Analysis and Verification of Software

Abstract Interpretation: an Overview

prof. Agostino Cortesi
Ca’ Foscari University, Venice, Italy
Overview

The goal: Define and prove automatically a property of the possible behaviors of a complex computer program, e.g. a security property.

The key-idea: The calculus can be done on an abstraction of these behaviors dealing only with those elements of the behaviors related to the considered property.

The theory: Abstract Interpretation [P.Cousot]
Abstract Interpretation

The central idea in abstract interpretation is to construct two different meanings of a programming language where the first gives the usual meaning of programs in the language, and the other can be used to answer certain questions about the runtime behavior of programs in the language.

The standard meaning of programs can typically be described by their input-output function, and the standard interpretation will then be a function which maps programs to their input-output functions.

The abstract meaning will be described by a function on to mathematical objects that represent the property be analyzed.

The correctness (or soundness) of this approach to program analysis can be established by proving a relationship between these two interpretations.
Abstract Interpretation

Figure by M. Rosendahl
Abstract Interpretation

- A theory of semantics approximation for computing conservative over-approximations of dynamic properties of programs.

- Successfully applied to infer run-time properties useful for a variety of tasks:
  - debugging (e.g. type inference)
  - code optimization (e.g., compile-time garbage collection)
  - program transformation (e.g., partial evaluation, parallelization)
  - program correctness proofs (e.g., safety, termination, proof of absence of run-time errors, semantic tattooing/watermarking).

- Still a large variety of tasks in the software engineering process that could greatly benefit from it, because of its firm theoretical foundations and mechanical nature.
Trajectories

The concrete semantics of a program formalizes the set of all its possible executions in all possible execution environments.
Undecidability

- The concrete semantics of a program is not computable (unless the program is trivial);
- All non trivial properties on the concrete program semantics are either undecidable or "very hard" to solve (either NP complete or co-NP complete).
Safety properties

Safety properties of a program express that no possible execution in any possible execution environment can reach an erroneous state.

We should prove that the intersection of the program concrete semantics and the forbidden zone is empty. This is an undecidable problem, as the concrete semantics is not computable.
What about testing?

Testing is not a good solution:

- it considers only a subset of the possible executions;
- it does not provide a correctness proof;
- the issue is "coverage"!

Figure by P. Cousot
Over-Approximation of the Concrete Semantics

Abstract Interpretation consists in considering an abstract semantics, that is an over-approximation of the concrete semantics of the program;

- The abstract semantics covers all possible concrete cases;
- It guarantees correctness: if it is safe (it does not intersect the forbidden zone) then so is the concrete semantics.
Correctness

Correctness in this approach is a MUST. The abstract semantics should not disregard any trajectory!
Correctness

Correctness in this approach is a MUST. The abstract semantics should not disregard any trajectory!!!
Imprecision

The over-approximation may result into a lack of precision, i.e. on generating false alarms.

Figure by P. Cousot
The starting point is a formal description of an (operational) concrete semantics

Start from a standard operational semantics that describes formally:

- states: data values of program variables,
- transitions: elementary computation steps;
- traces: sequences (possibly infinite) of states corresponding to executions described by transitions
Concrete Semantics: small-steps transitions

Figure by P.Cousot
Collecting Semantics

The collecting semantics is the set of observable behaviors in the operational semantics. It is the starting point of any analysis design.

- The set of all descendants of the initial state
- The set of all descendants of the initial state that can reach a final state
- The set of all finite traces from the initial state
- The set of all finite and infinite traces from the initial state etc.
Collecting semantics: trajectories of sets of states

Figure by P. Cousot

Università Ca’ Foscari Venezia, Italy
Concrete Semantics: trajectories of sets of states

\[ F(X) = \mathcal{I} \cup \{ s' \mid \exists s \in X : s \xrightarrow{t} s' \} \]

\[ \mathcal{R} = \operatorname{lfp}_\subseteq F \]

\[ = \bigcup_{n=0}^{+\infty} F^n(\emptyset) \quad \text{where} \quad f^0(x) = x \]

\[ f^{n+1}(x) = f(f^n(x)) \]
Abstracting Data and Operations

In order to get an abstract semantics we need:

- abstract program data,
- abstract program basic operations;
- abstract program control (iteration, procedure, concurrency, . . . ;

The abstraction can be parameterized, in order to tune it with respect to different application domains.
Abstracting properties

- Choose an abstract domain, a complete lattice whose elements capture the information to be analyzed.

- An abstraction function $\alpha$ which maps a set of concrete objects to its most accurate representation in the abstract domain;

- The inverse concretization function $\gamma$ which maps an abstract object to the concrete ones it represents;

- Forget no concrete objects: (abstraction from above) $S \subseteq \gamma(\alpha(S))$
Lattices

The idea behind a partial order is "a \( \subseteq \) b" means "a is more informative than b".

