Verification by Abstract Interpretation: Soundness and Abstract Induction

Patrick Cousot

cims.nyu.edu/~pcousot

PPDP — LOPSTR
July 14 – 16, 2015 — Siena, Italy

Motivation

Formal methods

Reasonings on programs are

• Reasonings on properties of their semantics (i.e. execution behaviors)

• Always involve some form of abstraction

Abstract interpretation

A theory establishing a correspondance between

• Concrete semantic properties
  ↑ what you want to prove on the semantics

• Abstract properties
  ↑ how to prove it in the abstract

Objective: formalize

• formal methods

• algorithms for reasoning on programs
Fundamental motivations

Scientific research

in **Mathematics/Physics:**

- trend towards **unification** and **synthesis** through universal principles

in **Computer science:**

- trend towards **dispersion** and **parcelization** through a collection of local techniques for specific applications

An exponential process, will stop!

Example: reasoning on computational structures
Example: reasoning on computational structures

Abstract interpretation

Practical motivations

Informal examples of abstraction


© P Cousot

All computer scientists have experienced bugs

Ariane 5.01 failure (overflow)
Patriot failure (float rounding)
Mars orbiter loss (unit error)
Heartbleed (buffer overflow)

Checking the presence of bugs by debugging is great
Proving their absence by static analysis is even better!
Undecidability and complexity is the challenge for automation
Abstractions of Dora Maar by Picasso

Pixelation

An old idea...

20 000 years old picture in a spanish cave:

(the concrete is unknown)

Abstractions of a man / crowd

Height
Fingerprint
Eye color
DNA
Phone metadata

Individual heights
min, max
Numerical abstractions in Astrée

Abstract interpretation-based tools usually use several different abstract domains, since the design of a complex one is best decomposed into a combination of simpler abstract domains. Here are a few abstract domain examples used in the Astrée static analyzer:

- **Collecting semantics:**
  \[ x, y \]

- **Intervals:**
  \[ x \in [a, b] \]

- **Simple congruences:**
  \[ x \equiv a[b] \]

- **Octagons:**
  \[ \pm x \pm y \leq a \]

- **Ellipses:**
  \[ x^2 + by^2 - axy \leq d \]

- **Exponentials:**
  \[ -a^{bt} \leq y(t) \leq a^{bt} \]

Such abstract domains can be combined using a reduced product for Galois connection abstractions. In absence of a Galois connection or for performance reasons, the conjunction is performed using an easily computable but not optimal overapproximation of this combination of abstract domains.

Making it easy...

- **No induction:**
  - model-checking finite systems
  - decidable cases

- **No soundness:** the last trend to fall in the easy, e.g.
  - Analyze Linux the easy way (ignoring aliases, overflows, recursion, etc.) \( \rightarrow \) 700 potential bugs
  - Ask PhD students to analyze manually the potential bug (3mn per bug maximum)
  - Claim 50 true bugs \( \rightarrow \) best paper award

Abstract Interpretation

Abstract interpretation is all about:

- **Soundness**
- **Induction**
A very short introduction to abstract interpretation

Concrete properties

A concrete property is represented by the set of elements which have that property:

- universe (set of elements) $\mathcal{D}$ (e.g. a semantic domain)
- properties of these elements: $P \in \wp(\mathcal{D})$
- “$x$ has property $P$” is $x \in P$

$\langle \wp(\mathcal{D}), \subseteq, \cup, \cap, \ldots \rangle$ is a complete lattice for inclusion $\subseteq$ (i.e. logical implication)

Properties and their Abstractions

Abstract properties

Abstract properties: $Q \in \mathcal{A}$

Abstract domain $\mathcal{A}$: encodes a subset of the concrete properties (e.g. a program logic, type terms, linear algebra, etc)

Poset: $\langle \mathcal{A}, \subseteq, \cup, \cap, \ldots \rangle$

Partial order: $\subseteq$ is abstract implication
Concretization

\( \gamma(\mathcal{Q}) \) is the semantics (concrete meaning) of \( \mathcal{Q} \)

\( \gamma \) is increasing (so \( \subseteq \) abstracts \( \subseteq \))

The concrete properties in \( \gamma(\mathcal{A}) \) are exactly representable in the abstract \( \mathcal{A} \), all others in \( \wp(\mathcal{U} \rightarrow \wp(\mathcal{E})) \backslash \gamma(\mathcal{A}) \) can only be approximated in \( \mathcal{A} \)

Best abstraction

A concrete property \( P \in \wp(\mathcal{D}) \) has a best abstraction \( Q \in \mathcal{A} \) iff

• it is sound (over-approximation):
  \[ P \subseteq \gamma(Q) \]

• and more precise than any sound abstraction:
  \[ P \subseteq \gamma(Q') \implies Q \subseteq Q' \implies \gamma(Q) \subseteq \gamma(Q') \]

The best abstraction is unique (by antisymmetry)

Under-approximation is order-dual

Examples

Needness/strictness analysis (80’s)

Similar abstraction (\( \gamma(\top) \triangleq \{ \text{true, false} \} \)) for scalable hardware symbolic trajectory evaluation STE (90)


