorithms) that the model satisfies a specification given (as a set of traces φ) by a (linear) temporal formula: $M \subseteq \varphi$ or $M \cap \varphi \neq \emptyset$.

- The model and specification should be proved to be correct abstractions of the computer system (often taken for granted, could be done by abstract interpretation);

Abstractions in Model Checking

Main abstractions in model checking:

- Implicit abstraction: to informally design the model of reference;
- Polyhedral abstraction (with widening): synchronous, real-time & hybrid system verification;
- Finitary abstraction (without widening): hardware & protocole verification [21];

[21] Abstracting concrete transition systems to abstract transition systems so as to reuse existing model checkers in the abstract.

Model-checking itself is an abstraction

- Universal abstraction:

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \supseteq \rangle \underset{\alpha^\forall_M}{\overset{\gamma^\forall_M}{\leftrightarrows}} \langle \wp(\Sigma), \supseteq \rangle$$

$$\alpha^\forall_M(\Phi) \stackrel{\text{def}}{=} \{s \mid \{\sigma \in M \mid \sigma_0 = s\} \subseteq \Phi\}$$

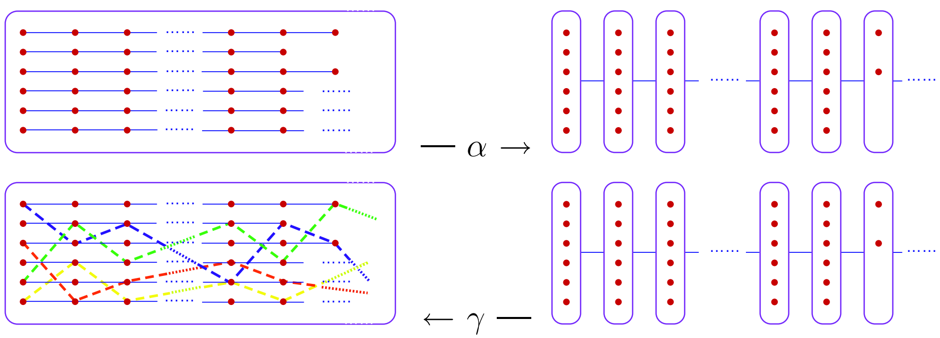
- Existential abstraction:

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \subseteq \rangle \underset{\alpha^\exists_M}{\overset{\gamma^\exists_M}{\leftrightarrows}} \langle \wp(\Sigma), \subseteq \rangle$$

$$\alpha^\exists_M(\Phi) \stackrel{\text{def}}{=} \{s \mid \{\sigma \in M \mid \sigma_0 = s\} \cap \Phi \neq \emptyset\}$$

These abstractions lead, by fixpoint approximation of the trace semantics, to the classical (finite-state or nonterminating) model-checking algorithms.

Implicit Abstraction in Model Checking



Spurious traces: , , , , ⋯ ;

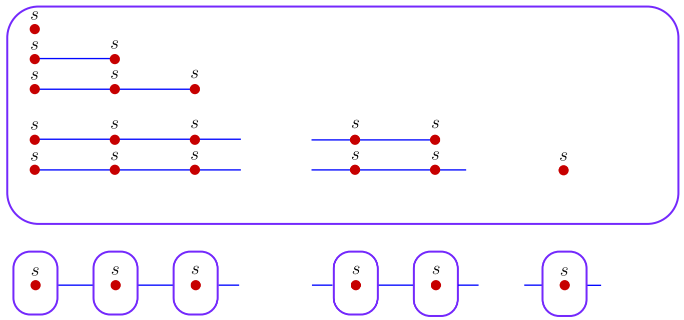
The semantics of the μ-calculus is closed under this abstraction.

Soundness

For a *given class* of properties, soundness means that:

Any property (in the *given class*) of the abstract world must hold in the concrete world;

Example for Unsoundness



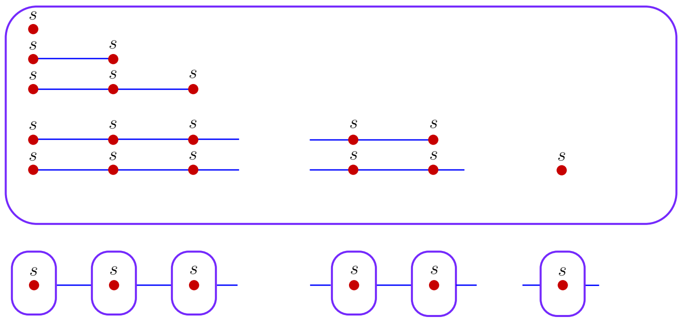
All abstract traces are infinite but not the concrete ones!

Completeness

For a *given class* of properties, completeness means that:

> Any property (in the *given class*) of the concrete world must hold in the abstract world;

Example for Incompleteness



All concrete traces are finite but not the abstract ones!

On the Completeness of Model-Checking

- Contrary to program analysis, model checking is complete;
- Completeness is relative to the model, not the program semantics;
- Completeness follows from restrictions on the models and specifications (e.g. closure under the implicit abstraction);
- There are models/specifications (such as the $\widehat{\mu}^\star$-calculus using bidirectional traces) for which:
  - The implicit abstraction is incomplete (POPL'00),
  - Any abstraction is incomplete (Ranzato, ESOP'01).

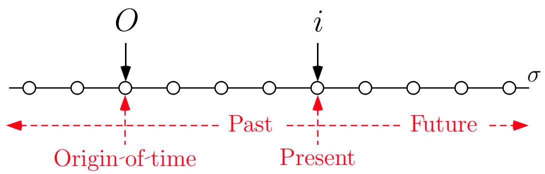
  in both cases, even for *finite* transition systems.

Bidirectional Traces

- $\langle i, \sigma \rangle$ bidirectional trace

  $\sigma \in \mathbb{Z} \longmapsto \Sigma$ trace

  $i \in \mathbb{Z}$ present time



The reversible $\mu^{\!\circ}$-calculus

$$
\begin{array}{lll}
\varphi ::= \boldsymbol{\sigma}_S{}^{22} & & [\![\boldsymbol{\sigma}_S]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \sigma_i \in S\} \\
\quad \mid \ \boldsymbol{\pi}_t{}^{23} & & [\![\boldsymbol{\pi}_t]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \langle \sigma_i, \sigma_{i+1}\rangle \in t\} \\
\quad \mid \ \oplus\varphi_1{}^{24} & & [\![\oplus\varphi_1]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \langle i+1, \sigma\rangle \in [\![\varphi_1]\!]\rho\} \\
\quad \mid \ \varphi_1{}^{\curvearrowleft} & & [\![\varphi_1{}^{\curvearrowleft}]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \langle -i, \lambda j . \sigma_{-j}\rangle \in [\![\varphi_1]\!]\rho\} \\
\quad \mid \ \varphi_1 \vee \varphi_2 & & [\![\varphi_1 \vee \varphi_2]\!]\rho \stackrel{\text{def}}{=} [\![\varphi_1]\!]\rho \cup [\![\varphi_2]\!]\rho \\
\quad \mid \ \neg\varphi_1 & & [\![\neg\varphi_1]\!]\rho \stackrel{\text{def}}{=} \neg[\![\varphi_1]\!]\rho
\end{array}
$$

The reversible $\mu^{\!\circ}$-calculus (cont'd)

$$
\begin{array}{ll}
\mid \ \ldots & \\
\mid \ X^{25} & [\![X]\!]\rho \stackrel{\text{def}}{=} \rho(X) \\
\mid \ \boldsymbol{\mu}\, X \cdot \varphi_1 & [\![\boldsymbol{\mu}\, X \cdot \varphi_1]\!]\rho \stackrel{\text{def}}{=} \operatorname{lfp}^{\subseteq} \boldsymbol{\lambda}\, x \cdot [\![\varphi_1]\!]\rho X x \\
\mid \ \boldsymbol{\nu}\, X \cdot \varphi_1 & [\![\boldsymbol{\nu}\, X \cdot \varphi_1]\!]\rho \stackrel{\text{def}}{=} \operatorname{gfp}^{\subseteq} \boldsymbol{\lambda}\, x \cdot [\![\varphi_1]\!]\rho X x \\
\mid \ \forall\varphi_1 : \varphi_2{}^{26} & [\![\forall\varphi_1 : \varphi_2]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \in [\![\varphi_1]\!]\rho \mid \\
& \qquad \{\langle i, \sigma'\rangle \in [\![\varphi_1]\!]\rho \mid \sigma'_i = \sigma_i\} \subseteq [\![\varphi_2]\!]\rho\}
\end{array}
$$

22 $S \in \wp(\Sigma)$.

