Abstract Interpretation & Applications

Patrick COUSOT

École normale supérieure, Paris, France

cousot@ens.fr www.di.ens.fr/~cousot

Seminar, MIT, April 3rd, 2006

École Normale Supérieure (ENS)

École Normale Supérieure (ENS)

Normale Sup. (ENS)

A few former students: Évariste Galois, Louis Pasteur, . . . ; Nobel prizes: Claude Cohen-Tannoudji, Pierre-Gilles de Gennes, Gabriel Lippmann, Louis Néel, Jean-Baptiste Perrin, Paul Sabatier, . . . ; Fields Medal holders: Laurent Schwartz, Jean-Pierre Serre (1st Abel Prize), René Thom, Alain Connes, Pierre-Louis Lions, Jean-Christophe Yoccoz, Laurent Lafforgue; Fictious mathematicians: Nicolas Bourbaki; Philosophers: Henri Bergson (Nobel Prize), Louis Althusser, Simone de Beauvoir, Émile Auguste Chartier "Alain", Raymond Aron, Jean-Paul Sartre, Maurice Merleau-Ponty, Michel Foucault, Jacques Derrida, Bernard-Henri Lévy. . . ; Politicians: Jean Jaurès, Léon Blum, Édouard Herriot, Georges Pompidou, Alain Juppé, Laurent Fabius, Léopold Sédar Senghor,. . . ; Sociologists: Émile Durkheim, Pierre Bourdieu, . . . ; Writers: Romain Rolland (Nobel Prize), Jean Giraudoux, Charles Péguy, Julien Gracq, . . . ;

Talk Outline

Motivation

Abstraction and Approximation

Two fundamental concepts in computer science (and engineering):

- Abstraction: to reason on complex systems;
- Approximation: to make undecidable reasoning computationally feasible.

Formalized by Abstract Interpretation.

References

[POPL '77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4^{th} ACM POPL*.

[Thesis '78] P. Cousot. Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes. Thèse ès sci. math. Grenoble, march 1978.

[POPL '79] P. Cousot & R. Cousot. Systematic design of program analysis frameworks. In *6^{th} ACM POPL*.

Abstract Interpretation

– Born to formalize static program analysis;
– Viewed today as a general formalism to reason about semantics of computer systems at different levels of abstraction;
– Successfully applied to automatic analysis of complex computer systems.

A Few Applications of Abstract Interpretation

A Few Applications of Abstract Interpretation

- **Static Program Analysis** [POPL '77], [POPL '78], [POPL '79] including **Dataflow Analysis** [POPL '79], [POPL '00], **Set-based Analysis** [FPCA '95], **Predicate Abstraction** [Manna's festschrift '03], ...
- **Syntax Analysis** [TCS 290(1) 2002]
- **Hierarchies of Semantics (including Proofs)** [POPL '92], [TCS 277(1–2) 2002]
- **Typing & Type Inference** [POPL '97]

A Few Applications of Abstract Interpretation (Cont'd)

- **(Abstract) Model Checking** [POPL '00]
- **Program Transformation** (partial evaluation, monitoring, ...) [POPL '02]
- **Software Watermarking** [POPL '04]
- **Bisimulations** [RT-ESOP '04]

All these techniques involve sound approximations that can be formalized by abstract interpretation

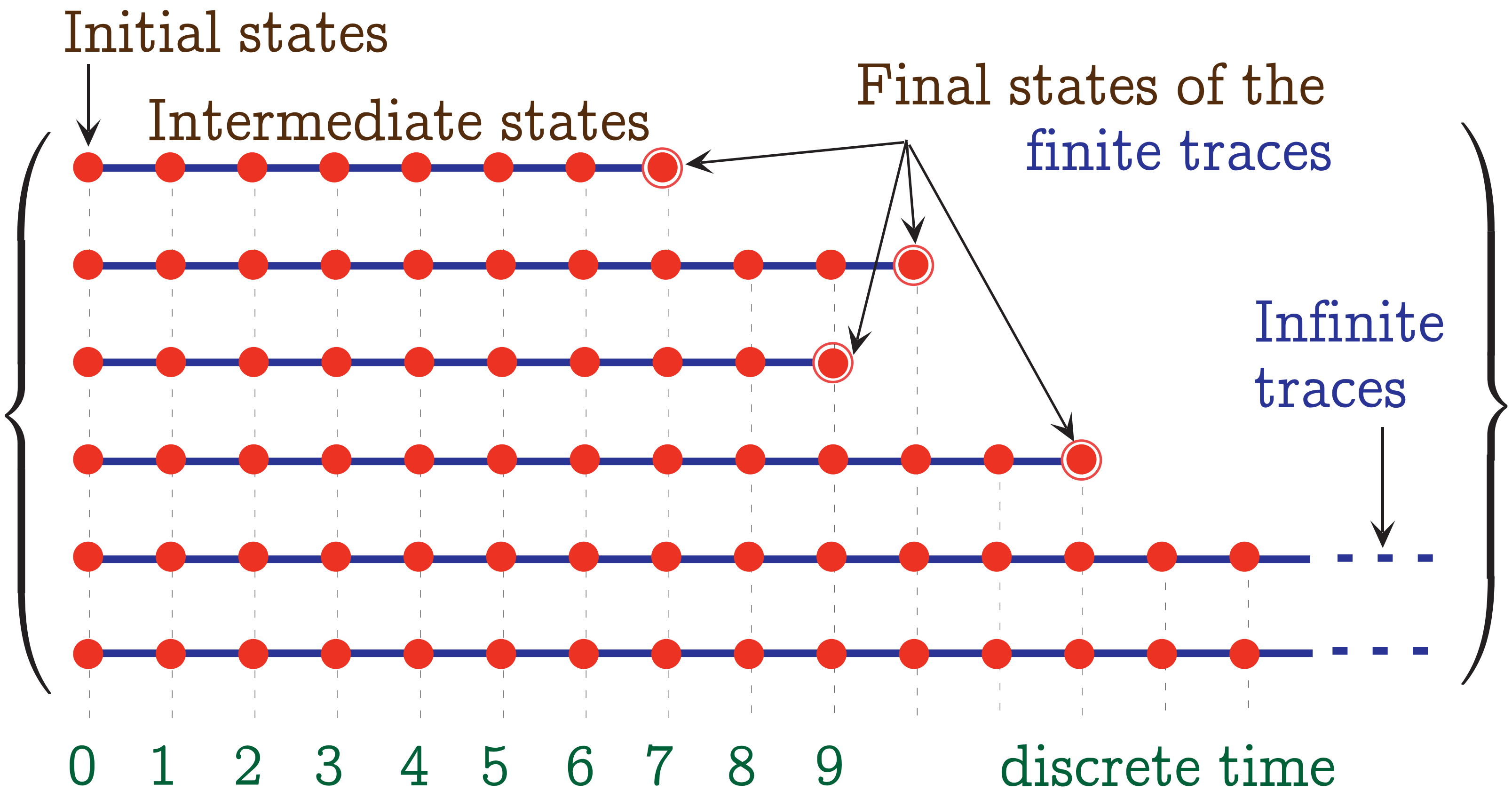
Elements of Abstract Interpretation

Program Semantics

Language Semantics

- A language $\mathcal{L}$ is a set of program texts $P \in \mathcal{L}$
- A semantic domain $\mathcal{D}$ is a set of program semantics
- A program semantics is a mathematical object formally describing program executions (i.e. the effect of running a program on a computer)
- A language semantics $\mathcal{S}$ maps programs $P \in \mathcal{L}$ to their semantics $\mathcal{S}[\![P]\!] \in \mathcal{D}$

Example: Trace Semantics



states $\Sigma = \{\bullet, \ldots, \odot \ldots\}$, transitions $\tau = \{\bullet \longrightarrow \bullet, \ldots, \bullet \longrightarrow \odot \ldots\}$

Formal Definition of the Language Semantics

- A language semantics $\mathcal{S} \in \mathcal{L} \mapsto \mathcal{D}$ is formally defined
  - denotationally: by *induction on the syntax of programs* $P \in \mathcal{L}$
  - compositionally: by *composing elementary mathematical objects and structures* (numbers, pairs, tuples, relations, orders, functions, functionals, fixpoints, etc)

