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 (α, γ) with $\alpha : L \to M$, $\gamma : M \to L$

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

Definition (Abstract semantics of WHILE)

Given $\alpha : 2^{\Sigma} \to Abs$, an abstract semantics is defined by a family of functions

$$\text{next}^{\#}_{c,c'} : Abs \to Abs$$

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

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

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

Extraction Functions

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

$$\begin{aligned} \alpha : L \to M : l \mapsto \beta(l) \quad & (= \{\beta(c) \mid c \in l\}) \\ \gamma : M \to L : m \mapsto \beta^{-1}(m) \quad & (= \{c \in C \mid \beta(c) \in m\}) \end{aligned}$$

Example

1. Parity abstraction (cf. Example 11.2): $\beta : \mathbb{Z} \to \{\text{even}, \text{odd}\}$ 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 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 $\beta : \mathbb{Z} \to A$ be an extraction function.

- An abstract (program) state is an element of the set
$$\{\rho \mid \rho : Var \to A\},$$
called the abstract state space.
- The abstract domain is denoted by $Abs := 2^{Var \to A}$.
- The abstraction function $\alpha : 2^{\Sigma} \to Abs$ is given by
$$\alpha(S) := \{\beta \circ \sigma \mid \sigma \in S\}$$
for every $S \subseteq \Sigma$.

Abstract Evaluation of Expressions

Definition (Abstract evaluation functions)

Let $\rho : Var \to A$ be an abstract state.

1. $val^\#_\rho : AExp \to 2^A$ is determined by (f arithmetic operation)
$$val^\#_\rho(z) := \{\beta(z)\}$$
$$val^\#_\rho(x) := \{\rho(x)\}$$
$$val^\#_\rho(f(a_1, \ldots, a_n)) := f^\#(val^\#_\rho(a_1), \ldots, val^\#_\rho(a_n))$$
2. $val^\#_\rho : BExp \to 2^{\mathbb{B}}$ is determined by (g/h relational/Boolean operation)
$$val^\#_\rho(t) := \{t\}$$
$$val^\#_\rho(g(a_1, \ldots, a_n)) := g^\#(val^\#_\rho(a_1), \ldots, val^\#_\rho(a_n))$$
$$val^\#_\rho(h(b_1, \ldots, b_n)) := h^\#(val^\#_\rho(b_1), \ldots, val^\#_\rho(b_n))$$

Example (Sign abstraction)

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

1. $val^\#_\rho(\mathtt{2 * x + y}) = \{+, -, 0\}$
2. $val^\#_\rho(\neg\mathtt{(x + 1 > y)}) = \{\mathsf{false}\}$

Abstract Semantics of WHILE I

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

Definition (Abstract execution relation for statements)

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

$$\text{(skip)}\frac{}{\langle \mathtt{skip}, abs\rangle \Rightarrow \langle \downarrow, abs\rangle}$$

$$\text{(asgn)}\frac{}{\langle x := a, abs\rangle \Rightarrow \langle \downarrow, \{\rho[x \mapsto a'] \mid \rho \in abs, a' \in val^{\#}_{\rho}(a)\}\rangle}$$

$$\text{(seq1)}\frac{\langle c_1, abs\rangle \Rightarrow \langle c'_1, abs'\rangle \quad c'_1 \neq \downarrow}{\langle c_1 ; c_2, abs\rangle \Rightarrow \langle c'_1 ; c_2, abs'\rangle}$$

