Static Program Analysis

Lecture 14: Abstract Interpretation IV
(Application Example: 16-Bit Multiplication)

Winter Semester 2016/17

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-1617/spa/
Recap: Abstract Semantics of WHILE

Safe Approximation of Execution Relations

Reminder: abstraction determined by Galois connection \((\alpha, \gamma)\) with \(\alpha : L \rightarrow M, \gamma : M \rightarrow L\)

- here: \(L := 2^\Sigma, M\) not fixed
- usually \(M = \text{Var} \rightarrow \ldots\) (more efficient) or \(M = 2^{\text{Var}\rightarrow\ldots}\) (more precise)
- write \(\text{Abs}\) in place of \(M\)
- thus \(\alpha : 2^\Sigma \rightarrow \text{Abs}\) and \(\gamma : \text{Abs} \rightarrow 2^\Sigma\)

Definition (Abstract semantics of WHILE)

Given \(\alpha : 2^\Sigma \rightarrow \text{Abs}\), an abstract semantics is defined by a family of functions

\[
\text{next}^\#_{c,c'} : \text{Abs} \rightarrow \text{Abs}
\]

where \(c \in \text{Cmd}, c' \in \text{Cmd} \cup \{\downarrow\}\), and each \(\text{next}^\#_{c,c'}\) is a safe approximation of \(\text{next}_{c,c'}\), i.e.,

\[
\alpha(\text{next}_{c,c'}(\gamma(\text{abs}))) \sqsubseteq_{\text{Abs}} \text{next}^\#_{c,c'}(\text{abs})
\]

for every \(\text{abs} \in \text{Abs}\) (notation: \(\langle c, \text{abs} \rangle \Rightarrow \langle c', \text{abs}' \rangle\) for \(\text{next}^\#_{c,c'}(\text{abs}) = \text{abs}'\)).
Recap: Abstract Semantics of WHILE

Extraction Functions

• **Assumption:** abstraction determined by *pointwise mapping* of concrete values
• If $L = 2^C$ and $M = 2^A$ with $\subseteq L = \subseteq M = \subseteq$, then $\beta : C \to A$ is called an *extraction function*
• $\beta$ determines *Galois connection* $(\alpha, \gamma)$ where

  $\alpha : L \to M : l \mapsto \beta(l)\ (= \{\beta(c) \mid c \in l\})$

  $\gamma : M \to L : m \mapsto \beta^{-1}(m)\ (= \{c \in C \mid \beta(c) \in m\})$

Example

1. Parity abstraction (cf. Example 11.2): $\beta : \mathbb{Z} \to \{\text{even, odd}\}$ where

   $\beta(z) := \begin{cases} 
   \text{even} & \text{if } z \text{ even} \\
   \text{odd} & \text{if } z \text{ odd}
   \end{cases}$

2. Sign abstraction (cf. Example 11.3): $\beta : \mathbb{Z} \to \{+, -, 0\}$ with $\beta = \text{sgn}$

3. Interval abstraction (cf. Example 11.4): not definable by extraction function (as $\text{int}$ is not of the form $2^A$)
Recap: Abstract Semantics of WHILE

Abstract Program States

Now: take values of variables into account

Definition (Abstract program state)

Let $\beta : \mathbb{Z} \rightarrow A$ be an extraction function.

- An abstract (program) state is an element of the set
  $$\{\rho \mid \rho : \text{Var} \rightarrow A\},$$
  called the abstract state space.

- The abstract domain is denoted by $Abs := 2^{\text{Var} \rightarrow A}$.

- The abstraction function $\alpha : 2^\Sigma \rightarrow Abs$ is given by
  $$\alpha(S) := \{\beta \circ \sigma \mid \sigma \in S\}$$
  for every $S \subseteq \Sigma$. 
Recap: Abstract Semantics of WHILE

Abstract Evaluation of Expressions

Definition (Abstract evaluation functions)

Let $\rho : \text{Var} \rightarrow A$ be an abstract state.