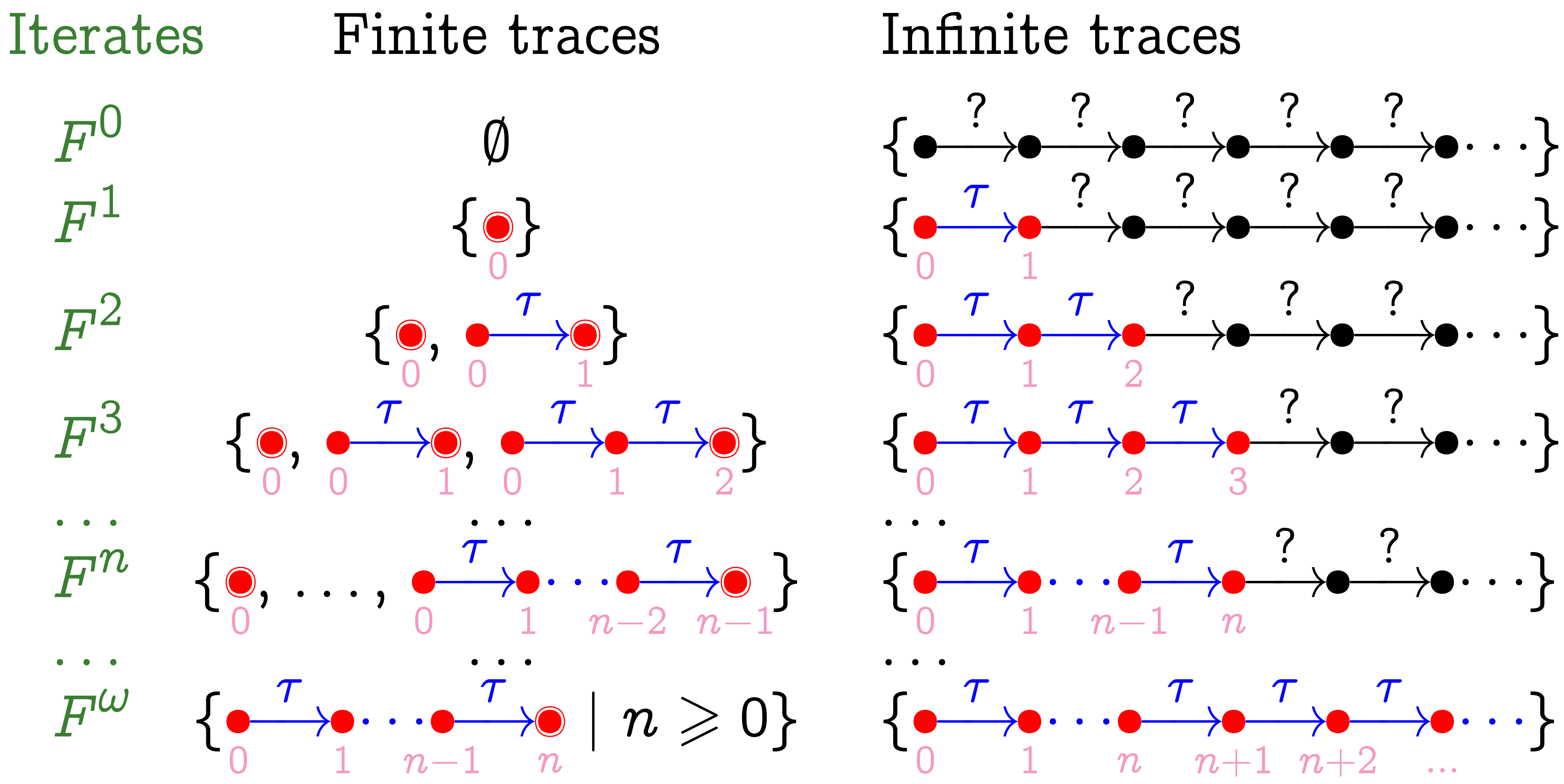
Least Fixpoint Trace Semantics

$$
\begin{aligned}
\text{Traces} = \; & \{\bullet \mid \bullet \text{ is a final state}\} \\
\cup \; & \{\bullet \longrightarrow \bullet \longrightarrow \ldots \longrightarrow \bullet \mid \bullet \longrightarrow \bullet \text{ is a transition step } \& \\
& \qquad \bullet \longrightarrow \ldots \longrightarrow \bullet \in \text{Traces}^{+}\} \\
\cup \; & \{\bullet \longrightarrow \bullet \longrightarrow \ldots \longrightarrow \ldots \mid \bullet \longrightarrow \bullet \text{ is a transition step } \& \\
& \qquad \bullet \longrightarrow \ldots \longrightarrow \ldots \in \text{Traces}^{\infty}\}
\end{aligned}
$$

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

"more finite traces & less infinite traces".

Iterative Fixpoint Calculation of the Trace Semantics



Trace Semantics

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

– $\Sigma^{+} \stackrel{\Delta}{=} \bigcup_{n>0} [0, n[\longmapsto \Sigma$ finite traces

– $\Sigma^{w} \stackrel{\Delta}{=} [0, \omega[\longmapsto \Sigma$ infinite traces

– $\mathcal{S}[\![\langle \Sigma, \tau \rangle]\!] = \mathsf{lfp}^{\sqsubseteq}\ F \in \mathcal{D} = \wp(\Sigma^{+} \cup \Sigma^{\omega})$ trace semantics

– $F(X) = \{s \in \Sigma^{+} \mid s \in \Sigma \wedge \forall s' \in \Sigma : \langle s, s' \rangle \not\in \tau\}$
$\cup \{ss'\sigma \mid \langle s, s' \rangle \in \tau \wedge s'\sigma \in X\}$ trace transformer

– $X \sqsubseteq Y \stackrel{\Delta}{=} (X \cap \Sigma^{+}) \subseteq (Y \cap \Sigma^{+}) \wedge (X \cap \Sigma^{\omega}) \supseteq (Y \cap \Sigma^{\omega})$
computational ordering

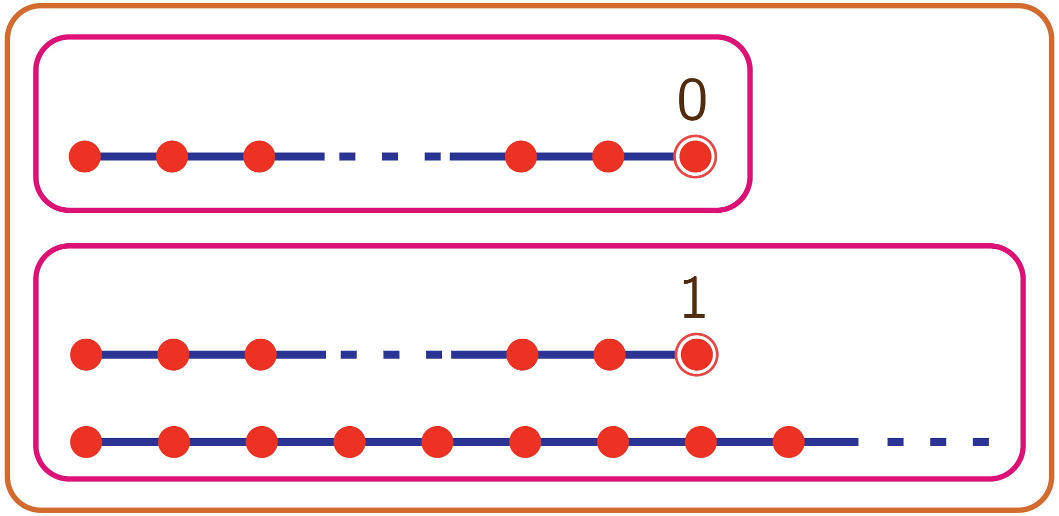
Program Properties

Program Properties & Static Analysis

- A program property $\mathcal{P} \in \wp(\mathcal{D})$ is a set of possible semantics for that program (hence a subset of the semantic domain $\mathcal{D}$)
- A property $\mathcal{P} \in \wp(\mathcal{D})$ is stronger (or more precise) than a property $\mathcal{Q} \in \wp(\mathcal{D})$ iff $P \subseteq Q$ (i.e. P implies Q, $P \Rightarrow Q$)
- The strongest program property [1] is $\{\mathcal{S}[\![P]\!]\} \in \wp(\mathcal{D})$
- A static analysis effectively approximates the strongest property of programs

[1] also called the *collecting semantics*

Example Program Property



– Correct implementations: `print 0`, `[print 1|loop]`, ...

– Excludes `[print 0|print 1]`

– Note for specialists: neither a safety nor a liveness property.

Abstraction of Program Properties

Abstraction

- Replace actual/concrete properties $\mathcal{P} \in \wp(\mathcal{D})$ by an approximate abstract properties $\alpha(\mathcal{P})$
- Examples:
  - engineering:

    α(property of an object) = property of a *model* of the object
  - partial correctness in computer science:

    α(program property) = restriction of the property to finite executions

Commonly Required Properties of the Abstraction

– [In this talk,] we consider sound overapproximations:

$$\mathcal{P} \subseteq \alpha(\mathcal{P})$$

  – If the abstract property $\alpha(\mathcal{P})$ does hold then so does the concrete properties $\mathcal{P}$
  – If the abstract property $\alpha(\mathcal{P})$ does not hold then the concrete properties $\mathcal{P}$ may hold or not! [2]

– All information is lost at once:

$$\alpha(\alpha(\mathcal{P})) = \alpha(\mathcal{P})$$

– The abstraction of more precise properties is more precise:

$$\text{if } \mathcal{P} \subseteq \mathcal{Q} \text{ then } \alpha(\mathcal{P}) \subseteq \alpha(\mathcal{Q})$$

---

[2] In this case we speak of *"false alarm"*.

Galois Connection

– We have got a Galois connection:

$$\langle \wp(\mathcal{D}),\ \subseteq \rangle \xleftrightarrow[\alpha]{1} \langle \wp(\mathcal{D}),\ \subseteq \rangle$$

$$\begin{array}{ccc} \uparrow & \qquad\qquad & \uparrow \\ \text{Concrete properties} & & \text{Abstract properties} \end{array}$$

– With an isomorphic mathematical/computer representation:

$$\langle \wp(\mathcal{D}),\ \subseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{D}^\sharp,\ \sqsubseteq \rangle$$

$$\begin{array}{ccc} \uparrow & \qquad\qquad & \uparrow \\ \text{Concrete properties} & & \text{Abstract domain} \end{array}$$

$$\forall \mathcal{P} \in \wp(\mathcal{D}) : \forall \mathcal{Q} \in \mathcal{D}^\sharp : \alpha(\mathcal{P}) \sqsubseteq \mathcal{Q} \iff \mathcal{P} \subseteq \gamma(\mathcal{Q})$$

Abstraction 1: Functions

- Let $\langle \wp(\mathcal{D}), \subseteq \rangle \xleftrightarrows[\alpha]{\gamma} \langle \mathcal{D}^\sharp, \sqsubseteq \rangle$
- How to abstract a *property transformer* $F \in \wp(\mathcal{D}) \stackrel{\mathrm{m}}{\longmapsto} \wp(\mathcal{D})$?
- The most precise sound overapproximation is

$$F^\sharp \in \mathcal{D}^\sharp \stackrel{\mathrm{m}}{\longmapsto} \mathcal{D}^\sharp$$
$$F^\sharp = \alpha \circ F \circ \gamma$$