23 $t \in \wp(\Sigma \times \Sigma)$.

24 $\oplus$ is next time.

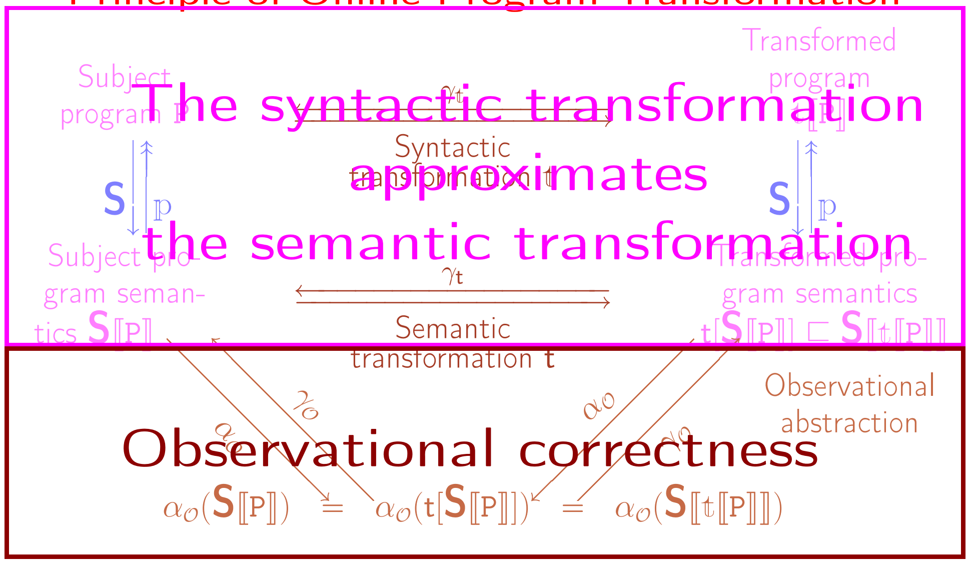
25 variable.

26 The traces of φ_1 such that all traces of φ_1 with same present state satisfy φ_2.

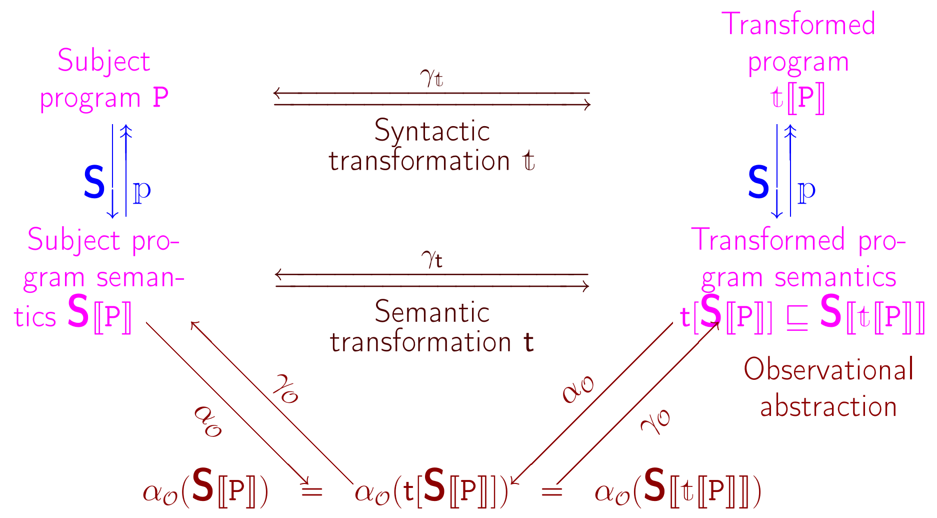
Application to Program Transformation

- P. Cousot & R. Cousot, *Systematic Design of Program Transformation Frameworks by Abstract Interpretation*, ACM 29th POPL, 2002, pp. 178—190.

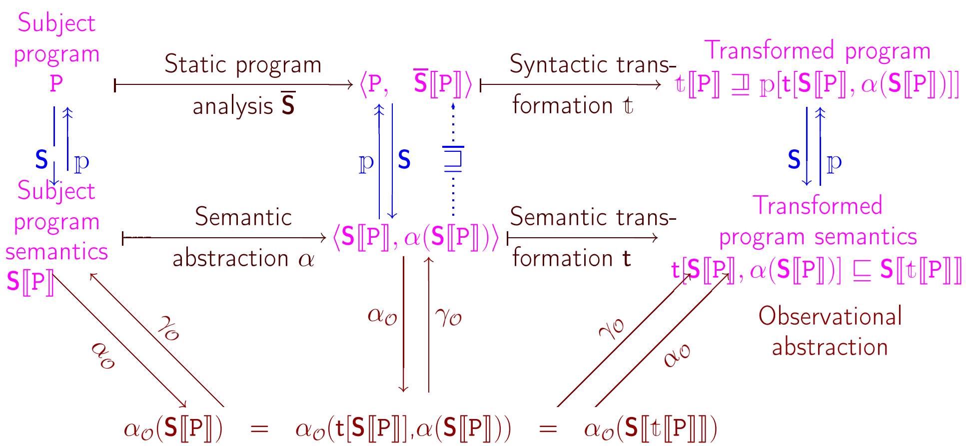
Principle of Online Program Transformation



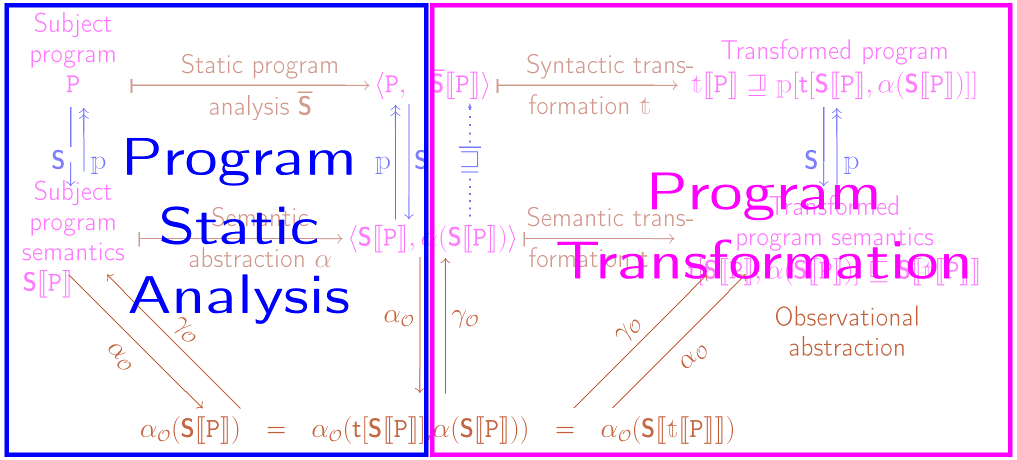
Principle of Online Program Transformation



Principle of Offline Program Transformation



Principle of Offline Program Transformation



Examples of Program Transformations

- Constant propagation;
- Online and offline partial evaluation;
- Slicing;
- Static program monitoring,

$$\alpha_{\mathcal{O}}(\mathbf{S}[\![\mathbb{t}[\![\mathtt{P},\mathtt{M}]\!]]\!]) = \alpha_{\mathcal{O}}(\mathbf{S}[\![\mathtt{P}]\!]) \sqcap \alpha_{\mathcal{O}}(\mathbf{S}[\![\mathtt{M}]\!]):$$

  - run-time checks elimination,
  - security policy enforcement,
  - proof by transformation $(\alpha_{\mathcal{O}}(\mathbf{S}[\![\mathtt{P}]\!]) = \alpha_{\mathcal{O}}(\mathbf{S}[\![\mathbb{t}[\![\mathtt{P},\mathtt{M}]\!]]\!]))$.
- Code and analysis translation [27].

[27] X. Rival. D.E.A. report, 2002.

Application to Static Program Analysis[28]

- 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.
- P. Cousot. *Semantic Foundations of Program Analysis.* Ch. 10 of *Program Flow Analysis: Theory and Applications*, S.S. Muchnick & N.D. Jones, pp. 303–342. Prentice-Hall, 1981.