Example: Homomorphic abstraction $\varphi(\mathcal{D}) \rightarrow \varphi(\mathcal{A})$

$h \in \mathcal{D} \rightarrow \mathcal{A}$

$\alpha \triangleq \lambda X \cdot \{ h(x) \mid x \in X \}$

$\gamma \triangleq \lambda Y \cdot \{ x \in \mathcal{D} \mid h(x) \in Y \}$

$\implies \langle \varphi(\mathcal{D}), \subseteq \rangle \overset{\gamma}{\rightarrow} \langle \varphi(\mathcal{A}), \subseteq \rangle$ (iff $h$ onto)

Example (*) rule of signs: $A = \mathbb{Z}, B = \{-1, 0, 1\}, h(z) = z/|z|$

Counter-example (**): intervals (octagons, polyhedra, etc)

(*) Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

(*** Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

---

Properties of Galois connections

$\alpha$ preserves existing lubs (by order-duality, $\gamma$ preserves existing glbs)

One adjoint uniquely determine the other

$\alpha$ is surjective (iff $\gamma$ injective iff $\alpha \circ \gamma = 1$), written

$\langle P, \leq \rangle \overset{\gamma}{\rightarrow} \langle Q, \subseteq \rangle$

The composition of Galois connections is a Galois connection

$\alpha(x)$ is the best over-approximation of $x \in P$:

- $x \leq \gamma(\alpha(x))$ over-approximation
- $x \leq \gamma(y) \implies \alpha(x) \subseteq y$ more precise than any other over-approximation

---

In absence of best abstraction?

Best abstraction of a disk by a rectangular parallelogram (intervals)

No best abstraction of a disk by a polyhedron (Euclid)

use only abstraction or concretization or widening (*)

Sound semantics abstraction

program \( P \in \mathbb{L} \)  
programming language

standard semantics \( S[P] \in \mathcal{D} \)  
semantic domain

collecting semantics \( \{S[P]\} \in \mathcal{D}(\mathcal{D}) \)  
semantic property

abstract semantics \( \overline{S}[P] \in \mathcal{A} \)  
abstract domain

concretization \( \gamma \in \mathcal{A} \rightarrow \mathcal{D}(\mathcal{D}) \)

soundness \( \{S[P]\} \subseteq \gamma(S[\overline{P}]) \)

i.e. \( S[P] \in \gamma(S[\overline{P}]), \)  
\( P \) has abstract property \( S[P] \)

Best abstract semantics

If \( \langle \mathcal{D}, \subseteq \rangle \xleftarrow{\gamma}{\alpha} \langle \mathcal{A}, \subseteq \rangle \) then the best abstract semantics is the abstraction of the collecting semantics \( \overline{S}[P] \not\equiv \alpha(\{S[P]\}) \)

Proof:

• It is sound: \( S[P] \not\equiv \alpha(\{S[P]\}) \subseteq S[P] \implies \{S[P]\} \subseteq \gamma(S[\overline{P}]) \)

• It is the most precise: \( S[P] \in \gamma(S[\overline{P}]) \implies \{S[P]\} \subseteq \gamma(S[\overline{P}]) \)

Example: functional connector

If \( g = \langle \mathcal{E}, \subseteq \rangle \xrightarrow{\gamma}{\alpha} \langle \mathcal{A}, \subseteq \rangle \) then

\[
g \implies g = \langle \mathcal{E} \rightarrow \mathcal{E}, \subseteq \rangle \xrightarrow{\lambda F. \gamma \circ F \circ \alpha} \langle \mathcal{A} \rightarrow \mathcal{A}, \subseteq \rangle
\]

Orthogonally, there are many styles of
• semantics (traces, relations, transformers, …)
• induction (transitional, structural, segmentation [POPL 2012])
• presentations (fixpoints, equations, constraints, rules [CAV 1995])
Fixpoint abstraction

Best abstraction (completeness case)

if $\alpha \cdot F = F \circ \bar{\alpha}$ then $F = \alpha \cdot F \circ \gamma$ and $\alpha(\text{lfp } F) = \text{lfp } F$

e.g. semantics, proof methods, static analysis of finite state systems

Best approximation (incompleteness case)

if $F = \alpha \cdot F \circ \gamma$ but $\alpha \cdot F \not\subseteq F \circ \bar{\alpha}$ then $\alpha(\text{lfp } F) \not\subseteq \text{lfp } F$

e.g. static analysis of infinite state systems

idem for equations, constraints, rule-based deductive systems, etc

---

Theorem

If $(C, \sqsubseteq) \xrightarrow{\gamma} (A, \preceq)$ in cpos for infinite/transfinite chains, $F \in C \hookrightarrow C$ and $G \in A \hookrightarrow A$ are continuous/increasing then

$$\alpha(\text{lfp}^\gamma F) = \text{lfp}^\gamma G \iff \alpha \circ F = G \circ \alpha$$  \hspace{1cm} \text{(commutation condition)}

$$\alpha(\text{lfp}^\gamma F) \preceq \text{lfp}^\gamma G \iff \alpha \circ F \preceq G \circ \alpha$$  \hspace{1cm} \text{(semi-commutation condition)}

---

[Cousot and Cousot, 1979b, theorem 7.1.0.4(2-3)], see also [de Bakker et al., 1984, lemma 4.3], [Apt and Plotkin, 1986, fact 2.3], [Backhouse, 2000, theorem 95], etc.