- This is a Galois connection

$$\langle \wp(\mathcal{D}) \stackrel{\mathrm{m}}{\longmapsto} \wp(\mathcal{D}), \subseteq \rangle \xleftrightarrows[\lambda F \cdot \alpha \circ F \circ \gamma]{\lambda F^\sharp \cdot \gamma \circ F^\sharp \circ \alpha} \langle \mathcal{D}^\sharp \stackrel{\mathrm{m}}{\longmapsto} \mathcal{D}^\sharp, \sqsubseteq \rangle$$

Abstraction 2: Fixpoints

- Let $\langle \wp(\mathcal{D}), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}^\sharp, \sqsubseteq \rangle$
- How to abstract a *fixpoint property* $\mathsf{lfp}^{\subseteq} F$ where $F \in \wp(\mathcal{D}) \xmapsto{\mathrm{m}} \wp(\mathcal{D})$?
- Approximate Sound Abstraction:

$$\mathsf{lfp}^{\subseteq} F \subseteq \gamma(\mathsf{lfp}^{\sqsubseteq} \alpha \circ F \circ \gamma)$$

- Complete Abstraction: if $\alpha \circ F = F^\sharp \circ \alpha$ then

$$F^\sharp = \alpha \circ F \circ \gamma, \text{ and}$$

$$\alpha(\mathsf{lfp}^{\subseteq} F) = \mathsf{lfp}^{\sqsubseteq} F^\sharp$$

Abstract Interpretation-Based Static Analysis

- an inductive compositional language semantics $\mathcal{S} \in \mathcal{L} \mapsto \mathcal{D}$
- program concrete properties $\wp(\mathcal{D})$
- an abstract domain $\langle \wp(\mathcal{D}),\ \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{D}^\sharp,\ \sqsubseteq \rangle$ designed inductively and compositionally to approximate the property to be analyzed
- the A.I. Theory is used to systematically derive the sound abstract semantics $\mathcal{S}^\sharp[\![P]\!] \sqsupseteq \alpha(\{\mathcal{S}[\![P]\!]\})$
- the static analysis algorithm is the computation of the abstract semantics and is correct by construction

Example 1: Trace Semantics Abstraction

Reference

[TCS '02] P. Cousot, Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation, Theoretical Computer Science,, 277(1—2):47—103, 2002. © Elsevier Science.

Objective

- A unifying formalization of the classical semantics as abstract interpretations of the trace semantics
- (... and of a few new ones)

Semantics Abstractions
1 — Relational Semantics Abstractions

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

$\uparrow$ Finite and infinite traces

$\uparrow$ Relation between initial and final states or $\bot$ [3]

[3] $\bot$ is Dana Scott's traditional notation for non-termination.

1 — Relational Semantics Abstractions (Cont'd)

- $\alpha^{\natural}(X) = \{\langle s, s'\rangle \mid s\sigma s' \in X \cap \Sigma^{+}\}$
  $\cup \{\langle s, \perp\rangle \mid s\sigma \in X \cap \Sigma^{\omega}\}$
  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

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

2 — Denotational Semantics Abstractions

$$\langle \wp(\Sigma \times (\Sigma \cup \{\bot\})), \subseteq \rangle \underset{\alpha^{\varphi}}{\overset{\gamma^{\varphi}}{\leftrightarrows}} \langle \Sigma \longmapsto \wp(\Sigma \cup \{\bot\}), \dot{\subseteq} \rangle$$

$\uparrow$ Relation between initial and final states or $\bot$

$\uparrow$ Map of initial states to sets of final states or $\bot$

– $\alpha^{\varphi}(X) = \lambda s.\{s' \in \Sigma \cup \{\bot\} \mid \langle s, s' \rangle \in X\}$

relational to denotational semantics

3 — Predicate Transformer Abstractions

$$\langle \Sigma \longmapsto \wp(\Sigma \cup \{\perp\}), \dot{\subseteq}\rangle \underset{\alpha^\pi}{\overset{\gamma^\pi}{\leftrightarrows}} \langle \wp(\Sigma) \overset{\cup}{\longmapsto} \wp(\Sigma \cup \{\perp\}), \dot{\subseteq}\rangle$$

$\uparrow$ Map of initial states to sets of final states or $\perp$

$\uparrow$ Map of sets of initial states to sets of final states or $\perp$

– $\alpha^\pi(\phi) = \lambda P.\{s' \in \Sigma \cup \{\perp\} \mid \exists s \in P : s' \in \phi(s)\}$

denotational to predicate transformer semantics

4 — Predicate Transformer Abstractions (Cont'd)

$$\begin{tikzcd}
\langle \wp(\Sigma) \overset{\cup}{\mapsto} \wp(\Sigma \cup \{\bot\}),\ \dot{\subseteq}\rangle \arrow[r, shift right, two heads, "\alpha^{\sim}"'] \arrow[d, shift right, two heads, "\alpha^{\cup}"'] & \langle \wp(\Sigma) \overset{\cap}{\mapsto} \wp(\Sigma \cup \{\bot\}),\ \dot{\supseteq}\rangle \arrow[l, shift right, "\gamma^{\sim}"'] \arrow[d, shift right, two heads, "\alpha^{\cap}"'] \\
\langle \wp(\Sigma \cup \{\bot\}) \overset{\cup}{\mapsto} \wp(\Sigma),\ \dot{\subseteq}\rangle \arrow[u, shift right, "\gamma^{\cup}"'] \arrow[r, shift right, two heads, "\alpha^{\sim}"'] & \langle \wp(\Sigma \cup \{\bot\}) \overset{\cap}{\mapsto} \wp(\Sigma),\ \dot{\supseteq}\rangle \arrow[u, shift right, "\gamma^{\cap}"'] \arrow[l, shift right, "\gamma^{\sim}"']
\end{tikzcd}$$

- $\alpha^{\sim}(\Phi) = \lambda P.\neg(\Phi(\neg P))$ conjugate [4]
- $\alpha^{\cup}(\Phi) = \lambda Q.\{s \in \Sigma \mid \Phi(\{s\}) \cap Q \neq \emptyset\}$ $\cup$-inversion [5]
- $\alpha^{\cap}(\Phi) = \lambda Q.\{s \in \Sigma \mid \Phi(\neg\{s\}) \cup Q = \Sigma \cup \{\bot\}\}$ $\cap$-inversion [6]

---

4 States that must reach P by state transformer Φ or block

5 Non-blocking states that may reach Q by state transformer Φ

6 Non-blocking states that must reach Q by state transformer Φ

5 — Hoare Logic Abstractions

$$\langle \wp(\Sigma) \stackrel{\cup}{\mapsto} \wp(\Sigma \cup \{\perp\}),\ \dot{\subseteq}\rangle \xrightleftharpoons[\alpha^H]{\gamma^H} \langle \wp(\Sigma) \otimes {}^{7}\wp(\Sigma \cup \{\perp\}), \dot{\supseteq}\rangle$$

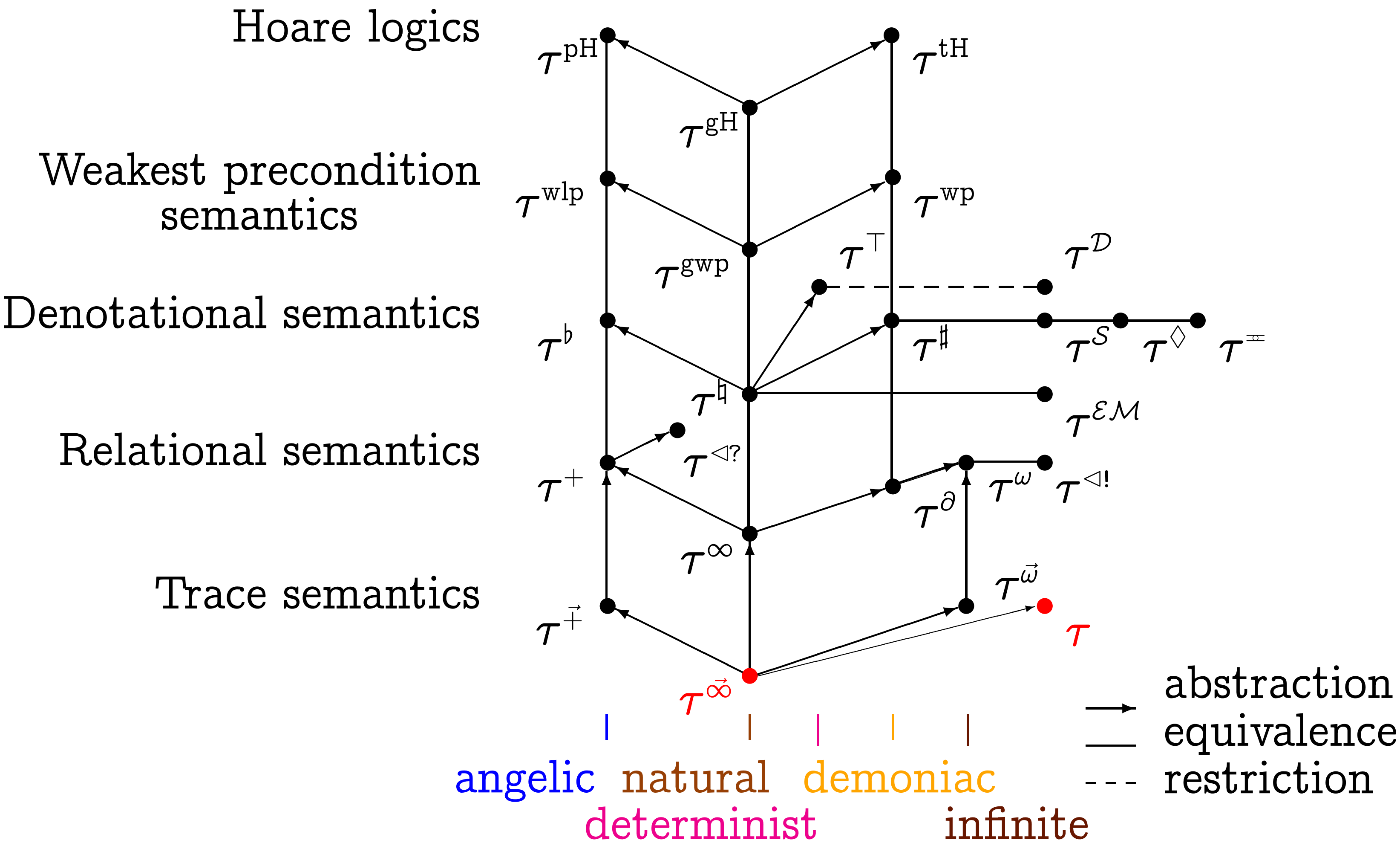
$\uparrow$ Map of sets of initial states to sets of final states or $\perp$

