Static Program Analysis

Lecture 11: Abstract Interpretation I (Theoretical Foundations)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://moves.rwth-aachen.de/teaching/ws-1415/spa/

Winter Semester 2014/15

Outline

Abstract Interpretation I

- **Summary:** a theory of sound approximation of the semantics of programs
- **Basic idea:** execution of program on abstract values (similar to type-level JVM bytecode interpreter)
- **Example:** parity (even/odd) rather than concrete numbers

Abstract Interpretation I

- **Summary:** a theory of sound approximation of the semantics of programs
- **Basic idea:** execution of program on abstract values (similar to type-level JVM 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

  $\Longrightarrow$ 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 tradeoff)
  - Numerous variants for different kinds of programs (functional, concurrent, ...)
  - Soundness is guaranteed if abstract operations are determined according to theory

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 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

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)

Outline

Definition 11.1 (Galois connection)

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

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

is called a Galois connection if

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

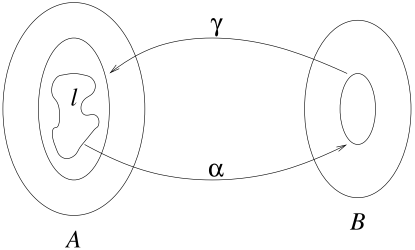
Evariste Galois
(1811–1832)

Interpretation:

- $L = \{\text{sets of concrete values}\}$, $M = \{\text{sets of abstract values}\}$
- α = abstraction function, γ = 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: $l \neq \gamma(\alpha(l))$, $\alpha(\gamma(m)) = m$

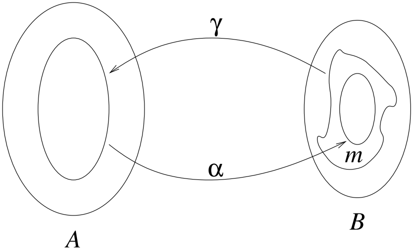
Galois Connections II

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



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

(α yields over-approximation)



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

(no loss of precision by abstraction after concretization)

Example 11.2 (Parity abstraction)

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

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

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

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

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

where

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

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

yields a Galois connection.

Galois Connections III

Example 11.2 (Parity abstraction)

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

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

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

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

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

where

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

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

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

$$\begin{aligned} \alpha : 2^{\mathbb{Z}} &\to 2^{\{+,-,0\}} \\ \alpha(Z) &:= \{\mathrm{sgn}(z) \mid z \in Z\} \\ \gamma : 2^{\{+,-,0\}} &\to 2^{\mathbb{Z}} \\ \gamma(S) &:= \bigcup_{s \in S} \mathbb{Z}_s \end{aligned}$$

where

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

$$\begin{aligned} \mathbb{Z}_+ &:= \{1, 2, 3, \ldots\} \\ \mathbb{Z}_- &:= \{-1, -2, -3, \ldots\} \\ \mathbb{Z}_0 &:= \{0\} \end{aligned}$$

yields a Galois connection.

Galois Connections IV

Example 11.3 (Sign abstraction)

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

$$\begin{aligned} \alpha : 2^{\mathbb{Z}} &\to 2^{\{+,-,0\}} \\ \alpha(Z) &:= \{\text{sgn}(z) \mid z \in Z\} \end{aligned}$$

$$\begin{aligned} \gamma : 2^{\{+,-,0\}} &\to 2^{\mathbb{Z}} \\ \gamma(S) &:= \textstyle\bigcup_{s \in S} \mathbb{Z}_s \end{aligned}$$

where

$$\begin{aligned} \text{sgn}(z) &:= \begin{cases} + & \text{if } z > 0 \\ - & \text{if } z < 0 \\ 0 & \text{otherwise} \end{cases} \\ \mathbb{Z}_+ &:= \{1, 2, 3, \ldots\} \\ \mathbb{Z}_- &:= \{-1, -2, -3, \ldots\} \\ \mathbb{Z}_0 &:= \{0\} \end{aligned}$$

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 7.17))

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

(where $Int = (\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{+\infty\}) \cup \{\emptyset\}$)

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

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

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

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

yields a Galois connection.

Example 11.4 (Interval abstraction (cf. Slide 7.17))

Concrete domain: $L = (2^{\mathbb{Z}}, \subseteq)$ Abstract domain: $M = (Int, \subseteq)$
(where $Int = (\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{+\infty\}) \cup \{\emptyset\}$)

$$\alpha : 2^{\mathbb{Z}} \to Int$$
$$\alpha(Z) := \begin{cases} \emptyset & \text{if } Z = \emptyset \\ [\bigsqcap Z, \bigsqcup Z] & \text{otherwise} \end{cases}$$

$$\gamma : Int \to 2^{\mathbb{Z}}$$
$$\gamma(J) := \begin{cases} \emptyset & \text{if } J = \emptyset \\ \{z \in \mathbb{Z} \mid z_1 \leq z \leq z_2\} & \text{if } J = [z_1, z_2] \end{cases}$$

yields a Galois connection. For example,

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

Properties of Galois Connections

Lemma 11.5

Let (α, γ) *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. $\alpha(l) \sqsubseteq_M m \iff l \sqsubseteq_L \gamma(m)$

Properties of Galois Connections

Lemma 11.5

Let (α, γ) *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. γ *is uniquely determined by* α *as follows:*

