Abstract Interpretation: A Theory of Approximation

Patrick COUSOT

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

MFPS XVIII Tulane University, New Orleans, LA, USA
March 23-26, 2002

Abstract

We rapidly introduce elements of abstract interpretation, a formalization of the (effective) conservative approximation of the semantics of programs and more generally software and hardware computer systems.

We argue that most semantics-based reasonings and computations on programs involve conservative approximations which are naturally formalized by abstract interpretation. This is illustrated on the syntax and semantics of programming languages, typing and type inference, program transformation, model-checking and static program analysis.

Content

Motivations

This work was supported in part by the RTD project IST-1999-20527 DAEDALUS of the european IST FP5 programme.

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 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

- Consider simple specifications or programs (hopeless);
- Consider decidable questions only or semi-algorithms (e.g. model-checking);
- Ask the programmer to help (e.g. proof assistants);
- Consider approximations to handle practical complexity limitations (the main application of abstract interpretation).

The Theory of Abstract Interpretation

- **Abstract interpretation**[1] 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

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

Informal Introduction to Abstract Interpretation 1 – Property Abstraction

- Program concrete/abstract properties are elements of posets/ lattices/. . . ;
- Program property abstraction is performed by (effective) conservative approximation of concrete properties;
- The abstract property (hence semantics) is sound but may be incomplete with respect to the concrete property (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, etc. [2]);
- Otherwise, weaker abstraction/ concretization correspondences are available [3];

[2] P. Cousot & R. Cousot. *Systematic design of program analysis frameworks.* ACM POPL'79, pp. 269–282, 1979.
[3] 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 [4]);
- 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 [5];

4 — Effective Analysis/Checking/ Verification Algorithms

- Computable abstract semantics lead to effective program analysis/verification algorithms;
- Furthermore fixpoints can be over-approximated iteratively by convergence acceleration through widening/narrowing that is non-standard induction [6].

[4] 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.
[5] P. Cousot & R. Cousot. *Inductive definitions, semantics and abstract interpretation.* POPL, 83–94, 1992.
[6] 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

Galois Connections[7]

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

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

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

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

Composing Galois Connections

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

$$\langle P, \leq\rangle \underset{\alpha_2 \circ \alpha_1}{\overset{\gamma_1 \circ \gamma_2}{\leftrightarrows}} \langle R, \preceq\rangle$$ [8]

Function Abstraction

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

$$\langle S \mapsto P, \dot{\leq}\rangle \underset{\boldsymbol{\lambda} f \bullet \boldsymbol{\lambda} x \bullet \alpha(f(x))}{\overset{\boldsymbol{\lambda} g \bullet \boldsymbol{\lambda} y \bullet \gamma(g(y))}{\leftrightarrows}} \langle S \mapsto Q, \dot{\sqsubseteq}\rangle$$

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

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

8 This would not be true with the original definition of Galois correspondences.

Kleenian Fixpoint Approximation

Let $F \in L \overset{m}{\longmapsto} L$ and $\bar{F} \in \bar{L} \overset{m}{\longmapsto} \bar{L}$ be respective monotone maps on the cpos $\langle L, \bot, \sqsubseteq\rangle$ and $\langle \bar{L}, \bar{\bot}, \bar{\sqsubseteq}\rangle$ and $\langle L, \bot\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \bar{\bot}, \bar{\sqsubseteq}\rangle$ such that $\alpha \circ F \circ \gamma \mathrel{\dot{\bar{\sqsubseteq}}} \bar{F}$. Then [9]:

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

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

Kleenian Fixpoint Abstraction

Moreover, the *commutation condition* $\bar{F} \circ \alpha = \alpha \circ F$ implies:

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

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

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

Systematic Design of an Abstract Semantics

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

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

(and similarly in case of approximation [10]).

