Sound Verification by Abstract Interpretation

Patrick Cousot

cims.nyu.edu/~pcousot

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
  \[ \uparrow \text{what you want to prove on the semantics} \]

• Abstract properties
  \[ \uparrow \text{how to prove it in the abstract} \]

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

<table>
<thead>
<tr>
<th>WCET</th>
<th>Axiomatic semantics</th>
<th>Confidentiality analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security protocol verification</td>
<td>Dataflow analysis</td>
<td>Denotational semantics</td>
</tr>
<tr>
<td>Systems biology analysis</td>
<td>Model checking</td>
<td>Theories combination</td>
</tr>
<tr>
<td>Database query</td>
<td>Dependence analysis</td>
<td>Abstract model checking</td>
</tr>
<tr>
<td>Operational semantics</td>
<td>Abstraction refinement</td>
<td>Type inference</td>
</tr>
<tr>
<td>Type theory</td>
<td>Symbolic execution</td>
<td>Interpolants</td>
</tr>
<tr>
<td>Quantum entanglement detection</td>
<td>Bisimulation</td>
<td>Shape analysis</td>
</tr>
<tr>
<td>SMT solvers</td>
<td>Code refactoring</td>
<td>Malware detection</td>
</tr>
<tr>
<td>Steganography</td>
<td>Tautology testers</td>
<td></td>
</tr>
</tbody>
</table>

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

Abstract interpretation

WCET
Axiomatic semantics
Confidentiality analysis
Program synthesis
Grammar analysis
Statistical model-checking
Invariance proof
Probabilistic verification
Parsing
Security protocol verification
Systems biology analysis
Operational semantics
Abstraction refinement
Type inference
Partial evaluation
Effect systems
Denotational semantics
Model checking
Database query
Dependence analysis
CEGAR
Program transformation
Obfuscation
Trace semantics
Theories combination
Interpolants
Quantum entanglement detection
Bisimulation
SMT solvers
Type theory
Shape analysis
Symbolic execution
Code contracts
Integrity analysis
Abstract model checking
Steganography
Tautology testers
Parsing
Grammar analysis
Bisimulation
Interpolants
Tautology testers

Practical motivations

Informal examples of abstraction

All computer scientists have experienced bugs

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

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...

Abstractions of a man / crowd
Numerical abstractions in Astrée

Collecting semantics: $[^1,5]
\text{partial traces}$

Intervals: $[^20]
x \in [a, b]$

Simple congruences: $[^{24}]$
\[ x \equiv a[b] \]

Octagons: $[^{25}]$
\[ \pm x \pm y \leq a \]

Ellipses: $[^{26}]$
\[ x^2 + by^2 - axy \leq d \]

Exponentials: $[^{27}]$
\[ -a^{bt} \leq y(t) \leq a^{bt} \]

Difficulties

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: $P \in \mathcal{P}(\mathcal{D})$
- “$x$ has property $P$” is $x \in P$

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

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

Partial order: $\subseteq$ is abstract implication

Properties and their Abstractions
Concretization

Concretization $\gamma \in \mathcal{A} \rightarrow \wp(\mathcal{D})$

$\gamma(Q)$ is the semantics (concrete meaning) of $Q$

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

The concrete properties in $\gamma(\mathcal{A})$ are exactly representable in the abstract $\mathcal{A}$, all others in $\wp(\mathcal{D}) \setminus \gamma(\mathcal{A})$ 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

Galois connection

Any $P \in \wp(\mathcal{D})$ has a (unique) best abstraction $\alpha(P)$ in $\mathcal{A}$ if and only if

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

written

$\langle \wp(\mathcal{D}), \subseteq \rangle \xrightarrow{\gamma} \langle \mathcal{A}, \subseteq \rangle$

$\Rightarrow$: over-approximation

$\Leftarrow$: best abstraction

Examples

Needness/strictness analysis (80’s)

