Static Program Analysis

Lecture 11: Abstract Interpretation I (Theoretical Foundations)

Winter Semester 2016/17

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-1617/spa/
Abstract Interpretation I

- **Summary**: a theory of sound approximation of the semantics of programs
- **Basic idea**: execution of program on abstract values
- **Examples**:
  - parity (even/odd) rather than concrete numbers
  - types rather than concrete values (similar to type-level JVM bytecode interpreter)
- **Procedure**: run program on (finite) set of abstract values that cover all concrete inputs using abstract operations (for basic operations, statements, ...) that cover all concrete outputs

  \[ \Rightarrow \text{soundness of approach} \]
- **Preciseness** of information again characterized by partial order
Abstract Interpretation II

- **Advantages:**
  - Abstract interpretation covers *conditional branches* (*if/while*) without further extension
  - Granularity of abstract domain influences *precision and complexity* of analysis (mutual trade-off)
  - Numerous variants for *different kinds of programs* (functional, concurrent, ...)
  - *Soundness* is guaranteed if abstract operations are determined according to theory

- **Disadvantages:**
  - Complexity generally higher than with dataflow analysis
  - Automatic derivation of abstract operations can be difficult
Introduction to Abstract Interpretation

Overview

1. Theoretical foundations (Galois connections)
2. (Concrete & Abstract semantics of WHILE programs
3. Automatic derivation of abstract semantics
4. Application: verification of 16-bit multiplication
5. Predicate abstraction
6. CEGAR (CounterExample-Guided Abstraction Refinement)
Galois Connections I

Definition 11.1 (Galois connection)

Let \((L, \sqsubseteq_L)\) and \((M, \sqsubseteq_M)\) be complete lattices. A pair \((\alpha, \gamma)\) of monotonic functions

\[
\alpha : L \rightarrow M \quad \text{and} \quad \gamma : M \rightarrow L
\]

is called a Galois connection if

\[
\forall l \in L : l \sqsubseteq_L \gamma(\alpha(l)) \quad \text{and} \quad \forall m \in M : \alpha(\gamma(m)) \sqsubseteq_M m
\]

Interpretation:

- \(L = \{\text{sets of concrete values}\}\), \(M = \{\text{sets of abstract values}\}\)
- \(\alpha = \text{abstraction function}, \gamma = \text{concretisation function}\)
- \(l \sqsubseteq_L \gamma(\alpha(l))\): \(\alpha\) yields over-approximation
- \(\alpha(\gamma(m)) \sqsubseteq_M m\): no loss of precision by abstraction after concretisation
- Usually: \(l \neq \gamma(\alpha(l)), \alpha(\gamma(m)) = m\)
Theoretical Foundations of Abstract Interpretation

Galois Connections II

For \( A = \{ \text{concrete values} \} \), \( B = \{ \text{abstract values} \} \), \( L = 2^A \), \( M = 2^B \):

\[
\forall l \in L : l \sqsubseteq_L \gamma(\alpha(l))
\]

(\( \alpha \) yields over-approximation)

\[
\forall m \in M : \alpha(\gamma(m)) \sqsubseteq_M m
\]

(no loss of precision by abstraction after concretisation)
Example 11.2 (Parity abstraction)

Concrete domain

\[ L := (2^\mathbb{Z}, \subseteq) \]

\[ \gamma : 2^{\{\text{even, odd}\}} \to 2^\mathbb{Z} \]

\[ \gamma(P) := \bigcup_{p \in P} \mathbb{Z}_p \]

where

\[ \mathbb{Z}_{\text{even}} := \{\ldots, -2, 0, 2, \ldots\} \]

\[ \mathbb{Z}_{\text{odd}} := \{\ldots, -3, -1, 1, 3, \ldots\} \]

Abstract domain

\[ M := (2^{\{\text{even, odd}\}}, \subseteq) \]

\[ \alpha : 2^\mathbb{Z} \to 2^{\{\text{even, odd}\}} \]

\[ \alpha(Z) := \begin{cases} 
\emptyset & \text{if } Z = \emptyset \\
\{\text{even}\} & \text{if } Z \subseteq \mathbb{Z}_{\text{even}} \\
\{\text{odd}\} & \text{if } Z \subseteq \mathbb{Z}_{\text{odd}} \\
\{\text{even, odd}\} & \text{otherwise} 
\end{cases} \]

yields a Galois connection. For example,

- \( \gamma(\alpha(\{1, 3, 7\})) = \gamma(\{\text{odd}\}) = \{\ldots, -3, -1, 1, 3, \ldots\} \supseteq \{1, 3, 7\} \)
- \( \alpha(\gamma(\{\text{even}\})) = \alpha(\{\ldots, -2, 0, 2, \ldots\}) = \{\text{even}\} \)
Example 11.3 (Sign abstraction)

