



Static Program Analysis

**Lecture 13: Abstract Interpretation IV
(Application Example: 16-Bit Multiplication)**

Summer Semester 2018

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-18/spa/>

Recap: Abstract Semantics of WHILE

Extraction Functions

Assumption: abstraction determined by **pointwise mapping** of concrete values

Definition (Extraction function)

- If $L = 2^C$ and $M = 2^A$ with $\sqsubseteq_L = \sqsubseteq_M = \subseteq$, then $\beta : C \rightarrow A$ is called an **extraction function**
- β determines **Galois connection** (α, γ) with

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

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

Example

1. Parity abstraction (cf. Ex. 10.2): $\beta : \mathbb{Z} \rightarrow \{\text{even}, \text{odd}\}$ 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. Ex. 10.3): $\beta : \mathbb{Z} \rightarrow \{+, -, 0\}$ with $\beta = \text{sgn}$
3. Interval abstraction (cf. Ex. 10.4): 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 $\beta : \mathbb{Z} \rightarrow A$ 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 Semantics of WHILE I

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

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:

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

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

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

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

Recap: Abstract Semantics of WHILE

Abstract Semantics of WHILE II

Definition (Abstract execution relation for statements; continued)

$$\exists \rho \in abs : true \in val_{\rho}^{\#}(b)$$

$$\text{(if1)} \frac{}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end, } abs \rangle \Rightarrow \langle c_1, abs \setminus \{ \rho \in abs \mid val_{\rho}^{\#}(b) = \{false\} \} \rangle}$$

$$\exists \rho \in abs : false \in val_{\rho}^{\#}(b)$$

$$\text{(if2)} \frac{}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end, } abs \rangle \Rightarrow \langle c_2, abs \setminus \{ \rho \in abs \mid val_{\rho}^{\#}(b) = \{true\} \} \rangle}$$

$$\exists \rho \in abs : true \in val_{\rho}^{\#}(b)$$

$$\text{(wh1)} \frac{}{\langle \text{while } b \text{ do } c \text{ end, } abs \rangle \Rightarrow \langle c; \text{while } b \text{ do } c \text{ end, } abs \setminus \{ \rho \in abs \mid val_{\rho}^{\#}(b) = \{false\} \} \rangle}$$

$$\exists \rho \in abs : false \in val_{\rho}^{\#}(b)$$

$$\text{(wh2)} \frac{}{\langle \text{while } b \text{ do } c \text{ end, } abs \rangle \Rightarrow \langle \downarrow, abs \setminus \{ \rho \in abs \mid val_{\rho}^{\#}(b) = \{true\} \} \rangle}$$

Recap: Abstract Semantics of WHILE

Abstract Semantics of WHILE III

Definition (Abstract transition function)

The **abstract transition function** is defined by the family of mappings

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

given by $\text{next}_{c,c'}^{\#}(abs) := \bigcup \{abs' \in Abs \mid \langle c, abs \rangle \Rightarrow \langle c', abs' \rangle\}$

Example (Hailstone Sequences; cf. Example 11.7)

```
[skip]1;  
while [¬(n = 1)]2 do  
  if [even(n)]3 then  
    [n := n / 2]4; [skip]5  
  else  
    [n := 3 * n + 1]6; [skip]7  
  end  
end
```

Execution relation with parity abstraction:
see following slide (courtesy B. König)

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

Proof (Lemma 12.14).

omitted

Proof (Theorem 12.13).

on the board

Application Example: 16-Bit Multiplication

A 16-Bit Multiplier

Example 13.1 (16-bit multiplier)

```
c = [out:=0]1; [ovf:=0]2;
  while [¬(f1=0) ∧ ovf=0]3 do
    if [lsb(f1)=1]4 then
      [(ovf, out) := (out:17)+f2]5
    else
      [skip]6
    end;
    [f1:=f1>>1]7;
    if [¬(f1=0) ∧ ovf=0]8 then
      [(ovf, f2) := (f2:17)<<1]9
    else
      [skip]10
    end
  end
end
```

Inputs:

- f_1, f_2 : 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 to x and y with split of z
- $\ll 1 / \gg 1$: left/right shift by 1 bit

Procedure: in each iteration,

1. if LSB of f_1 is set (4), add f_2 to out (5)
2. shift f_1 right (7)
3. shift f_2 left (9)

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 13.4)
 - **extraction function**: $\beta : \mathbb{Z} \rightarrow \{0, \dots, m-1\} : z \mapsto z \bmod m$
 - **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 13.2 (Properties of modulo congruence)

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

$$\begin{aligned}(z_1 + z_2) \bmod m &\equiv ((z_1 \bmod m) + (z_2 \bmod m)) \bmod m \\(z_1 - z_2) \bmod m &\equiv ((z_1 \bmod m) - (z_2 \bmod m)) \bmod m \\(z_1 \cdot z_2) \bmod m &\equiv ((z_1 \bmod m) \cdot (z_2 \bmod m)) \bmod m\end{aligned}$$

Thus: modulo value of expression determined by modulo values of subexpressions

Application Example: 16-Bit Multiplication

Abstract Interpretation of Multiplier

Example 13.3 (Abstraction of 16-bit multiplier; cf. Example 13.1)

Abstract execution for

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

⇒ initial abstract value:

$$abs_0 = \{[f1 \mapsto 101_2, f2 \mapsto \underbrace{74 \bmod 5}_4, out \mapsto r, ovf \mapsto b] \mid r \in \{0, \dots, 4\}, b \in \mathbb{B}\}$$

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

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

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

Application Example: 16-Bit Multiplication

Ensuring Correctness I

Theorem 13.4 (Chinese Remainder Theorem; without proof)

Let $m_1, \dots, m_k \geq 1$ be pairwise relatively prime (i.e., $\gcd(m_i, m_j) = 1$ for $1 \leq i < j \leq k$). Let $m := m_1 \cdot \dots \cdot m_k$, and let $z_1, \dots, z_k \in \mathbb{Z}$. 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 $i \in \{1, \dots, k\}$.

