On Various Abstract Understandings of Abstract Interpretation

Patrick Cousot

cims.nyu.edu/~pcousot

TASE 2015
The 9th International Symposium on Theoretical Aspects of Software Engineering
September 12—14, 2015 — Nanjing, China

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**

- WCET
- Axiomatic semantics
- Confidentiality analysis
- Program synthesis
- Grammar analysis
- Statistical model-checking
- Invariance proof
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Partial evaluation
- Effect systems
- Denotational semantics
- Theories combination
- Code contracts
- Symbolic execution
- Quantum entanglement detection
- SMT solvers
- Abstraction refinement
- Type inference
- Separation logic
- CEGAR
- Program transformation
- Shape analysis
- Abstract model checking
- Install the model
- Malware detection
- Code refactoring
- WCET
- Axiomatic semantics
- Confidentiality analysis
- Program synthesis
- Grammar analysis
- Statistical model-checking
- Invariance proof
- Probabilistic verification
- Parsing
- Security protocol verification
- Dataflow analysis
- Model checking
- Partial evaluation
- Effect systems
- Denotational semantics
- Theories combination
- Code contracts
- Symbolic execution
- Quantum entanglement detection
- SMT solvers
- Abstraction refinement
- Type inference
- Separation logic
- CEGAR
- Program transformation
- Shape analysis
- Abstract model checking
- Install the model
- Malware detection
- Code refactoring
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

Dataflow analysis
Model checking
Denotational semantics

Abstraction refinement
Type inference

Operational semantics

Database query
Dependence analysis
CEGAR
Program transformation

Invariance proof
Separation logic
Termination proof

Abstract model checking

Trace semantics

Shape analysis

Quantum entanglement detection
SMT solvers
...

Database query
Security protocol verification
...

Practical motivations

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

Theories combination

Ariane 5.01 failure
Patriot failure
Mars orbiter loss
Heartbleed

Boeing 787 Dreamliners contain a potentially catastrophic software bug

A software vulnerability in Boeing’s new 787 Dreamliner jet has the potential to cause pilots to lose control of the aircraft, possibly in mid-flight, Federal Aviation Administration officials warned airlines recently.

The bug—which is either a classic integer overflow or one very much resembling it—resides in one of the electrical systems responsible for generating power, according to a memo the FAA issued last week. The vulnerability, which Boeing reported to the FAA, is triggered when a generator has been running continuously for a little more than eight months. As a result, FAA officials have adopted a new airworthiness directive (AD) that airlines will be required to follow, at least until the underlying flaw is fixed.

"This AD was prompted by the determination that a Model 787 airplane that has been powered continuously for 248 days can lose all alternating current (AC) electrical power due to the generator control units (GCUs) simultaneously going into failsafe mode," the memo stated. "This condition is caused by a software counter internal to the GCUs that will overflow after 248 days of continuous power. We are issuing this AD to prevent loss of all AC electrical power, which could result in loss of control of the airplane."
Informal examples of abstraction

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

Individual heights

min, max

Numerical abstractions in Astrée

Collecting semantics:
\[
\begin{align*}
\text{partial traces} & \\
\text{Intervals:} & \quad x \in [a, b] \\
\text{Octagons:} & \quad \pm x \pm y \leq a \\
\text{Ellipses:} & \quad x^2 + by^2 - axy \leq d \\
\text{Exponentials:} & \quad -a^t \leq y(t) \leq a^t
\end{align*}
\]

Simple congruences:
\[
x = a[b]
\]

An informal introduction to abstract interpretation

1) Define the programming language

Formalize the concrete execution of programs (e.g., transition system)

Trajectory in state space \( \Sigma \)

Space/time trajectory

© P Cousot

II) Define the program properties of interest
Formalize what you are interested to know about program behaviors

III) Define which specification must be checked
Formalize what you are interested to prove about program behaviors

IV) Choose the appropriate abstraction
Abstract away all information on program behaviors irrelevant to the proof

V) Mechanically verify in the abstract
The proof is fully automatic
Soundness of the abstract verification

Never forget any possible case so the abstract proof is correct in the concrete

Forbidden zone

Possible trajectories

Abstraction of the trajectories

Unsound validation: testing

Try a few cases

Forbidden zone

Possible trajectories

Test of a few trajectories

Unsound validation: bounded model-checking

Simulate the beginning of all executions

Forbidden zone

Possible trajectories

Bounded model-checking

Erroneous trajectory abstraction

Unsound validation: static analysis

Many static analysis tools are unsound (e.g. Coverity, etc.) so inconclusive

Forbidden zone

Possible trajectories

Error !!!

