Abstract interpretation with numeric intervals

Second assignment of the Software Verification course, A.Y. 2022/2023

Christian Micheletti
October 24, 2023

Università degli Studi di Padova

Outline

The Language

The language is a variation of the While language seen in class. It differs on:

- it admits some syntactic sugar (it's not minimal);
- its semantic functions are modified to allow divergence and state changes in both arithmetic and boolean expressions.

Arithmetic Expressions (1)

$$\begin{aligned} AExp ::= & n \mid x \mid -e \mid (e) \\ & \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 / e_2 \\ & \mid x{+}{+} \mid {+}{+}x \mid x{-}{-} \mid {-}{-}x \end{aligned}$$

$\mathcal{A} : AExp \to State \hookrightarrow \mathbb{Z} \times State$

$$\begin{aligned} \mathcal{A}[\![n]\!]\varphi &= (n_{\mathbb{Z}}, \varphi) \\ \mathcal{A}[\![x]\!]\varphi &= (\varphi(x), \varphi) \\ \mathcal{A}[\![(e)]\!]\varphi &= \mathcal{A}[\![e]\!]\varphi \\ \mathcal{A}[\![-e]\!]\varphi &= \begin{cases} (-a, \varphi') & \mathcal{A}[\![e]\!]\varphi = (a, \varphi') \\ \uparrow & (\mathcal{A}[\![e]\!]\varphi)\uparrow \end{cases} \end{aligned}$$

Arithmetic Expressions (2)

$\mathcal{A} : AExp \to State \hookrightarrow \mathbb{Z} \times State$

$$\mathcal{A}[\![e_1/e_2]\!]\varphi = \begin{cases} (a_1 \div a_2, \varphi'') & \mathcal{A}[\![e_1]\!]\varphi = (a_1, \varphi') \\ & \wedge\ \mathcal{A}[\![e_2]\!]\varphi' = (a_2, \varphi'') \\ & \wedge\ a_2 \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mathcal{A}[\![e_1\ \mathbf{op}\ e_2]\!]\varphi = \begin{cases} (a_1\ op\ a_2, \varphi'') & \mathcal{A}[\![e_1]\!]\varphi = (a_1, \varphi') \\ & \wedge\ \mathcal{A}[\![e_2]\!]\varphi' = (a_2, \varphi'') \\ \uparrow & \text{otherwise} \end{cases}$$

Arithmetic Expressions (3)

$\mathcal{A} : AExp \to State \hookrightarrow \mathbb{Z} \times State$

$$
\begin{aligned}
\mathcal{A}[\![x{+}{+}]\!]\varphi &= (\varphi(x), \varphi[x \mapsto x+1]) \\
\mathcal{A}[\![{+}{+}x]\!]\varphi &= \mathit{let}\ \varphi' = \varphi[x \mapsto x+1] \\
&\quad\ \mathit{in}\ (\varphi'(x), \varphi') \\
\mathcal{A}[\![x{-}{-}]\!]\varphi &= (\varphi(x), \varphi[x \mapsto x-1]) \\
\mathcal{A}[\![{-}{-}x]\!]\varphi &= \mathit{let}\ \varphi' = \varphi[x \mapsto x-1] \\
&\quad\ \mathit{in}\ (\varphi'(x), \varphi')
\end{aligned}
$$

Boolean Expressions (1)

$$BExp ::= \texttt{true} \mid \texttt{false} \mid (b) \mid b_1 \texttt{ and } b_2 \mid b_1 \texttt{ or } b_2 \mid e_1 \texttt{ = } e_2 \mid e_1 \texttt{ != } e_2 \mid e_1 \texttt{ < } e_2 \mid e_1 \texttt{ >= } e_2$$

