

Static Program Analysis

Lecture 12: Abstract Interpretation I (Theoretical Foundations)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

`noll@cs.rwth-aachen.de`

`http://www-i2.informatik.rwth-aachen.de/i2/spa11/`

Summer Semester 2011

- 1 Introduction to Abstract Interpretation
- 2 Theoretical Foundations of Abstract Interpretation
- 3 Excursus: Concrete Semantics of WHILE Programs

- **Summary:** a theory of **sound approximation** of the semantics of programs
- **Basic idea:** execution of program on **abstract values** (similar to type-level bytecode interpreter)
- **Example:** parity (even/odd) rather than concrete numbers
- **Procedure:** run program on finite set of abstract values that **cover all concrete inputs** using abstract operations that **cover all concrete outputs**
 \implies **soundness** of approach
- **Preciseness** of information again characterized by **partial order**

- **Advantages:**

- Abstract interpretation covers **conditional branches** (**if/while**) without further extension
- Granularity of abstract domain influences **precision and complexity** of analysis (mutual tradeoff)
- 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

- 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)

- 1 Introduction to Abstract Interpretation
- 2 Theoretical Foundations of Abstract Interpretation
- 3 Excursus: Concrete Semantics of WHILE Programs

Definition 12.1 (Galois connection)

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

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

is called a **Galois connection** if

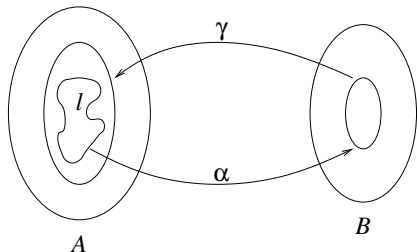
$$\forall l \in L : l \sqsubseteq_L \gamma(\alpha(l)) \quad \text{and} \quad \forall m \in M : \alpha(\gamma(m)) \sqsubseteq_M m$$

Interpretation:

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

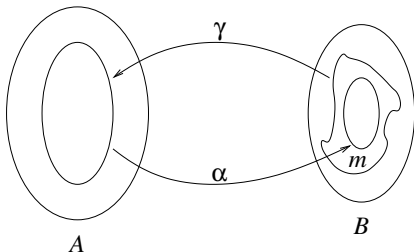
Galois Connections II

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



$$\forall I \in L : I \sqsubseteq_L \gamma(\alpha(I))$$

(α yields over-approximation)



$$\forall m \in M : \alpha(\gamma(m)) \sqsubseteq_M m$$

(no loss of precision by
abstraction after concretization)

Example 12.2 (Parity abstraction)

Concrete domain: $L = (2^{\mathbb{Z}}, \subseteq)$ Abstract domain: $M = (2^{\{\text{even}, \text{odd}\}}, \subseteq)$
 $\alpha : 2^{\mathbb{Z}} \rightarrow 2^{\{\text{even}, \text{odd}\}}$

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

$$\gamma : 2^{\{\text{even}, \text{odd}\}} \rightarrow 2^{\mathbb{Z}}$$
$$\gamma(P) := \bigcup_{p \in P} \mathbb{Z}_p$$

where

$$\mathbb{Z}_{\text{even}} := \{\dots, -2, 0, 2, \dots\}$$
$$\mathbb{Z}_{\text{odd}} := \{\dots, -3, -1, 1, 3, \dots\}$$

yields a Galois connection. For example,

- $\gamma(\alpha(\{1, 3, 7\})) = \gamma(\{\text{odd}\}) = \{\dots, -3, -1, 1, 3, \dots\} \supseteq \{1, 3, 7\}$
- $\alpha(\gamma(\{\text{even}\})) = \alpha(\{\dots, -2, 0, 2, \dots\}) = \{\text{even}\}$

Example 12.3 (Sign abstraction)

Concrete domain: $L = (2^{\mathbb{Z}}, \subseteq)$ Abstract domain: $M = (2^{\{+, -, 0\}}, \subseteq)$

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

$$\alpha(Z) := \{\text{sgn}(z) \mid z \in Z\}$$

$$\gamma : 2^{\{+, -, 0\}} \rightarrow 2^{\mathbb{Z}}$$

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

where

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

$$\mathbb{Z}_+ := \{1, 2, 3, \dots\}$$

$$\mathbb{Z}_- := \{-1, -2, -3, \dots\}$$

$$\mathbb{Z}_0 := \{0\}$$

yields a Galois connection. For example,

- $\gamma(\alpha(\{0, 1, 3\})) = \gamma(\{+, 0\}) = \{0, 1, 2, 3, \dots\} \supseteq \{0, 1, 3\}$
- $\alpha(\gamma(\{+, -\})) = \alpha(\mathbb{Z} \setminus \{0\}) = \{+, -\}$

Example 12.4 (Interval abstraction (cf. Slide 8.5))