1. $\text{val}_\rho^\# : \text{AExp} \rightarrow 2^A$ is determined by ($f$ arithmetic operation)
   \[
   \text{val}_\rho^\#(z) := \{\beta(z)\} \\
   \text{val}_\rho^\#(x) := \{\rho(x)\} \\
   \text{val}_\rho^\#(f(a_1, \ldots, a_n)) := f^\#(\text{val}_\rho^\#(a_1), \ldots, \text{val}_\rho^\#(a_n))
   \]

2. $\text{val}_\rho^\# : \text{BExp} \rightarrow 2^B$ is determined by ($g/h$ relational/Boolean operation)
   \[
   \text{val}_\rho^\#(t) := \{t\} \\
   \text{val}_\rho^\#(g(a_1, \ldots, a_n)) := g^\#(\text{val}_\rho^\#(a_1), \ldots, \text{val}_\rho^\#(a_n)) \\
   \text{val}_\rho^\#(h(b_1, \ldots, b_n)) := h^\#(\text{val}_\rho^\#(b_1), \ldots, \text{val}_\rho^\#(b_n))
   \]

Example (Sign abstraction)

Let $\rho(x) = +$ and $\rho(y) = -$.

1. $\text{val}_\rho^\#(2 \ast x + y) = \{+, -, 0\}$
2. $\text{val}_\rho^\#(\neg(x + 1 > y)) = \{\text{false}\}$
Recap: Abstract Semantics of WHILE

Abstract Semantics of WHILE I

Reminder: abstract domain is $Abs := 2^{\text{Var} \rightarrow A}$

Definition (Abstract execution relation for statements)

If $c \in \text{Cmd}$ and $abs \in Abs$, then $\langle c, abs \rangle$ is called an abstract configuration. The abstract execution relation is defined by the following rules:

- **(skip)**
  \[
  \langle \text{skip}, abs \rangle \Rightarrow \langle \bot, abs \rangle
  \]

- **(asgn)**
  \[
  \langle x := a, abs \rangle \Rightarrow \langle \bot, \{ \rho[x \mapsto a'] | \rho \in abs, a' \in val^\rho(a) \} \rangle
  \]

- **(seq1)**
  \[
  \langle c_1, abs \rangle \Rightarrow \langle c'_1, abs' \rangle \quad c'_1 \neq \bot
  \]
  \[
  \langle c_1 ; c_2, abs \rangle \Rightarrow \langle c'_1 ; c_2, abs' \rangle
  \]

- **(seq2)**
  \[
  \langle c_1, abs \rangle \Rightarrow \langle \bot, abs' \rangle
  \]
  \[
  \langle c_1 ; c_2, abs \rangle \Rightarrow \langle c_2, abs' \rangle
  \]
### Recap: Abstract Semantics of WHILE

#### Abstract Semantics of WHILE II

**Definition (Abstract execution relation for statements; continued)**

- **(if1)**
  \[ \exists \rho \in \text{abs} : \text{true} \in \text{val}^\rho(b) \]
  \[ \langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \text{abs} \rangle \Rightarrow \langle c_1, \text{abs} \setminus \{ \rho \in \text{abs} | \text{val}^\rho(b) = \{\text{false}\} \} \rangle \]

- **(if2)**
  \[ \exists \rho \in \text{abs} : \text{false} \in \text{val}^\rho(b) \]
  \[ \langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end}, \text{abs} \rangle \Rightarrow \langle c_2, \text{abs} \setminus \{ \rho \in \text{abs} | \text{val}^\rho(b) = \{\text{true}\} \} \rangle \]

- **(wh1)**
  \[ \exists \rho \in \text{abs} : \text{true} \in \text{val}^\rho(b) \]
  \[ \langle \text{while } b \text{ do } c \text{ end}, \text{abs} \rangle \Rightarrow \langle c; \text{while } b \text{ do } c \text{ end}, \text{abs} \setminus \{ \rho \in \text{abs} | \text{val}^\rho(b) = \{\text{false}\} \} \rangle \]

- **(wh2)**
  \[ \exists \rho \in \text{abs} : \text{false} \in \text{val}^\rho(b) \]
  \[ \langle \text{while } b \text{ do } c \text{ end}, \text{abs} \rangle \Rightarrow \langle \downarrow, \text{abs} \setminus \{ \rho \in \text{abs} | \text{val}^\rho(b) = \{\text{true}\} \} \rangle \]
Recap: Abstract Semantics of WHILE

Correctness of Abstract Semantics

**Theorem (Soundness of abstract semantics)**

For each \( c \in \text{Cmd} \) and \( c' \in \text{Cmd} \cup \{\downarrow\} \), \( \text{next}^\#_{c,c'} \) is a **safe approximation** of \( \text{next}_{c,c'} \), i.e., for every \( \text{abs} \in \text{Abs} \),

\[
\alpha(\text{next}_{c,c'}(\gamma(\text{abs}))) \subseteq \text{next}^\#_{c,c'}(\text{abs}).
\]

The soundness proof employs the following auxiliary lemma.