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

Abstract Semantics of WHILE II

Definition (Abstract execution relation for statements; continued)

$$\text{(if1)}\ \frac{\exists\rho \in abs : \mathrm{true} \in val^{\#}_{\rho}(b)}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}, abs\rangle \Rightarrow \langle c_1, abs \setminus \{\rho \in abs \mid val^{\#}_{\rho}(b) = \{\mathrm{false}\}\}\rangle}$$

$$\text{(if2)}\ \frac{\exists\rho \in abs : \mathrm{false} \in val^{\#}_{\rho}(b)}{\langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}, abs\rangle \Rightarrow \langle c_2, abs \setminus \{\rho \in abs \mid val^{\#}_{\rho}(b) = \{\mathrm{true}\}\}\rangle}$$

$$\text{(wh1)}\ \frac{\exists\rho \in abs : \mathrm{true} \in val^{\#}_{\rho}(b)}{\begin{array}{l}\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, abs\rangle \\ \Rightarrow \langle c\texttt{;while}\ b\ \texttt{do}\ c\ \texttt{end}, abs \setminus \{\rho \in abs \mid val^{\#}_{\rho}(b) = \{\mathrm{false}\}\}\rangle\end{array}}$$

$$\text{(wh2)}\ \frac{\exists\rho \in abs : \mathrm{false} \in val^{\#}_{\rho}(b)}{\langle \texttt{while}\ b\ \texttt{do}\ c\ \texttt{end}, abs\rangle \Rightarrow \langle \downarrow, abs \setminus \{\rho \in abs \mid val^{\#}_{\rho}(b) = \{\mathrm{true}\}\}\rangle}$$

Correctness of Abstract Semantics

Theorem (Soundness of abstract semantics)

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

The soundness proof employs the following auxiliary lemma.

Lemma (Soundness of abstract evaluation)

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

1. *For every* $a \in AExp$ *and* $\sigma \in \Sigma$, $\beta(val_\sigma(a)) \in val^{\#}_{\beta\circ\sigma}(a)$.
2. *For every* $b \in BExp$ *and* $\sigma \in \Sigma$, $val_\sigma(b) \in val^{\#}_{\beta\circ\sigma}(b)$.

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)

$$
\begin{array}{l}
c = [\texttt{out:=0}]^1;[\texttt{ovf:=0}]^2;\\
\quad \texttt{while } [\neg\texttt{(f1=0)} \wedge \texttt{ ovf=0}]^3 \texttt{ do}\\
\quad\quad \texttt{if } [\texttt{lsb(f1)=1}]^4 \texttt{ then}\\
\quad\quad\quad [\texttt{(ovf,out):=(out:17)+f2}]^5\\
\quad\quad \texttt{else}\\
\quad\quad\quad [\texttt{skip}]^6\\
\quad\quad \texttt{end;}\\
\quad\quad [\texttt{f1:=f1>>1}]^7;\\
\quad\quad \texttt{if } [\neg\texttt{(f1=0)} \wedge \texttt{ ovf=0}]^8 \texttt{ then}\\
\quad\quad\quad [\texttt{(ovf,f2):=(f2:17)<<1}]^9\\
\quad\quad \texttt{else}\\
\quad\quad\quad [\texttt{skip}]^{10}\\
\quad\quad \texttt{end}\\
\quad \texttt{end}
\end{array}
$$

Inputs:

- `f1`, `f2`: 16-bit input factors

Outputs:

- `out`: 16-bit result
- `ovf`: overflow bit

Operations:

- `lsb(`z`)`: least significant bit of z
- `(`z`:`k`)`: extension of z to k bits by adding leading zeros
- `(`x`,`y`) :=` z: simultaneous assignment with split of z
- `<<1` / `>>1`: left/right shift by 1 bit

Procedure: in each iteration,

1. if LSB of `f1` is set (4), add `f2` to `out` (5)
2. shift `f1` right (7)
3. shift `f2` left (9)

Expected result:

if $\langle c, \sigma\rangle \to^{+} \langle \downarrow, \sigma'\rangle$, then

- $\sigma'(\texttt{out}) = \sigma(\texttt{f1}) \cdot \sigma(\texttt{f2})$ or
- $\sigma'(\texttt{ovf}) = 1$

(termination is trivial)

Example run: on the board

Application Example: 16-Bit Multiplication

The Abstraction

(see E.M. Clarke, O. Grumberg, D.A. Peled: *Model Checking*, MIT Press, 1999, pp. 205)