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


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

\[
h \in \mathcal{D} \rightarrow \mathcal{A}
\]

\[
\alpha \triangleq \lambda X. \{ h(x) \mid x \in X \}
\]

\[
\gamma \triangleq \lambda Y. \{ x \in \mathcal{D} \mid h(x) \in Y \}
\]

\[
\Rightarrow \langle \varphi(\mathcal{D}), \subseteq \rangle \xrightarrow{\gamma \alpha} \langle \varphi(\mathcal{A}), \subseteq \rangle \quad (\rightarrow \text{iff } h \text{ onto})
\]

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

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

The composition of Galois connections is a Galois connection

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

\[ x \leq \gamma(\alpha(x)) \]

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: $P \in L$ (programming language)
- Standard semantics: $S[P] \in \mathcal{D}$ (semantic domain)
- Collecting semantics: $\{S[P]\} \in \wp(\mathcal{D})$ (semantic property)
- Abstract semantics: $\overline{S}[P] \in \mathcal{A}$ (abstract domain)
- Soundness: $\{S[P]\} \subseteq \gamma(S[\overline{P}])$
  
  i.e. $S[P] \in \gamma(S[\overline{P}])$, $P$ has abstract property $S[\overline{P}]$

Best abstract semantics

If $\langle \varnothing(\mathcal{D}), \subseteq \rangle \xrightarrow{\gamma}{\alpha} \langle \mathcal{A}, \subseteq \rangle$ then the best abstract semantics is the abstraction of the collecting semantics

$$S[P] \triangleq \alpha(\{S[P]\})$$

Proof:

- It is sound: $S[\overline{P}] \triangleq \alpha(\{S[\overline{P}]\}) \subseteq S[P] \implies \{S[P]\} \subseteq \gamma(S[\overline{P}])$

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

Calculational design of the abstract semantics

The (standard hence collecting) semantics are defined by composition of mathematical structures (such as set unions, products, functions, fixpoints, etc)

If you know best abstractions of properties, you also know best abstractions of these mathematical structures

So, by composition, you also know the best abstraction of the collecting semantics — calculational design of the abstract semantics

Orthogonally, there are many styles of

- semantics (traces, relations, transformers, etc)
- induction (transitional, structural, segmentation [POPL 2012])
- presentations (fixpoints, equations, constraints, rules [CAV 1995])

Example: functional connector

If $g = \langle \mathcal{C}, \subseteq \rangle \xrightarrow{\gamma}{\alpha} \langle \mathcal{A}, \subseteq \rangle$ then

$$g \implies g = \langle \mathcal{C} \xrightarrow{\gamma} \mathcal{C}, \subseteq \rangle \xrightarrow{\lambda F \cdot \gamma \circ F \circ \alpha}{\alpha \cdot F \circ \gamma} \langle \mathcal{A} \xrightarrow{\gamma} \mathcal{A}, \subseteq \rangle$$

($\implies$ is a called a Galois connector)
Fixpoint abstraction

Best abstraction (completeness case)

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

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

Best approximation (incompleteness case)

if $\overline{F} = \alpha \circ F \circ \gamma$ but $\alpha \circ F \subseteq \overline{F} \circ \alpha$ then $\alpha(\text{lfp } F) \subseteq \text{lfp } \overline{F}$

e.g. static analysis of infinite state systems

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

Fixpoint abstraction

Theorem 1

If $(C, \sqsubseteq, (A, \preceq))$ in cpos for infinite/transfinite chains, $F \subseteq C \mapsto C$ and $G \subseteq A \mapsto A$ are continuous/increasing then

- $\alpha(\text{lfp }^C F) = \text{lfp }^C G$ if $\alpha \circ F = G \circ \alpha$ (commutation condition)
- $\alpha(\text{lfp }^C F) \preceq \text{lfp }^C G$ if $\alpha \circ F \preceq G \circ \alpha$ (semi-commutation condition)


Exact fixpoint abstraction

Abstract domain

Concrete domain

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

Approximate fixpoint abstraction

Abstract domain

Concrete domain

$\text{lfp } F \subseteq \gamma(\text{lfp } F^\sharp)$
Duality

Order duality: join (∪) or meet (∩)

Inversion duality: forward (→) or backward (← = (→)^{-1})