**Lemma (Soundness of abstract evaluation)**

Let \( \beta : \mathbb{Z} \rightarrow A \) be an extraction function.

1. For every \( a \in A\text{Exp} \) and \( \sigma \in \Sigma \), \( \beta(\text{val}_\sigma(a)) \in \text{val}^\#_{\beta_\sigma}(a) \).
2. For every \( b \in B\text{Exp} \) and \( \sigma \in \Sigma \), \( \text{val}_\sigma(b) \in \text{val}^\#_{\beta_\sigma}(b) \).

**Proof (Lemma 13.13).**

omitted

**Proof (Theorem 13.12).**

on the board
Application Example: 16-Bit Multiplication

A 16-Bit Multiplier

Example 14.1 (16-bit multiplier)

\[ c = [\text{out}:0][\text{ovf}:0] \]
\[ \text{while } [\neg (\text{f1}=0) \land \text{ovf}=0] \text{ do} \]
\[ \text{if } [\text{lsb(f1)}=1] \text{ then} \]
\[ ([\text{ovf}, \text{out}]:=(\text{out}:17)+\text{f2}) \]
\[ \text{else} \]
\[ [\text{skip}] \]
\[ \text{end;} \]
\[ [\text{f1}:=(\text{f1}>>1)] \]
\[ \text{if } [\neg (\text{f1}=0) \land \text{ovf}=0] \text{ then} \]
\[ ([\text{ovf}, \text{f2}]:=(\text{f2}:17)<<1) \]
\[ \text{else} \]
\[ [\text{skip}] \]
\[ \text{end} \]

Procedure: in each iteration,

1. if LSB of \( \text{f1} \) is set (4), add \( \text{f2} \) to \( \text{out} \) (5)

Outputs:

- \( \text{out} \): 16-bit result
- \( \text{ovf} \): overflow bit

Operations:

- \( \text{lsb}(z) \): least significant bit of \( z \)
- \( \text{extension of } z \) to \( k \) bits by adding leading zeros

Expected result: (termination is trivial)

- \( \langle c, \sigma \rangle \overset{+}{\rightarrow} \langle \downarrow \sigma', \rangle \)
- \( \sigma'(\text{out}) = \sigma(\text{f1}) \cdot \sigma(\text{f2}) \)
- \( \sigma'(\text{ovf}) = 1 \)

Example run: on the board

- \( <<1/>>1 \): left/right shift by 1 bit

Input:

- \( \text{f1}, \text{f2} \): 16-bit input factors

Outputs:

- \( \text{out} \): 16-bit result
- \( \text{ovf} \): overflow bit

Operations:

- \( (x, y) \rightarrow z \): simultaneous assignment with split of \( z \)
Application Example: 16-Bit Multiplication

The Abstraction


- **f1**: no abstraction (as \( f1 \) controls multiplication)
- **f2**: congruence modulo \( m \) (for specific values of \( m \geq 2 \) – see Theorem 14.4)
  - extraction function: \( \beta : \mathbb{Z} \rightarrow \{0, \ldots, m - 1\} : z \mapsto z \mod m \)
  - congruence: \( z_1 \equiv z_2 \pmod{m} \) iff \( z_1 \mod m = z_2 \mod m \)
- **out**: congruence modulo \( m \)
- **ovf**: no abstraction (single bit)

Lemma 14.2 (Properties of modulo congruence)

For every \( z_1, z_2 \in \mathbb{Z} \) and \( m \geq 1 \),

\[
\begin{align*}
(z_1 + z_2) \mod m &\equiv ((z_1 \mod m) + (z_2 \mod m)) \mod m \\
(z_1 - z_2) \mod m &\equiv ((z_1 \mod m) - (z_2 \mod m)) \mod m \\
(z_1 \cdot z_2) \mod m &\equiv ((z_1 \mod m) \cdot (z_2 \mod m)) \mod m
\end{align*}
\]

Thus: modulo value of expression determined by modulo values of subexpressions
Application Example: 16-Bit Multiplication

Abstract Interpretation of Multiplier

Example 14.3 (Abstraction of 16-bit multiplier; cf. Example 14.1)

Abstract execution for
- \( f_1 = 101_2 (= 5) \), \( f_2 = 1001010_2 (= 74) \)
- \( \text{out}, \text{ovf} \) with arbitrary initial values
- \( m = 5 \)