---

### Exact fixpoint abstraction

Abstract domain

$\perp \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha$

Concrete domain

$\perp \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha$

$\alpha \cdot F = F^\# \circ \alpha \Rightarrow \alpha(\text{lfp } F) = \text{lfp } F^\#$

---

### Approximate fixpoint abstraction

Abstract domain

$\perp \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha$

Concrete domain

$\perp \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha \quad \alpha$

$\text{lfp } F \not\subseteq \gamma(\text{lfp } F^\#)$
Duality

Order duality: join ($\cup$) or meet ($\cap$)

Inversion duality: forward ($\rightarrow$) or backward ($\leftarrow = (\rightarrow)^{-1}$)

Fixpoint duality: least ($\downarrow$) or greatest ($\uparrow$)

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

Patrick Cousot, Radhia Cousot: Systematic Design of Program Analysis Frameworks. POPL 1979: 269-282

Why abstracting properties of semantics, not semantics?

1. Abstract interpretation = a non-standard semantics (computations on values in the standard semantics are replaced by computations on abstract values) $\Rightarrow$ extremely limited

2. Abstract interpretation = an abstraction of the standard semantics $\Rightarrow$ limited

3. Abstract interpretation = an abstraction of properties of the standard semantics $\Rightarrow$ more

i.e. (1) is an abstraction of (2), (2) is an abstraction of (3)

Example: trace semantics properties

Domain of [in]finite traces on states: $\mathcal{P}$

“Standard” trace semantics domain: $\mathcal{D} = \wp(\mathcal{P})$

“Standard” trace semantics $S[P] \in \mathcal{D} = \wp(\mathcal{P})$

Domain of semantics properties is $\wp(\mathcal{D}) = \wp(\wp(\mathcal{P}))$

Collecting semantics $C[P] \triangleq \{ S[P] \} \in \wp(\mathcal{D}) = \wp(\wp(\mathcal{P}))$
How to abstract the standard semantics?

The join abstraction:

\[ (\mathcal{P}(\mathcal{P}(\Pi)), \subseteq) \xrightarrow{\gamma \cup} (\mathcal{P}(\Pi), \subseteq) \]
\[ \alpha_{\cup}(X) \triangleq \bigcup X \]
\[ \gamma_{\cup}(Y) \triangleq \mathcal{P}(Y) \]

Join abstraction of the collecting semantics:

\[ \alpha_{\cup}(C[\llbracket P \rrbracket]) \triangleq \bigcup \{ S[\llbracket P \rrbracket] \} \triangleq S[\llbracket P \rrbracket] \]

(i.e. the semantics is the join abstraction of its strongest property)

Loss of information

“Always terminate with the same value, either 0 or 1”

\[ P = \]

Join abstraction:

\[ \alpha_{\cup}(P) = \]

“Always terminate, either with 0 or 1”

Limitations of the union abstraction

Complete iff any property of the semantics \( S[\llbracket P \rrbracket] \) is also valid for any subset \( \gamma(S[\llbracket P \rrbracket]) = \mathcal{P}(S[\llbracket P \rrbracket]) \):

- Examples: safety, liveness
- Counter-example: security (e.g. authentication using a random cryptographic nonce)
Exact abstractions

The concrete properties of the standard semantics $S[P]$ that you want to prove can always be proved in the abstract (which is simpler):

$$\forall Q \in \mathcal{A}: S[P] \subseteq \gamma(Q) \iff S[P] \subseteq Q$$

where

$$S[P] \triangleq \alpha \circ S[P] \circ \gamma$$

Example 1 of exact abstractions: grammars


Example: Grammars

**Context-free grammar** on alphabet $A = Num \cup Var \cup \{+,-, (,\ldots\}:

$$E ::= Num | Var | E + E | -E | (E)$$

Chomsky-Schützenberger *fixpoint semantics*:

$$S[E] = \mathit{fp} \subseteq F[E]$$

$$F[E]X \triangleq S[Num] \cup S[Var]$$

$$\cup \{ e_1 + e_2 | e_1, e_2 \in X \}$$

$$\cup \{-e | e \in X\} \cup \{(e) | e \in X\}$$

**Example: Grammars (cont’d)**

**FIRST abstraction** of a language $X \in A^*$:

$$\alpha_F(X) \triangleq \{ \ell | \exists \sigma \in A^*: \ell \sigma \in X \} \cup \{ \epsilon | \epsilon \in X \}$$