Tarskian Fixpoint Abstraction

Let $F \in L \overset{m}{\longmapsto} L$ and $\bar{F} \in \bar{L} \overset{m}{\longmapsto} \bar{L}$ be respective monotone maps on the complete lattices $\langle L, \sqsubseteq, \bot, \top, \sqcup, \sqcap\rangle$ and $\langle \bar{L}, \bar{\sqsubseteq}, \bar{\bot}, \bar{\top}, \bar{\sqcup}, \bar{\sqcap}\rangle$ and the abstraction function $\alpha \in L \overset{\sqcap}{\longmapsto} \bar{L}$ be a complete $\sqcap$-morphism satisfying:

- the *commutation inequality* $\bar{F} \circ \alpha \mathrel{\bar{\sqsubseteq}} \alpha \circ F$, and
- the *post-fixpoint correspondence* $\forall y \in \bar{L} : \bar{F}(y) \mathrel{\bar{\sqsubseteq}} y \Rightarrow \exists x \in L : \alpha(x) = y \wedge F(x) \sqsubseteq x$.

Then [11] $\alpha(\mathrm{lfp}^{\sqsubseteq} F) = \mathrm{lfp}^{\bar{\sqsubseteq}} \bar{F}$.

10 A complete example is handled in "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."

11 P. Cousot, *Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation*, TCS, 2002, to appear. Similar results hold for sound fixpoint approximation.

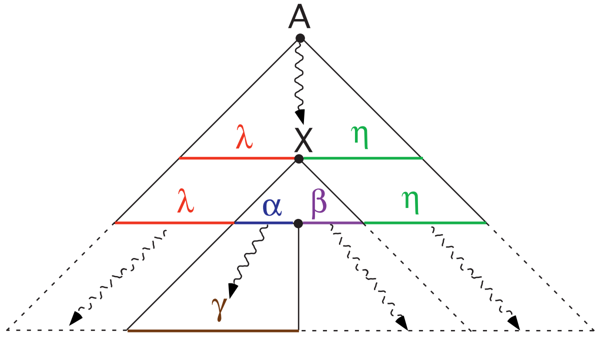
A Potpourri of Applications of Abstract Interpretation

Application to Syntax

P. Cousot & R. Cousot. *Parsing as Abstract Interpretation of Grammar Semantics*, TCS, 2002, to appear.

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

$$S = \text{lfp}^{\subseteq}\, F$$

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

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

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

  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, 2002, to appear.

Trace Semantics

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
- $S = \text{lfp}^{\sqsubseteq}\, F \in \Sigma^+ \cup \Sigma^\omega$ 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

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

1 — Relational Semantics Abstractions (Cont'd)

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

2 — Functional/Denotational Semantics Abstractions

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

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

3 — Predicate Transformer Semantics Abstractions

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

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

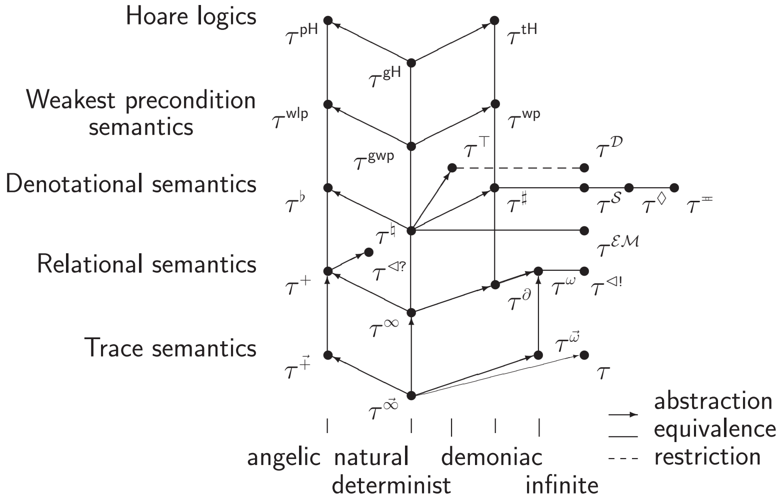
5 — Hoare Logic Semantics Abstractions

$$\langle \wp(\Sigma) \overset{\cap}{\longmapsto} \wp(\Sigma \cup \{\bot\}), \dot{\supseteq}\rangle \underset{\alpha^H}{\overset{\gamma^H}{\leftrightarrows}} \wp(\Sigma) \otimes^{12} \wp(\Sigma \cup \{\bot\}), \dot{\supseteq}\rangle$$