**Concrete domain**

\[
L := (2^\mathbb{Z}, \subseteq)
\]

\[
\gamma : \mathbb{Z} \rightarrow \mathbb{Z}^+
\]

\[
\gamma(S) := \bigcup_{s \in S} \mathbb{Z}_s
\]

where

\[
\mathbb{Z}_+ := \{1, 2, 3, \ldots\}
\]

\[
\mathbb{Z}_- := \{-1, -2, -3, \ldots\}
\]

\[
\mathbb{Z}_0 := \{0\}
\]

**Abstract domain**

\[
M := (2^{\{+,-,0\}}, \subseteq)
\]

\[
\alpha : \mathbb{Z} \rightarrow 2^{\{+,-,0\}}
\]

\[
\alpha(Z) := \{\text{sgn}(z) \mid z \in \mathbb{Z}\}
\]

where

\[
\text{sgn}(z) := \begin{cases} + & \text{if } z > 0 \\ - & \text{if } z < 0 \\ 0 & \text{otherwise} \end{cases}
\]

yields a Galois connection. For example,

- \[\gamma(\alpha(\{0, 1, 3\})) = \gamma(\{+, 0\}) = \{0, 1, 2, 3, \ldots\} \supseteq \{0, 1, 3\}\]
- \[\alpha(\gamma(\{+, -\})) = \alpha(\mathbb{Z} \setminus \{0\}) = \{+, -\}\]
### Galois Connections V

**Example 11.4 (Interval abstraction (cf. Slide 7.16))**

<table>
<thead>
<tr>
<th>Concrete domain</th>
<th>Abstract domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L := (2^\mathbb{Z}, \subseteq)$</td>
<td>$M := (\text{Int}, \subseteq)$</td>
</tr>
<tr>
<td>$\gamma : \text{Int} \rightarrow 2^\mathbb{Z}$</td>
<td>where $\text{Int} := (\mathbb{Z} \cup { -\infty }) \times (\mathbb{Z} \cup { +\infty }) \cup { \emptyset }$</td>
</tr>
<tr>
<td>$\gamma(J) := \begin{cases} \emptyset &amp; \text{if } J = \emptyset \ { z \in \mathbb{Z} \mid z_1 \leq z \leq z_2 } &amp; \text{if } J = [z_1, z_2] \end{cases}$</td>
<td>$\alpha : 2^\mathbb{Z} \rightarrow \text{Int}$</td>
</tr>
<tr>
<td>$\alpha(\mathbb{Z}) := \begin{cases} \emptyset &amp; \text{if } \mathbb{Z} = \emptyset \ \left[ \prod \mathbb{Z}, \bigcup \mathbb{Z} \right] &amp; \text{otherwise} \end{cases}$</td>
<td></td>
</tr>
</tbody>
</table>

yields a Galois connection. For example,

- $\gamma(\alpha(\{1, 3, 5, \ldots \})) = \gamma([1, +\infty]) = \{1, 2, 3, 4, 5, \ldots \} \supseteq \{1, 3, 5, \ldots \}$
- $\alpha(\gamma([-1, 1])) = \alpha(\{-1, 0, 1\}) = [-1, 1]$
Theoretical Foundations of Abstract Interpretation

Properties of Galois Connections

Lemma 11.5

Let \((\alpha, \gamma)\) be a Galois connection with \(\alpha : L \to M\) and \(\gamma : M \to L\), and let \(l \in L\), \(m \in M\), \(L' \subseteq L\), \(M' \subseteq M\).

1. \(\alpha(l) \sqsubseteq_M m \iff l \sqsubseteq_L \gamma(m)\)

2. \(\gamma\) is uniquely determined by \(\alpha\) as follows: \(\gamma(m) = \bigsqcup \{l \in L \mid \alpha(l) \sqsubseteq_M m\}\)

3. \(\alpha\) is uniquely determined by \(\gamma\) as follows: \(\alpha(l) = \bigcap \{m \in M \mid l \sqsubseteq_L \gamma(m)\}\)

4. \(\alpha\) is completely distributive: for every \(L' \subseteq L\), \(\alpha(\bigsqcup L') = \bigsqcup \{\alpha(l) \mid l \in L'\}\)

5. \(\gamma\) is completely multiplicative: for every \(M' \subseteq M\), \(\gamma(\bigcap M') = \bigcap \{\gamma(m) \mid m \in M'\}\)

Proof.

on the board
Excursus: Concrete Semantics of WHILE Programs

Reminder: Syntax of WHILE

The syntax of WHILE Programs is defined by the following context-free grammar (cf. Definition 1.3):

\[ a ::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \in AExp \]
\[ b ::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \land b_2 \mid b_1 \lor b_2 \in BExp \]
\[ c ::= \text{skip} \mid x := a \mid c_1 ; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end} \mid \text{while } b \text{ do } c \text{ end} \in Cmd \]
Excursus: Concrete Semantics of WHILE Programs

Program States

- **Meaning of expression** = value (in the usual sense)
- Depends on the **values of the variables** in the expression

Definition 11.6 (Program state)

A (program) state is an element of the set

\[ \Sigma := \{ \sigma \mid \sigma : \text{Var} \rightarrow \mathbb{Z} \} \]

called the **state space**.