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

Properties of Galois Connections

Lemma 11.5

Let (α, γ) *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. γ *is uniquely determined by* α *as follows:*
$$\gamma(m) = \bigsqcup\{l \in L \mid \alpha(l) \sqsubseteq_M m\}$$
3. α *is uniquely determined by* γ *as follows:*
$$\alpha(l) = \bigsqcap\{m \in M \mid l \sqsubseteq_L \gamma(m)\}$$

Properties of Galois Connections

Lemma 11.5

Let (α, γ) *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. γ *is uniquely determined by* α *as follows:*
$$\gamma(m) = \bigsqcup \{l \in L \mid \alpha(l) \sqsubseteq_M m\}$$
3. α *is uniquely determined by* γ *as follows:*
$$\alpha(l) = \bigsqcap \{m \in M \mid l \sqsubseteq_L \gamma(m)\}$$
4. α *is completely distributive:* $\alpha(\bigsqcup L') = \bigsqcup \{\alpha(l) \mid l \in L'\}$

Properties of Galois Connections

Lemma 11.5

Let (α, γ) *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. γ *is uniquely determined by* α *as follows:*
$$\gamma(m) = \bigsqcup\{l \in L \mid \alpha(l) \sqsubseteq_M m\}$$
3. α *is uniquely determined by* γ *as follows:*
$$\alpha(l) = \bigsqcap\{m \in M \mid l \sqsubseteq_L \gamma(m)\}$$
4. α *is completely distributive:* $\alpha(\bigsqcup L') = \bigsqcup\{\alpha(l) \mid l \in L'\}$
5. γ *is completely multiplicative:* $\gamma(\bigsqcap M') = \bigsqcap\{\gamma(m) \mid m \in M'\}$

Properties of Galois Connections

Lemma 11.5

Let (α, γ) *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. γ *is uniquely determined by* α *as follows:*
$$\gamma(m) = \bigsqcup \{l \in L \mid \alpha(l) \sqsubseteq_M m\}$$
3. α *is uniquely determined by* γ *as follows:*
$$\alpha(l) = \bigsqcap \{m \in M \mid l \sqsubseteq_L \gamma(m)\}$$
4. α *is completely distributive:* $\alpha(\bigsqcup L') = \bigsqcup \{\alpha(l) \mid l \in L'\}$
5. γ *is completely multiplicative:* $\gamma(\bigsqcap M') = \bigsqcap \{\gamma(m) \mid m \in M'\}$

Proof.

on the board □

Outline

Reminder: Syntax of WHILE

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

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

Program States

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

- 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

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

called the state space.

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

Evaluation of Expressions

Definition 11.7 (Evaluation function)

Let $\sigma \in \Sigma$ be a state.

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

Evaluation of Expressions

Definition 11.7 (Evaluation function)

Let $\sigma \in \Sigma$ be a state.

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

Example 11.8

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

1. $val_\sigma(\mathtt{2 * x + y}) = 4$
2. $val_\sigma(\neg(\mathtt{x + 1 > y})) = \mathsf{true}$

Derivation Rules

- 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

Derivation Rules

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

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:

$$\text{(skip)}\ \frac{}{\langle \texttt{skip}, \sigma \rangle \to \langle \downarrow, \sigma \rangle}$$

$$\text{(asgn)}\ \frac{}{\langle x \ := a, \sigma \rangle \to \langle \downarrow, \sigma[x \mapsto val_\sigma(a)] \rangle}$$

$$\text{(seq1)}\ \frac{\langle c_1, \sigma \rangle \to \langle c_1', \sigma' \rangle \quad c_1' \neq \downarrow}{\langle c_1 ; c_2, \sigma \rangle \to \langle c_1' ; c_2, \sigma' \rangle}$$

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

Execution of Statements II

Definition 11.9 (Execution relation for statements; continued)

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

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

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

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

Remark: $\downarrow$ indicates successful termination of the program

An Execution Example

Example 11.10

- $c := \underbrace{\texttt{y := 1; while}\ \underbrace{\neg(\texttt{x=1})}_{b}\ \texttt{do}\ \underbrace{\underbrace{\texttt{y := y*x}}_{c_1};\underbrace{\texttt{x := x-1}}_{c_2}}_{c_0}}$
- Claim: $\langle c, \sigma \rangle \to^{+} \langle \downarrow, \sigma_{1,6} \rangle$ for every $\sigma \in \Sigma$ with $\sigma(\texttt{x}) = 3$
- Notation: $\sigma_{i,j}$ means $\sigma(\texttt{x}) = i$, $\sigma(\texttt{y}) = j$
- Derivation: on the board

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 $c \in Cmd$, $\sigma \in \Sigma$ *and* $\kappa_1, \kappa_2 \in (Cmd \cup \{\downarrow\}) \times \Sigma$ *such that* $\langle c, \sigma \rangle \rightarrow \kappa_1$ *and* $\langle c, \sigma \rangle \rightarrow \kappa_2$*, then* $\kappa_1 = \kappa_2$.

Proof.

omitted □

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 $c \in Cmd$, $\sigma \in \Sigma$ *and* $\kappa_1, \kappa_2 \in (Cmd \cup \{\downarrow\}) \times \Sigma$ *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 forthcoming summer semester