What is static program analysis?

- Automatic static/compile time determination of dynamic/run-time properties of programs;
- **Basic idea:** use effective computable approximations of the program semantics;

  Advantage: fully automatic, no need for error-prone user designed model or costly user interaction;

  Drawback: can only handle properties captured by the approximation.

[28] Now called *software model checking*!

Collecting Semantics Abstractions

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \subseteq\rangle \xleftarrow[\alpha]{\gamma} \langle \wp(\Sigma), \subseteq\rangle$$

Example 1: reachable states (forward analysis)

$$\alpha_I(X) \stackrel{\text{def}}{=} \{\sigma_i \mid \sigma \in X \wedge \sigma_0 \in I \wedge i \in \mathrm{Dom}(\sigma)\}$$

Example 2: ancestor states (backward analysis)

$$\alpha_F(X) \stackrel{\text{def}}{=} \{\sigma_i \mid \sigma \in X \wedge \exists n \in \mathrm{Dom}(\sigma) : 0 \leq i \leq n \wedge \sigma_n \in F\}$$

Partitioning

- If $\Sigma = C \times M$ (control and store state) and C is finite[29], we can partition:

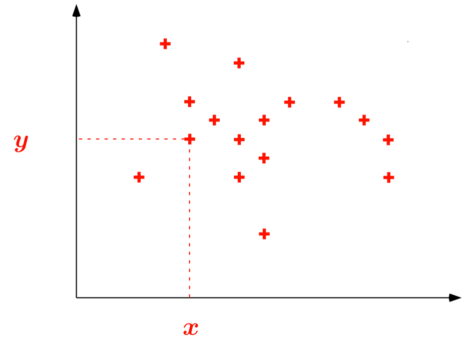
$$\langle \wp(C \times M), \subseteq\rangle \xleftarrow[\alpha_c]{\gamma_c} \langle C \mapsto \wp(M), \dot{\subseteq}\rangle$$

$$\alpha_c(S) \;=\; \boldsymbol{\lambda} c \in C \cdot \{m \mid \langle c, m\rangle \in S\}$$

- It remains to find abstractions of the store $M = V \mapsto D$ (variables to data) e.g. of [in]finite set of points of the euclidian space.

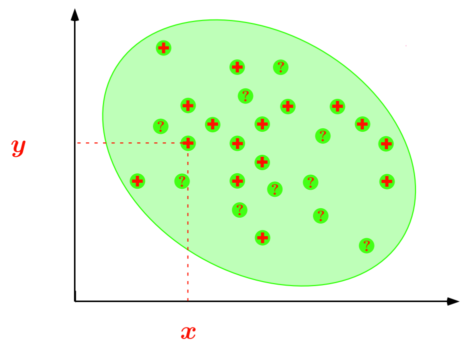
[29] use e.g. dynamic partitioning if C is infinite

Approximations of an [in]finite set of points:



$$\{\ldots, \langle 19,\ 77\rangle, \ldots,\\ \langle 20,\ 02\rangle, \ldots\}$$

Approximations of an [in]finite set of points: From Above

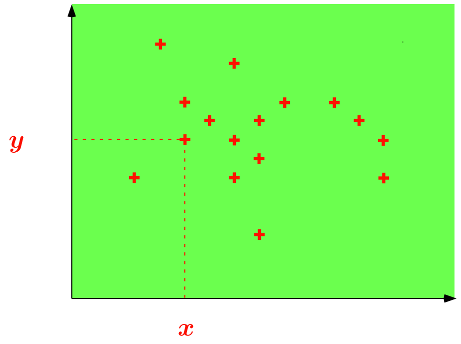


$$\{\ldots, \langle 19,\ 77\rangle, \ldots,\\ \langle 20,\ 02\rangle, \langle ?,\ ?\rangle, \ldots\}$$

From Below: dual[30] + combinations.

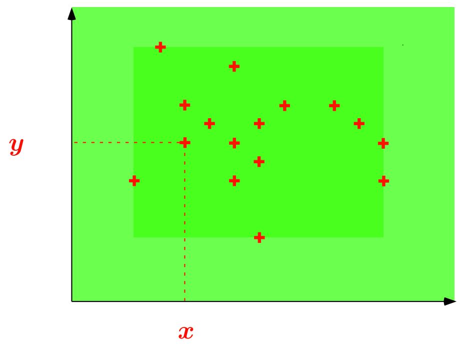
[30] Trivial for finite states (liveness model-checking), more difficult for infinite states (variant functions).

Effective computable approximations of an [in]finite set of points; Signs[31]



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

Effective computable approximations of an [in]finite set of points; Intervals[32]

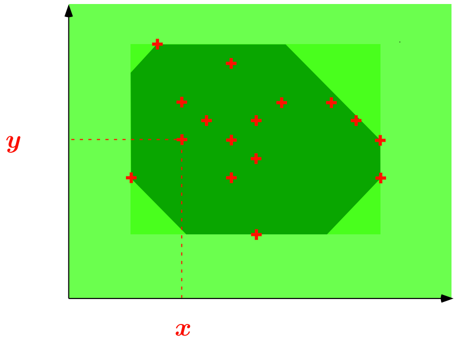


$$\begin{cases} x \in [19,\ 77] \\ y \in [20,\ 02] \end{cases}$$

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

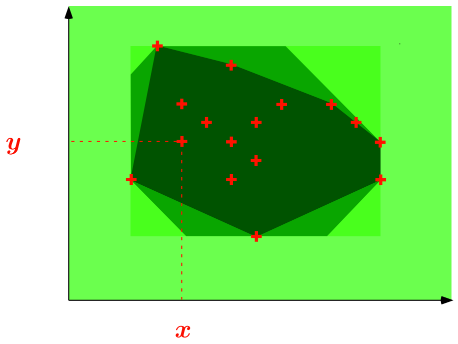
[32] P. Cousot & R. Cousot. *Static determination of dynamic properties of programs.* Proc. 2nd Int. Symp. on Programming, Dunod, 1976.

Effective computable approximations of an [in]finite set of points; Octagons [33]



$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Polyhedra [34]

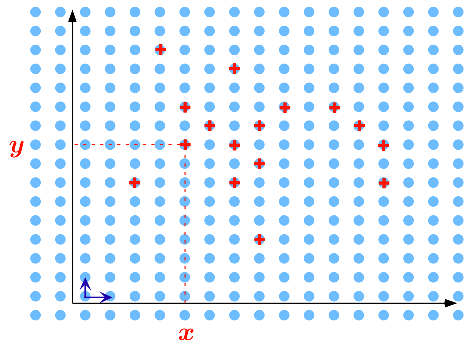


$$\begin{cases} 19x + 77y \leq 2002 \\ 20x + 02y \geq 0 \end{cases}$$

[33] A. Miné. *A New Numerical Abstract Domain Based on Difference-Bound Matrices.* PADO '2001. LNCS 2053, pp. 155–172. Springer 2001.

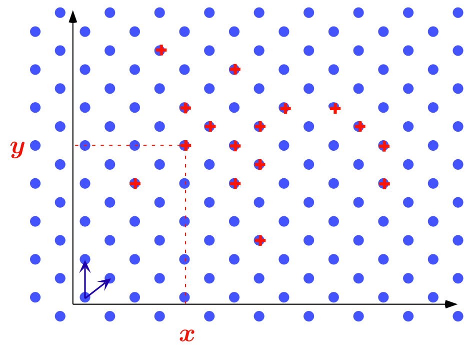
[34] P. Cousot & N. Halbwachs. *Automatic discovery of linear restraints among variables of a program.* ACM POPL, 1978, pp. 84–97.

Effective computable approximations of an [in]finite set of points; Simple congruences [35]



$$\begin{cases} x = 19 \bmod 77 \\ y = 20 \bmod 99 \end{cases}$$

Effective computable approximations of an [in]finite set of points; Linear congruences [36]

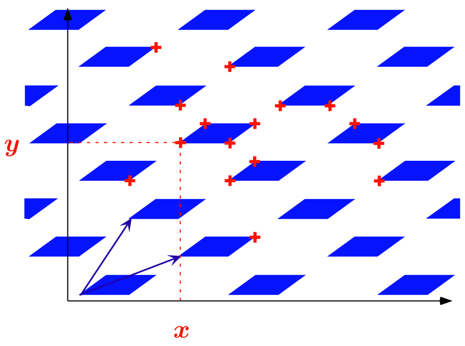


$$\begin{cases} 1x + 9y = 7 \bmod 8 \\ 2x - 1y = 9 \bmod 9 \end{cases}$$

[35] Ph. Granger. *Static Analysis of Arithmetical Congruences.* Int. J. Comput. Math. 30, 1989, pp. 165–190.