$\uparrow$ Set of all Hoare triples (generalized to non-termination)

– $\alpha^H(\Phi) = \{\langle P, Q\rangle \mid \Phi(P) \subseteq Q\}$
predicate transformer to Hoare logic semantics

7 Semi-dual Shmuely tensor product.

Lattice of Semantics



6 — Safety Abstraction

- Disjunctive abstraction: $\alpha_u(P) \stackrel{\Delta}{=} \bigcup P$

  $$\langle \wp(\wp(\Sigma^+ \cup \Sigma^\omega)),\ \subseteq\rangle \xleftrightarrow[\alpha_u]{\gamma_u} \langle \wp(\Sigma^+ \cup \Sigma^\omega),\ \subseteq\rangle$$

- Prefix abstraction (time invariance):

  $$\alpha_p(P) \stackrel{\Delta}{=} \{\sigma \in \Sigma^+ \mid \exists \sigma' \in: \Sigma^+ \cup \Sigma^\omega : \sigma\sigma' \in P\}$$

  $$\langle \wp(\Sigma^+ \cup \Sigma^\omega),\ \subseteq\rangle \xleftrightarrow[\alpha_p]{\gamma_p} \langle \wp(\Sigma^+ \cup \Sigma^\omega),\ \subseteq\rangle$$

- Limit abstraction (infinite behaviors are not observable):

  $$\alpha_\ell(P) \stackrel{\Delta}{=} \{\sigma \in \Sigma^\omega \mid \alpha_p(\{\sigma\}) \subseteq P\}$$

  $$\langle \wp(\Sigma^+ \cup \Sigma^\omega),\ \subseteq\rangle \xleftrightarrow[\alpha_\ell]{\gamma_\ell} \langle \wp(\Sigma^+ \cup \Sigma^\omega),\ \subseteq\rangle$$

- Safety abstraction (can be monitored at runtime):

  $$\langle \wp(\wp(\Sigma^+ \cup \Sigma^\omega)),\ \subseteq\rangle \xleftrightarrow[\alpha_\ell \circ \alpha_p \circ \alpha_u]{\gamma_u \circ \gamma_\ell \circ \gamma_p} \langle \wp(\Sigma^+ \cup \Sigma^\omega),\ \subseteq\rangle$$

Example 2: Typing

Reference

[POPL '97] P. Cousot. Types as Abstract Interpretations. In Conference Record of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages, pages 316–331, Paris, France, 1997. ACM Press, New York, U.S.A.

Objective

- Show that static typing and type inference are abstract interpretations of a semantics with runtime type checking
- (... and consider nontermination in type soundness)

Syntax of the Eager Lambda Calculus

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

Semantic Domains

Ω	wrong/runtime error value
$\perp$	non-termination
$\mathbb{W} \triangleq \{\Omega\}$	wrong
$z \in \mathbb{Z}$	integers
$u, f, \varphi \in \mathbb{U} \cong \mathbb{W}_\perp \oplus \mathbb{Z}_\perp \oplus [\mathbb{U} \mapsto \mathbb{U}]^{8}{}_\perp$	values
$R \in \mathbb{R} \triangleq \mathbb{X} \mapsto \mathbb{U}$	environments
$\phi \in \mathbb{S} \triangleq \mathbb{R} \mapsto \mathbb{U}$	semantic domain

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

Denotational Semantics with Run-Time Type Checking

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

$$\begin{aligned}\mathbf{S}[\![e_1 - e_2]\!]\mathsf{R} \stackrel{\Delta}{=} (\, & \mathbf{S}[\![e_1]\!]\mathsf{R} = \bot \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \bot \ ? \ \bot \\ | \, & \mathbf{S}[\![e_1]\!]\mathsf{R} = z_1 \wedge \mathbf{S}[\![e_2]\!]\mathsf{R} = z_2 \ ? \ z_1 - z_2 \\ | \, & \Omega \,)\end{aligned}$$

$$\begin{aligned}\mathbf{S}[\![(e_1 \,?\, e_2 : e_3)]\!]\mathsf{R} \stackrel{\Delta}{=} (\, & \mathbf{S}[\![e_1]\!]\mathsf{R} = \bot \ ? \ \bot \\ | \, & \mathbf{S}[\![e_1]\!]\mathsf{R} = z \neq 0 \ ? \ \mathbf{S}[\![e_2]\!]\mathsf{R} \\ | \, & \mathbf{S}[\![e_1]\!]\mathsf{R} = 0 \ ? \ \mathbf{S}[\![e_3]\!]\mathsf{R} \\ | \, & \Omega \,)\end{aligned}$$

$$\mathbf{S}[\![\mathtt{x}]\!]\mathsf{R} \triangleq \mathsf{R}(\mathtt{x})$$

$$\begin{aligned}\mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x}\cdot e]\!]\mathsf{R} \triangleq \lambda \mathsf{u}\,.\,(\;&\mathsf{u} = \bot \;?\; \bot\\ &|\; \mathsf{u} = \Omega \;?\; \Omega\\ &|\; \mathbf{S}[\![e]\!]\mathsf{R}[\mathtt{x}\leftarrow\mathsf{u}]\;)\end{aligned}$$

$$\begin{aligned}\mathbf{S}[\![e_1(e_2)]\!]\mathsf{R} \triangleq (\;&\mathbf{S}[\![e_1]\!]\mathsf{R} = \bot \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \bot \;?\; \bot\\ &|\; \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{f} \in [\mathbb{U} \mapsto \mathbb{U}] \;?\; \mathsf{f}\big(\mathbf{S}[\![e_2]\!]\mathsf{R}\big)\\ &|\; \Omega\;)\end{aligned}$$

$$\mathbf{S}[\![\boldsymbol{\mu}\mathtt{f}\cdot\boldsymbol{\lambda}\mathtt{x}\cdot e]\!]\mathsf{R} \triangleq \mathbf{lfp}^{\sqsubseteq}\,\lambda\varphi\,.\,\mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x}\cdot e]\!]\mathsf{R}[\mathtt{f}\leftarrow\varphi]$$

Standard Denotational and Collecting Semantics

- The denotational semantics is:

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

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

$$P \in \wp(\mathbb{S})$$

- The standard collecting semantics is the strongest concrete property:

$$\mathbf{C}[\![\bullet]\!] \in \mathbb{E} \mapsto \wp(\mathbb{S}) \qquad \mathbf{C}[\![e]\!] \stackrel{\Delta}{=} \{\mathbf{S}[\![e]\!]\}$$

Abstracting with Church/Curry Monotypes

– Simple types are monomorphic:

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

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

$$H \in \mathbb{H}^{\mathrm{c}} \triangleq \mathbb{X} \mapsto \mathbb{M}^{\mathrm{c}} \qquad \text{type environment}$$

Abstracting with Church/Curry Monotypes (Cont'd)

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

$$\theta \in \mathbb{I}^{\mathrm{c}} \triangleq \mathbb{H}^{\mathrm{c}} \times \mathbb{M}^{\mathrm{c}} \qquad \text{typing}$$

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

$$T \in \mathbb{T}^{\mathrm{c}} \triangleq \wp(\mathbb{I}^{\mathrm{c}}) \qquad \text{program type}$$

Concretization Function

The meaning of types is a program property, as defined by the concretization function γ^{c}:[9]

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

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

---

9 For short up/down lifting/injection are omitted.

– type environment $\gamma_2^{\mathrm{c}} \in \mathbb{H}^{\mathrm{c}} \mapsto \wp(\mathbb{R})$:

$$\gamma_2^{\mathrm{c}}(H) \stackrel{\Delta}{=} \{\mathsf{R} \in \mathbb{R} \mid \forall \mathrm{x} \in \mathbb{X} : \mathsf{R}(\mathrm{x}) \in \gamma_1^{\mathrm{c}}(H(\mathrm{x}))\}$$

– typing $\gamma_3^{\mathrm{c}} \in \mathbb{I}^{\mathrm{c}} \mapsto \wp(\mathbb{S})$:

$$\gamma_3^{\mathrm{c}}(\langle H,\ m\rangle) \stackrel{\Delta}{=} \{\phi \in \mathbb{S} \mid \forall \mathsf{R} \in \gamma_2^{\mathrm{c}}(H) : \phi(\mathsf{R}) \in \gamma_1^{\mathrm{c}}(m)\}$$