Error !!!
Incompleteness

When abstract proofs may fail while concrete proofs would succeed

By soundness an alarm must be raised for this overapproximation!

False alarm

The abstract alarm may correspond to no concrete error (false negative)

False alarm

What to do about false alarms?

- Consider special cases: finite (small) models (model-checking), decidable cases (SMT solvers), human interaction (theorem provers, proof verifiers), ...
- Automatic refinement: inefficient and may not terminate (Gödel, see next slide)
- Domain-specific abstraction:
  - Adapt the abstraction to the programming paradigms typically used in given domain-specific applications
  - e.g. synchronous control/command: no recursion, simple memory allocation, maximum execution time, etc.
In general refinement does not terminate

- Example: filter invariant abstraction:

2nd order filter:

Unstable polyhedral abstraction:

Counter-example guided refinement will indefinitely add missing points according to the execution trace:

Stable ellipsoidal abstraction:

Abstract Interpretation

Abstract interpretation is all about:

- Soundness
- Induction

Properties and their Abstractions

A very short more formal 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 $x \in P$

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

Example of Property

- Odd natural numbers $O = \{ 1, 3, 5, 7, \ldots \}$
- $x$ is odd
  $x \in O$
- “$x$ has property $O$”
- $x$ is 2
  $x \in \{2\}$
- the strongest property of 2
  $\{2\}$
- 2 and 4 are even
  $\{2, 4\} \subseteq \{0, 2, 4, 6, 8, \ldots \}$

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**

Example of Abstract Properties

$\mathcal{A} = \{ \bot, O, E, T \}$

$T = \mathbb{N}$

$O \subseteq E$

$\bot = \emptyset$
Concretization

\[ \gamma \in \mathcal{A} \rightarrow \wp(\mathcal{D}) \]

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

\( \gamma \) is \textit{increasing} (so \( \subseteq \) abstracts \( \subseteq \))

Best abstraction

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

- it is \textit{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

Example of Concretization

\[ \{0,1,2,3,4,5,...\} = \mathbb{N} \]

\[ \{0,2,4,...\} \quad \gamma \quad O \]

\[ \{1,3,5,...\} \quad \gamma \quad E \quad \gamma \quad \emptyset \]

Galois connection

Any \( P \in \wp(\mathcal{D}) \) has a (unique) \textit{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

\[ \langle \wp(\mathcal{D}), \subseteq \rangle \leftarrow[\gamma]_{\alpha} \langle \mathcal{A}, \subseteq \rangle \]

\( \Rightarrow \) : over-approximation

\( \Leftarrow \) : best abstraction
Examples

Needness/strictness analysis (80's)

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

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)

Sound semantics abstraction

program $P \in \mathcal{L}$ 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

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

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

i.e. $S[P] \in \gamma(S[P])$, $P$ has abstract property $S[P]$
Best abstract semantics

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

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

Proof:

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

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

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,…)
- induction (transitional, structural, segmentation [POPL 2012])
- presentations (fixpoints, equations, constraints, rules [CAV 1995])

Example: functional connector

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

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

\( \xrightarrow{\alpha} \) is a called a Galois connector.