Galois connection:

$$\langle \varphi(A^*), \subseteq \rangle \xrightleftharpoons[\gamma_F \alpha_F]{\gamma_F \alpha_F} \langle \varphi(A \cup \{\epsilon\}), \subseteq \rangle$$

where

$$\gamma_F(Y) \triangleq \{ \ell \sigma | \ell \in Y \land \sigma \in A^* \} \cup \{ \epsilon | \epsilon \in Y \}$$
Example: Grammars (cont’d)

Commutation:
\[ \alpha_F \circ \mathcal{F}[E] = \mathcal{F}[E] \circ \alpha_F \]

where for \( E ::= \text{Num} | \text{Var} | E + E | -E | (E) \)
\[ \mathcal{F}[E] Y \triangleq S[\text{Num}] \cup S[\text{Var}] \cup (Y \setminus \{\varepsilon\}) \cup \{+\ | \epsilon \in Y\} \cup \{-,\} \]

FIRST abstract semantics:
\[ \mathcal{S}[E] \triangleq \alpha_F(\mathcal{S}[E]) \]
\[ = \alpha_F(\text{lfp} \circ \mathcal{F}[E]) \quad \text{(Chomsky-Schützenberger)} \]
\[ = \text{lfp} \circ \mathcal{F}[E] \quad \text{(fixpoint abstraction th.)} \]

Algorithm

Read the grammar \( G \), establish the system of equations
\[ Y = \mathcal{F}[G](Y) \], solve by chaotic iterations

This is, up to [en]coding details, the classical algorithm:

for each \( \alpha \in (T \cup e) \)
\[ \text{FIRST}(\alpha) \leftarrow \alpha \]
for each \( A \in NT \)
\[ \text{FIRST}(A) \leftarrow \emptyset \]
while (FIRST sets are still changing)
   for each \( p \in P \), where \( p \) has the form \( A \rightarrow \beta \)
      if \( \beta \leq 1 \)
         \[ \text{FIRST}(A) \leftarrow \text{FIRST}(A) \cup (\text{FIRST}(\beta_k) - \{\varepsilon\}) \]
         \[ i \leftarrow 1 \]
         while \((\varepsilon \in \text{FIRST}(\beta_i)) \) and \( i \leq k \)
            \[ \text{FIRST}(A) \leftarrow \text{FIRST}(A) \cup (\text{FIRST}(\beta_{i+1}) - \{\varepsilon\}) \]
            \[ i \leftarrow i + 1 \]
      if \( i = k \) and \( \varepsilon \in \text{FIRST}(\beta_k) \)
      then \[ \text{FIRST}(A) \leftarrow \text{FIRST}(A) \cup \{\varepsilon\} \]

Hierarchies of abstractions
Comparison of abstractions

\[\langle P, \leq \rangle \xrightarrow{\gamma_1} \alpha_1 \langle Q, \sqsubseteq \rangle\]

is more precise than

\[\langle P, \leq \rangle \xrightarrow{\gamma_2} \alpha_2 \langle R, \preceq \rangle\]

iff \(\gamma_2(R) \subseteq \gamma_1(Q)\)

(every abstraction in \(R\) is exactly expressible by \(Q\))

We say that \(Q\) is a refinement of \(R\) and \(R\) that is an abstraction of \(Q\)

A pre-order

---

**Example of proto-derivation**

---

**Example of proto-derivation tree**

---

**Example 11 of exact abstraction: graphs**
Transition system

Transition system: \( \langle \Sigma, A, \rightarrow \rangle \)

transition relation: \( \rightarrow \in \varphi(\Sigma \times A \times \Sigma) \)

transitions/edges: \( \sigma \xrightarrow{A} \sigma' \)

Example: non-negatively weighted graphs \( A \triangleq \mathbb{N} \)

Fixpoint characterization

Pointwise fixpoint characterization:

\[
\Pi = \text{ifp}^\subseteq F \\
F \in ((\Sigma \times \Sigma) \rightarrow \varphi(\Theta^+)) \rightarrow ((\Sigma \times \Sigma) \rightarrow \varphi(\Theta^+)) \\
F(X)(\sigma, \sigma') = \{ \sigma = \sigma' \land \{ \sigma \cup \{ \sigma \xrightarrow{A} \sigma'' \pi \mid \sigma \xrightarrow{A} \sigma' \land \sigma'' \pi \in X(\sigma'', \sigma') \} \}
\]

(a path of \( n \) transitions is either a single vertex \( (n = 0) \) or an edge followed by a path of \( n - 1 \) transitions)

Finite paths

Finite paths:

\[
\Theta^+ \triangleq \{ \sigma_0 \xrightarrow{A_0} \sigma_1 \ldots \sigma_{n-1} \xrightarrow{A_{n-1}} \sigma_n \mid n \geq 0 \land \forall i \in [0, n] : \sigma_i \in \Sigma \land \forall i \in [0, n) : A_i \in A \}
\]

Paths between two vertices:

\[
\Pi \in (\Sigma \times \Sigma) \rightarrow \varphi(\Theta^+) \\
\Pi(\sigma, \sigma') \triangleq \{ \sigma_0 \xrightarrow{A_0} \sigma_1 \ldots \sigma_{n-1} \xrightarrow{A_{n-1}} \sigma_n \mid \sigma = \sigma_0 \land n \geq 0 \land \forall i \in [0, n-1] : \sigma_i \xrightarrow{A_i} \sigma_{i+1} \land \sigma_n = \sigma' \}
\]

