Software Verification

Sebastian Nanz

Lecture 10: Abstract Interpretation
Abstract Interpretation

Introduction
In the past lectures we have introduced a particular style of program analysis: data flow analysis.

For these types of analyses, and others, a main concern is correctness: how do we know that a particular analysis produces sound results (does not forget possible errors)?

In the following we discuss abstract interpretation, a general framework for describing program analyses and reasoning about their correctness.
Main ideas: Concrete computations

- An ordinary program describes computations in some **concrete domain** of values.
  - **Example**: program states that record the integer value of every program variable.

\[ \sigma \in \text{State} = \text{Var} \rightarrow \mathbb{Z} \]

- Possible computations can be described by the **concrete semantics** of the programming language used.
Main ideas: Abstract computations

- Abstract interpretation of a program describes computation in a different, abstract domain.

  **Example:** program states that only record a specific property of integers, instead of their value: their sign, whether they are even/odd, or contained in \([-32768, 32767]\) etc.

\[\sigma \in \text{AbstractState} = \text{Var} \rightarrow \{\text{even, odd}\}\]

- In order to obtain abstract computations, an abstract semantics for the programming language has to be defined.
- Abstract interpretation provides a framework for proving that the abstract semantics is sound with respect to the concrete semantics.
The collecting semantics

We assume the state of a program to be modeled as:

\[\sigma \in \text{State} = \text{Var} \rightarrow \mathbb{Z}\]

We will use the following notation for function update:

\[\sigma[x \mapsto k](y) = \begin{cases} k & \text{if } x = y \\ \sigma(y) & \text{otherwise} \end{cases}\]

We construct the collecting semantics as a function which gives for every program label the set of all possible states.

\[C : \text{Labels} \rightarrow \mathcal{P}(\text{State})\]
Rules of the collecting semantics

\[ I \downarrow \]
\[ [x := e] \downarrow \]
\[ I' \]

\[ I \downarrow \]
\[ [b] \xrightarrow{I_{false}} \downarrow \]
\[ I_{false} \]
\[ [b] \xrightarrow{I_{true}} \downarrow \]
\[ I_{true} \]
\[ I_1 \quad I_2 \]
\[ l' \]

\[ C_{l} = \{\sigma[x \mapsto n] \mid \sigma \in C_{l} \text{ and } C[e]\sigma = n\}\]

\[ C_{ltrue} = \{\sigma \mid \sigma \in C_{l} \text{ and } C[b]\sigma = true\}\]

\[ C_{lfalse} = \{\sigma \mid \sigma \in C_{l} \text{ and } C[b]\sigma = false\}\]

\[ C_{l} = C_{l1} \cup C_{l2} \]

**Note:** In difference to the lecture on program analysis, labels are not on blocks, but on edges.
Example: Collecting semantics

Assume $x > 0$.

\[
\begin{align*}
C_1 &= \{\sigma \mid \sigma(x) > 0\} \\
C_2 &= \{\sigma[y \mapsto 1] \mid \sigma \in C_1\} \cup \{\sigma[x \mapsto \sigma(x) - 1] \mid \sigma \in C_4\} \\
C_3 &= C_2 \cap \{\sigma \mid \sigma(x) \neq 0\} \\
C_4 &= \{\sigma[y \mapsto \sigma(x) \cdot \sigma(y)] \mid \sigma \in C_3\} \\
C_5 &= C_2 \cap \{\sigma \mid \sigma(x) = 0\}
\end{align*}
\]
Solving the equations

- The equation system we obtain has variables $C_1, ..., C_5$ which are interpreted over the complete lattice $\mathcal{P}(\text{State})$.
- We can express the equation system as a monotone function $F : \mathcal{P}(\text{State})^5 \rightarrow \mathcal{P}(\text{State})^5$
  \[
  F(C_1, ..., C_5) = (\{\sigma \mid \sigma(x) > 0\}, ..., C_2 \cap \{\sigma \mid \sigma(x) = 0\})
  \]
- Using Tarski's Fixed Point Theorem, we know that a least fixed point exists.
- We have seen: The least fixed point can be computed by repeatedly applying $F$, starting with the bottom element $\perp = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ of the complete lattice until stabilization.