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

  predicate transformer to Hoare logic semantics

Lattice of Semantics



12 Semi-dual Shmuely tensor product.

Application to Typing

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

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
$\mid$	$(e_1 \mathbin{\mathbf{?}} e_2 \mathbin{\mathbf{:}} e_3)$	conditional

Semantic Domains

$$
\begin{array}{lr}
\Omega & \text{wrong/runtime error value} \\
\perp & \text{non-termination} \\
\mathbb{W} \stackrel{\text{def}}{=} \{\Omega\} & \text{wrong} \\
\mathsf{z} \in \mathbb{Z} & \text{integers} \\
\mathsf{u}, \mathsf{f}, \varphi \in \mathbb{U} \cong \mathbb{W}_\perp \oplus \mathbb{Z}_\perp \oplus [\mathbb{U} \mapsto \mathbb{U}]^{13}{}_\perp & \text{values} \\
\mathsf{R} \in \mathbb{R} \stackrel{\text{def}}{=} \mathbb{X} \mapsto \mathbb{U} & \text{environments} \\
\phi \in \mathbb{S} \stackrel{\text{def}}{=} \mathbb{R} \mapsto \mathbb{U} & \text{semantic domain}
\end{array}
$$

Denotational Semantics with Run-Time Type Checking

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

$$
\begin{aligned}
\mathbf{S}[\![e_1 - e_2]\!]\mathsf{R} \stackrel{\text{def}}{=} & \;(\, \mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \perp \;?\; \perp \\
& \mid \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z}_1 \wedge \mathbf{S}[\![e_2]\!]\mathsf{R} = \mathsf{z}_2 \;?\; \mathsf{z}_1 - \mathsf{z}_2 \\
& \mid \Omega \,)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{S}[\![(e_1 \,\mathbf{?}\, e_2 \,\mathbf{:}\, e_3)]\!]\mathsf{R} \stackrel{\text{def}}{=} & \;(\, \mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \;?\; \perp \\
& \mid \mathbf{S}[\![e_1]\!]\mathsf{R} = 0 \;?\; \mathbf{S}[\![e_2]\!]\mathsf{R} \\
& \mid \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{z} \neq 0 \;?\; \mathbf{S}[\![e_3]\!]\mathsf{R} \\
& \mid \Omega \,)
\end{aligned}
$$

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

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

$$
\begin{aligned}
\mathbf{S}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!]\mathsf{R} \stackrel{\text{def}}{=} \boldsymbol{\lambda}\,\mathsf{u} \bullet & (\, \mathsf{u} = \perp \;?\; \perp \\
& \mid \mathsf{u} = \Omega \;?\; \Omega \\
& \mid \mathbf{S}[\![e]\!]\mathsf{R}[\mathtt{x} \leftarrow \mathsf{u}] \,)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{S}[\![e_1(e_2)]\!]\mathsf{R} \stackrel{\text{def}}{=} & \;(\, \mathbf{S}[\![e_1]\!]\mathsf{R} = \perp \vee \mathbf{S}[\![e_2]\!]\mathsf{R} = \perp \;?\; \perp \\
& \mid \mathbf{S}[\![e_1]\!]\mathsf{R} = \mathsf{f} \in [\mathbb{U} \mapsto \mathbb{U}] \;?\; \mathsf{f}\big(\mathbf{S}[\![e_2]\!]\mathsf{R}\big) \\
& \mid \Omega \,)
\end{aligned}
$$

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

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:

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

- The standard collecting semantics is the strongest concrete property:

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

Church/Curry Monotypes

- Simple types are monomorphic:

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

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

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

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:

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

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

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

Concretization Function

The meaning of types is a program property, as defined by the concretization function γ^{c}: [14]

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

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

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

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

- typing $\gamma_3^{c} \in \mathbb{I}^{c} \mapsto \mathbb{P}$:

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

- program type $\gamma^{c} \in \mathbb{T}^{c} \mapsto \mathbb{P}$:

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

$$\gamma^{c}(\emptyset) \stackrel{\text{def}}{=} \mathbb{S}$$

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

Program Types

- Galois connection:

$$\langle \mathbb{P}, \subseteq, \emptyset, \mathbb{S}, \cup, \cap \rangle \xleftarrow[\alpha^c]{\gamma^c} \langle \mathbb{T}^c, \supseteq, \mathbb{I}^c, \emptyset, \cap, \cup \rangle$$

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

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

Typable Programs Cannot Go Wrong

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

Church/Curry Monotype Abstract Semantics

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

$$\mathbf{T}[\![\boldsymbol{\lambda}\mathtt{x} \cdot e]\!] \stackrel{\text{def}}{=} \{\langle H, m_1 \to m_2\rangle \mid \langle H[\mathtt{x} \leftarrow m_1], m_2\rangle \in \mathbf{T}[\![e]\!]\} \tag{ABS}$$

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

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

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

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

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

The Herbrand Abstraction to Get Hindley's Type Inference Algorithm

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

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);

Model-checking is an abstraction

- Universal abstraction:

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \supseteq \rangle \underset{\alpha_M^\forall}{\overset{\gamma_M^\forall}{\leftrightarrows}} \langle \wp(\Sigma), \supseteq \rangle$$

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

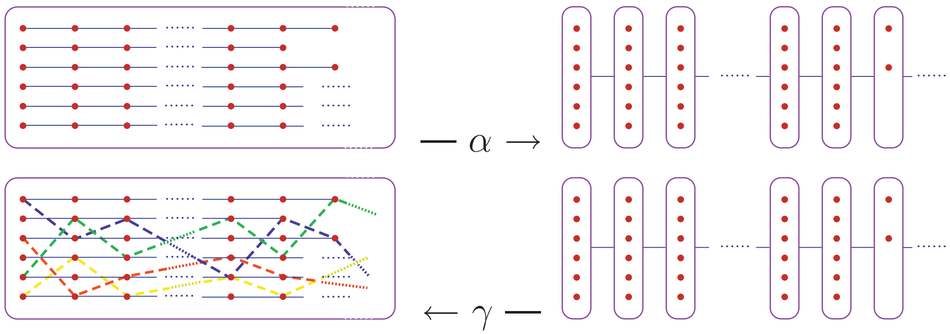
- Existential abstraction:

$$\langle \wp(\Sigma^+ \cup \Sigma^\omega), \subseteq \rangle \underset{\alpha_M^\exists}{\overset{\gamma_M^\exists}{\leftrightarrows}} \langle \wp(\Sigma), \subseteq \rangle$$

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

These abstractions lead to the classical (finite-state or nonterminating) model-checking algorithms.

Implicit Abstraction in Model Checking



Spurious traces: - - -, - - -, - - -, - - -, ··· ;

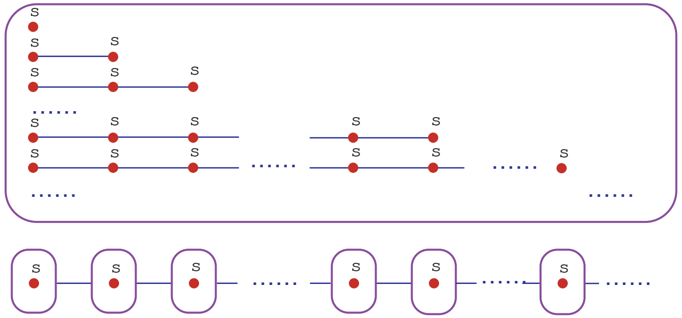
The semantics of the μ-calculus is closed under this abstraction.