A lattice \( L \) is a partially ordered set \( (L, \subseteq) \) with:

- Least upper bounds \( \sqcup \) and greatest lower bounds \( \sqcap \) operators
- A least element \( \bot \)
- A greatest element \( \top \)

\( L \) is complete if for each subset of \( L \) there exist a (unique) least upper bound.
Abstracting by Sign

We may in some situations be able to predict whether the result of an expression is positive or negative by only using the sign of the constants in the expression.

As an example consider the expression $-413 \times (2571 + 879)$. We can deduce that the result is negative without actually performing the addition and the multiplication since we know that adding two positive numbers gives a positive result and that multiplying a negative and a positive number gives a negative result.
Abstracting by Sign

In the sign interpretation we will operate with four different values: zero, pos, neg, num, where zero indicates that the number is zero, pos that the number is positive, neg that the number is negative, and num that we don’t know. We will call the set of these values Sign.

The rule-of-sign interpretation of addition and multiplication can then be specified in two tables. These interpretations may be viewed as "abstract" additions and multiplications.

⊕ : Sign × Sign → Sign

<table>
<thead>
<tr>
<th>⊕</th>
<th>zero</th>
<th>pos</th>
<th>neg</th>
<th>num</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>zero</td>
<td>pos</td>
<td>neg</td>
<td>num</td>
</tr>
<tr>
<td>pos</td>
<td>pos</td>
<td>pos</td>
<td>num</td>
<td>num</td>
</tr>
<tr>
<td>neg</td>
<td>neg</td>
<td>num</td>
<td>neg</td>
<td>num</td>
</tr>
<tr>
<td>num</td>
<td>num</td>
<td>num</td>
<td>num</td>
<td>num</td>
</tr>
</tbody>
</table>

⊗ : Sign × Sign → Sign

<table>
<thead>
<tr>
<th>⊗</th>
<th>zero</th>
<th>pos</th>
<th>neg</th>
<th>num</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>zero</td>
<td>zero</td>
<td>zero</td>
<td>zero</td>
</tr>
<tr>
<td>pos</td>
<td>zero</td>
<td>pos</td>
<td>neg</td>
<td>num</td>
</tr>
<tr>
<td>neg</td>
<td>zero</td>
<td>neg</td>
<td>pos</td>
<td>num</td>
</tr>
<tr>
<td>num</td>
<td>zero</td>
<td>num</td>
<td>num</td>
<td>num</td>
</tr>
</tbody>
</table>
Abstract domains for approximating an (infinite) set of values of two variables

\[ \{ \ldots, (19, 77), \ldots, (20, 03), \ldots \} \]

Figure by P. Cousot
The Domain of Signs

Figure by P. Cousot
Operations on Sign

\[(3, 1) + (2, 3) = (5, 4)\]
\[(\text{pos, pos}) \oplus (\text{pos, pos}) = (\text{pos, pos})\]
\[(0, 3) \ast (-2, -4) = (0, -12)\]
\[(\text{zero, pos}) \otimes (\text{neg, neg}) = (\text{zero, neg})\]
The Domain of Intervals
Abstracting by Intervals

\[ \{x : [1, 99], y : [2, 77]\} \]

Figure by P. Cousot
Concretization of Intervals

\{x : [1, 99], y : [2, 77]\}

Figure by P. Cousot
Monotonicity of the Abstraction

\[ X \subseteq Y \Rightarrow \alpha(X) \subseteq \alpha(Y) \]

Figure by P. Cousot
Monotonicity of the Concretization

\[ \{x : [33, 89], y : [48, 61] \} \subseteq \{x : [1, 99], y : [2, 90] \} \]

\[ X \subseteq Y \Rightarrow \gamma(X) \subseteq \gamma(Y) \]

Figure by P. Cousot
The composition $\gamma \circ \alpha$ is extensive

If we apply $\alpha$ first, and then $\gamma$, we over-approximate the initial set.

$X \subseteq \gamma \circ \alpha(X)$
The composition $\alpha \circ \gamma$ is reductive

If we apply $\gamma$ first, and then $\alpha$, we don’t loose precision.

$\{x : [1, 99], y : [2, 77]\} = \sqsubseteq \{x : [1, 99], y : [2, 77]\}$

$\alpha \circ \gamma(Y) = \sqsubseteq Y$
Galois Connection

A Galois Connection between a concrete p.o. and an abstract p.o arises when:

- The Concretization and Abstraction functions are monotone
- The composition $\gamma \circ \alpha$ is extensive
- The composition $\alpha \circ \gamma$ is reductive


Galois Connection

A Galois Connection between a concrete p.o. and an abstract p.o arises when:

- The Concretization and Abstraction functions are monotone
- The composition \( \gamma \circ \alpha \) is extensive
- The composition \( \alpha \circ \gamma \) is reductive

\[
\langle D, \subseteq \rangle \xrightarrow[\alpha \circ \gamma]{\gamma} \langle \overline{D}, \sqsubseteq \rangle
\]

iff

\[
\forall x, y \in D : x \subseteq y \Rightarrow \alpha(x) \sqsubseteq \alpha(y)
\]

\[
\wedge \forall x, \overline{y} \in \overline{D} : x \sqsubseteq \overline{y} \Rightarrow \gamma(x) \subseteq \gamma(\overline{y})
\]

\[
\wedge \forall x \in D : x \subseteq \gamma(\alpha(x))
\]

\[
\wedge \forall \overline{y} \in \overline{D} : \alpha(\gamma(\overline{y})) \sqsubseteq \overline{x}
\]

iff

\[
\forall x \in D, \overline{y} \in \overline{D} : \alpha(x) \sqsubseteq \overline{y} \iff x \subseteq \gamma(y)
\]
The Domain of Octagons

Figure by P. Cousot

\[
\begin{align*}
1 & \leq x \leq 9 \\
x + y & \leq 77 \\
1 & \leq y \leq 9 \\
x - y & \leq 99
\end{align*}
\]
The Domain of Polyhedra

\[
\begin{align*}
19x + 77y & \leq 2004 \\
20x + 03y & \geq 0
\end{align*}
\]

Figure by P.Cousot
Abstract Functions

There is always an optimal abstraction of the concrete function $F$, defined by

$$F^\# = \alpha \circ F \circ \gamma.$$  

However, for the correctness of the analysis it is sufficient that

$$\forall a \in D^\# : \gamma(F^\#(a)) \succeq F(\gamma(a)).$$
Fixpoint Theorems

- **Knaster-Tarski theorem**: If $F : L \to L$ is monotone and $L$ is a complete lattice, the set of fixpoints of $F$ is also a complete lattice.

- **Kleene theorem**: If $F : L \to L$ is monotone, $L$ is a complete lattice and $F$ preserves all least upper bounds then $\text{lfp}(F)$ is the limit of the sequence:

\[
\begin{align*}
F_0 &= \bot \\
F_{n+1} &= F(F_n)
\end{align*}
\]
Approximate Semantics

\[ \alpha(\text{lfp } F') \sqsubseteq \text{lfp } F'\]

Figure by P. Cousot
Widening and Narrowing

Figure by A. Venet
Convergence Acceleration by Widening

A widening operator on a partial order \( D \) is binary operator \( \nabla : D \to D \) such that:

- It is an **upper bound operator**: 

  \[
  \forall x, y \in D : x \subseteq x \nabla y, y \subseteq x \nabla y
  \]

- It enforces convergence:

  for all increasing chains \( x_0, x_1, \ldots \), the chain defined by \( y_0 = x_0, y_{i+1} = y_i \nabla x_{i+1} \) is not strictly increasing (i.e. it converges after a finite number of steps).
Recovering Accuracy by Narrowing

A narrowing operator on a partial order \( D \) is a binary operator \( \triangle : D \rightarrow D \) such that:

- It is an **abstract intersection operator**:

\[
\forall x, y \in D : x \sqcap y \subseteq x \triangle y
\]

- It **enforces convergence**:

  for all decreasing chains \( x_0, x_1, \ldots \), the chain defined by \( y_0 = x_0 \), \( y_{i+1} = y_i \triangle x_{i+1} \) is not strictly decreasing, i.e. it converges after a finite number of steps.
Widening on the interval Domain (threshold)

Let $k$ be a fixed positive integer constant.

\[
\bot \nabla_k x = x \\
x \nabla_k \bot = x \\
[l_0, u_0] \nabla_k [l_1, u_1] = \begin{cases} 
\min(l_0, l_1) & \text{if } \min(l_0, l_1) > -k, \\
\max(u_0, u_1) & \text{if } \max(u_0, u_1) < k, \\
-\infty & \text{else.}
\end{cases}
\]

Observe that for all $k$, $\nabla_k$ is commutative, associative, and order-preserving.

However, it is not reflexive. For instance, if $k = 7$ we get:

\[[-8, 4] \nabla [-8, 4] = [-\infty, 4]\]
Example: Convergence Acceleration by Widening

Figure by P. Cousot
Example: Convergence Acceleration by Widening!

Figure by P. Cousot
Example: Convergence Acceleration by Widening !!
Example: Convergence Acceleration by Widening !!!
Refining the Abstract Semantics

Figure by P.Cousot
Refining the Abstract Semantics: Partitioning

Figure by P. Cousot
Refining the Abstract Semantics: Partitioning

Figure by P. Cousot
Overall Architecture

Concrete Semantics

Collecting Semantics

Partitioning

Abstract Semantics

Iterative Resolution Algorithms

Tuners

Abstract Domain

Abstract Domain

Figure by A. Venet
References

- Home page of Patrick Cousot:
  
  http://www.di.ens.fr/~cousot/

- My papers:
  
  http://www.dsi.unive.it/~cortesi/publ.htm