[36] Ph. Granger. *Static Analysis of Linear Congruence Equalities among Variables of a Program.* TAPSOFT '91, pp. 169–192. LNCS 493, Springer, 1991.

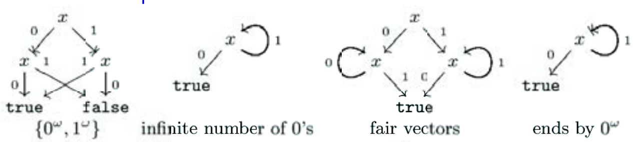
Effective computable approximations of an [in]finite set of points; Trapezoidal linear congruences[37]



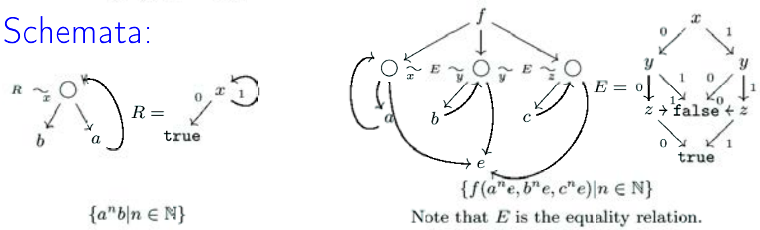
$$\begin{cases} 1x + 9y \in [0, 77] \bmod 10 \\ 2x - 1y \in [0, 99] \bmod 11 \end{cases}$$

Example of Effective Abstractions of Infinite Sets of Infinite Trees[38]

Binary Decision Graphs:



Tree Schemata:



[37] F. Masdupuy. *Array Operations Abstraction Using Semantic Analysis of Trapezoid Congruences.* ACM ICS '92.

[38] L. Mauborgne. *Improving the Representation of Infinite Trees to Deal with Sets of Trees.* ESOP'00. LNCS 1782, pp. 275–289, Springer, 2000.

On the Design of Abstractions for Software Checking

P. Cousot. *Partial Completeness of Abstract Fixpoint Checking, invited paper.* In *Proc. 4th Int. Symp. SARA '2000*, B.Y. Choueiry and T. Walsh (Eds). Horseshoe Bay, Texas, USA, 26–29 Jul. 2000, LNAI 1864. Springer-Verlag, pp. 1–25, 2000.

Discovery of Abstractions

- In static program analysis:
  - task of the program analyzer designer (abstract domains),
  - find a sound abstraction providing useful information for all programs,
  - essentially manual,
  - partially automated e.g. by combination & refinement of abstract domains;
- In model checking:
  - task of the user (model),
  - find a sound & complete abstraction required to verify one model,
  - looking for automation (e.g. starting from a trivial or user provided guess and refining by trial and error).

In what consists abstraction discovery?

- Understand the logical nature of the problem of finding an appropriate abstraction (for proving safety properties).

Formalization of the Abstraction Design Problem

Fixpoint Checking

- Model-checking safety properties of transition systems:

$$\operatorname{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S \ ?$$

- Program static analysis by abstract interpretation:

$$\gamma(\operatorname{lfp}^{\leq} \boldsymbol{\lambda} X \cdot \alpha(I \vee F(\gamma(X)))) \leq S \ ?$$

Soundness / (Partial) Completeness

Soundness: a positive abstract answer implies a positive concrete answer. So no error is possible when reasoning in the abstract;

Completeness: a positive concrete answer can always be found in the abstract;

Partial completeness: in case of termination of the abstract fixpoint checking algorithm, no positive answer can be missed.

Termination/resource limitation is therefore considered a separate problem (widening/narrowing, etc.).

Practical Question

Is it possible to automatize the discovery of complete abstractions?

Objective (Formally)

Constructively characterize the abstractions $\langle \alpha, \gamma \rangle$ for which abstract fixpoint algorithms are partially complete.

Concrete Fixpoint Checking

Concrete Fixpoint Checking Problem

- Complete lattice $\langle L, \leq, 0, 1, \vee, \wedge \rangle$;
- Monotonic transformer $F \in L \xrightarrow{\text{mon}} L$;
- Specification $\langle I, S \rangle \in L^2$;

$$\operatorname{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S \ ?$$

Example

- Set of states: Σ;
- Initial states: $I \subseteq \Sigma$;
- Transition relation: $\tau \subseteq \Sigma \times \Sigma$;
- Transition system: $\langle \Sigma, \tau, I \rangle$;
- Complete lattice: $\langle \wp(\Sigma), \subseteq, \emptyset, \Sigma, \cup, \cap \rangle$;
- Right-image of $X \subseteq \Sigma$ by τ:
  $post[\tau](X) \stackrel{\text{def}}{=} \{s' \mid \exists s \in X : \langle s, s' \rangle \in \tau\}$;
- Reflexive transitive closure of τ: $\tau^\star$

Example (contd.)

- Safety specification: $S \subseteq \Sigma$
- Reachable states from I:

$$post[\tau^\star](I) = \operatorname{lfp}^{\subseteq} \boldsymbol{\lambda} X \cdot I \cup post[\tau](X) \ ;$$

- Satisfaction of the safety specification ($post[\tau^\star](I) \subseteq S$):

$$\operatorname{lfp}^{\subseteq} \boldsymbol{\lambda} X \cdot I \vee post[\tau](X) \subseteq S \ ?$$

Concrete Fixpoint Checking Algorithm [39]

Algorithm 1

$X := I;\ Go := (X \leq S);$
while Go **do**
  $X' := I \vee F(X);$
  $Go := (X \neq X')\ \&\ (X' \leq S);$
  $X := X';$
od;
return $(X \leq S)$;

Partial correctness of Alg. 1

Alg. 1 is partially correct: if it ever terminates then it returns $\operatorname{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$.

[39] P. Cousot & R. Cousot, POPL'77

Concrete Invariants

$A \in L$ is an *invariant* for $\langle F,\ I,\ S\rangle$ if and only if $I \leq A \,\&\, F(A) \leq A \,\&\, A \leq S$;

Note 1 (Floyd's proof method): $\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$ if and only if there exists an invariant $A \in L$ for $\langle F,\ I,\ S\rangle$;

Note 2: if Alg. 1 terminates successfully, then it has computed an invariant $(X = \mathrm{lfp}^{\leq} \boldsymbol{\lambda} X' \cdot I \vee F(X'))$.

Galois connection

A *Galois connection*, written

$$\langle L, \leq\rangle \underset{f}{\overset{g}{\leftrightarrows}} \langle M, \sqsubseteq\rangle,$$

is such that:

- $\langle L, \leq\rangle$ and $\langle M, \sqsubseteq\rangle$ are posets;
- the maps $f \in L \mapsto M$ and $g \in M \mapsto L$ satisfy

$$\forall x \in L : \forall y \in M : f(x) \sqsubseteq y \text{ if and only if } x \leq g(y)\ .$$

Dual and Adjoined Concrete Fixpoint Checking

Concrete Adjoinedness

In general, F has an *adjoint* $\widetilde{F}$ such that $\langle L, \leq\rangle \underset{F}{\overset{\widetilde{F}}{\leftrightarrows}} \langle L, \leq\rangle$.