Fixpoint duality: least (↓) or greatest (↑)

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) ⟹ extremely limited
2. Abstract interpretation = an abstraction of the standard semantics ⟹ limited
3. Abstract interpretation = an abstraction of properties of the standard semantics ⟹ 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: \( \Pi \)

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

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

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

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

The join abstraction:

\[ (\mathcal{P}(\mathcal{P}(\Pi)), \subseteq) \text{ and } (\mathcal{P}(\Pi), \subseteq) \]

\[ \alpha_U(X) \triangleq \bigcup X \]
\[ \gamma_U(Y) \triangleq \mathcal{P}(Y) \]

Join abstraction of the collecting semantics:

\[ \alpha_U(\mathcal{C}[P]) \triangleq \bigcup \{S[P]\} \triangleq S[P] \]

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

Limitations of the union abstraction

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

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

Loss of information

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

\[ P = \]

Join abstraction:

\[ \alpha_U(P) = \]

“Always terminate, either with 0 or 1”

Exact abstractions
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):

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

where

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

Example 1 of exact abstraction: grammars


Example: Grammars

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

$$E ::= \text{Num} | \text{Var} | E + E | -E | (E)$$

Chomsky-Schützenberger fixpoint semantics:

$$S[E] = \text{fp}_\mathcal{F}[E]$$

$$\mathcal{F}[E] X \triangleq S[\text{Num}] \cup S[\text{Var}]$$

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

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

Example: Grammars (cont’d)

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

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

Galois connection:

$$\langle \varphi(A^*), \subseteq \rangle \xrightarrow{\gamma_F} \langle \varphi(A \cup \{\epsilon\}), \subseteq \rangle$$

where

$$\gamma_F(Y) \triangleq \{\ell \sigma \mid \ell \in Y \land \sigma \in A^*\} \cup \{\epsilon \mid \epsilon \in Y\}$$
Commutation:
\[ \alpha_F \circ \mathcal{F}[E] = \mathcal{F}[E] \circ \alpha_F \]