Thus \( \sigma(x) \) denotes the value of \( x \in \text{Var} \) in state \( \sigma \in \Sigma \).
Excursus: Concrete Semantics of WHILE Programs

Evaluation of Expressions

Definition 11.7 (Evaluation function)

Let $\sigma \in \Sigma$ be a state.
1. $val_\sigma : AExp \rightarrow \mathbb{Z} : a \rightarrow val_\sigma(a)$ yields the value of $a$ in state $\sigma$
2. $val_\sigma : BExp \rightarrow \mathbb{B} : b \rightarrow val_\sigma(b)$ yields the value of $b$ in state $\sigma$

Example 11.8

Let $\sigma(x) = 1$ and $\sigma(y) = 2$.
1. $val_\sigma(2 \times x + y) = 4$
2. $val_\sigma(\neg(x + 1 > y)) = \text{true}$
Excursus: Concrete Semantics of WHILE Programs

Derivation Rules

- Definition employs derivation rules of the form

<table>
<thead>
<tr>
<th>Name</th>
<th>Premise(s)</th>
<th>Conclusion</th>
</tr>
</thead>
</table>

- meaning: if every premise is fulfilled, then conclusion can be drawn
- a rule with no premises is called an axiom

- Iterated application yields complete derivation tree
  - initial program and state at root
  - premises as children of inner nodes
  - axioms at leaves
Execution of Statements I

Definition 11.9 (Execution relation for statements)

If \( c \in Cmd \) and \( \sigma \in \Sigma \), then \( \langle c, \sigma \rangle \) is called a configuration. The execution relation

\[
\to \subseteq (Cmd \times \Sigma) \times ((Cmd \cup \{\downarrow\}) \times \Sigma)
\]

is defined by the following rules:

\[
\begin{align*}
\text{(skip)} & \quad \langle \text{skip}, \sigma \rangle \to \langle \downarrow, \sigma \rangle \\
\text{(asgn)} & \quad \langle x := a, \sigma \rangle \to \langle \downarrow, \sigma[x \mapsto \text{val}_\sigma(a)] \rangle \\
\text{(seq1)} & \quad \langle c_1, \sigma \rangle \to \langle c'_1, \sigma' \rangle \quad c'_1 \neq \downarrow \\
\text{(seq2)} & \quad \langle c_1 ; c_2, \sigma \rangle \to \langle c'_1 ; c_2, \sigma' \rangle \\
\text{(seq1)} & \quad \langle c_1, \sigma \rangle \to \langle \downarrow, \sigma' \rangle \\
\text{(seq2)} & \quad \langle c_1 ; c_2, \sigma \rangle \to \langle c_2, \sigma' \rangle
\end{align*}
\]
**Definition 11.9 (Execution relation for statements; continued)**

\[
\begin{align*}
\text{(if1)} & \quad \text{val}_\sigma(b) = \text{true} \\
& \quad \langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma \rangle \rightarrow \langle c_1, \sigma \rangle \\
\text{(if2)} & \quad \text{val}_\sigma(b) = \text{false} \\
& \quad \langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \sigma \rangle \rightarrow \langle c_2, \sigma \rangle \\
\text{(wh1)} & \quad \text{val}_\sigma(b) = \text{true} \\
& \quad \langle \text{while } b \text{ do } c \text{ end}, \sigma \rangle \rightarrow \langle c; \text{while } b \text{ do } c \text{ end}, \sigma \rangle \\
\text{(wh2)} & \quad \text{val}_\sigma(b) = \text{false} \\
& \quad \langle \text{while } b \text{ do } c \text{ end}, \sigma \rangle \rightarrow \langle \downarrow, \sigma \rangle
\end{align*}
\]

**Remark:** \(\downarrow\) indicates **successful termination** of the program.
**Example 11.10**

- \( y := 1; \) while \( \neg(x=1) \) do \( y := y \times x; x := x - 1 \) end

- **Claim:** \( \langle c, \sigma \rangle \rightarrow^+ \langle \downarrow, \sigma_{1,6} \rangle \) for every \( \sigma \in \Sigma \) with \( \sigma(x) = 3 \)

- **Notation:** \( \sigma_{i,j} \) means \( \sigma(x) = i, \sigma(y) = j \)

- **Derivation:** on the board
Excursus: Concrete Semantics of WHILE Programs

Determinism Property of Execution Relation

This operational semantics is well defined in the following sense:

**Theorem 11.11**

The execution relation for statements is deterministic, i.e., whenever \( c \in \text{Cmd} \), \( \sigma \in \Sigma \) and \( \kappa_1, \kappa_2 \in (\text{Cmd} \cup \{\uparrow\}) \times \Sigma \) such that \( \langle c, \sigma \rangle \rightarrow \kappa_1 \) and \( \langle c, \sigma \rangle \rightarrow \kappa_2 \), then \( \kappa_1 = \kappa_2 \).

**Proof.**

omitted

More on formal semantics of programming languages: *Semantics and Verification of Software* in next winter semester.