Example of Concrete Adjoinedness

- τ^{-1} is the inverse of τ;
- $pre[\tau] \stackrel{\text{def}}{=} post[\tau^{-1}]$;
- Set complement $\neg X \stackrel{\text{def}}{=} \Sigma \setminus X$;
- $\widetilde{pre}[\tau](X) \stackrel{\text{def}}{=} \neg pre[\tau](\neg X)$;

$$\langle \wp(\Sigma), \subseteq \rangle \underset{post[\tau]}{\overset{\widetilde{pre}[\tau]}{\leftrightarrows}} \langle \wp(\Sigma), \subseteq \rangle \ .$$

Fixpoint Concrete Adjoinedness

$$\langle L, \leq \rangle \underset{\boldsymbol{\lambda} I \cdot \text{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X)}{\overset{\boldsymbol{\lambda} S \cdot \text{gfp}^{\leq} \boldsymbol{\lambda} X \cdot S \wedge \widetilde{F}(X)}{\leftrightarrows}} \langle L, \leq \rangle$$

Proof:

$$\begin{aligned}
& \text{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S \\
\Longleftrightarrow \ & \exists A \in L : I \leq A \,\&\, F(A) \leq A \,\&\, A \leq S \qquad (1) \\
\Longleftrightarrow \ & \exists A \in L : I \leq A \,\&\, A \leq \widetilde{F}(A) \,\&\, A \leq S \\
\Longleftrightarrow \ & I \leq \text{gfp}^{\leq} \boldsymbol{\lambda} X \cdot S \wedge \widetilde{F}(X) \ .
\end{aligned}$$

The Complete Lattice of Concrete Invariants

- The set $\mathcal{I}$ of invariants for $\langle F, I, S \rangle$ is a complete lattice $\langle \mathcal{I}, \leq, \text{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X), \text{gfp}^{\leq} \boldsymbol{\lambda} X \cdot S \wedge \widetilde{F}(X), \vee, \wedge \rangle$.

Dual Concrete Fixpoint Checking Algorithm [40]

Algorithm 2

$Y := S;\ Go := (I \leq Y);$
while Go **do**
  $Y' := S \wedge \widetilde{F}(Y);$
  $Go := (Y \neq Y') \,\&\, (I \leq Y');$
  $Y := Y';$
od;
return $(I \leq Y);$

[40] P. Cousot, 1981; E.M. Clarke & E.A. Emerson, 1981; J.-P. Queille and J. Sifakis, 1982.

Partial correctness of Alg. 2

Alg. 2 is partially correct: if it ever terminates then it returns $\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$.

On (Dual) Fixpoint Checking

$$\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$$

if and only if

$$I \leq \mathrm{gfp}^{\leq} \boldsymbol{\lambda} X \cdot S \wedge \widetilde{F}(X).$$

if and only if

$$\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq \mathrm{gfp}^{\leq} \boldsymbol{\lambda} X \cdot S \wedge \widetilde{F}(X)$$

The Adjoined Concrete Fixpoint Checking Algorithm

Algorithm 3

$X := I;\ Y := S;\ Go := (X \leq Y);$
while Go **do**
  $X' := I \vee F(X);\ Y' := S \wedge \widetilde{F}(Y);$
  $Go := (X \neq X')\ \&\ (Y \neq Y')\ \&\ (X' \leq Y');$
  $X := X';\ Y := Y';$
od;
return $(X \leq Y)$;

Partial correctness of Alg. 3

Alg. 3 is partially correct: if it ever terminates then it returns $\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$.

Abstract Fixpoint Checking

Abstract Interpretation

- Concrete complete lattice: $\langle L,\ \leq,\ 0,\ 1,\ \vee,\ \wedge\rangle$;
- Abstract complete lattice: $\langle M,\ \sqsubseteq,\ \bot,\ \top,\ \sqcap,\ \sqcup\rangle$;
- Abstraction/concretization pair $\langle \alpha, \gamma\rangle$;
- Galois connection:

$$\langle L, \leq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M, \sqsubseteq\rangle.$$

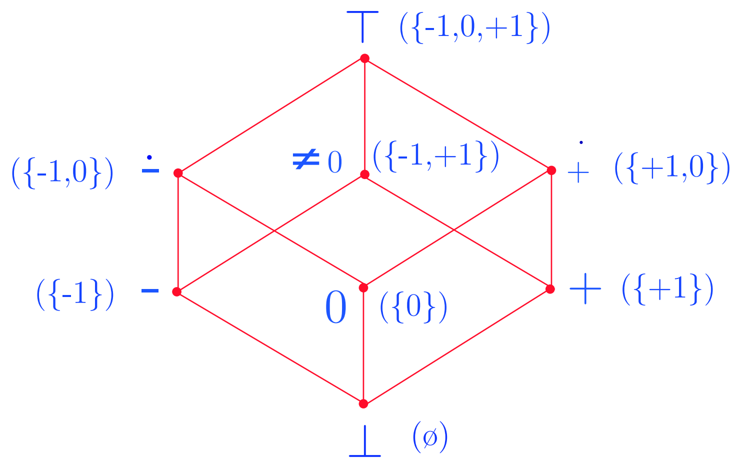
Example: the Recurrent Abstraction in Abstract Model-Checking

- State abstraction: $h \in \Sigma \mapsto \overline{\Sigma}$;
- Property abstraction: $\alpha_h(X) \stackrel{\text{def}}{=} \{h(x) \mid x \in X\} = post[h]$ [41];
- Property concretization: $\gamma_h(Y) \stackrel{\text{def}}{=} \{x \mid h(x) \in Y\} = \widetilde{pre}[h]$;
- Galois connection:

$$\langle \wp(\Sigma), \subseteq\rangle \underset{\alpha_h}{\overset{\gamma_h}{\leftrightarrows}} \langle \wp(\overline{\Sigma}), \subseteq\rangle.$$

- Example (rule of signs): $\Sigma = \mathbb{Z}$ so choose $h(z)$ to be the sign of z.

Example: the Sign Abstraction



[41] Considering the function h as a relation.

Abstract Fixpoint Checking Algorithm [42]

Algorithm 4

$X := \alpha(I);\ Go := (\gamma(X) \leq S);$
while Go **do**
  $X' := \alpha(I \vee F(\gamma(X)));$
  $Go := (X \neq X')\ \&\ (\gamma(X') \leq S);$
  $X := X';$
od;
return if $(\gamma(X) \leq S)$ ***then*** *true* ***else*** *I don't know;*

Partial correctness of Alg. 4

Alg. 4 is partially correct: if it terminates and returns "*true*" then $\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$.

[42] In P. Cousot & R. Cousot, POPL'77, $(\gamma(X) \leq S)$ is $X \sqsubseteq S'$ where $S' = \alpha(S)$.

Dual and Adjoined Abstract Fixpoint Checking

Dual Abstraction

$$\langle L, \geq \rangle \underset{\widetilde{\alpha}}{\overset{\widetilde{\gamma}}{\leftrightarrows}} \langle M, \sqsupseteq \rangle.$$

Example of Dual Abstraction

If

- $\langle L,\ \leq,\ 0,\ 1,\ \vee,\ \wedge,\ \neg\rangle$ is a complete boolean lattice;
- $\langle M,\ \sqsubseteq,\ \bot,\ \top,\ \sqcap,\ \sqcup,\ \backsim\rangle$ is a complete boolean lattice;
- $\langle L,\leq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M,\sqsubseteq\rangle$;
- $\widetilde{\alpha} \stackrel{\text{def}}{=} \backsim \circ \alpha \circ \neg$ and $\widetilde{\gamma} \stackrel{\text{def}}{=} \neg \circ \gamma \circ \backsim$

then

$$\langle L,\geq\rangle \underset{\widetilde{\alpha}}{\overset{\widetilde{\gamma}}{\leftrightarrows}} \langle M,\sqsupseteq\rangle$$

Example of Dual Abstraction (Contd.)

For the recurrent abstraction in abstract model-checking $\alpha_h(X) \stackrel{\text{def}}{=} \{h(x) \mid x \in X\} = post[h]$ we have:

- $\langle \wp(\Sigma), \subseteq\rangle \underset{post[h]}{\overset{\widetilde{pre}[h]}{\leftrightarrows}} \langle \wp(\Sigma), \subseteq\rangle$;
- $\widetilde{pre}[h](X) = \neg pre[h](\neg X)$ and $\widetilde{post}[h](X) = \neg post[h](\neg X)$, so:
- $\langle \wp(\Sigma), \supseteq\rangle \underset{\widetilde{post}[h]}{\overset{pre[h]}{\leftrightarrows}} \langle \wp(\Sigma), \supseteq\rangle$.