Minimal path length abstraction

Edges have non-negative lengths \( A = \mathbb{N} \)

Abstraction:

\[
\alpha \in \Theta^+ \mapsto \mathbb{N} \\
\alpha(\sigma) \triangleq 0 \\
\alpha(\sigma \xrightarrow{n} \sigma' \pi) \triangleq n + \alpha(\sigma' \pi) \\
\alpha(\sigma) \triangleq \min\{\alpha(\pi) \mid \pi \in X\}
\]

where \( \min(\emptyset) = +\infty \)

\( \mathbb{N}^\infty \triangleq \mathbb{N} \cup \{+\infty\} \)

\( \langle \mathbb{N}^\infty, \geq, \min \rangle \) is a complete lattice
Galois connection

\[\langle \varphi(\Theta^+), \subseteq \rangle \xleftrightarrow{\gamma}{\alpha} \langle \mathbb{N}^\infty, \supseteq \rangle\]

Pointwise extension:

\[\hat{\alpha}(X)(\sigma, \sigma') \triangleq \alpha(X, \sigma')\]

Pointwise Galois connection:

\[\langle (\Sigma \times \Sigma) \mapsto \varphi(\Theta^+), \subseteq \rangle \xleftrightarrow{\gamma}{\alpha} \langle (\Sigma \times \Sigma) \mapsto \mathbb{N}^\infty, \supseteq \rangle\]

Calculational design of the shortest distance algorithm

\[\hat{\alpha} \circ F\]

\[= \lambda X \cdot \hat{\alpha}(F(X))\quad \text{[def. \(F\)]}\]

\[= \lambda (\sigma, \sigma') \cdot \lambda X \cdot \hat{\alpha}(F(X)) (\sigma, \sigma')\quad \text{[def. \(\lambda x \cdot e\)]}\]

\[= \lambda (\sigma, \sigma') \cdot \lambda X \cdot \hat{\alpha}(\lambda (\sigma, \sigma')) \cdot \left\{ \sigma = \sigma' \quad ? \{ \sigma \} \right\} \triangleq \bigcup_{\sigma' \in \Sigma} \left\{ \sigma \xrightarrow{n} \sigma'' \pi \mid \sigma \xrightarrow{n} \sigma'' \land \sigma'' \pi \in X(\sigma', \sigma') \right\}\quad \text{[def. \(F\)]}\]

\[= \lambda (\sigma, \sigma') \cdot \lambda X \cdot \alpha(\sigma = \sigma' \quad ? \{ \sigma \}) \triangleq \bigcup_{\sigma' \in \Sigma} \left\{ \sigma \xrightarrow{n} \sigma'' \pi \mid \sigma \xrightarrow{n} \sigma'' \land \sigma'' \pi \in X(\sigma', \sigma') \right\}\quad \text{[def. conditional \(\ldots ? \ldots \)]}\]

\[= \lambda (\sigma, \sigma') \cdot \lambda X \cdot \left\{ \sigma = \sigma' \quad ? \alpha(\{ \sigma \}) \right\} \triangleq \bigcup_{\sigma' \in \Sigma} \left\{ \sigma \xrightarrow{n} \sigma'' \pi \mid \sigma \xrightarrow{n} \sigma'' \land \sigma'' \pi \in X(\sigma', \sigma') \right\}\quad \text{[join preservation in Galois C.]}\]

\[= \lambda (\sigma, \sigma') \cdot \lambda X \cdot \left\{ \sigma = \sigma' \quad ? \min_{\pi \in \{ \sigma \}}(\alpha(\pi) \mid \pi \in \{ \sigma \}) \right\} \triangleq \min_{\pi \in \{ \sigma \}}(\alpha(\pi) \mid \pi \in \{ \sigma \}) \quad \text{[def. min in \(\hat{\alpha}\)]}\]

\[\text{by defining}\]

\[G(X)(\sigma, \sigma') = \left\{ \sigma = \sigma' \quad ? \min_{\pi \in \{ \sigma \}}(X(\sigma', \sigma') \mid \sigma \xrightarrow{n} \sigma' \pi) \right\}\quad \text{[def. \(G \circ \hat{\alpha}\)]}\]

\[= \lambda X \cdot G(\hat{\alpha}(X))\quad \text{[def. \(G \circ \hat{\alpha}\)]}\]

Shortest distance

Shortest distance \(\Delta(\sigma, \sigma')\) between any two vertices

\[\Delta \in (\Sigma \times \Sigma) \mapsto \mathbb{N}^\infty\]

\[\Delta \triangleq \hat{\alpha}(\Pi) = \hat{\alpha}(\text{lfp} \subseteq F)\]