\[
F(\perp) \subseteq F(F(\perp)) \subseteq ... \subseteq F^n(\perp) = F^{n+1}(\perp)
\]
Example: Fixed Point Computation

\[
\text{Example: Fixed Point Computation}
\]

\[
\begin{array}{c}
\downarrow^1 \varnothing \{[x \mapsto m, y \mapsto n] \mid m > 0\} \\
[y := 1] \\
\downarrow^2 \varnothing \{[x \mapsto m, y \mapsto 1] \mid m > 0\} \cup \{[x \mapsto m-1, y \mapsto m] \mid m > 0\} \\
[x \neq 0] \rightarrow \varnothing \{[x \mapsto 0, y \mapsto m] \mid m > 0\} \quad \ldots \text{ etc.} \\
\downarrow^3 \varnothing \{[x \mapsto m, y \mapsto 1] \mid m > 0\} \\
[y := x \cdot y] \\
\downarrow^4 \varnothing \{[x \mapsto m, y \mapsto m] \mid m > 0\} \\
[x := x - 1]
\end{array}
\]

\[
C_1 = \{\sigma \mid \sigma(x) > 0\} \\
C_2 = \{\sigma[y \mapsto 1] \mid \sigma \in C_1\} \cup \{\sigma[x \mapsto \sigma(x) - 1] \mid \sigma \in C_4\} \\
C_3 = C_2 \cap \{\sigma \mid \sigma(x) \neq 0\} \\
C_4 = \{\sigma[y \mapsto \sigma(x) \cdot \sigma(y)] \mid \sigma \in C_3\} \\
C_5 = C_2 \cap \{\sigma \mid \sigma(x) = 0\}
\]
Domain for Sign Analysis

We want to focus on the sign of integers, using the domain

\[ \sigma \in \text{AbstractState} = \text{Var} \rightarrow \text{Signs} \]

where Signs is the following structure:

- \( \top \) represents all integers
- + the positive integers
- - the negative integers
- 0 the set \( \{0\} \)
- \( \bot \) the empty set

How is such a structure called?
A complete lattice
Example: Sign Analysis

Assume $x > 0$. Use the abstract domain for sign analysis.

\[ A_1 = [x \mapsto +, y \mapsto T] \]

\[ A_2 = A_1[y \mapsto +] \sqcup \]

\[ A_4[x \mapsto A_4(x) \ominus +] \]

\[ A_3 = A_2 \]

\[ A_4 = A_3[y \mapsto A_3(x) \otimes A_3(y)] \]

\[ A_5 = A_2 \sqcap [x \mapsto 0, y \mapsto T] \]
Abstract Interpretation

Foundations
Introductory example: Expressions

A little language of expressions

Syntax
\[ e ::= n \mid e \cdot e \]

Concrete semantics
\[ C[n] = n \]
\[ C[e \cdot e] = C[e] \cdot C[e] \]

Example
\[ C[-3 \cdot 2 \cdot -5] = C[-3 \cdot 2] \cdot C[-5] = C[-3 \cdot 2] \cdot (-5) = ... = 30 \]
Introductory example: Abstraction

Assume that we are not interested in the value of an expression but only in its **sign**:

- Negative:  - 
- Zero:       0 
- Positive:   + 

**Abstract semantics**

\[ A[n] = \text{sign}(n) \]

\[ A[e \times e] = A[e] \otimes A[e] \]

**Example**

\[ A[-3 \times 2 \times -5] = A[-3 \times 2] \otimes A[-5] = A[-3 \times 2] \otimes (-) = \ldots = \]

\[ = (-) \otimes (+) \otimes (-) = (+) \]
Introductory example: Soundness

- We want to express that the abstract semantics correctly describes the sign of a corresponding concrete computation.
- For this we first link each concrete value to an abstract value:

  **Representation function**

\[ \beta : \mathbb{Z} \rightarrow \{-, 0, +\} \]

\[ \beta(n) = \begin{cases} 
  - & \text{if } n < 0 \\
  0 & \text{if } n = 0 \\
  + & \text{if } n > 0 
\end{cases} \]
Introductory example: Soundness

Conversely, we can also link abstract values to the set of concrete values they describe:

**Concretization function**

\[
\gamma : \{-, 0, +\} \rightarrow \mathcal{P}(\mathbb{Z})
\]

\[
\gamma(s) = \begin{cases} 
\{n \mid n < 0\} & \text{if } s = - \\
\{0\} & \text{if } s = 0 \\
\{n \mid n > 0\} & \text{if } s = + 
\end{cases}
\]

**Soundness** then describes intuitively that the concrete value of an expression is described by its abstract value:

\[
\forall e. C[e] \subseteq \gamma(A[e])
\]
Extending the language

Syntax
\[ e ::= n \mid e * e \mid e + e \mid -e \]

Abstract semantics
\[ A[n] = \text{sign}(n) \]
\[ A[-e] = \oplus A[e] \]

Observation: The abstract domain \{-,0,+\} is not closed under the interpretation of addition.
Extending the abstract domain

We have to introduce an additional abstract value:

⊤  "top" - (any value)

<table>
<thead>
<tr>
<th>⊕</th>
<th>-</th>
<th>0</th>
<th>+</th>
<th>⊤</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>⊤</td>
<td>⊤</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>+</td>
<td>⊤</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>⊤</td>
<td></td>
</tr>
<tr>
<td>⊤</td>
<td></td>
<td></td>
<td></td>
<td>⊤</td>
</tr>
</tbody>
</table>
The new abstract domain

We can extend the concretization function to the new abstract domain \{-,0,+, \top, \bot\} (add \bot for completeness):

\[ \gamma(\top) = \mathbb{Z} \quad \gamma(\bot) = \emptyset \]

We obtain the following structure when drawing the partial order induced by

\[ a \leq b \text{ iff } \gamma(a) \subseteq \gamma(b) \]

How is such a structure called?
A complete lattice
Construction of complete lattices

- If we know some complete lattices, we can construct new ones by combining them.
- Such constructions become important when designing new analyses with complex analysis domains.

**Example**: Total function space

Let \((D_1, \sqsubseteq_1)\) be a complete lattice and let \(S\) be a set. Then \((D, \sqsubseteq)\), defined as follows, is a complete lattice:

- \(D = S \rightarrow D_1\) ("space of total functions")
- \(f \sqsubseteq f' \iff \forall s \in S : f(s) \sqsubseteq_1 f'(s)\) ("point-wise ordering")
The framework of abstract interpretation

- Starting from a concrete domain $C$, define an abstract domain $(A, \sqsubseteq)$, which must be a complete lattice.
- Define a representation function $\beta$ that maps a concrete value to its best abstract value:
  \[ \beta : C \rightarrow A \]
- From this we can derive the concretization function $\gamma$
  \[ \gamma : A \rightarrow \mathcal{P}(C) \]
  \[ \gamma(a) = \{ c \in C \mid \beta(c) \sqsubseteq a \} \]
- and abstraction function $\alpha$ for sets of concrete values
  \[ \alpha : \mathcal{P}(C) \rightarrow A \]
  \[ \alpha(C) = \bigcup \{ \beta(c) \mid c \in C \} \]
**Galois connections**

- The following properties of $\alpha$ and $\gamma$ hold:

  **Monotonicity**
  
  (1) $\alpha$ and $\gamma$ are monotone functions

  **Galois connection**
  
  (2) $c \subseteq \gamma(\alpha(c))$ for all $c \in \mathcal{P}(C)$
  
  (3) $a \supseteq \alpha(\gamma(a))$ for all $a \in A$

  **Galois connection**: This property means intuitively that the functions $\alpha$ and $\gamma$ are "almost inverses" of each other.
Figure: Galois connection
Galois insertions

- For a Galois connection, there may be several elements of A that describe the same element in C.
- As a result, A may contain elements which are irrelevant for describing C.
- The concept of Galois insertion fixes this:

**Monotonicity**

(1) $\alpha$ and $\gamma$ are monotone functions

**Galois insertion**

(2) $c \subseteq \gamma(\alpha(c))$ for all $c \in \mathcal{P}(C)$

(3) $a = \alpha(\gamma(a))$ for all $a \in A$
Figure: Galois insertion

\[ \gamma(a(c)) \]
A Galois connection can be used to induce the abstract operations from the concrete ones.

We can show that the induced operation $\text{op} = a \circ \text{op} \circ \gamma$ is the most precise abstract operation in this setting.

The induced operation might not be computable. In this case we can define an upper approximation $\text{op}^\#$, $\text{op} \subseteq \text{op}^\#$, and use this as abstract operation.
Abstract Interpretation

Widening
To introduce the notion of widening, we have a look at range analysis, which provides for every variable an over-approximation of its integer value range.

We are left with the task of choosing a suitable abstract domain: the interval lattice suggests itself.

Interval = \{ \bot \} \cup \{ [x,y] \mid x \leq y, x \in \mathbb{Z} \cup \{\infty\}, y \in \mathbb{Z} \cup \{\infty\} \}
Consider the following program:

At program point 2, the following sequence of abstract states arises: $[x \mapsto [1,1]], [x \mapsto [1,2]], [x \mapsto [1,3]], ...$

**Consequence:** The analysis never terminates (or, if $n$ is statically known, converges only very slowly).
The ascending chain condition

- Using an arbitrary complete lattice as abstract domain, the solution is not computable in general.
- The reason for that is the fact that the value space might be unbounded, containing infinite ascending chains:
  \[(l_n)_n \text{ is such that } l_1 \subseteq l_2 \subseteq l_3 \subseteq \cdots,\]
  but there exists no \(n\) such that \(l_n = l_{n+1} = \cdots\)
- If we replace it with an abstract space that is finite (or does not possess infinite ascending chains), then the computation is guaranteed to terminate.
- In general, we want an abstract domain to satisfy the ascending chain condition, i.e. each ascending chain eventually stabilises:
  if \((l_n)_n\) is such that \(l_1 \subseteq l_2 \subseteq l_3 \subseteq \cdots\),
  then there exists \(n\) such that \(l_n = l_{n+1} = \cdots\)
The reason for the non-termination in the example is that the interval lattice contains infinite ascending chains.

Trick, if we cannot eliminate ascending chains: We redefine the join operator of the lattice to jump to the extremal value more quickly.

Before: \([1,1] \sqcup [2,2] = [1,2]\)    
Now: \([1,1] \triangledown [2,2] = [1,\infty]\)
A widening ∇ : D × D → D on a partially ordered set (D, ⊑) satisfies the following properties:

1. For all x, y ∈ D. x ⊑ x ∇ y and y ⊑ x ∇ y
2. For all ascending chains x_1 ⊑ x_2 ⊑ x_3 ⊑ · · · the ascending chain y_1 = x_1 ⊑ y_2 = y_1 ∇ x_2 ⊑ · · · ⊑ y_{n+1} = y_n ∇ x_{n+1} eventually stabilizes.

- Widening is used to accelerate the convergence towards an upper approximation of the least fixed point.
Example (continued)

- Assume we have a widening operator $\nabla$ that is defined such that $[1,1] \nabla [2,2] = [1, +\infty]$

The analysis converges quickly.
Reading


Chapter 1: Section 1.5
Chapter 4 (advanced material)