Static Program Analysis

Lecture 15: Abstract Interpretation IV (Correctness of Abstract Semantics)

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

Outline

Safe Approximation of Execution Relation

- **Reminder:** abstraction determined by Galois connection (α, γ) with $\alpha : L \to M$ and $\gamma : M \to L$
  - here: $L := 2^\Sigma$, M not fixed (usually $M = Var \to \ldots$ or $M = 2^{Var \to \cdots}$)
  - write Abs in place of M
  - thus $\alpha : 2^\Sigma \to Abs$ and $\gamma : Abs \to 2^\Sigma$
- Yields abstract semantics:

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 $\text{next}^\#_{c,c'}(abs) = abs'$ and
- $\langle c, abs \rangle \Rightarrow abs'$ for $\text{next}^\#_{c,\downarrow}(a) = abs'$

Extraction Functions

- **Assumption:** abstraction determined by pointwise mapping of concrete elements
- 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
$$\alpha : L \to M : l \mapsto \{\beta(c) \mid c \in l\}$$
and
$$\gamma : M \to L : m \mapsto \beta^{-1}(m) \; (= \{c \in C \mid \beta(c) \in m\})$$

Example

1. Parity abstraction (cf. Example 12.2): $\beta : \mathbb{Z} \to \{\text{even}, \text{odd}\}$ 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 12.3): $\beta : \mathbb{Z} \to \{+, -, 0\}$ with $\beta = \text{sgn}$
3. Interval abstraction (cf. Example 12.4): not definable by extraction function (as *Int* is not of the form 2^A)

Safe Approximation by Extraction Functions

Reminder: safe approximation condition (Definition 13.3)

$$\alpha(f(\gamma(m_1), \ldots, \gamma(m_n))) \sqsubseteq_M f^\#(m_1, \ldots, m_n).$$

Theorem

Let $L = 2^C$ and $M = 2^A$ with $\sqsubseteq_L = \sqsubseteq_M = \subseteq$, $\beta : C \to A$ be an extraction function, and $f : C^n \to C$. Then

$$f^\# : M^n \to M : (m_1, \ldots, m_n) \mapsto \{\beta(f(c_1, \ldots, c_n)) \mid \forall i \in \{1, \ldots, n\} : c_i \in \beta^{-1}(m_i)\}$$

is a safe approximation of f.

Proof.

on the board □

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

$$\begin{aligned} val^\#_\rho(z) &:= \{\beta(z)\} \\ val^\#_\rho(x) &:= \{\rho(x)\} \\ val^\#_\rho(f(a_1, \ldots, a_n)) &:= f^\#(val^\#_\rho(a_1), \ldots, val^\#_\rho(a_n)) \end{aligned}$$

2. $val^\#_\rho : BExp \to 2^{\mathbb{B}}$ is determined by

$$\begin{aligned} val^\#_\rho(t) &:= \{t\} \\ val^\#_\rho(f(a_1, \ldots, a_n)) &:= f^\#(val^\#_\rho(a_1), \ldots, val^\#_\rho(a_n)) \\ val^\#_\rho(g(b_1, \ldots, b_n)) &:= g^\#(val^\#_\rho(b_1), \ldots, val^\#_\rho(b_n)) \end{aligned}$$

Example (Sign abstraction)

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

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

Outline

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

Definition 15.1 (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:

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

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

$$\text{(seq1)}\ \frac{\langle c_1, abs \rangle \Rightarrow \langle c_1', abs' \rangle}{\langle c_1 \texttt{;} c_2, abs \rangle \Rightarrow \langle c_1' \texttt{;} c_2, abs' \rangle}$$

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

Definition 15.1 (Abstract execution relation for statements; cont.)

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

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

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

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

Definition 15.2 (Abstract transition function)

The abstract transition function is defined by the family of mappings

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

given by

$$\mathrm{next}^{\#}_{c,c'}(abs) := \bigsqcup\{abs' \in Abs \mid \langle c, abs\rangle \Rightarrow \langle c', abs'\rangle\}$$
$$\mathrm{next}^{\#}_{c,\downarrow}(abs) := \bigsqcup\{abs' \in Abs \mid \langle c, abs\rangle \Rightarrow abs'\}$$

Theorem 15.3 (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 $abs \in Abs$,

$$\alpha(\mathrm{next}_{c,c'}(\gamma(abs))) \sqsubseteq \mathrm{next}^{\#}_{c,c'}(abs).$$

The soundness proof employs the following auxiliary lemma.

Lemma 15.4 (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 15.4).

omitted □

Proof (Theorem 15.3).

on the board □