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

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

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 \( \alpha \in (T \cup \epsilon) \)
   FIRST(\alpha) \leftarrow \alpha
for each \( A \in N \)
   FIRST(A) \leftarrow \emptyset
while (FIRST sets are still changing)
   for each \( p \in P \) where \( p \) has the form \( A \rightarrow \beta \)
      if \( \beta \) is \( \beta_1, \beta_2, \ldots, \beta_k \), where \( \beta_i \in T \cup N \)
      then
         FIRST(A) \leftarrow FIRST(A) \cup (FIRST(\beta_i) - \{\epsilon\})
      i \leftarrow 1
      while (i \leq k - 1)
         FIRST(A) \leftarrow FIRST(A) \cup (FIRST(\beta_i) - \{\epsilon\})
         i \leftarrow i + 1
      if i = k and \( \epsilon \in \text{FIRST}(\beta_k) \)
      then FIRST(A) \leftarrow FIRST(A) \cup \{\epsilon\}
```

Hierarchies of abstractions
Comparison of abstractions

\[ \langle P, \leq \rangle \overset{\gamma_1}{\underset{\alpha_1}{\iff}} \langle Q, \sqsubseteq \rangle \]

is more precise than

\[ \langle P, \leq \rangle \overset{\gamma_2}{\underset{\alpha_2}{\iff}} \langle R, \preceq \rangle \]

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

Hierarchies of Grammar Semantics

Chomsky–Schützenberger terminal language

Example of proto-derivation tree

Example of proto-derivation

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

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

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

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

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

---

**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:

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

---

**Fixpoint characterization**

Pointwise fixpoint characterization:

\[ \Pi = \text{lfp}^\subseteq F \]

\[ F \in (\Sigma \times \Sigma) \rightarrow \wp(\Theta^+) \rightarrow (\Sigma \times \Sigma) \rightarrow \wp(\Theta^+) \]

\[ F(X)(\sigma, \sigma') = \{ \sigma = \sigma' \mid \sigma \in \bigcup_{\sigma'' \in \Sigma} \{ \sigma \xrightarrow{A} \sigma'' \pi \mid \sigma \xrightarrow{A} \sigma'' \land \sigma'' \pi \in X(\sigma'', \sigma') \} \} \]

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

---

**Minimal path length abstraction**

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

Abstraction:

\[ \alpha \in \Theta^+ \rightarrow \mathbb{N} \]

\[ \alpha(\sigma) \triangleq 0 \]

\[ \alpha(\sigma \xrightarrow{n} \sigma' \pi) \triangleq n + \alpha(\sigma' \pi) \]

\[ \alpha(X) \triangleq \min\{\alpha(\pi) \mid \pi \in X\} \]

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

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

Pointwise extension:

\[ \hat{\alpha} \in (\Sigma \times \Sigma \mapsto \varphi(\Theta^+)) \mapsto (\Sigma \times \Sigma \mapsto N^\infty) \]

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

Pointwise Galois connection:

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

Calcalculational design of the shortest distance algorithm

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

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

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

\[ = \lambda (\sigma, \sigma') \cdot \lambda X \cdot \hat{\alpha}(\lambda(\sigma, \sigma')) \{ \sigma = \sigma' \Rightarrow \{ \sigma \} \ni \bigcup_{\sigma'' \in \Sigma} \{ \sigma \mapsto \sigma'' \pi \mid \sigma \hat{\sim} \sigma'' \wedge \sigma'' \pi \in \}

\[ X(\sigma'', \sigma') \} \{\text{def. } \hat{\alpha}\} \]

\[ = \lambda (\sigma, \sigma') \cdot \lambda X \cdot \hat{\alpha}(\lambda(\sigma, \sigma')) \{ \sigma = \sigma' \Rightarrow \{ \sigma \} \ni \bigcup_{\sigma'' \in \Sigma} \{ \sigma \mapsto \sigma'' \pi \mid \sigma \hat{\sim} \sigma'' \wedge \sigma'' \pi \in X(\sigma'', \sigma') \} \{\text{def. conditional } [\ldots \Rightarrow \ldots \Rightarrow \ldots] \}

\[ = \lambda (\sigma, \sigma') \cdot \lambda X \cdot \hat{\alpha}(\lambda(\sigma, \sigma')) \{ \sigma = \sigma' \Rightarrow \{ \sigma \} \ni \bigcup_{\sigma'' \in \Sigma} \{ \sigma \mapsto \sigma'' \pi \mid \sigma \hat{\sim} \sigma'' \wedge \sigma'' \pi \in X(\sigma'', \sigma') \} \{\text{join preservation in Galois C.}\}