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



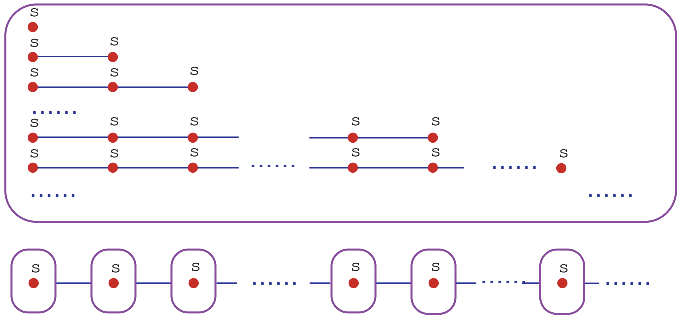
All abstract traces are infinite but not the concrete ones!

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 Soundness/Completeness of Model-Checking

- Model checking is sound and complete (for the model);
- This is due to 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 $\widehat{\mu}^{\star}$-calculus (cont'd)

$$
\begin{array}{llr}
\varphi ::= \boldsymbol{\sigma}_S{}^{15} & & [\![\boldsymbol{\sigma}_S]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \sigma_i \in S\} \\
\quad \mid \boldsymbol{\pi}_t{}^{16} & & [\![\boldsymbol{\pi}_t]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \langle \sigma_i, \sigma_{i+1}\rangle \in t\} \\
\quad \mid \oplus\varphi_1{}^{17} & & [\![\oplus\varphi_1]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \langle i+1, \sigma\rangle \in [\![\varphi_1]\!]\rho \\
\quad \mid \varphi_1^{\curvearrowleft} & & [\![\varphi_1^{\curvearrowleft}]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \mid \langle -i, \lambda j.\sigma_{-j}\rangle \in [\![\varphi_1]\!]\rho\} \\
\quad \mid \varphi_1 \vee \varphi_2 & & [\![\varphi_1 \vee \varphi_2]\!]\rho \stackrel{\text{def}}{=} [\![\varphi_1]\!]\rho \cup [\![\varphi_2]\!]\rho \\
\quad \mid \neg\,\varphi_1 & & [\![\neg\,\varphi_1]\!]\rho \stackrel{\text{def}}{=} \neg[\![\varphi_1]\!]\rho
\end{array}
$$

The reversible $\widehat{\mu}^{\star}$-calculus (cont'd)

$$
\begin{array}{ll}
\mid \ldots & \\
\mid X^{18} & [\![X]\!]\rho \stackrel{\text{def}}{=} \rho(X) \\
\mid \boldsymbol{\mu}\, X \cdot \varphi_1 & [\![\boldsymbol{\mu}\, X \cdot \varphi_1]\!]\rho \stackrel{\text{def}}{=} \operatorname{lfp}^{\subseteq} \boldsymbol{\lambda}\, x \cdot [\![\varphi_1]\!]\rho X x \\
\mid \boldsymbol{\nu}\, X \cdot \varphi_1 & [\![\boldsymbol{\nu}\, X \cdot \varphi_1]\!]\rho \stackrel{\text{def}}{=} \operatorname{gfp}^{\subseteq} \boldsymbol{\lambda}\, x \cdot [\![\varphi_1]\!]\rho X x \\
\mid \forall\, \varphi_1 : \varphi_2{}^{19} & [\![\forall\, \varphi_1 : \varphi_2]\!]\rho \stackrel{\text{def}}{=} \{\langle i, \sigma\rangle \in [\![\varphi_1]\!]\rho \mid \\
& \qquad \{\langle i, \sigma'\rangle \in [\![\varphi_1]\!]\rho \mid \sigma'_i = \sigma_i\} \subseteq [\![\varphi_2]\!]\rho\}
\end{array}
$$

15 $S \in \wp(\Sigma)$.

16 $t \in \wp(\Sigma \times \Sigma)$.

17 $\oplus$ is next time.