– program type $\gamma^{\mathrm{c}} \in \mathbb{T}^{\mathrm{c}} \mapsto \wp(\mathbb{S})$:

$$\gamma^{\mathrm{c}}(T) \stackrel{\Delta}{=} \bigcap_{\theta \in T} \gamma_3^{\mathrm{c}}(\theta)$$

$$\gamma^{\mathrm{c}}(\emptyset) \stackrel{\Delta}{=} \mathbb{S}$$

Program Types

– Galois connection:

$$\langle \wp(\mathbb{S}),\ \subseteq,\ \emptyset,\ \mathbb{S},\ \cup,\ \cap \rangle \xrightleftharpoons[\alpha^{\mathrm{C}}]{\gamma^{\mathrm{C}}} \langle \mathbb{T}^{\mathrm{C}},\ \supseteq,\ \mathbb{I}^{\mathrm{C}},\ \emptyset,\ \cap,\ \cup \rangle$$

– Types $\mathbf{T}[\![e]\!]$ of an expression e:

$$\mathbf{T}[\![e]\!] \subseteq \alpha^{\mathrm{C}}(\mathbf{C}[\![e]\!]) = \alpha^{\mathrm{C}}(\{\mathbf{S}[\![e]\!]\})$$

Typable Programs Cannot Go Wrong

$$\Omega \in \gamma^{\mathrm{C}}(\mathbf{T}[\![e]\!]) \quad \Longleftrightarrow \quad \mathbf{T}[\![e]\!] = \emptyset$$

Church/Curry Monotype Abstract Semantics

$$\mathbf{T}[\![\mathrm{x}]\!] \stackrel{\Delta}{=} \{\langle H,\ H(\mathrm{x})\rangle \mid H \in \mathbb{H}^{\mathrm{c}}\} \tag{VAR}$$

$$\begin{aligned}\mathbf{T}[\![\boldsymbol{\lambda}\mathrm{x}\cdot e]\!] \stackrel{\Delta}{=} \{\langle H,\ m_1 \to m_2\rangle \mid\ & \\ & \langle H[\mathrm{x}\leftarrow m_1],\ m_2\rangle \in \mathbf{T}[\![e]\!]\}\end{aligned} \tag{ABS}$$

$$\begin{aligned}\mathbf{T}[\![e_1(e_2)]\!] \stackrel{\Delta}{=} \{\langle H,\ m_2\rangle \mid \langle H,\ m_1 \to m_2\rangle \in \mathbf{T}[\![e_1]\!] & \\ \wedge \langle H,\ m_1\rangle \in \mathbf{T}[\![e_2]\!]\}&\end{aligned} \tag{APP}$$

$$\mathbf{T}[\![\mu \mathtt{f} \cdot \boldsymbol{\lambda} \mathtt{x} \cdot e]\!] \triangleq \{\langle H,\ m\rangle \mid \langle H[\mathtt{f} \leftarrow m],\ m\rangle \in \mathbf{T}[\![\boldsymbol{\lambda} \mathtt{x} \cdot e]\!]\} \tag{REC}$$

$$\mathbf{T}[\![\mathbf{1}]\!] \triangleq \{\langle H,\ \mathtt{int}\rangle \mid H \in \mathbb{H}^{\mathrm{c}}\} \tag{CST}$$

$$\mathbf{T}[\![e_1 - e_2]\!] \triangleq \{\langle H,\ \mathtt{int}\rangle \mid \langle H,\ \mathtt{int}\rangle \in \mathbf{T}[\![e_1]\!] \cap \mathbf{T}[\![e_2]\!]\} \tag{DIF}$$

$$\mathbf{T}[\![(e_1 \mathbin{?} e_2 : e_3)]\!] \triangleq \{\langle H,\ m\rangle \mid \langle H,\ \mathtt{int}\rangle \in \mathbf{T}[\![e_1]\!] \wedge \langle H,\ m\rangle \in \mathbf{T}[\![e_2]\!] \cap \mathbf{T}[\![e_3]\!]\} \tag{CND}$$

The Herbrand Abstraction to Get Hindley's Type Inference Algorithm

$$\langle \wp(\mathsf{ground}(T)),\ \subseteq,\ \emptyset,\ \mathsf{ground}(T),\ \cup,\ \cap\rangle \xrightleftharpoons[\mathsf{lcg}]{\mathsf{ground}} \langle T^{\emptyset}/_{\equiv},\ \leq,\ \emptyset,\ [\,'\mathsf{a}]_{\equiv},\ \mathsf{lcg},\ \mathsf{gci}\rangle$$

where:

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

Example 3: Termination Proofs

References

[VMCAI '05] P. Cousot. Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. In *Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, pages 1–24, Paris, France, January 17-19, 2005. Lecture Notes in Computer Science, volume 3385, Springer, Berlin.

Objective

- Show that program termination proofs are abstract interpretations of a relational semantics
- (... and automatize such proofs)

Termination Proof

- Problem: prove that all executions of a program loop terminate
- Principle[10]: Exhibit a *ranking function* of the program variables in a well-founded set that strictly decreases at each program step for reachable states.

10 Robert Floyd, 1967, note the similarity with Lyapunov, 1890, *"an invariant set of a differential equation is stable in the sense that it attracts all solutions if one can find a function that is bounded from below and decreases along all solutions outside the invariant set"*.

Termination Proof by Static Analysis

1. Perform an iterated forward/backward relational static analysis of the loop to determine a *necessary* termination precondition

2. Assuming the *termination precondition*, perform an forward relational static analysis of the loop to determine the loop invariant (overapproximating reachable states)

3. Assuming the loop invariant, perform an forward relational static analysis of the loop body to determine the loop abstract operational semantics

4. Assuming the loop semantics, use an abstraction of Floyd's ranking function method to prove termination of the loop

Example (Arithmetic Mean)

```
{x=y+2k,x>=y}  ← necessary termination precondition
  while (x <> y) do
    {x=y+2k,x>=y+2}  ← loop invariant
      {(x=x0)&(y=y0)&(k=k0)}
      k := k - 1;
      x := x - 1;
      y := y + 1
      {x+2=y+2k0,y=y0+1,   ← loop abstract
       x+1=x0,x=y+2k,x>=y}   operational semantics
  od
{k=0}
```

$$\bigwedge_{i=1}^{N} \sigma_i(k_0, x_0, y_0, k, x, y) \geqslant_i 0$$

Floyd's Ranking Function Method

Find an $\mathbb{R}/\mathbb{Q}/\mathbb{Z}$-valued unkown rank function r and $\eta > 0$ such that:

– The rank is *nonnegative*:

$$\forall\, x_0, x : \bigwedge_{i=1}^{N} \sigma_i(x_0, x) \geqslant_i 0 \Rightarrow r(x_0) \geq 0$$

– The rank is *strictly decreasing*:

$$\forall\, x_0, x : \bigwedge_{i=1}^{N} \sigma_i(x_0, x) \geqslant_i 0 \Rightarrow r(x_0) - r(x) - \eta \geq 0$$

Abstraction

1. Eliminate $\bigwedge$ and $\Rightarrow$ by Lagrangian relaxation [11]
2. Choose a parametric abstraction r_a for the ranking function r in term of unkown parameters a e.g. $r_a(x) = a.x^\top$ (linear), $r_a(x) = a.(x\ 1)^\top$ (affine) or $r_a(x) = (x\ 1).a.(x\ 1)^\top$ (quadratic)
3. Eliminate the universal quantification $\forall$ using linear matrix inequalities (LMIs) in favor of positive semidefiniteness i.e. $M(\lambda) \succcurlyeq 0 = \forall X \in \mathbb{R}^N : X^\top M(\lambda) X \geqslant 0$ where $M(\lambda) = M_0 + \sum_{i=1}^{N} \lambda_i M_i$

[11] $[\forall x : (\bigwedge_i f_i(x) \geqslant 0) \Rightarrow (g(x) \geqslant 0)] \Longleftarrow [\exists \lambda_i \geqslant 0 : \forall x : g(x) - \sum_i \lambda_i f_i(x) \geqslant 0]$, sound by Lagrange, complete by Farkas in linear case and Yakubovich's S-procedure with one quadratic constraint)

Abstract Floyd's Ranking Function Method

Find $\mathbb{R}/\mathbb{Q}/\mathbb{Z}$-valued unkown parameters a, such that:

– *Nonnegative*: $\exists \lambda \in [1, N] \mapsto \mathbb{R}^{+i}$:

$$\forall\ x_0, x : r_a(x_0) - \sum_{i=1}^{N} \lambda_i (x_0\ x\ 1) M_i (x_0\ x\ 1)^\top \geq 0$$

– *Strictly decreasing*: $\exists \eta > 0 : \exists \lambda' \in [1, N] \mapsto \mathbb{R}^{+i}$:

$$\forall\ x_0, x : (r_a(x_0) - r_a(x) - \eta) - \sum_{i=1}^{N} \lambda'_i (x_0\ x\ 1) M_i (x_0\ x\ 1)^\top \geq 0$$

Finally, solve these convex constraints by semidefinite programming to get the parameters a (and λ)

Example (Arithmetic Mean)

```
{x=y+2k,x>=y}  ← necessary termination precondition
  while (x <> y) do
      k := k - 1;
      x := x - 1;
      y := y + 1
  od
```

$$r(x,y,k) = +4.k - 2$$

Generalization: non-convex polynomial constraints can be approximated in semidefinite programming form as SOS.