\[ = \lambda (\sigma, \sigma') \cdot \lambda X \cdot \hat{\alpha}(\lambda(\sigma, \sigma')) \{ \sigma = \sigma' \Rightarrow \{ \sigma \} \ni \bigcup_{\sigma'' \in \Sigma} \{ \sigma \mapsto \sigma'' \pi \mid \sigma \hat{\sim} \sigma'' \wedge \sigma'' \pi \in X(\sigma'', \sigma') \} \{\text{join preservation in Galois C.}\}

by defining

\[ G(X)(\sigma, \sigma') = \{ \sigma = \sigma' \Rightarrow \{ n \in X(\sigma'', \sigma') \mid \sigma \hat{\sim} \sigma'' \} \}

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

Shortest distance

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

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

\[ \Delta \triangleq \hat{\alpha}(\Pi) = \hat{\alpha}(1_{\text{fp}} \subseteq F) \]

Calculational design of the shortest distance algorithm

\[ = \lambda (\sigma, \sigma') \cdot \lambda X \cdot \{ \sigma = \sigma' \Rightarrow \{ \sigma \} \ni \bigcup_{\sigma'' \in \Sigma} \{ \sigma \mapsto \sigma'' \pi \mid \sigma \hat{\sim} \sigma'' \wedge \sigma'' \pi \}

\[ X(\sigma'', \sigma') \} \{\text{def. conditional } [\ldots \Rightarrow \ldots \Rightarrow \ldots] \}

\[ = \lambda (\sigma, \sigma') \cdot \lambda X \cdot \{ \sigma = \sigma' \Rightarrow \{ \sigma \} \ni \bigcup_{\sigma'' \in \Sigma} \{ \sigma \mapsto \sigma'' \pi \mid \sigma \hat{\sim} \sigma'' \wedge \sigma'' \pi \}

\[ X(\sigma'', \sigma') \} \{\text{join preservation in Galois C.}\}

\[ = \lambda (\sigma, \sigma') \cdot \lambda X \cdot \{ \sigma = \sigma' \Rightarrow \{ \sigma \} \ni \bigcup_{\sigma'' \in \Sigma} \{ \sigma \mapsto \sigma'' \pi \mid \sigma \hat{\sim} \sigma'' \wedge \sigma'' \pi \}

\[ X(\sigma'', \sigma') \} \{\text{join preservation in Galois C.}\}

\[ \text{by defining}

\[ G(X)(\sigma, \sigma') = \{ \sigma = \sigma' \Rightarrow \{ n \in X(\sigma'', \sigma') \mid \sigma \hat{\sim} \sigma'' \} \}

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

By the fixpoint abstraction theorem

\[ \Delta = \alpha(1fp \subseteq F) = 1fp \subseteq G = \min_{n \in \mathbb{N}} G^n(\lambda, \sigma, \sigma') \cdot + \infty \]

where the iterates are

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

Example III of exact abstractions: semantics

Trace semantics

Initial states

Intermediate states

Final states of the finite traces

Infinite traces

\[ 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ \ldots \ \text{discrete time} \]

Not Floyd-Warshall? Take instead:

\[ \alpha(\sigma) \triangleq 0 \]
\[ \alpha(\sigma \xrightarrow{n} \sigma') \triangleq n \]
\[ \alpha(\pi \sigma \pi') \triangleq \alpha(\pi \sigma) + \alpha(\sigma \pi') \]
Abstraction to denotational/natural semantics

Trace semantics → Denotational semantics → Natural semantics

Abstraction to small-steps operational semantics

Initial states → Transitions → Final states

(Small-Step) Operational Semantics

Abstraction to reachability/invariance

Initial states → Reachable states → Final states

Partial Correctness / Invariance Semantics

Abstraction to Hoare logic

Initial states → Intermediate states → Final states

{P}C{Q} ⇔ {● | ● ∈ P ∧ ... ∈ \llbracket C \rrbracket} ⊆ Q
Verification/static analysis by abstract interpretation

Define the syntax of programs $P \in L$.

Define the concrete semantics of programs:
- $\mathcal{D}[P]$ - concrete semantic domain
- $\forall P \in L : S[P] \in \mathcal{D}[P]$ - concrete semantics

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

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

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

Verification/static analysis by abstract interpretation

Define the abstraction:
- $\langle \mathcal{D}(\mathcal{D}[P]), \subseteq \rangle \xrightarrow{\alpha[P]} \langle \mathcal{A}[P], \subseteq \rangle$

Calculate the abstract semantics:
- $S^\#[P] = \alpha[P](\{S[P]\})$ - exact abstraction
- $S^\#[P] \supseteq \alpha[P](\{S[P]\})$ - approximate abstraction

Soundness (by construction):
- $\forall P \in L : \forall Q \in \mathcal{A} : S^\#[P] \subseteq Q \implies S[P] \in \gamma[P](Q)$