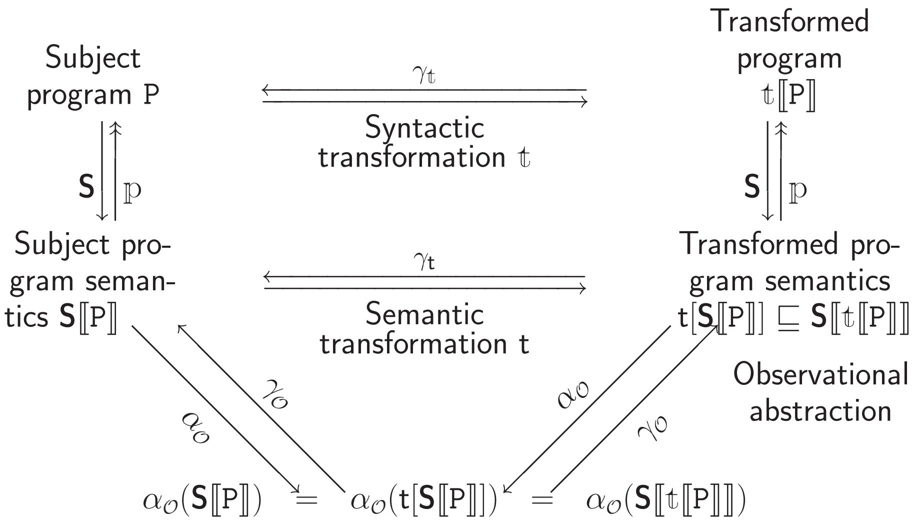
18 variable.