Termination of a Fair Parallel Program

```
[[ while [(x>0)|(y>0) do x := x - 1] od ||
   while [(x>0)|(y>0) do y := y - 1] od ]]
```

interleaving
+ scheduler
$\longrightarrow$

{m>=1} $\longleftarrow$ termination precondition determined by iterated forward/backward polyhedral analysis

```
t := ?;
assume (0 <= t & t <= 1);
s := ?;
assume ((1 <= s) & (s <= m));
while ((x > 0) | (y > 0)) do
  if (t = 1) then
     x := x - 1
  else
     y := y - 1
  fi;
  s := s - 1;
    if (s = 0) then
      if (t = 1) then
         t := 0
      else
         t := 1
      fi;
      s := ?;
      assume ((1 <= s) & (s <= m))
    else
      skip
    fi
od;;
```

penbmi: r(x,y,m,s,t) = +1.000468e+00.x +1.000611e+00.y +2.855769e-02.m -3.929197e-07.s +6.588027e-06.t +9.998392e+03

Example of Challenge in Embedded Software Verification

Given a control/command program, prove that requests have responses in bounded time:

- solved for synchronous programs by abstract interpretation-based *worst-case execution time* (WCET) static analysis; does scale up [12]!
- Opened challenge to scale up for asynchronous control/command programs with real-time scheduling

[12] See aiT WCET Analyzers of AbsInt Angewandte Informatik GmbH

Example 4: Hardware Verification

Objective

- Show that hardware verification is an abstract interpretation of a monitored operational semantics
- (... and automatize such a verification without state explosion)

Hardware Verification in VHDL (Code[13])

```
loop
  clk <= not clk;
  wait for 1;
end;
```

```
loop
  if clk then
    o <= x and not y;
  wait on clk;
end;
```

Clock `clk` $= 0\ 1\ 0$
$1\ 0\ 1\ 0\ 1\ \ldots$

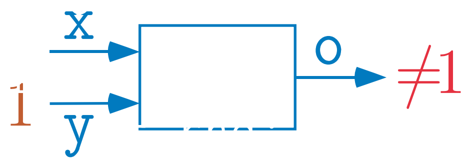
Action $o := x \wedge \neg y$ on
"`clk` $= 1$" events

[13] Very High Speed Integrated Circuit Hardware Description Language (VHDL) pseudo-code at the Behavioral Level.

Hardware Verification in VHDL (Specification)

```
loop
  clk <= not clk;
  wait for 1;
end;
```

```
loop
  if clk then
    o <= x and not y;
  wait on clk;
end;
```



Clock `clk` = 0 1 0 1 0 1 0 1 ...

Action $o := x \wedge \neg y$ on "`clk` = 1" events

`Specification`

Hardware Verification in VHDL (Monitoring)

```
loop
  clk <= not clk;
  wait for 1;
end;
```

Clock `clk` = 0 1 0 1 0 1 0 1 ...

```
loop
  if clk then
    o <= x and not y;
  wait on clk;
end;
```

Action $o := x \wedge \neg y$ on "`clk` = 1" events

```
x <= 0; y <= 1;
wait on clk;
loop
    x <= rnd;
    assert (o != 1);
  wait on clk;
end;
```

Runtime monitor:

- Generates all possible entries
- Checks the property

Hardware Verification in VHDL (Proof)

```
loop
  clk <= not clk;
  wait for 1;
end;
```

Clock `clk` = 0 1 0 1 0 1 0 1 ...

```
loop
  if clk then
    o <= x and not y;
  wait on clk;
end;
```

Action $o := x \wedge \neg y$ on "`clk` = 1" events

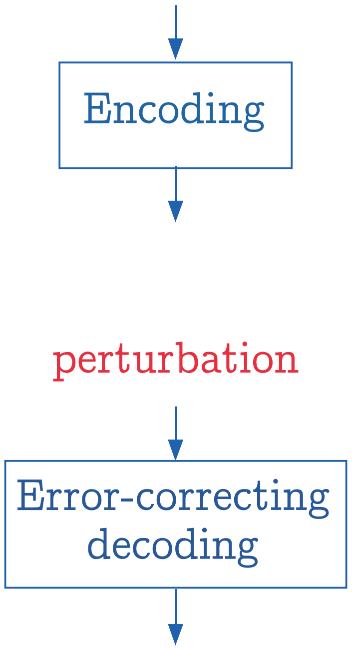
```
x <= 0; y <= 1;
wait on clk;
loop
    x <= rnd;
    assert (o != 1);
  wait on clk;
end;
```

Runtime monitor:

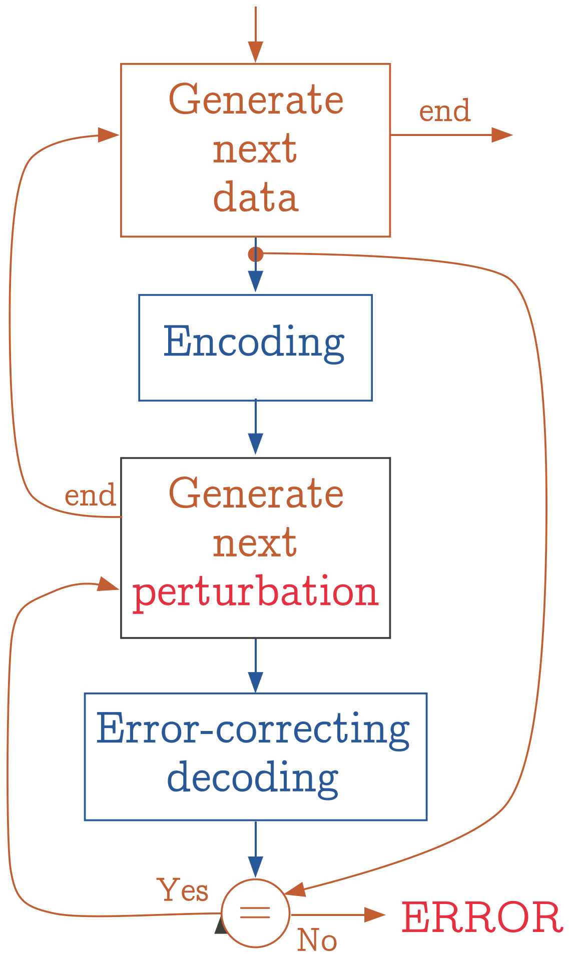
- Generates all possible entries
- Checks the property

Model checking/static analysis show the assertion to always hold

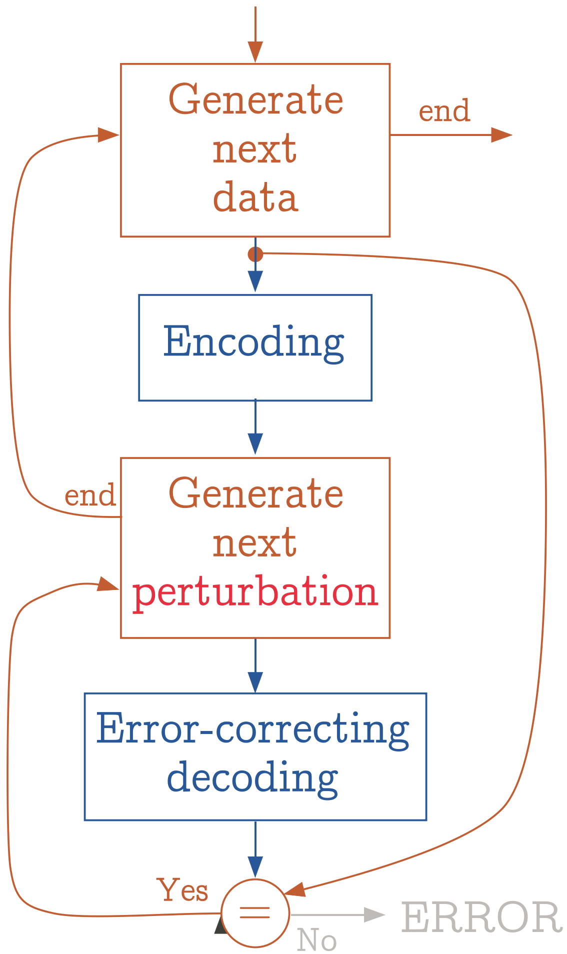
Hardware Verification (Reed-Solomon – Code)



Hardware Verification (Reed-Solomon – Monitor)



Hardware Verification (Reed-Solomon – Proof)



Simulation:	not exhaustive
Model-checking:	state explosion
Static analysis:	exhaustive verification

Example of Challenge in Hardware/Software Verification

- Data transmission using USB/AFDX is now preferred to avionic ARINC 429 transmit and receive channels
- Challenge: prove communications correct on a USB port, given
  - a software driver in C;
  - a hardware controler in VHDL;
  - a formal specification of "correct communication".

Example 5: Static Analysis of Avionic Safety-Critical Software

References

[ASTRÉE] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE analyser. *ESOP 2005*, Edinburgh, LNCS 3444, pp. 21–30, Springer, 2005. www.astree.ens.fr

Objective

- Show that static analysis by abstract interpretation does scale up
- (... and report on an industrialization success story)

The Static Analysis Problem

– Given a C control/command program and a configuration file [13],

– effectively compute a computer representation of an overapproximation of the reachable program states from the initial states,

– in order to statically prove the absence of runtime and user-defined errors.