Concrete domain: $L = (2^{\mathbb{Z}}, \subseteq)$

Abstract domain: $M = (Int, \subseteq)$

$$\alpha : 2^{\mathbb{Z}} \rightarrow Int$$

$$\alpha(Z) := \begin{cases} \emptyset & \text{if } Z = \emptyset \\ [\sqcap Z, \sqcup Z] & \text{otherwise} \end{cases}$$

$$\gamma : Int \rightarrow 2^{\mathbb{Z}}$$

$$\gamma(I) := \begin{cases} \emptyset & \text{if } I = \emptyset \\ \{z \in \mathbb{Z} \mid z_1 \leq z \leq z_2\} & \text{if } I = [z_1, z_2] \end{cases}$$

yields a Galois connection. For example,

- $\gamma(\alpha(\{1, 3, 5, \dots\})) = \gamma([1, +\infty]) = \{1, 2, 3, 4, 5, \dots\} \supseteq \{1, 3, 5, \dots\}$
- $\alpha(\gamma([-1, 1])) = \alpha(\{-1, 0, 1\}) = [-1, 1]$

Lemma 12.5

Let (α, γ) be a Galois connection with $\alpha : L \rightarrow M$ and $\gamma : M \rightarrow L$, and let $I \in L$, $m \in M$, $L' \subseteq L$, $M' \subseteq M$.

① $\alpha(I) \sqsubseteq_M m \iff I \sqsubseteq_L \gamma(m)$

② γ is *uniquely determined by α* as follows:

$$\gamma(m) = \bigsqcup \{I \in L \mid \alpha(I) \sqsubseteq_M m\}$$

③ α is *uniquely determined by γ* as follows:

$$\alpha(I) = \bigsqcap \{m \in M \mid I \sqsubseteq_L \gamma(m)\}$$

④ α is *completely distributive*:

$$\alpha(\bigsqcup L') = \bigsqcup \{\alpha(I) \mid I \in L'\}$$

⑤ γ is *completely multiplicative*:

$$\gamma(\bigsqcap M') = \bigsqcap \{\gamma(m) \mid m \in M'\}$$

Proof.

on the board



- 1 Introduction to Abstract Interpretation
- 2 Theoretical Foundations of Abstract Interpretation
- 3 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):

$$a ::= z \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in AExp$$
$$b ::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \in BExp$$
$$c ::= \text{skip} \mid x := a \mid c_1 ; c_2 \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c \in Cmd$$

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

Definition 12.6 (Program state)

A **(program) state** is an element of the set

$$\Sigma := \{\sigma \mid \sigma : Var \rightarrow \mathbb{Z}\},$$

called the **state space**.

Thus $\sigma(x)$ denotes the value of $x \in Var$ in state $\sigma \in \Sigma$.

Definition 12.7 (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 σ**
- 2 $val_\sigma : BExp \rightarrow \mathbb{B} : b \rightarrow val_\sigma(b)$
yields the **value of b in state σ**

Example 12.8

Let $\sigma(\mathbf{x}) = 1$ and $\sigma(\mathbf{y}) = 2$.

- 1 $val_\sigma(2 * \mathbf{x} + \mathbf{y}) = 4$
- 2 $val_\sigma(\neg(\mathbf{x} + 1 > \mathbf{y})) = \text{true}$

Definition employs **derivation rules** of the form

$$\text{Name} \frac{\text{Premise(s)}}{\text{Conclusion}}$$

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

Definition 12.9 (Execution relation for statements)

If $c \in \text{Cmd}$ and $\sigma \in \Sigma$, then $\langle c, \sigma \rangle$ is called a **configuration**. The **execution relation** (on configurations and states) is defined by the following rules:

$$\text{(skip)} \frac{}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma}$$

$$\text{(asgn)} \frac{}{\langle x := a, \sigma \rangle \rightarrow \sigma[x \mapsto \text{val}_\sigma(a)]}$$

$$\text{(seq1)} \frac{\langle c_1, \sigma \rangle \rightarrow \langle c'_1, \sigma' \rangle}{\langle c_1 ; c_2, \sigma \rangle \rightarrow \langle c'_1 ; c_2, \sigma' \rangle}$$

$$\text{(seq2)} \frac{\langle c_1, \sigma \rangle \rightarrow \sigma'}{\langle c_1 ; c_2, \sigma \rangle \rightarrow \langle c_2, \sigma' \rangle}$$

Definition 12.9 (Execution relation for statements; continued)

$$\text{(if1)} \frac{val_{\sigma}(b) = \text{true}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle c_1, \sigma \rangle}$$

$$\text{(if2)} \frac{val_{\sigma}(b) = \text{false}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \rightarrow \langle c_2, \sigma \rangle}$$

$$\text{(wh1)} \frac{val_{\sigma}(b) = \text{true}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \langle c; \text{while } b \text{ do } c, \sigma \rangle}$$

$$\text{(wh2)} \frac{val_{\sigma}(b) = \text{false}}{\langle \text{while } b \text{ do } c, \sigma \rangle \rightarrow \sigma}$$