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

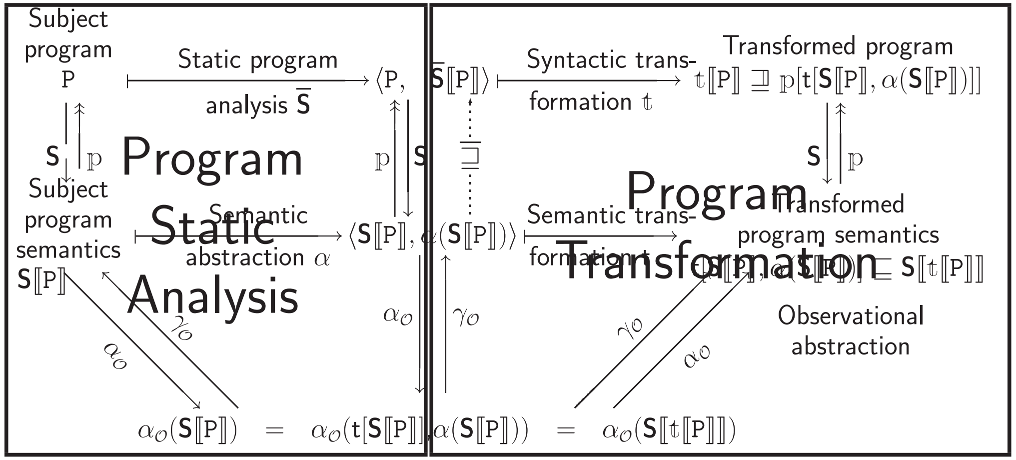
Application to Program Transformation

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

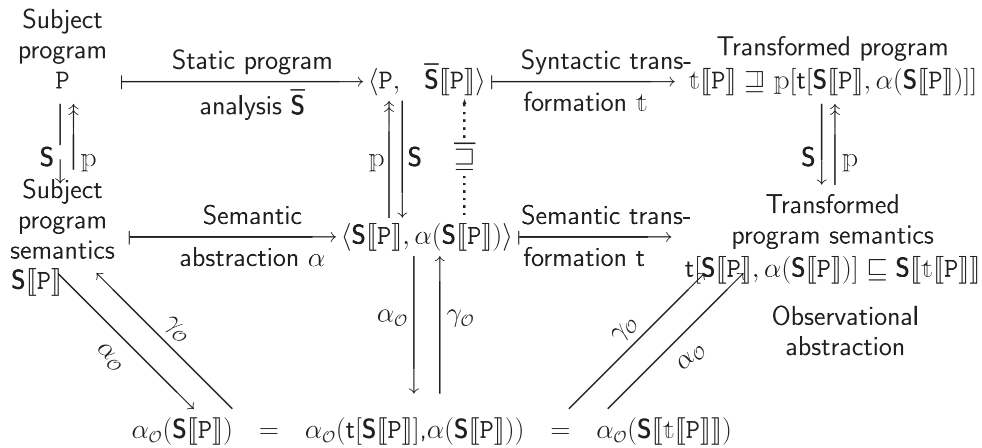
Online Program Transformation



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

Application to Static Program Analysis

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.

Collecting Semantics Abstractions

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

Example 1: reachable states [20]

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

Example 2: ancestor states [21]

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

[20] Abusively called "software model-checking"!

[21] Now "property based slicing"!

Partitioning

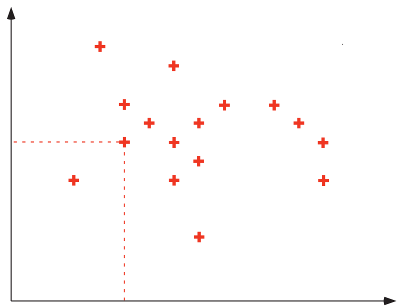
- If $\Sigma = C \times M$ (control and store state) and C is finite[22], we can partition:

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

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

- 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.

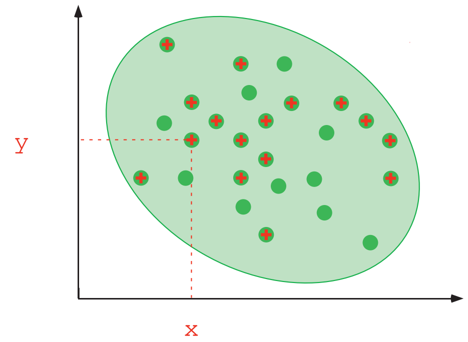
Approximations of an [in]finite set of points;



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

22 use e.g. dynamic partitioning if C is infinite

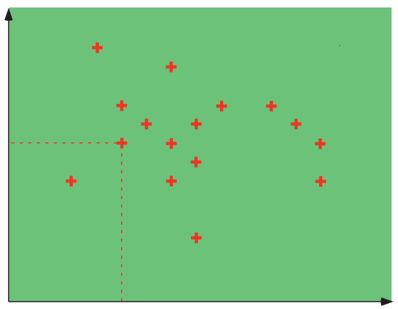
Approximations of an [in]finite set of points: From Above



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

From Below: dual[23] + combinations.

Effective computable approximations of an [in]finite set of points; Signs[24]

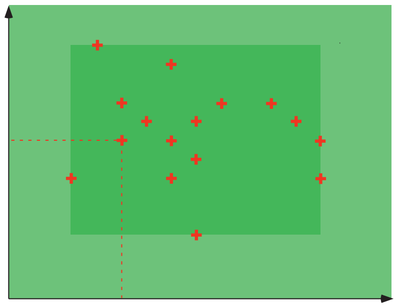


$$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$$

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

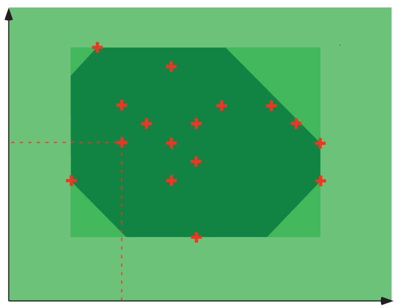
24 P. Cousot & R. Cousot. *Systematic design of program analysis frameworks.* ACM POPL'79, pp. 269–282, 1979.