Abstract Adjoinedness

$\langle L,\leq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M,\sqsubseteq\rangle$, $\langle L,\leq\rangle \underset{F}{\overset{\widetilde{F}}{\leftrightarrows}} \langle L,\leq\rangle$ and $\langle L,\geq\rangle \underset{\widetilde{\alpha}}{\overset{\widetilde{\gamma}}{\leftrightarrows}} \langle M,\sqsupseteq\rangle$ imply:

$$\langle M,\sqsubseteq\rangle \underset{\alpha\circ F\circ\widetilde{\gamma}}{\overset{\widetilde{\alpha}\circ\widetilde{F}\circ\gamma}{\leftrightarrows}} \langle M,\sqsubseteq\rangle.$$

The Dual Abstract Fixpoint Checking Algorithm

Algorithm 5

> $Y := \widetilde{\alpha}(S)$; $Go := (I \leq \widetilde{\gamma}(Y))$;
> **while** Go **do**
> $Y' := \widetilde{\alpha}(S \wedge \widetilde{F}(\widetilde{\gamma}(Y)))$;
> $Go := (Y \neq Y')\ \&\ (I \leq \widetilde{\gamma}(Y'))$;
> $Y := Y'$;
> **od**;
> **return if** $(I \leq \widetilde{\gamma}(Y))$ ***then*** *true* ***else*** *I don't know*;

Partial correctness of Alg. 5

Alg. 5 is partially correct: if it terminates and returns "*true*" then $\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$.

The Particular Case of Complement Abstraction

1. $\langle L, \leq, 0, 1, \vee, \wedge, \neg\rangle$ is a complete boolean lattice;
2. $\langle M, \sqsubseteq, \bot, \top, \sqcup, \sqcap, \backsim\rangle$ is a complete boolean lattice;
3. $\langle L, \leq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M, \sqsubseteq\rangle$;
4. $\langle L, \leq\rangle \underset{F}{\overset{\widetilde{F}}{\leftrightarrows}} \langle L, \leq\rangle$;
5. $\widetilde{F} \stackrel{\text{def}}{=} \neg \circ F \circ \neg$, $\widetilde{\alpha} \stackrel{\text{def}}{=} \backsim \circ \alpha \circ \neg$ and $\widetilde{\gamma} \stackrel{\text{def}}{=} \neg \circ \gamma \circ \backsim$.

The Contrapositive Abstract Fixpoint Checking Algorithm

Alg. 5 becomes:

Algorithm 6

$Z := \alpha(\neg S)$; $Go := (I \wedge \gamma(Z) = 0)$;
while Go **do**
  $Z' := \alpha(\neg S \vee F(\gamma(Z)))$;
  $Go := (Z \neq Z')$ & $(I \wedge \gamma(Z') = 0)$;
  $Z := Z'$;
od;
return if $(I \wedge \gamma(Z) = 0)$ ***then*** *true* ***else*** *I don't know*;

Partial correctness of Alg. 6

Alg. 6 is partially correct: if it terminates and returns "*true*" then $\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$.

The Adjoined Abstract Fixpoint Checking Algorithm

Algorithm 7

$X := \alpha(I);\ Y := \widetilde{\alpha}(S);\ Go := (\gamma(X) \leq S)\ \&\ (I \leq \widetilde{\gamma}(Y));$
while Go **do**
  $X' := \alpha(I \vee F \circ \gamma(X));\ Y' := \widetilde{\alpha}(S \wedge \widetilde{F} \circ \widetilde{\gamma}(Y));$
  $Go := (X \neq X')\ \&\ (Y \neq Y')\ \&\ (\gamma(X') \leq S)\ \&\ (I \leq \widetilde{\gamma}(Y'));$
  $X := X';\ Y := Y';$
od;
return if $(\gamma(X) \leq S) \mid (I \leq \widetilde{\gamma}(Y))$ ***then*** *true*
  else *I don't know;*

Partial correctness of Alg. 7

Alg. 7 is partially correct: if it terminates and returns "*true*" then $\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$.

Program Static Analysis

Further Requirements for Program Static Analysis

- In program static analysis, one *cannot* compute γ, $\widetilde{\gamma}$ and $\leq$ and sometimes neither I nor S may even be machine representable;
- So Alg. 7, which can be useful in model-checking, is of *limited interest* in program static analysis;
- Such problems do no appear in abstract model checking since the concrete model is almost always machine-representable (although sometimes too large).

Additional Hypotheses

In order to be able to check termination in the abstract, we assume:

1. $\forall X \in L : \gamma \circ \widetilde{\alpha}(X) \leq X$;
2. $\forall X \in L : X \leq \widetilde{\gamma} \circ \alpha(X)$.

Example: the Recurrent Abstraction in Abstract Model-Checking

Continuing with the abstraction of p. 140 with

$$\begin{array}{lll} & \alpha \stackrel{\text{def}}{=} post[h] & \gamma \stackrel{\text{def}}{=} \widetilde{pre}[h] \\ \text{and} & \widetilde{\alpha} \stackrel{\text{def}}{=} \widetilde{post}[h] & \widetilde{\gamma} \stackrel{\text{def}}{=} pre[h], \end{array}$$

we have:

1. $\forall X \in L : \gamma \circ \widetilde{\alpha}(X) \subseteq X$;
2. $\forall X \in L : X \subseteq \widetilde{\gamma} \circ \alpha(X)$.

The Adjoined Abstract Fixpoint Abstract Checking Algorithm

Algorithm 8

$X := \alpha(I);\ Y := \widetilde{\alpha}(S);\ Go := (X \sqsubseteq Y);$
while Go **do**
  $X' := \alpha(I) \sqcup \alpha \circ F \circ \gamma(X);\ Y' := \widetilde{\alpha}(S) \sqcap \widetilde{\alpha} \circ \widetilde{F} \circ \widetilde{\gamma}(Y);$
  $Go := (X \neq X')\ \&\ (Y \neq Y')\ \&\ (X' \sqsubseteq Y');$
  $X := X';\ Y := Y';$
od;
return if $X \sqsubseteq Y$ ***then*** *true* ***else*** *I don't know;*

Partial correctness of Alg. 8

Alg. 8 is partially correct: if it ever terminates and returns "*true*" then $\operatorname{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$.

Partially Complete Abstraction

Partially Complete Abstraction (definition) [43]

Definition 9 The abstraction $\langle \alpha, \gamma \rangle$ is *partially complete* if, whenever Alg. 4 terminates and $\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$ then the returned result is "*true*".

[43] Observe that this notion of *partial completeness* is different from the notions of *fixpoint completeness* ($\alpha(\mathrm{lfp}^{\leq} G) = \mathrm{lfp}^{\sqsubseteq} \alpha \circ G \circ \gamma$) and the stronger one of *local completeness* ($\alpha \circ G = \alpha \circ G \circ \gamma \circ \alpha$) considered in P. Cousot & R. Cousot, POPL'79.

Characterization of Partially Com- plete Abstractions for Algorithm 4

Theorem 9 The abstraction $\langle \alpha, \gamma \rangle$ is partially complete for Alg. 4 if and only if $\alpha(L)$ contains an abstract value A such that $\gamma(A)$ is an invariant for $\langle F,\ I,\ S \rangle$.

<u>Intuition:</u> *finding a partially complete abstraction is logically equivalent to making an invariance proof.*

The <u>Most</u> Abstract Partially Complete Abstraction (Definition)

Definition 10 The *most abstract partially complete abstraction* $\langle \overline{\alpha}, \overline{\gamma} \rangle$, if it exists, is defined such that:

1. The *abstract domain* $\overline{M} = \overline{\alpha}(L)$ has the smallest possible cardinality;
2. If another abstraction $\langle \alpha', \gamma' \rangle$ is a partially complete abstraction with the same cardinality, then there exists a bijection β such that $\forall x \in \overline{M} : \gamma'(\beta(x)) \leq \overline{\gamma}(x)$ [44].

[44] Otherwise stated, the abstract values in $\overline{\alpha}(L)$ are more approximate than the corresponding elements in $\alpha'(L)$.