Calculational design of the shortest distance algorithm

\[= \lambda (\sigma, \sigma') \cdot \lambda X \cdot \left\{ \sigma = \sigma' \quad ? \min_{\pi \in \{ \sigma \}}(\alpha(\pi) \mid \pi \in \{ \sigma \}) \right\} \triangleq \min_{\pi \in \{ \sigma \}}(\alpha(\pi) \mid \pi \in \{ \sigma \}) \quad \text{[def. \(G \circ \hat{\alpha}\)]}\]

\[= \lambda X \cdot G(\hat{\alpha}(X))\quad \text{[def. \(G \circ \hat{\alpha}\)]}\]
Shortest distance in fixpoint form

By the fixpoint abstraction theorem

\[ \Delta = \alpha(\text{fix} G) \]
\[ = \text{fix} G \]
\[ = \min_{n \in \mathbb{N}} G^n(\lambda, \sigma', \lambda) \cdot +\infty \]

where the iterates are

1. \( G^0(\lambda) = \lambda \)
2. \( G^{n+1} = G \circ G^n, n \in \mathbb{N} \)

Example III of exact abstractions: semantics

Not Floyd-Warshall? Take instead:

\[ \alpha(\sigma) \triangleq 0 \]
\[ \alpha(\sigma \overset{n}{\rightarrow} \sigma') \triangleq n \]
\[ \alpha(\pi \sigma \pi') \triangleq \alpha(\pi \sigma) + \alpha(\sigma \pi') \]

Trace semantics

Initial states

\[ a \]
\[ b \]
\[ c \]
\[ d \]

Intermediate states

\[ e \]
\[ g \]
\[ i \]
\[ k \]

Final states of the finite traces

\[ j \]
\[ h \]

Infinite traces

0 1 2 3 4 5 6 7 8 9 ... discrete time

Abstraction to denotational/natural semantics

Abstraction to small-steps operational semantics

Abstraction to reachability/invariance

Abstraction to Hoare logic
Verification/static analysis by abstract interpretation

Define the syntax of programs $P \in \mathbb{L}$

Define the concrete semantics of programs:

- $\mathbb{D}[P]$ concrete semantic domain

- $\forall P \in \mathbb{L}: S[P] \in \mathbb{D}[P]$ concrete semantics

Concrete/semantic properties: $\mathcal{F}(\mathbb{D}[P])$

Collecting semantics: $\{S[P]\} \in \mathcal{F}(\mathbb{D}[P])$

(the strongest property of the semantics, which implies all other semantic properties)

Verifikation/statisk analys av abstrakt interpretasjon

Definer syntaksen for program $P \in \mathbb{L}$

Definer konkrete semantikk for program:

- $\mathbb{D}[P]$ konkrete semantisk domen

- $\forall P \in \mathbb{L}: S[P] \in \mathbb{D}[P]$ konkrete semantikk

Konkrete/semantiske egenskaper: $\mathcal{F}(\mathbb{D}[P])$

Samlende semantikk: $\{S[P]\} \in \mathcal{F}(\mathbb{D}[P])$

(the strongest property of the semantics, which implies all other semantic properties)
Verification/static analysis by abstract interpretation

Completeness (for exact abstractions only)


Methodology:

• Structural induction on programs P
• Compositional definition(*) of S[P] and α[P]/γ[P]
• Fixpoint abstraction/approximation for recursion

Verification for fixpoints is the main problem:

\[ \text{lfp} F# [P] \subseteq Q \]


Approximate abstractions

The concrete properties of the standard semantics S[P] that you want to prove may not always be provable in the abstract:

∀ Q ∈ S[P] ∈ γ(Q) ⟷ S'[P] ⊆ Q

where

S[P] ⊆ αoldsymbol; \cdot S[P] • γ
Why abstraction may be approximate?

Example

\[
\begin{align*}
\{ x = y \land 0 \leq x \leq 10 \} \\
x := x - y; \\
\{ x = 0 \land 0 \leq y \leq 10 \}
\end{align*}
\]

Interval abstraction:

\[
\begin{align*}
\{ x \in [0, 10] \land y \in [0, 10] \} \\
x := x - y; \\
\{ x \in [-10, 10] \land y \in [0, 10] \}
\end{align*}
\]

(but for constants, the interval abstraction can’t express equality)

---

Refinement

Problem: how to prove a valid abstract property \( \alpha(\{ \text{ lfp } F[\mathcal{P}] \}) \subseteq Q \) when \( \alpha \circ F \subseteq F^\# \circ \alpha \) but \( \text{lfp } F^\#[\mathcal{P}] \nsubseteq Q \)?