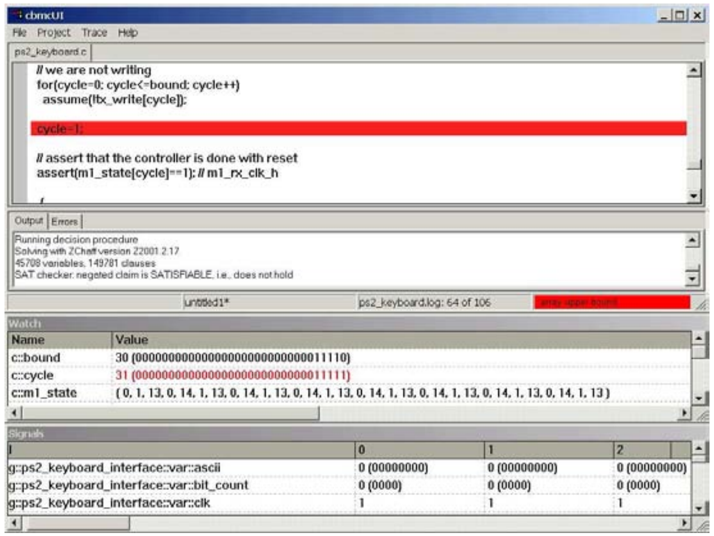
– **Extremely difficult to scale up!**

---

13 Physical range hypotheses for some sensor inputs

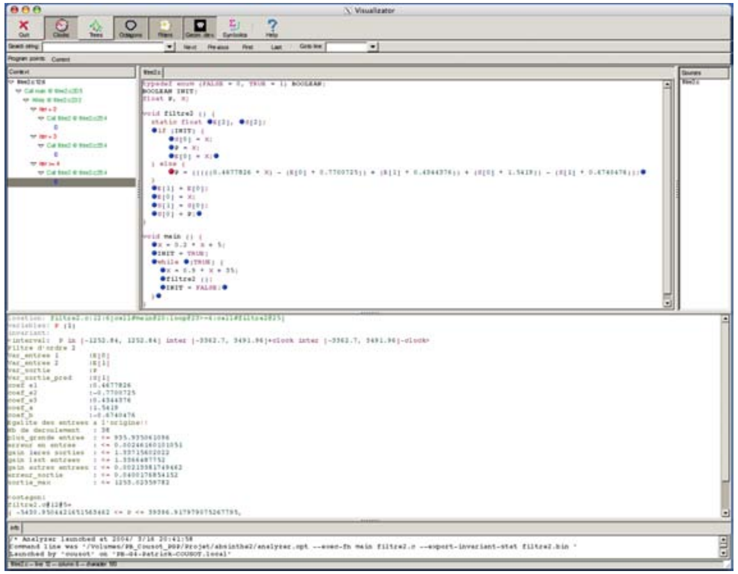
Example 1: CBMC

- CBMC is a Bounded Model Checker for ANSI-C programs (started at CMU in 1999).
- Allows verifying array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions.
- Aimed for embedded software, also supports dynamic memory allocation using `malloc`.
- Done by unwinding the loops in the program and passing the resulting equation to a SAT solver.
- **Problem (a.o.): does <u>not</u> scale up!**



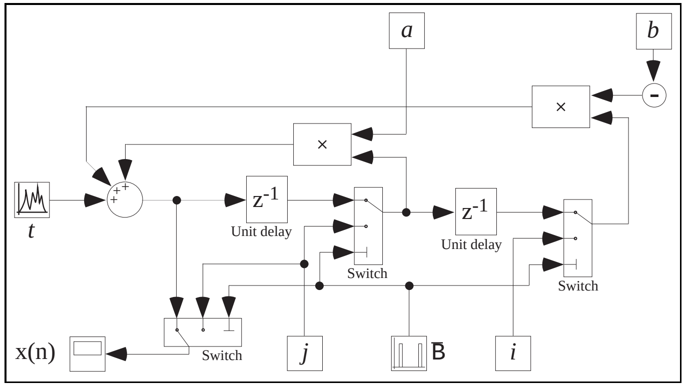
Example 2: ASTRÉE

- ASTRÉE is an abstract interpretation-based static analyzer for ANSI-C programs (started at ENS in 2001).
- Allows verifying array bounds (buffer overflows), pointer safety, exceptions and user-specified assertions.
- Aimed for embedded software, does not support dynamic memory allocation.
- Done by abstracting the reachability fixpoint equations for the program operational semantics.
- **Advantage (a.o.): <u>does</u> scale up!**



Ellipsoid Abstract Domain for Filters

2^{d} Order Digital Filter:

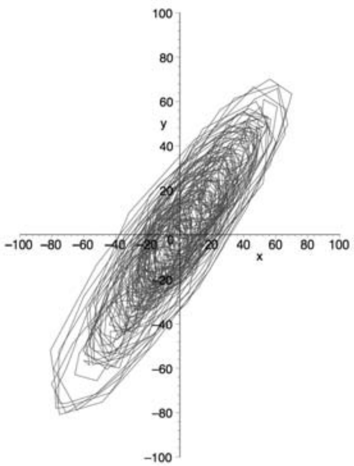


– Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$

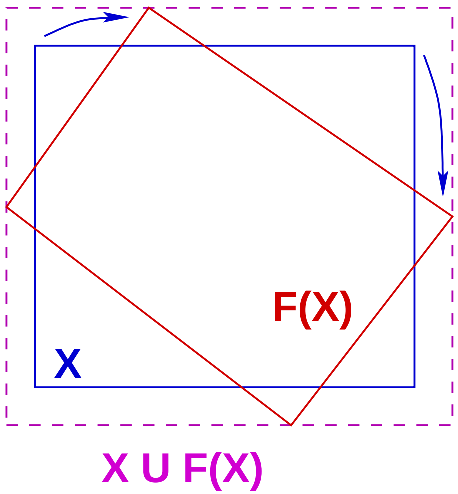
– The concrete computation is bounded, which must be proved in the abstract.

– There is no stable interval or octagon.

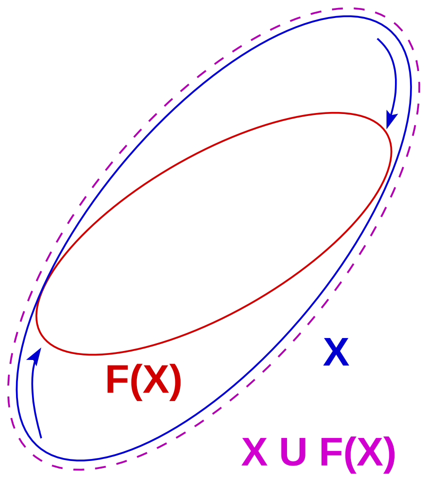
– The simplest stable surface is an ellipsoid.



execution trace



unstable interval



stable ellipsoid

Filter Example [6]

```
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float P, X;
void filter () {
  static float E[2], S[2];
  if (INIT) { S[0] = X; P = X; E[0] = X; }
  else { P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
             + (S[0] * 1.5)) - (S[1] * 0.7)); }
  E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
  /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}
void main () { X = 0.2 * X + 5; INIT = TRUE;
  while (1) {
    X = 0.9 * X + 35; /* simulated filter input */
    filter (); INIT = FALSE; }
}
```

Success Story

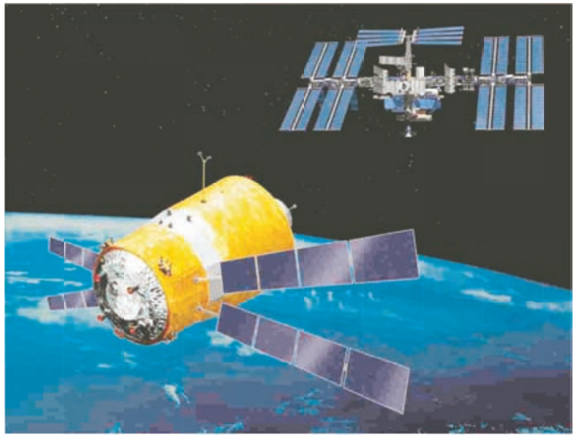
– A340 family (200/300/500/600): ASTRÉE is now part of the production line of the Primary Flight Control Software (130-250 000 lines)

– A380: ASTRÉE is still being tuned up to handle the Primary Flight Control Software (1000 000 lines) without false alarms

Projects

ASTRÉE Follow-on (I)
Space Software Validation by Abstract Interpretation

- ESA ITI Initiative, 2006–2008
- ENS + CEA + EADS SPACE Transportation
- Verification of the MSU software of the ATV docking the ISS [14]



[14] MSU:Monitoring and Safety Unit, ATV:Automated Transfer Vehicule, ISS:International Space Station.

ASTRÉE Follow-on (II) [15]

– Aeronautics, space, automotive, railway, medical industries
– 2006–2008 / 2007–2008
– ENS + Airbus + Astrium + Barco + CS SI + Daimler-Chrysler AG + Siemens VDO / Transportation + Thales Avionics + ...
– Static analysis verification tools for embedded software:


CNES Pleiade observation satellite


CNES MYRIADE micro-satellite series


Barco Medical imaging


Engine Management System 2nd Generation

[15] Outils de Vérification par Analyse Statique de Logiciels Embarqués/Embedded Software Product-based Assurance

THÉSÉE

- Verification of absence of runtime errors, data races and deadlocks in asynhronous safety-critical real-time embedded control/command software
- 2006–2009
- ENS + Airbus + EDF International (1600-megawatt EPR (Evolutionary Power Reactor) for the Finnish Olkiluoto 3 plant unit, to be operational in 2009)

ASBAPROD

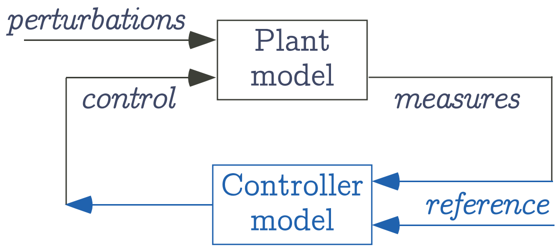
- Translation validation (Scade $\rightarrow$ C $\rightarrow$ ASM)
- Verification of functional properties of safety-critical real-time embedded synhronous electric flight control software, for example:
  - One and only one computer has control at any time,
  - If some input i changes by Δ_i then some output o changes by at most Δ_o, etc
- 2006–2010
- ENS + Airbus



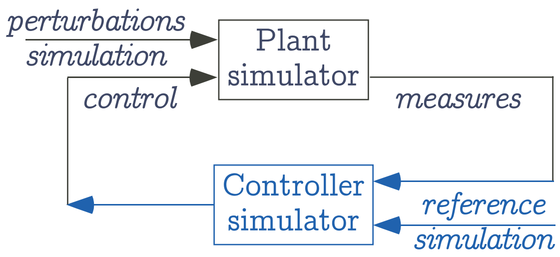

CONTROVERT

- CONTROl system VERificaTion
- 2006–2009
- ENS (computer scientists) + ONERA Toulouse (control theoreticians)