Characterization of the Most Abstract Complete Abstraction

Theorem 11 The most abstract partially complete abstraction for Alg. 4 is such that:

- if $S = 1$ then $\overline{M} = \{\top\}$ where $\overline{\alpha} \stackrel{\text{def}}{=} \boldsymbol{\lambda} X \cdot \top$ and $\overline{\gamma} \stackrel{\text{def}}{=} \boldsymbol{\lambda} Y \cdot 1$;
- if $S \neq 1$ then $\overline{M} = \{\perp, \top\}$ where $\perp \sqsubseteq \perp \sqsubset \top \sqsubseteq \top$ with $\langle \overline{\alpha}, \overline{\gamma} \rangle$ such that:

$$
\begin{aligned}
\overline{\alpha}(X) &\stackrel{\text{def}}{=} \text{if } X \leq \operatorname{gfp}^{\leq} \boldsymbol{\lambda} X \cdot S \wedge \widetilde{F}(X) \text{ then } \perp \text{ else } \top \\
\overline{\gamma}(\perp) &\stackrel{\text{def}}{=} \operatorname{gfp}^{\leq} \boldsymbol{\lambda} X \cdot S \wedge \widetilde{F}(X) \\
\overline{\gamma}(\top) &\stackrel{\text{def}}{=} 1
\end{aligned}
\tag{2}
$$

The Least Abstract Partially Complete Abstraction (Definition)

Definition 12 Dually, the *least abstract partially complete abstraction* $\langle \overline{\alpha}, \overline{\gamma} \rangle$, if it exists, is defined such that:

1. The *abstract domain* $\overline{M} = \overline{\alpha}(L)$ has the smallest possible cardinality;
2. If another abstraction $\langle \alpha', \gamma' \rangle$ is a partially complete abstraction with the same cardinality, then there exists a bijection β such that $\forall x \in \overline{M} : \overline{\gamma}(x) \leq \gamma'(\beta(x))$ [45].

[45] Otherwise stated, the abstract values in $\overline{\alpha}(L)$ are less approximate than the corresponding elements in $\alpha'(L)$.

Characterization of the Least Abstract Complete Abstraction

Theorem 13 Dually, the least abstract partially complete abstraction for Alg. 4 is such that:

- if $I = 1$ then $\underline{M} = \{\top\}$ where $\underline{\alpha} \stackrel{\text{def}}{=} \boldsymbol{\lambda} X \cdot \top$ and $\underline{\gamma} \stackrel{\text{def}}{=} \boldsymbol{\lambda} Y \cdot 1$;
- if $I \neq 1$ then $\underline{M} = \{\perp, \top\}$ where $\perp \sqsubseteq \perp \sqsubset \top \sqsubseteq \top$ with $\langle \underline{\alpha}, \underline{\gamma} \rangle$ such that:

$$
\begin{aligned}
\underline{\alpha}(X) &\stackrel{\text{def}}{=} \text{if } X \leq \operatorname{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \text{ then } \perp \text{ else } \top \\
\underline{\gamma}(\perp) &\stackrel{\text{def}}{=} \operatorname{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \\
\underline{\gamma}(\top) &\stackrel{\text{def}}{=} 1
\end{aligned}
\tag{3}
$$

The Minimal Partially Complete Abstractions for Algorithm 4

Theorem 14

- The set $\mathcal{A}$ of partially complete abstractions of minimal cardinality for Alg. 4 is the set of all abstract domains $\langle M, \sqsubseteq, \alpha, \gamma \rangle$ such that $M = \{\perp, \top\}$ with $\perp \sqsubseteq \perp \sqsubseteq \top \sqsubseteq \top$, $\langle L, \leq \rangle \underset{\alpha}{\overset{\gamma}{\longleftrightarrow}} \langle M, \sqsubseteq \rangle$, $\gamma(\perp) \in \mathcal{I}$ and $\perp = \top$ if and only if $\gamma(\top) \in \mathcal{I}$.

The Complete Lattice of Minimal Complete Abstractions for Alg. 4

Theorem 15

- The relation $\langle\{\bot, \top\}, \sqsubseteq, \alpha, \gamma\rangle \preceq \langle\{\bot', \top'\}, \sqsubseteq', \alpha', \gamma'\rangle$ if and only if $\gamma(\bot) \leq \gamma'(\bot')$ is a pre-ordering on $\mathcal{A}$.
- Let $\langle\{\bot, \top\}, \sqsubseteq, \alpha, \gamma\rangle \cong \langle\{\bot', \top'\}, \sqsubseteq', \alpha', \gamma'\rangle$ if and only if $\gamma(\bot) = \gamma'(\bot')$ be the corresponding equivalence.
- The quotient $\mathcal{A}_{/\cong}$ is a complete lattice[46] for $\preceq$ with infimum class representative $\langle\underline{M}, \underline{\sqsubseteq}, \underline{\alpha}, \underline{\gamma}\rangle$ and supremum $\langle\overline{M}, \overline{\sqsubseteq}, \overline{\alpha}, \overline{\gamma}\rangle$.

Intuition for Minimal Partially Complete Abstractions

- There is a one to one correspondance between partially complete abstractions of minimal cardinality for Alg. 4 and the set of invariants for proving $\mathrm{lfp}^{\leq} \boldsymbol{\lambda} X \cdot I \vee F(X) \leq S$;

- Similar results hold for the other Algs. 6, 7 & 8.

[46] Observe however that it is not a sublattice of the lattice of abstract interpretations of P. Cousot & R. Cousot, POPL'77, POPL'79 with reduced product as glb.

Conclusion on Abstraction Design

On Complete Abstraction Design

- For a single given program, it is always possible to find a finite and small model and so to check it;

  $\Rightarrow$ *by putting enough effort on the design of the model, model-checking will always succeed without false alarm;*
- Finding a model is difficult since logically equivalent to discovering a program invariant;

  $\Rightarrow$ *no problem, its given by the user of the model-checker;*
- proving the correctness of the model is logically equivalent to an invariant proof obligation (Th. 10);

  $\Rightarrow$ *no problem, the end-user often does not care (he trusts himself).*

On Complete Abstraction Design (contd.)

- For a infinitely many programs, it is impossible to find a finite abstraction or widening that will work for all programs;

  $\Rightarrow$ *whichever effort is put on the design of the static analyzer, there will always be false alarms for some program;*
- Finding an abstraction/widening is difficult since logically equivalent to discovering a map from programs to invariants;

  $\Rightarrow$ *never given by the user, a problem for the designer;*
- Proving the correctness of the static analyzer is logically equivalent to a proof obligation for all programs (Th. 10);

  $\Rightarrow$ *a definite problem, the end-user does care (he distrusts the designer).*

On Widenings [47]

[47] P. Cousot, R. Cousot: Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. PLILP 1992: 269-295

Widening Operator

A widening operator $\nabla \in \overline{L} \times \overline{L} \mapsto \overline{L}$ is such that:

- Correctness:
  - $\forall x, y \in \overline{L} : \gamma(x) \sqsubseteq \gamma(x \nabla y)$
  - $\forall x, y \in \overline{L} : \gamma(y) \sqsubseteq \gamma(x \nabla y)$
- Convergence:
  - for all increasing chains $x^0 \sqsubseteq x^1 \sqsubseteq \ldots$, the increasing chain defined by $y^0 = x^0, \ldots, y^{i+1} = y^i \nabla x^{i+1}, \ldots$ is not strictly increasing.

Fixpoint Approximation with Widening

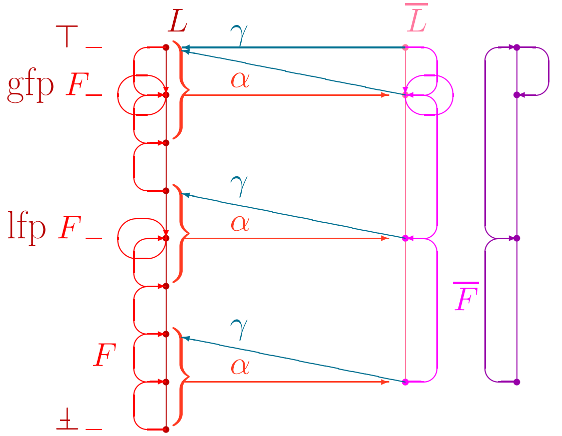
The upward iteration sequence with widening:

- $\hat{X}^0 = \overline{\bot}$ (infimum)
- $$\begin{aligned} \hat{X}^{i+1} &= \hat{X}^i && \text{if } \overline{F}(\hat{X}^i) \sqsubseteq \hat{X}^i \\ &= \hat{X}^i \nabla F(\hat{X}^i) && \text{otherwise} \end{aligned}$$

is ultimately stationary and its limit $\hat{A}$ is a sound upper approximation of $\mathrm{lfp}^{\overline{\bot}}\, \overline{F}$:

$$\mathrm{lfp}^{\overline{\bot}}\, \overline{F} \sqsubseteq \hat{A}$$

Fixpoint Approximation with Widening/Narrowing



$$\check{X}^2 = \check{X}^1 \nabla \overline{F}(\hat{X}^1)$$
$$= \overline{\top} = \check{X}^0$$
$$\check{X}^1 = \check{X}^0 \Delta \overline{F}(\check{X}^0)$$
$$= \text{gfp}\overline{F} = \text{lfp}\overline{F}$$
$$\hat{X}^1 = \hat{X}^0 \nabla \overline{F}(\hat{X}^0)$$
$$\hat{X}^0 = \overline{\perp}$$