It is always possible to refine \( \langle \mathcal{A}, \subseteq \rangle \) into a most abstract more precise abstraction \( \langle \mathcal{A}', \subseteq' \rangle \) such that

\[
\langle \varphi(\mathcal{D}), \subseteq \rangle \xrightarrow{\gamma'} \langle \mathcal{A}', \subseteq' \rangle
\]

and \( \alpha' \circ F = F' \circ \alpha \) with \( \text{lfp } F'[\mathcal{P}] \subseteq' \alpha' \circ \gamma(Q) \)

(again proving \( \text{lfp } F'[\mathcal{P}] \in \gamma'(Q) \) which implies \( \text{lfp } F'[\mathcal{P}] \in \gamma(Q) \))

Refinement: good news

Refinement: bad news

But, refinements of an abstraction can be intrinsically incomplete

The only complete refinement of that abstraction for the collecting semantics is:

- the identity (i.e. no abstraction at all)

In that case, the only complete refinement of the abstraction is to the collecting semantics and any other refinement is always imprecise
Example of intrinsic approximate refinement

Consider executions traces $\langle i, \sigma \rangle$ with infinite past and future:

$$\begin{align*}
&\vdots \sigma_{-2} \sigma_{-1} \sigma_0 \sigma_1 \sigma_2 \sigma_3 \sigma_4 \vdots \\
&\text{states} \quad \text{time origin} \quad \text{present time} \quad \text{past} \quad \text{future}
\end{align*}$$

Patrick Cousot, Radhia Cousot: Temporal Abstract Interpretation. POPL 2000: 12-25
In general refinement does not terminate

Narrowing is needed to stop infinite iterated automatic refinements:

e.g. SLAM stops refinement after 20mn

Intelligence is needed for refinement:
e.g. human-driven refinement of Astrée


Finite versus infinite abstractions
[In]finite abstractions

Given a program $\mathcal{P}$ and a program property $Q$ which holds (i.e. $\text{lfp } F[\mathcal{P}] \in Q$) there exists a most abstract abstraction in a finite domain $\mathcal{A}[\mathcal{P}]$ to prove it. (*)

Example:

\[
\begin{align*}
  x &= 0; \text{ while } x < 1 \text{ do } x \text{++} \rightarrow \{\bot, [0,0], [0,1], [-\infty, \infty]\} \\
  x &= 0; \text{ while } x < 2 \text{ do } x \text{++} \rightarrow \{\bot, [0,0], [0,1], [0,2], [-\infty, \infty]\} \\
  \vdots \\
  x &= 0; \text{ while } x < n \text{ do } x \text{++} \rightarrow \{\bot, [0,0], [0,1], [0,2], [0,3], \ldots, [0,n], [-\infty, \infty]\} \\
  \vdots 
\end{align*}
\]

Convergence acceleration

Infinite iteration

Accelerated iteration with widening
(e.g. with a widening based on the derivative as in Newton-Raphson method)


Problem with infinite abstractions

For non-Noetherian iterations, we need

- finitary abstract induction, and
- finitary passage to the limit

\[ X^0 = ⊥, \ldots, X^{n+1} = \mathcal{F}(X^0, \ldots, X^n, F(X^0), \ldots, F(X^n)), \ldots, \lim_{n \to \infty} X^n \]

iteration converging

<table>
<thead>
<tr>
<th>Iteration starting from</th>
<th>above the limit</th>
<th>below the limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>below the limit</td>
<td>widening ▼</td>
<td>dual narrowing △</td>
</tr>
<tr>
<td>above the limit</td>
<td>narrowing △</td>
<td>dual widening △</td>
</tr>
</tbody>
</table>

[Semi-]dual abstract induction methods

(co-induction)

(induction)

(terminate from termination conditions)
Examples of widening/narrowing

Abstract induction for intervals:

• a widening \[^{[1,2]}\]

\[
[a_1, b_1] \triangleright [a_2, b_2] = \\
\begin{cases} 
\text{if } a_2 < a_1 \text{ then } \Rightarrow \text{ else } a_1 \text{ fi}, \\
\text{if } b_2 > b_1 \text{ then } \Leftarrow \text{ else } b_1 \text{ fi}
\end{cases}
\]

• a narrowing \[^{[2]}\]

\[
[a_1, b_1] \triangleleft [a_2, b_2] = \\
\begin{cases} 
\text{if } a_1 = \Leftarrow \text{ then } a_2 \text{ else MIN } (a_1, a_2), \\
\text{if } b_1 = \Rightarrow \text{ then } b_2 \text{ else MAX } (b_1, b_2)
\end{cases}
\]

On widening/narrowing/and their duals

Because the abstract domain is non-Noetherian, any widening/narrowing/duals can be strictly improved infinitely many times (i.e. no best widening)

E.g. widening with thresholds

\[
\forall x \in L_2, \perp \triangleright V(f)x = x \triangleright V(f)\perp = x
\]

\[
[u_1, u_2] V(x)[u_1, u_2] = [if 0 \leq l_2 < l_1 \text{ then 0 elsif } l_2 < l_1 \text{ then } -b \text{ else } l_1 \text{ fi},
\]

if \(u_1 < u_2 \leq 0 \text{ then 0 elsif } u_1 < u_2 \text{ then b else } u_1 \text{ fi}\]

Any terminating widening is not increasing (in its 1 parameter)

Any abstraction done with Galois connections can be done with widenings (i.e. a widening calculus)

Infinitary static analysis with abstract induction

Widening

\(\langle \mathcal{A}, \sqsubseteq \rangle\) poset

\(\triangledown \in \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}\)

Sound widening (upper bound):