Simple example

\[
F(x_2) = \{x+2 \mid x \in x_2\}
\]

\[
\begin{align*}
\alpha \circ F \circ \gamma(\bot) &= \alpha(\{x+2 \mid x \in \emptyset\}) = \alpha(\emptyset) = \bot \\
\alpha \circ F \circ \gamma(O) &= \alpha(\{x+2 \mid x \in \gamma(O)\}) = \alpha(\{3,5,7,\ldots\}) = O
\end{align*}
\]
Fixpoints of increasing functions (Tarski)

Another fixpoint at $+\infty \uparrow$

Best abstraction (completeness case)

if $\alpha \circ F = \exists F \circ \alpha$ then $\exists F = \alpha \circ F \circ \gamma$ and $\alpha(lfp F) = lfp \exists F$

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

Best approximation (incompleteness case)

if $\exists F = \alpha \circ F \circ \gamma$ but $\alpha \circ F \sqsubseteq \exists F \circ \alpha$ then $\alpha(lfp F) \sqsubseteq lfp \exists F$

e.g. static analysis of infinite state systems

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

Simple Example

0: $x := 1$
1: while $x < 10$ do
2: $x := x + 2$
3: od;
4:

$(x_0, \ldots, x_4) = F(x_0, \ldots, x_4)$

$x_0 = \{\ldots -2, -1, 0, 1, 2, \ldots\}$
$x_1 = \{1\}$
$x_2 = (x_1 \cup x_3) \cap \{\ldots, -8, -9\}$
$x_3 = \{\text{x+2} \mid x \in x_2\}$
$x_4 = (x_1 \cup x_3) \cap \{10, 11, \ldots\}$

$(x_0, \ldots, x_4) = \exists F(x_0, \ldots, x_4)$

$x_0 = T$
$x_1 = O$
$x_2 = (x_1 \cup x_3) \cap T$
$x_3 = x_2 \oplus E$
$x_4 = (x_1 \cup x_3) \cap T$

Iterative resolution

$x_0 = T$
$x_1 = O$
$x_2 = (x_1 \cup x_3) \cap T$
$x_3 = x_2 \oplus E$
$x_4 = (x_1 \cup x_3) \cap T$

<table>
<thead>
<tr>
<th>iteration</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0$</td>
<td>$\perp$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td>$T$</td>
<td></td>
</tr>
<tr>
<td>$x_1$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$O$</td>
<td>$O$</td>
<td>$O$</td>
<td>$O$</td>
<td>$O$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$O$</td>
<td>$O$</td>
<td>$O$</td>
<td>$O$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$O$</td>
<td>$O$</td>
<td>$O$</td>
</tr>
<tr>
<td>$x_4$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td>$O$</td>
<td>$O$</td>
</tr>
</tbody>
</table>
Exact fixpoint abstraction

Abstract domain

Concrete domain

\[ \alpha \circ F = F^\# \circ \alpha \Rightarrow \alpha(lfp F) = lfp F^\# \]

Approximate fixpoint abstraction

Abstract domain

Concrete domain

\[ lfp F \subseteq \gamma(lfp F^\#) \]

Duality

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

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

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

Why abstracting properties of semantics, not semantics or models?
Understandings of Abstract Interpretation

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 \(\wp(\wp(\mathcal{P}))) = \wp(\wp(\wp(\mathcal{P}))))\)

Collecting semantics \(C[P] = \{S[P]\} \in \wp(\mathcal{D}) = \wp(\wp(\mathcal{P})))\)

How to abstract the standard semantics?

The join abstraction:

\[
\begin{align*}
\wp(\wp(\mathcal{P})), \subseteq & \xrightarrow{\gamma_U} \wp(\mathcal{P}), \subseteq \\
\alpha_U(X) & \triangleq \bigcup X \\
\gamma_U(Y) & \triangleq \wp(Y)
\end{align*}
\]

Join abstraction of the collecting semantics:

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

(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:

\[
\alpha_U(P) = \]

“Always terminate, either with 0 or 1”

\(\alpha_U(P) \in \wp(\mathcal{P})\)

\(P \in \wp(\wp(\mathcal{P}))\)

always the same result

results can be different
Limitations of the union abstraction

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

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

Exact abstractions

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

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

where

$$S\llbracket P \rrbracket \triangleq \alpha \circ S\llbracket P \rrbracket \circ \gamma$$

Example III of exact abstractions: semantics

Trace semantics

Initial states
Intermediate states
Final states of the finite traces

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

Infinite traces

Abstraction to denotational/natural semantics

Initial states
Intermediate states
Final states of finite traces

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
Approximate abstractions

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

$$\forall Q \in S: S[P] \in \gamma(Q) \iff S[P] \subseteq Q$$

where

$$S[P] \triangleq \alpha \circ S[P] \circ \gamma$$
Why abstraction may be approximate?