Effective computable approximations of an [in]finite set of points; Intervals [25]



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

Effective computable approximations of an [in]finite set of points; Octagons [26]

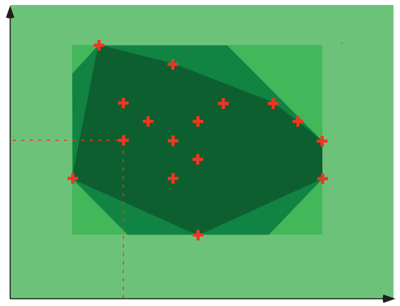


$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 9 \\ x - y \leq 99 \end{cases}$$

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

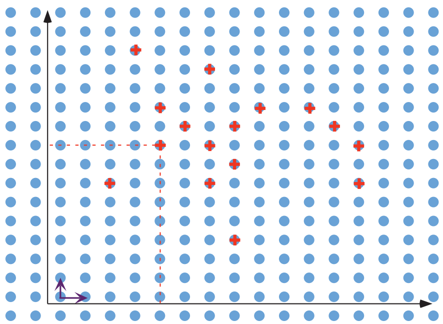
[26] A. Miné. *A New Numerical Abstract Domain Based on Difference-Bound Matrices.* PADO '2001. LNCS 2053, pp. 155–172. Springer 2001.

Effective computable approximations of an [in]finite set of points; Polyhedra [27]



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

Effective computable approximations of an [in]finite set of points; Simple congruences [28]

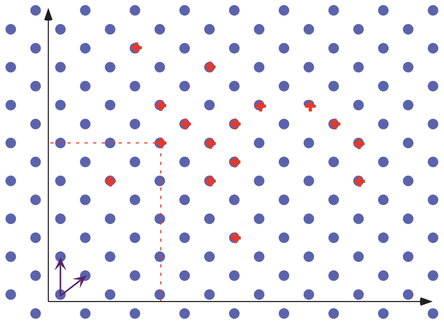


$$\begin{cases} x = 19 \bmod 77 \\ y = 20 \bmod 99 \end{cases}$$

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

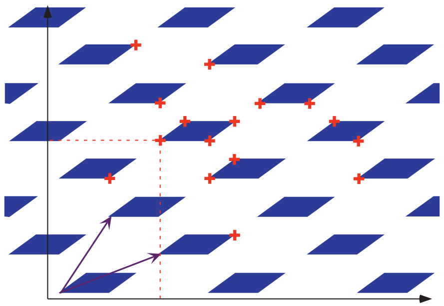
[28] Ph. Granger. *Static Analysis of Arithmetical Congruences.* Int. J. Comput. Math. 30, 1989, pp. 165–190.

Effective computable approximations of an [in]finite set of points; Linear congruences [29]



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

Effective computable approximations of an [in]finite set of points; Trapezoidal linear congruences [30]



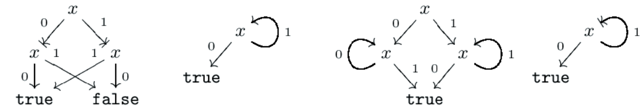
$$\begin{cases} 1x + 9y \in [0, 77] \bmod 10 \\ 2x - 1y \in [0, 99] \bmod 11 \end{cases}$$

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

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

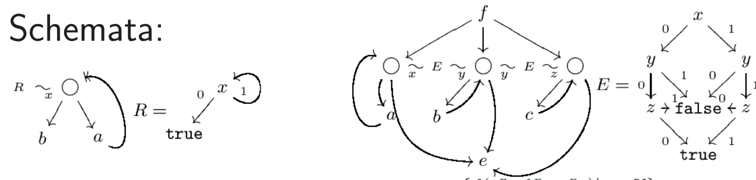
Example of Effective Abstractions of Infinite Sets of Infinite Trees [31]

Binary Decision Graphs:



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

Tree Schemata:



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

$\{a^n b | n \in \mathbb{N}\}$ Note that E is the equality relation.

Conclusion

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

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 semantic-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 (e.g. critical synchronous real-time embedded systems) so as to design *very efficient analyzers* with *almost zero-false alarm* [32].

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

THE END

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