\(\forall x, y \in \mathcal{A}: x \sqsubseteq x \triangledown y \land y \sqsubseteq x \triangledown y\)

Terminating widening: for any \(<x^0 \in \mathcal{A}, n \in \mathbb{N}>\), the sequence \(y^0 \trianglelefteq x^0, \ldots, y^{n+1} \trianglelefteq y^n \triangledown x^n, \ldots\) is ultimately stationary (\(\exists \alpha \in \mathbb{N}: \forall n \geq \alpha: y^n = y^\alpha\))

(Note: sound and terminating are independent properties)

Example: (simple) widening for polyhedra

Iterates

\[
\begin{align*}
\overline{F^n} & \\
\overline{F(F^n)} & \\
\overline{F^n} \lor \overline{F(F^n)} &
\end{align*}
\]

Widening

\[
\begin{align*}
\overline{F^n} \lor \overline{F(F^n)} &
\end{align*}
\]

Iteration with widening for static analysis

Problem: compute \( I \) such that \( \text{lfp} \preceq F \preceq I \preceq Q \)

Compute \( I \) as the limit of the iterates:

\[
\begin{align*}
X^0 & \triangleq \bot, \\
X^{n+1} & \triangleq X^n \quad \text{when } F(X^n) \preceq X^n \text{ so } I = X^n \\
X^{n+1} & \triangleq (X^n \triangledown F(X^n)) \triangle Q \quad \text{otherwise}
\end{align*}
\]

\( I \) can be improved by an iteration with narrowing \( \triangledown \)

Check that \( F(I) \subseteq Q \)

Example: Astrée

Patrick Cousot, Radhia Cousot: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252

Dual narrowing

\[ \langle \mathcal{A}, \sqsubseteq \rangle \text{ poset} \]
\[ \triangle \sim \sqsubseteq \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A} \]

Sound dual narrowing (interpolation):

\[
\forall x, y \in \mathcal{A}: x \sqsubseteq y \implies x \sqsubseteq x \triangle \sim y \sqsubseteq y
\]

Terminating dual narrowing: for any \( \langle a^n \in \mathcal{A}, n \in \mathbb{N} \rangle \), the sequence \( y^0 \triangleq a^0, \ldots, y^{n+1} \triangleq a^n \triangle \sim y^n, \ldots \) is ultimately stationary \((\exists \varepsilon \in \mathbb{N}: \forall n \geq \varepsilon: y^n = y^\varepsilon)\)

(Note: sound and terminating are independent properties)


Iteration with dual narrowing for static checking

Problem: find \( I \) such that \( \text{lfp} \preceq F \preceq I \preceq Q \)

Compute \( I \) as the limit of the iterates:

\[
\begin{align*}
X^0 & \triangleq \bot, \\
X^{n+1} & \triangleq X^n \quad \text{when } F(X^n) \preceq X^n \text{ so } I = X^n \\
X^{n+1} & \triangleq F(X^n) \triangle \sim Q \quad \text{otherwise}
\end{align*}
\]

Check that \( F(I) \subseteq Q \)

Example: First-order logic + Craig interpolation (with some choice of one of the solutions, control of combinatorial explosion, and convergence enforcement)

Kenneth L. McMillan: Applications of Craig Interpolants in Model Checking. TACAS 2005: 1-12
Example of domain-specific abstraction: ellipses

typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;

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; }
}
Comments on screenshot (courtesy Francesco Logozzo)

1. A screenshot from Clousot/cccheck on the classic binary search.
2. The screenshot shows from left to right and top to bottom:
   1. C# code + CodeContracts with a buggy BinarySearch
   2. cccheck integration in VS (right pane with all the options integrated in the VS project system)
   3. cccheck messages in the VS error list
3. The features of cccheck that it shows are:
   1. basic abstract interpretation:
      1. the loop invariant to prove the array access correct and that the arithmetic operation may overflow is inferred fully automatically
      2. different from deductive methods as e.g. ESC/Java or Boogie where the loop invariant must be provided by the end-user
   2. inference of necessary preconditions:
      1. Clousot finds that array may be null (message 3)
      2. Clousot suggests and propagates a necessary precondition invariant (message 1)
   3. array analysis (+ disjunctive reasoning):
      1. to prove the postcondition should infer property of the content of the array
      2. please note that the postcondition is true even if there is no precondition requiring the array to be sorted.
4. verified code repairs:
   1. from the inferred loop invariant does not follow that index computation does not overflow

Abstract interpretation

Intellectual tool (not to be confused with its specific application to iterative static analysis with ▽ & △)

No cathedral would have been built without plumb-line and square, certainly not enough for skyscrapers:

Powerful tools are needed for progress and applicability of formal methods

Conclusion

Varieties of researchers in formal methods:

(i) explicitly use abstract interpretation, and are happy to extend its scope and broaden its applicability
(ii) implicitly use abstract interpretation, and hide it
(iii) pretend to use abstract interpretation, but misuse it
(iv) don’t know that they use abstract interpretation, but would benefit from it

Never too late to upgrade
The End

Thank You