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

$$(z_1 + z_2) \bmod m \equiv ((z_1 \bmod m) + (z_2 \bmod m)) \bmod m$$
$$(z_1 - z_2) \bmod m \equiv ((z_1 \bmod m) - (z_2 \bmod m)) \bmod m$$
$$(z_1 \cdot z_2) \bmod m \equiv ((z_1 \bmod m) \cdot (z_2 \bmod m)) \bmod m$$

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

- $\mathtt{f1} = 101_2 \ (= 5)$, $\mathtt{f2} = 1001010_2 \ (= 74)$
- $\mathtt{out}$, $\mathtt{ovf}$ with arbitrary initial values
- $m = 5$

$\Rightarrow$ initial abstract value:

$$abs = \{[\mathtt{f1} \mapsto 101_2, \mathtt{f2} \mapsto \underbrace{74 \bmod 5}_{4}, \mathtt{out} \mapsto r, \mathtt{ovf} \mapsto b] \mid r \in \{0, \ldots, 4\}, b \in \mathbb{B}\}$$

- first transitions: on the board
- generally: for all initial (abstract) values of $\mathtt{f1}$ and $\mathtt{f2}$, abstract results $\langle\downarrow, abs'\rangle$, and $\rho' \in abs'$,

$$\rho'(\mathtt{ovf}) = 1 \vee \rho'(\mathtt{out}) = (\mathtt{f1} \cdot (\mathtt{f2} \bmod 5)) \bmod 5$$

Problem: choose which values of m to deduce correctness of concrete result from correctness of all abstract results?

Application Example: 16-Bit Multiplication

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 `f1` and `f2`,

- z = concrete final value of `out`
- z_i = abstract final value of `out` $\pmod{m_i}$
- $k := 5$, $m_1 := 5$, $m_2 := 7$, $m_3 := 9$, $m_4 := 11$, $m_5 := 32$ (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: $\forall y_1, y_2 \in \mathbb{B}^{16}, \sigma, \sigma' \in \Sigma : \sigma(\texttt{f1}) = y_1, \sigma(\texttt{f2}) = y_2, \langle c, \sigma \rangle \to^+ \langle \downarrow, \sigma' \rangle$
$\implies \sigma'(\texttt{ovf}) = 1 \vee \sigma'(\texttt{out}) = y_1 \cdot y_2$

Known: $\forall i \in \{1, \ldots, 5\}, y_1, y_2 \in \mathbb{B}^{16}, abs, abs' \in Abs : \langle c, abs \rangle \Rightarrow^+ \langle \downarrow, abs' \rangle,$
$abs = \{[\texttt{f1} \mapsto y_1, \texttt{f2} \mapsto y_2^\#, \texttt{out} \mapsto r, \texttt{ovf} \mapsto b] \mid r \in \{0, \ldots, m_i - 1\}, b \in \mathbb{B}\}$
$\implies \left(\forall \rho' \in abs' : \rho'(\texttt{ovf}) = 1 \vee \rho'(\texttt{out}) \overset{(*)}{=} (y_1 \cdot y_2^\#)^\#\right)$ (where $x^\# := x \bmod m_i$)

Proof:
- Let $y_1, y_2 \in \mathbb{B}^{16}$, $\sigma(\texttt{f1}) = y_1$, $\sigma(\texttt{f2}) = y_2$, $\langle c, \sigma \rangle \to^+ \langle \downarrow, \sigma' \rangle$, $\sigma'(\texttt{ovf}) = 0$, 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 $\rho'(\texttt{ovf}) = 0$ and

$$\begin{aligned}(\sigma'(\texttt{out}))^\# &= \rho'(\texttt{out}) && \text{(correctness of abstraction)}\\ &= (y_1 \cdot y_2^\#)^\# && (*)\\ &= (y_1 \cdot y_2)^\# && \text{(Lemma 14.2)}\end{aligned}$$

$\implies \sigma'(\texttt{out}) = z = y_1 \cdot y_2$

$\square$