Interval Widening

- $\overline{L} = \{\perp\} \cup \{[\ell,\, u] \mid \ell \in \mathbb{Z} \cup \{-\infty\} \wedge u \in \mathbb{Z} \cup \{+\infty\} \wedge \ell \le u\}$
- The widening extrapolates unstable bounds to infinity:

$$\perp \nabla X = X$$
$$X \nabla \perp = X$$
$$[\ell_0,\, u_0] \nabla [\ell_1,\, u_1] = [\text{if } \ell_1 < \ell_0 \text{ then } -\infty \text{ else } \ell_0,$$
$$\text{if } u_1 > u_0 \text{ then } +\infty \text{ else } u_0]$$

Not monotone. For example $[0,\, 1] \sqsubseteq [0,\, 2]$ but $[0,\, 1] \nabla [0,\, 2] = [0,\, +\infty] \not\sqsubseteq [0,\, 2] = [0,\, 2] \nabla [0,\, 2]$

Interval Widening with Thresholds

- Extrapolate to thresholds, zero, one or infinity:

$$[\ell_0,\, u_0] \nabla [\ell_1,\, u_1] = [\text{if } \ell \le \ell_1 < \ell_0 \wedge \ell \in \{1, 0, -1\} \text{ then } l$$
$$\text{elsif } \ell_1 < \ell_0 \text{ then } -\infty$$
$$\text{else } \ell_0,$$
$$\text{if } u_0 < u_1 \le u \wedge u \in \{-1, 0, 1\} \text{ then } u$$
$$\text{elsif } u_0 < u_1 \text{ then } +\infty$$
$$\text{else } u_0]$$

- So the analysis is always as good as the sign analysis.

Non-Existence of Finite Abstractions

Let us consider the infinite family of programs parameterized by the *mathematical constants* n_1, n_2 $(n_1 \le n_2)$:

```
X := n1;
while X <= n2 do
  X := X + 1;
od
```

- An interval analysis with widening/narrowing will discover the loop invariant $X \in [n_1, n_2]$;
- To handle all programs in the family without false alarm, the abstract domain must contain all such intervals;

  $\Rightarrow$ No single finite abstract domain will do for all programs!

- Yes, but predicate abstraction with refinement will do (?) for each program in the family (since it is equivalent to a widening)[48]!
- Indeed no, since:
  - Predicate abstraction is unable to express limits of infinite sequences of predicates;
  - Not all widening proceed by eliminating constraints;
  - A narrowing is necessary anyway in the refinement loop (to avoid infinitely many refinements);
  - Not speaking of costs!

On the Design of Program Static Analyzers

- P. Cousot. *The Calculational Design of a Generic Abstract Interpreter*. In *Calculational System Design*, M. Broy and R. Steinbrüggen (Eds). Vol. 173 of NATO Science Series, Series F: Computer and Systems Sciences. IOS Press, pp. 421–505, 1999.
- The corresponding *generic abstract interpreter* (written in Ocaml) is available at URL `www.di.ens.fr/~cousot`

48 T. Ball, A. Podelski, S.K. Rajamani. Relative Completeness of Abstraction Refinement for Software Model Checking. TACAS 2002: 158-172.

On the Design of Program Analyzers

- The abstract interpretation theory provides the design principles;
- In practice, one must find the appropriate tradeoff between generality, precision and efficiency;
- There is a full range of program analyzers from

  general purpose analyzers for programming languages

  to

  specific analyzers for a given program (software model checking).

Specific Static Program Analyzers

- A complete specific analyzer[49] (for a given software or hardware program) can always use a finite abstract domain[50];
- The design of a complete specific analyzer is logically equivalent to a correctness proof of the program;
- Such analyzers are precise but not reusable hence very costly to develop.

49 Called a *software model checker?*

50 P. Cousot. *Partial completeness of abstract fixpoint checking*. SARA'2000. LNAI 1864, pp. 1–25. Springer.

General-Purpose Static Program Analyzers

- To handle infinitely many programs for non-trivial properties, a general-purpose analyser must use an infinite abstract domain [51];
- Such analyzers are huge for complex languages hence very costly to develop but reusable;
- There are always programs for which they lead to false alarms;
- Although incomplete, they are very useful for verifying/testing/debugging.

Parametric Specializable Static Program Analyzers

- The abstraction can be tailored to significant classes of programs (e.g. critical synchronous real-time embedded systems);
- This leads to *very efficient analyzers* with ~~almost no~~ *zero-false alarm* even for large programs.

[51] P. Cousot & R. Cousot. *Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation*. PLILP'92. LNCS 631, pp. 269–295. Springer.

Experience Report on a Parametric Specializable Program Static Analyzer

B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, X. Rival.

Example of Parametric Specializable Static Program Analyzers

Analyzer under development, very first results!

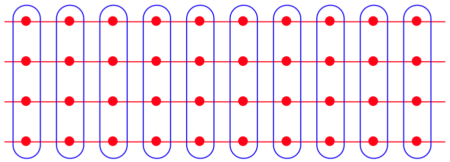
- C programs: safety critical embedded real-time synchronous software for non-linear control of complex systems;
- 10 000 LOCs, 1300 global variables (booleans, integers, real, arrays, macros, non-recursive procedures);
- Implicit specification: absence of runtime errors (no integer/floating point arithmetic overflow, no array bound overflow);
- Initial design: 2h, 110 false alarms (general purpose analyzer);

Experience report

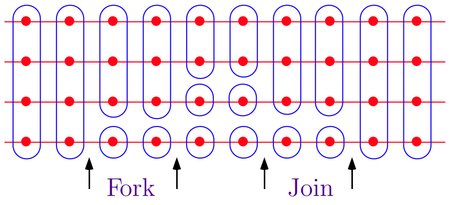
- Comparative results (commercial software):
  - 70 false alarms, 2 days, 500 Megabytes;
- Initial redesign:
  - Weak relational domain with time;
- Parametrisation:
  - Hypotheses on volatile inputs;
  - Staged widenings with thresholds;
  - Local refinements of the parameterized abstract domains;
- Results:
  - No false alarm, 14s, 20 Megabytes.

Example of refinement: trace partitionning

Control point partitionning:

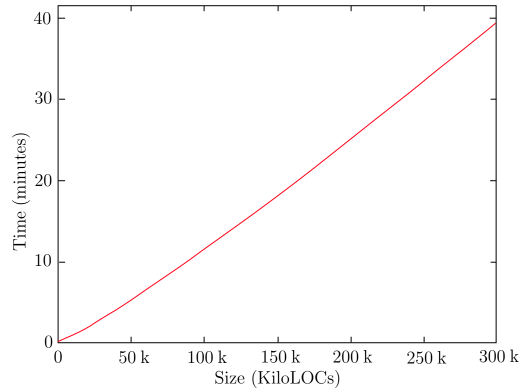


Trace partitionning:



Performance: Space and Time

$$\text{Space} = \mathcal{O}(\text{LOCs})$$
$$\text{Time} = \mathcal{O}(\text{LOCs} \times (\ln(\text{LOCs}))^{1.5})$$



Conclusion

Conclusion on Formal Methods

- Formal methods concentrate on the deductive/exhaustive verification of (abstract) models of the execution of programs;
- Most often this abstraction into a model is *manual* and left completely *informal*, if not tortured to meet the tool limitations;
- Semantics concentrates on the rigorous formalization of the execution of programs;
- So models should abstract the program semantics. This is the whole purpose of Abstract Interpretation!

Conclusion on Abstract Interpretation

- Abstract interpretation provides mathematical foundations of most semantics-based program verification and manipulation techniques;
- In abstract interpretation, the abstraction of the program semantics into an approximate semantics is automated so that one can go *much beyond* examples modelled by hand;
- The abstraction can be tailored to classes of programs so as to design *very efficient analyzers* with ~~*almost no*~~ *zero-false alarm*.

THE END

More references at URL www.di.ens.fr/~cousot.