Example

\{ x = y \land 0 \leq x \leq 10 \}

\begin{align*}
\ 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)

[In]finite abstractions

Given a program \( P \) and a program property \( Q \) which holds (i.e. \( \text{lpf} \ F \left[ [P] \right] \in Q \)) there exists a most abstract abstraction in a finite domain \( \mathcal{A} \left[ [P] \right] \) to prove it \( ^\text{(4)} \)

Example:

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

\( ^\text{(4)} \) Patrick Cousot: Partial Completeness of Abstract Fixpoint Checking. SARA 2000: 1-25

Finite versus infinite abstractions

No such domain exists for infinitely many programs

1. \( \bigcup_{P \in \mathcal{L}} \mathcal{A} \left[ [P] \right] \) is infinite

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

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

\( \longrightarrow \) 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(*)

Problem with infinite abstractions

For non-Noetherian iterations, we need

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

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

<table>
<thead>
<tr>
<th>Iteration starting from</th>
<th>Iteration converging</th>
</tr>
</thead>
<tbody>
<tr>
<td>below the limit</td>
<td>widening ( \nabla )</td>
</tr>
<tr>
<td></td>
<td>dual narrowing ( \tilde{\nabla} )</td>
</tr>
<tr>
<td>above the limit</td>
<td>narrowing ( \Delta )</td>
</tr>
<tr>
<td></td>
<td>dual widening ( \tilde{\Delta} )</td>
</tr>
</tbody>
</table>

Examples of widening/narrowing

Abstract induction for intervals:

- a widening \([1,2]\)

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

- a narrowing \([2]\)

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

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 \([1]\)

\[ \forall x \in L_1, \perp \triangleright V_S(f) x = x \triangleright V_S(f) \perp = x \]

\[ [v_1, v_2] \triangleright V_S(f) [u_1, u_2] = \begin{cases} \text{if } 0 \leq v_2 < l_1 \text{ then } v_1 \text{ else } l_1 \triangleright \ell, \\ \text{if } u_1 < u_2 \leq 0 \text{ then } v_1 \text{ else } b \triangleright v_1 \ell \end{cases} \]

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)

---

2. Patrick Cousot, Radhia Cousot: Abstract Interpretation: A United Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL 1977: 238-252
Infinitary static analysis with abstract induction

Example: (simple) widening for polyhedra

Iterates

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

Widening

\[
\mathcal{F}^n \sqcup \mathcal{F}(\mathcal{F}^n)
\]

Widening

\[
\mathcal{F}^n \sqcup \mathcal{F}(\mathcal{F}^n)
\]

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

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

Sound widening (upper bound):

\[
\forall x, y \in \mathcal{A}: x \sqsubseteq x \bigtriangledown y \land y \sqsubseteq x \bigtriangledown y
\]

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

(Note: sound and terminating are independent properties)

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

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

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

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

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

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

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

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

\[ \text{Sound dual narrowing (interpolation):} \]

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

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

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

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

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

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


Astrée

Commercially available: [www.absint.com/astree/](http://www.absint.com/astree/)

Effectively used in production to qualify truly large and complex software in transportation, communications, medicine, etc.
Example of domain-specific abstraction: ellipses

typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float P, X;

void filter () {
    static float E[2], S[2];
    if (INIT) { S[0] = X; P = X; E[0] = X; }
    else { P = ((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4)) + (S[0] * 1.5)) - (S[1] * 0.7)); }
E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
/* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}

void main () { X = 0.2 * X + 5; INIT = TRUE;
while (1) {
    X = 0.9 * X + 35; /* simulated filter input */
    filter (); INIT = FALSE; }
}

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. 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

Example III: CodeHawk

- http://www.kestreltechnology.com
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

Abstract interpretation

Intellectual tool (not to be confused with its specific application to iterative static analysis with $\nabla$ & $\Delta$)

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

The End
The End
Thank You