⇒ initial abstract value:
\[
asb = \{[f_1 \mapsto 101_2, f_2 \mapsto 74 \mod 5, \text{out} \mapsto r, \text{ovf} \mapsto b] \mid r \in \{0, \ldots, 4\}, b \in \mathbb{B}\}
\]
- first transitions: on the board
- generally: for all initial (abstract) values of \( f_1 \) and \( f_2 \), abstract results \( \langle \downarrow, \text{abs}' \rangle \), and \( \rho' \in \text{abs}' \),
\[
\rho'(\text{ovf}) = 1 \lor \rho'(\text{out}) = (f_1 \cdot (f_2 \mod 5)) \mod 5
\]

Problem: choose which values of \( m \) to deduce correctness of concrete result from correctness of all abstract results?
Ensuring Correctness I

Theorem 14.4 (Chinese Remainder Theorem; without proof)

Let \( m_1, \ldots, m_k \geq 1 \) be pairwise relatively prime (i.e., \( \gcd(m_i, m_j) = 1 \) for \( 1 \leq i < j \leq k \)). Let \( m := m_1 \cdot \ldots \cdot m_k \), and let \( z_1, \ldots, z_k \in \mathbb{Z} \). Then there is a unique \( z \in \mathbb{Z} \) such that \( 0 \leq z < m \) and \( z \equiv z_i \pmod{m_i} \) for all \( i \in \{1, \ldots, k\} \).

Application: for fixed initial (abstract) value of \( f_1 \) and \( f_2 \),

- \( z \) = concrete final value of \text{out}  
- \( z_i \) = abstract final value of \text{out} \ (\text{mod} \ m_i)  
- \( k := 5, \ m_1 := 5, \ m_2 := 7, \ m_3 := 9, \ m_4 := 11, \ m_5 := 32 \)  
  (thus \( m = 5 \cdot 7 \cdot 9 \cdot 11 \cdot 32 = 110880 > 2^{16} \))  
- Theorem 14.4 yields unique \( z < m \) with \( z \equiv z_i \pmod{m_i} \)  
- \( m > 2^{16} \implies z \) is correct result of multiplication (see next slide)  
- thus termination implies correct result or overflow

Efficiency:

- Exhaustive testing: \( 2^{16} \cdot 2^{16} = 2^{32} = 4.29 \cdot 10^9 \) runs  
- Abstract interpretation: \( 2^{16} \cdot (5 + 7 + 9 + 11 + 32) = 4.19 \cdot 10^6 \) runs
Ensuring Correctness II

Proof (Correctness of abstraction).

To show: \( \forall y_1, y_2 \in \mathbb{B}^{16}, \sigma, \sigma' \in \Sigma : \sigma(f1) = y_1, \sigma(f2) = y_2, \langle c, \sigma \rangle \rightarrow^+ \langle \downarrow, \sigma' \rangle \)

\[ \implies \sigma'(ovf) = 1 \lor \sigma'(out) = y_1 \cdot y_2 \]

Known: \( \forall i \in \{1, \ldots, 5\}, y_1, y_2 \in \mathbb{B}^{16}, abs, abs' \in Abs : \langle c, abs \rangle \Rightarrow^+ \langle \downarrow, abs' \rangle, \)

\[ abs = \{ [f1 \mapsto y_1, f2 \mapsto y_2^\#, out \mapsto r, ovf \mapsto b] \mid r \in \{0, \ldots, m_i - 1\}, b \in \mathbb{B} \} \]

\[ \implies \left( \forall \rho' \in abs' : \rho'(ovf) = 1 \lor \rho'(out) \equiv (y_1 \cdot y_2^\#)^\# \right) \quad \text{(where } x^\# := x \mod m_i) \]

Proof:

- Let \( y_1, y_2 \in \mathbb{B}^{16}, \sigma(f1) = y_1, \sigma(f2) = y_2, \langle c, \sigma \rangle \rightarrow^+ \langle \downarrow, \sigma' \rangle, \sigma'(ovf) = 0, \) and \( z_i := (y_1 \cdot y_2)^\# \) for \( i \in \{1, \ldots, 5\} \)
- Thm. 14.4 yields unique \( z < m \) such that \( z \equiv z_i \pmod{m_i} \) for all \( i \in \{1, \ldots, 5\} \)
- On the other hand, correctness of modulo abstraction implies \( \rho'(ovf) = 0 \) and \( (\sigma'(out))^\# = \rho'(out) \) (correctness of abstraction)

\[ = (y_1 \cdot y_2^\#)^\# \quad \text{(correctness of abstraction)} \]

\[ = (y_1 \cdot y_2^\#)^\# \quad \text{(Lemma 14.2)} \]

\[ \implies \sigma'(out) = z = y_1 \cdot y_2 \]

\[ \square \]