Verification/static analysis by abstract interpretation

Completeness (for exact abstractions only)
\[ \forall P \in \mathcal{L} : \forall Q \in \mathcal{A}[P] : S[P] \in \gamma[P](Q) \Rightarrow S'[P] \sqsubseteq Q \]

Methodology:
- Structural induction on programs \( P \)
- Compositional definition\(^(*)\) of \( \mathcal{A}[P] \) and \( \alpha[P] / \gamma[P] \)
- Fixpoint abstraction/approximation for recursion

Verification for fixpoints is the main problem:
\[ \text{lfp} \subseteq F'[P] \sqsubseteq Q \]


Approximate abstractions

The concrete properties of the standard semantics \( S[P] \) that you want to prove may not always be provable in the abstract:
\[ \forall Q \in \mathcal{A}: S[P] \in \gamma(Q) \iff S[P] \sqsubseteq Q \]
where
\[ \overline{S}[P] \overset{\Delta}{=} \alpha \circ S[P] \circ \gamma \]
Why abstraction may be approximate?

Example
\{ x = y \land 0 \leq x \leq 10 \}
\x := x - y;
\{ x = 0 \land 0 \leq y \leq 10 \}

Interval abstraction:
\{ x \in [0, 10] \land y \in [0, 10] \}
\x := x - y;
\{ x \in [-10, 10] \land y \in [0, 10] \}

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

Refinement: good news

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

It is always possible to refine \( \langle \mathcal{A}, \sqsubseteq \rangle \) into a most abstract more precise abstraction \( \langle \mathcal{A}', \sqsubseteq' \rangle \) such that
\[
\langle g(\mathcal{D}), \sqsubseteq \rangle \xrightarrow[\alpha']{\gamma'} \langle \mathcal{A}', \sqsubseteq' \rangle
\]
and \( \alpha' \circ F = F' \circ \alpha \) with \( \text{lfp } F'[\|P\|] \sqsubseteq' \alpha' \circ \gamma (Q) \)

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

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:

\[
\begin{array}{cccccc}
\sigma_{-2} & \sigma_{-1} & \sigma_0 & \sigma_1 & \sigma_2 & \cdots \sigma_i \\
-2 & -1 & 0 & 1 & 2 & \cdots
\end{array}
\]

Example of intrinsic approximate refinement

Consider universal model-checking abstraction:

\[
MC^\forall_M(\phi) = \alpha^\forall_M([\phi]) = \varphi(\text{Traces}) \rightarrow \varphi(\text{States})
\]

\[
= \{ s \in \text{States} \mid \forall (i, \sigma) \in \text{Traces}_M . (\sigma_i = s) \Rightarrow \langle i, \sigma \rangle \in [\phi] \}
\]

where \( M \) is defined by a transition system

(and dually the existential model-checking abstraction)

Example of intrinsic approximate refinement

Consider the temporal specification language \( \mu^3 \)
(containing LTL, CTL, CTL*, and Kozen’s \( \mu \)-calculus as fragments):

\[
\begin{align*}
\varphi & ::= \sigma_S & S \in \varphi(S) & \text{state predicate} \\
& | \pi_t & t \in \varphi(S \times S) & \text{transition predicate} \\
& | \oplus \varphi_1 & \text{next} \\
& | \varphi_1 \lor \varphi_2 & \text{disjunction} \\
& | \neg \varphi_1 & \text{negation} \\
& | X & X \in X & \text{variable} \\
& | \mu X \cdot \varphi_1 & \text{least fixpoint} \\
& | \nu X \cdot \varphi_1 & \text{greatest fixpoint} \\
& | \forall \varphi_1 : \varphi_2 & \text{universal state closure}
\end{align*}
\]

The abstraction from a set of traces to a trace of sets is sound but \textit{incomplete}, even for finite systems \(^{(1)}\)

\[
\begin{array}{c}
\text{Any refinement of this abstraction is incomplete} \text{ (but to the infinite past/future trace semantics itself)} \quad \text{\((**\)}\)
\end{array}
\]

\(^{(1)}\) 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, now abandoned

Intelligence is needed for refinement:

e.g. human-driven refinement of Astrée


[In]finite abstractions

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

Example:

\[ x=0; \text{while } x<1 \text{ do } x++ \longrightarrow \{\bot, [0,0], [0,1], [-\infty,\infty]\} \]

\[ x=0; \text{while } x<2 \text{ do } x++ \longrightarrow \{\bot, [0,0], [0,1], [0,2], [-\infty,\infty]\} \]

...\]

\[ x=0; \text{while } x<n \text{ do } x++ \longrightarrow \{\bot, [0,0], [0,1], [0,2], [0,3], \ldots, [0,n], [-\infty,\infty]\} \]

...\]


---

No such domain exists for infinitely many programs

1. $\bigcup_{P \in \mathcal{L}} \mathcal{A}[\![P]\!]$ is infinite

Example: $\{\bot, [0,0], [0,1], [0,2], [0,3], \ldots, [0,n], [0,n+1], \ldots, [-\infty,\infty]\}$