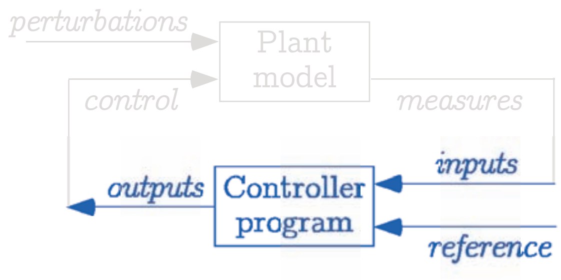
The Current Situation [16]



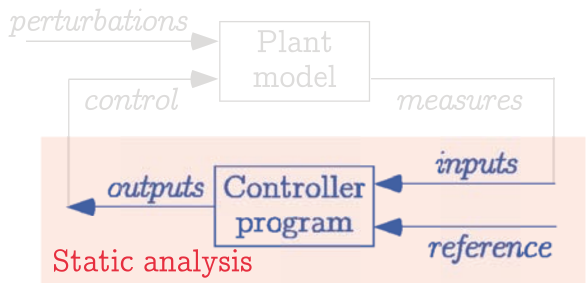
(1) Model design



(2) Simulation



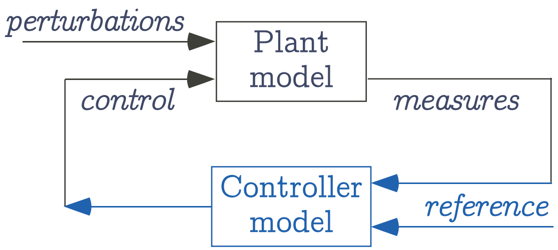
(3) Implementation



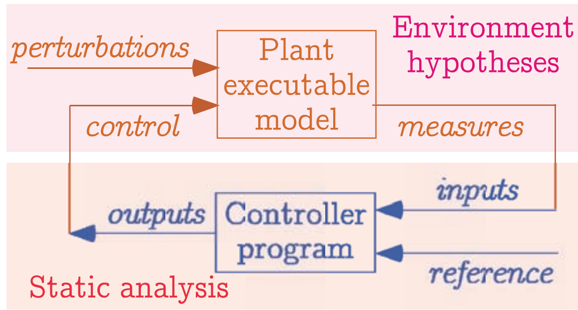
(4) Program analysis

16 greatly simplified, system dependability is simply ignored!

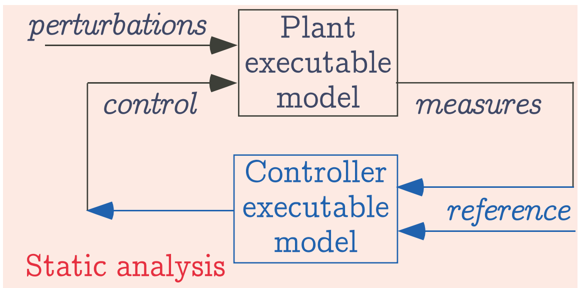
The Project [17]



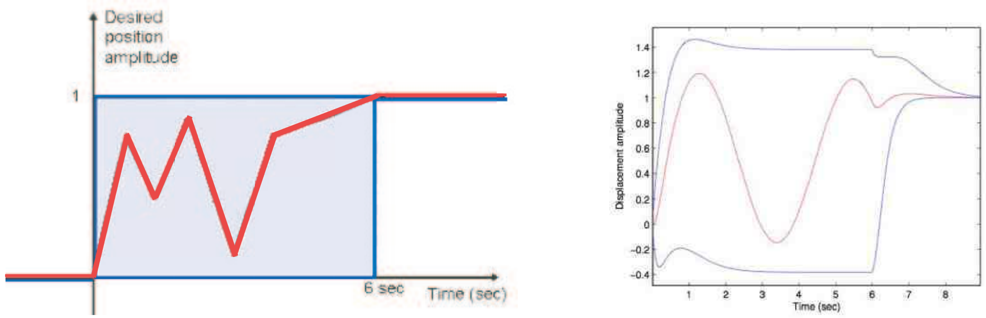
(1) Model design



(3) Program analysis



(2) Model analysis



Example (response analysis)

[17] greatly simplified, system dependability is simply ignored!

Conclusion

Formal Methods

– Formal methods have made considerable academic progress these last 30 years
– Automatic formal methods still have to scale up for everyday industrial practice
– The high-technology industries have imperative needs in software design & verification
– Static program analysis is progressively becoming an advanced industrial practice
– Automatic verification from specification design downto program implementation is a challenge

Abstract Interpretation

- Theoretical foundations: deep unification of formal methods, semantics, modularity/incrementability, parallelism/distribution/mobility, object-orientation, complex hardware/software/communication systems, integration of continuous/discrete/probabilistic models of the physical world/user interaction, ...
- Abstractions: abstract domains for safety, security, ..., controlability, robustness, ...
- Applications: beyond computer science, control/command, biology, ...

THE END, THANK YOU

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

References

[1] `www.astree.ens.fr` [3, 4, 5, 6, 7, 10, 11, 12, 13]

[2] 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, Université scientifique et médicale de Grenoble, Grenoble, France, 21 March 1978.

[3] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, LNCS 2566, pp. 85–108. Springer, 2002.

[4] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. *PLDI'03*, San Diego, pp. 196–207, ACM Press, 2003.

[POPL '77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY, USA.

[POPL '78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY, U.S.A.

[PACJM '79] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. Pacific Journal of Mathematics 82(1):43–57 (1979).

[POPL '79] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY, U.S.A.

[POPL '92] P. Cousot and R. Cousot. Inductive Definitions, Semantics and Abstract Interpretation. In Conference Record of the 19^{th} ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages, pages 83–94, Albuquerque, New Mexico, 1992. ACM Press, New York, U.S.A.

[FPCA '95] P. Cousot and R. Cousot. Formal Language, Grammar and Set-Constraint-Based Program Analysis by Abstract Interpretation. In *SIGPLAN/SIGARCH/WG2.8 7^{th} Conference on Functional Programming and Computer Architecture, FPCA'95*. La Jolla, California, U.S.A., pages 170–181. ACM Press, New York, U.S.A., 25-28 June 1995.

[POPL '97] P. Cousot. Types as Abstract Interpretations. In Conference Record of the 24^{th} ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages, pages 316–331, Paris, France, 1997. ACM Press, New York, U.S.A.

[POPL '00] P. Cousot and R. Cousot. Temporal abstract interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Mass., January 2000. ACM Press, New York, NY.

[POPL '02] P. Cousot and R. Cousot. Systematic Design of Program Transformation Frameworks by Abstract Interpretation. In *Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 178–190, Portland, Oregon, January 2002. ACM Press, New York, NY.

[TCS 277(1–2) 2002] P. Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Theoretical Computer Science* 277(1–2):47–103, 2002.

[TCS 290(1) 2002] P. Cousot and R. Cousot. Parsing as abstract interpretation of grammar semantics. *Theoret. Comput. Sci.*, 290:531–544, 2003.

[Manna's festschrift '03] P. Cousot. Verification by Abstract Interpretation. *Proc. Int. Symp. on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday*, N. Dershowitz (Ed.), Taormina, Italy, June 29 – July 4, 2003. Lecture Notes in Computer Science, vol. 2772, pp. 243–268. © Springer-Verlag, Berlin, Germany, 2003.

[POPL '04] P. Cousot and R. Cousot. An Abstract Interpretation-Based Framework for Software Watermarking. In *Conference Record of the Thirtyfirst Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 173–185, Venice, Italy, January 14-16, 2004. ACM Press, New York, NY.

[VMCAI '05] P. Cousot. Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. In *Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, pages 1–24, Paris, France, January 17-19, 2005. Lecture Notes in Computer Science, volume 3385, Springer, Berlin.

[5] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE analyser. *ESOP 2005*, Edinburgh, LNCS 3444, pp. 21–30, Springer, 2005.

[6] J. Feret. Static analysis of digital filters. *ESOP'04*, Barcelona, LNCS 2986, pp. 33—-48, Springer, 2004.

[7] J. Feret. The arithmetic-geometric progression abstract domain. In *VMCAI'05*, Paris, LNCS 3385, pp. 42–58, Springer, 2005.

[8] C. Hymans. Checking safety properties of behavioral VHDL descriptions by abstract interpretation. *SAS'02*, LNCS 2477.

[9] C. Hymans. Design and implementation of an abstract interpreter for VHDL. *CHARME '03*, LNCS 2860.

[10] Laurent Mauborgne & Xavier Rival. Trace Partitioning in Abstract Interpretation Based Static Analyzers. *ESOP'05*, Edinburgh, LNCS 3444, pp. 5–20, Springer, 2005.

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

[12] A. Miné. Relational abstract domains for the detection of floating-point run-time errors. *ESOP'04*, Barcelona, LNCS 2986, pp. 3—17, Springer, 2004.

[13] A. Miné. Weakly Relational Numerical Abstract Domains. *PhD Thesis*, École Polytechnique, 6 december 2004.

[DPG-ICALP '05] M. Dalla Preda and R. Giacobazzi. Semantic-based Code Obfuscation by Abstract Interpretation. In Proc. 32nd Int. Colloquium on Automata, Languages and Programming (ICALP'05 – Track B). LNCS, 2005 Springer-Verlag. July 11-15, 2005, Lisboa, Portugal.

[EMSOFT '01] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. *EMSOFT (2001)*, LNCS 2211, 469–485.

[RT-ESOP '04] F. Ranzato and F. Tapparo. Strong Preservation as Completeness in Abstract Interpretation. ESOP 2004, Barcelona, Spain, March 29 - April 2, 2004, D.A. Schmidt (Ed), LNCS 2986, Springer, 2004, pp. 18–32.