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 13.4 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

Application Example: 16-Bit Multiplication

Ensuring Correctness II

Proof (Correctness of abstraction).

To show: $\forall y_1, y_2 \in \mathbb{B}^{16}, \sigma, \sigma' \in \Sigma : \sigma(\text{f1}) = y_1, \sigma(\text{f2}) = y_2, \langle c, \sigma \rangle \rightarrow^+ \langle \downarrow, \sigma' \rangle$
 $\implies \sigma'(\text{ovf}) = 1 \vee \sigma'(\text{out}) = y_1 \cdot y_2$

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

Proof: • Let $y_1, y_2 \in \mathbb{B}^{16}, \sigma(\text{f1}) = y_1, \sigma(\text{f2}) = y_2, \langle c, \sigma \rangle \rightarrow^+ \langle \downarrow, \sigma' \rangle, \sigma'(\text{ovf}) = 0,$ and
 $z_i := (y_1 \cdot y_2)^\#$ for $i \in \{1, \dots, 5\}$

- Thm. 13.4 yields unique $z < m$ such that $z \equiv z_i \pmod{m_i}$ for all $i \in \{1, \dots, 5\}$
- On the other hand, correctness of modulo abstraction implies $\rho'(\text{ovf}) = 0$ and
 $(\sigma'(\text{out}))^\# = \rho'(\text{out})$ (correctness of abstraction)
 $= (y_1 \cdot y_2^\#)^\#$ (*)
 $= (y_1 \cdot y_2)^\#$ (Lemma 13.2)

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

□

Overview of Numerical Abstraction Domains

Non-Relational Abstraction Domains

Here, abstract values are independently referring to single variables.

Example 13.5 (Non-relational domains)

- **Signs** (cf. Example 10.3): $\text{sgn}(x) = s$ ($x \in \text{Var}$, $s \in \{+, -, 0\}$)
- **Intervals** (cf. Example 10.4): $x \in J$ ($x \in \text{Var}$, $J \in (\mathbb{Z} \cup \{-\infty\}) \times (\mathbb{Z} \cup \{+\infty\}) \cup \{\emptyset\}$)
- **Parities** (cf. Example 10.2): $x \in \mathbb{Z}_p$ ($x \in \text{Var}$, $p \in \{\text{even}, \text{odd}\}$)
- **Congruences** (cf. Lemma 13.2): $x \bmod m = k$ ($x \in \text{Var}$, $m > 1$, $k \in \{0, \dots, m - 1\}$)

Observations

- Expressive power:
 - Signs $<$ Intervals (since $+ \cong [1, +\infty]$, ...)
 - Parities $<$ Congruences (since $x \text{ even} \iff x \bmod 2 = 0$, ...)
 - Intervals and Congruences are “incomparable”
- Congruences can prove disequalities (“ $x \neq y$ ”) but not inequalities (“ $x \leq y$ ”)
 - e.g., $x \bmod m = k_x$, $y \bmod m = k_y$, $k_x \neq k_y \implies$ no zero division in $1/(x - y)$
 - but $x \leq y$ generally not representable
- Non-relational domains efficient to represent and manipulate
- Signs, Parities, and Congruences definable by extraction functions

Overview of Numerical Abstraction Domains

Relational Abstraction Domains

Here, interdependencies between variables are captured:

Example 13.6 (Relational domains)

- **Difference Bound Matrices (DBMs)**: conjunctions of $x - y \leq c$ and $\pm x \leq c$ ($x, y \in Var, c \in \mathbb{Z}$)
- **Octagons**: conjunctions of $ax + by \leq c$ ($x, y \in Var, a, b \in \{-1, 0, 1\}, c \in \mathbb{Z}$)
- **Octahedra**: conjunctions of $a_1x_1 + \dots + a_nx_n \leq c$ ($x_i \in Var, a_i \in \{-1, 0, 1\}, c \in \mathbb{Z}$)
- **Polyhedra**: conjunctions of $a_1x_1 + \dots + a_nx_n \leq c$ ($x_i \in Var, a_i \in \mathbb{Z}, c \in \mathbb{Z}$)

Observations

- Expressive power:
 - DBMs < Octagons < Octahedra < Polyhedra
 - Intervals < DBMs (since $x \in [c_1, c_2] \iff -x \leq -c_1 \wedge x \leq c_2$)
- Can prove inequalities but not (general) disequalities
- Representation and manipulation generally more involved
 - Polyhedra require computation of convex hulls (exponential in $|Var|$)

Overview of Numerical Abstraction Domains

Combining Non-Relational and Relational Domains

Linear Congruences combine features of Congruences and Polyhedra:

- Given by conjunctions of

$$(a_1x_1 + \dots + a_nx_n) \bmod m = z$$

$$(x_i \in \text{Var}, a_i \in \mathbb{Z}, m > 1, z \in \mathbb{Z})$$

- Typical application:

$$2x + 1 \bmod m = k_x, y \bmod m = k_y, k_x \neq k_y \implies \text{no zero division in } 1/(2x + 1 - y)$$

- Again usable for proving disequalities but not inequalities