2. $\lambda P \in \mathcal{L}. \mathcal{A}[\![P]\!]$ is not computable (for undecidable properties)

$\implies$ finite abstractions will fail infinitely often while infinite abstractions will succeed!

---

Fixpoint approximation in infinite abstractions

Abstract Induction (in non-Noetherian domains)
Convergence acceleration

Infinite iteration

Accelerated iteration with widening (e.g. with a widening based on the derivative as in Newton-Raphson method\(^{(9)}\))

---

Problem with infinite abstractions

For non-Noetherian iterations, we need

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

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

iteration converging

<table>
<thead>
<tr>
<th>( \mathcal{A} )</th>
<th>above the limit</th>
<th>below the limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>below the limit</td>
<td>widening ( \bigtriangledown )</td>
<td>dual narrowing ( \bigtriangleup )</td>
</tr>
<tr>
<td>above the limit</td>
<td>narrowing ( \bigtriangledown )</td>
<td>dual widening ( \bigtriangleup )</td>
</tr>
</tbody>
</table>

---


---

[Semi-]dual abstract induction methods

\( X \in \mathbb{F}(X) \)

\( \mathcal{A} \)

\( \bigtriangleup \)

\( \bigtriangledown \)

\( \top \)

\( \bot \)

(co-induction)

(induction)

(separate from termination conditions)
Examples of widening/narrowing

Abstract induction for intervals:

• a widening \[1,2\]

\[[a_1, b_1] \triangleright [a_2, b_2] = \]

\[\text{if } a_2 < a_1 \text{ then } \rightarrow \text{ else } a_1 \text{ fi,}\]

\[\text{if } b_2 > b_1 \text{ then } \leftarrow \text{ else } b_1 \text{ fi}\]

• a narrowing \[2\]

\[[a_1, b_1] \triangleleft [a_2, b_2] = \]

\[\text{if } a_1 = \rightarrow \text{ then } a_2 \text{ else } \text{MIN } (a_1, a_2),\]

\[\text{if } b_1 = \leftarrow \text{ then } b_2 \text{ else } \text{MAX } (b_1, b_2)\]

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 \[1\]

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

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

Infinitary static analysis with abstract induction

Widening

\[\langle \mathcal{A}, \sqsubseteq \rangle \text{ poset}\]

\[\triangledown \in \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}\]

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 \(\langle x^n \in \mathcal{A}, n \in \mathbb{N}\rangle\), the sequence \(y^0 \triangleq x^0, \ldots, y^{n+1} \triangleq y^n \triangledown x^n, \ldots\) is ultimately stationary \((\exists e \in \mathbb{N}: \forall n \geq e: y^n = y^e)\)

(Note: sound and terminating are independent properties)
Example: (simple) widening for polyhedra

Iterates

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

Widening

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

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

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

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

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

Example: Astrée

Dual narrowing

\[ \langle \mathcal{A}, \subseteq \rangle \text{ poset} \]

\[ \triangle \in \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A} \]

Sound dual narrowing (interpolation):

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

Terminating dual narrowing: for any \( \langle x^n \in \mathcal{A}, n \in \mathbb{N} \rangle \), the sequence \( y^0 \triangleq x^0, \ldots, y^{n+1} \triangleq y^{n} \triangle x^{n}, \ldots \) 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 \( \text{lfp} F \subseteq I \subseteq Q \)

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

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

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

Example: First-order logic + Graig 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;

BOOLEAN INIT;

typedef enum
{
E[6] = E[6];
E[6] = X;
S[6] = S[6];
S[6] = P;
} BOOLEAN;

else

static float

{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 $\nabla$ & $\triangle$)

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

Abstract interpretation

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