$\mathcal{B} : BExp \to State \hookrightarrow \mathbb{T} \times State$

$$\mathcal{B}[\![\texttt{true}]\!]\varphi = (\mathbf{tt}, \varphi)$$
$$\mathcal{B}[\![\texttt{false}]\!]\varphi = (\mathbf{ff}, \varphi)$$
$$\mathcal{B}[\![(b)]\!]\varphi = \mathcal{B}[\![b]\!]\varphi$$

Boolean Expressions (2)

Operators between booleans short circuits results:

$\mathcal{B} : \mathit{BExp} \to \mathit{State} \hookrightarrow \mathbb{T} \times \mathit{State}$

$$\mathcal{B}[\![b_1 \texttt{ and } b_2]\!]\varphi = \begin{cases} (\mathbf{ff}, \varphi') & \mathcal{B}[\![b_1]\!]\varphi = (\mathbf{ff}, \varphi') \\ \mathcal{B}[\![b_2]\!]\varphi' & \mathcal{B}[\![b_1]\!]\varphi = (\mathbf{tt}, \varphi') \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mathcal{B}[\![b_1 \texttt{ or } b_2]\!]\varphi = \begin{cases} (\mathbf{tt}, \varphi') & \mathcal{B}[\![b_1]\!]\varphi = (\mathbf{tt}, \varphi') \\ \mathcal{B}[\![b_2]\!]\varphi' & \mathcal{B}[\![b_1]\!]\varphi = (\mathbf{ff}, \varphi') \\ \uparrow & \text{otherwise} \end{cases}$$

Boolean Expressions (3)

Comparison operations propagate updates in the state:

$$\mathcal{B} : \mathit{BExp} \to \mathit{State} \hookrightarrow \mathbb{T} \times \mathit{State}$$

$$\mathcal{B}[\![e_1 \texttt{ = } e_2]\!]\varphi = \begin{cases} (a_1 = a_2, \varphi'') & \mathcal{A}[\![e_1]\!]\varphi = (a_1, \varphi') \\ & \wedge\ \mathcal{A}[\![e_2]\!]\varphi' = (a_2, \varphi'') \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mathcal{B}[\![e_1 \texttt{ < } e_2]\!]\varphi = \begin{cases} (a_1 < a_2, \varphi'') & \mathcal{A}[\![e_1]\!]\varphi = (a_1, \varphi') \\ & \wedge\ \mathcal{A}[\![e_2]\!]\varphi' = (a_2, \varphi'') \\ \uparrow & \text{otherwise} \end{cases}$$

Boolean Expressions (4)

$\mathcal{B} : BExp \to State \hookrightarrow \mathbb{T} \times State$

$$\mathcal{B}[\![e_1 \texttt{ != } e_2]\!]\varphi = \begin{cases} (a_1 \neq a_2, \varphi'') & \mathcal{A}[\![e_1]\!]\varphi = (a_1, \varphi') \\ & \wedge\ \mathcal{A}[\![e_2]\!]\varphi' = (a_2, \varphi'') \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mathcal{B}[\![e_1 \texttt{ >= } e_2]\!]\varphi = \begin{cases} (a_1 \geq a_2, \varphi'') & \mathcal{A}[\![e_1]\!]\varphi = (a_1, \varphi') \\ & \wedge\ \mathcal{A}[\![e_2]\!]\varphi' = (a_2, \varphi'') \\ \uparrow & \text{otherwise} \end{cases}$$

Boolean Expressions (5)

Negation is expressed by syntactic sugar:

$$\texttt{not true} \stackrel{\text{def}}{=} \texttt{false}$$

$$\texttt{not false} \stackrel{\text{def}}{=} \texttt{true}$$

$$\texttt{not } (b_1 \texttt{ and } b_2) \stackrel{\text{def}}{=} \texttt{not } b_1 \texttt{ or not } b_2$$

$$\texttt{not } (b_1 \texttt{ or } b_2) \stackrel{\text{def}}{=} \texttt{not } b_1 \texttt{ and not } b_2$$

$$\texttt{not } e_1 \texttt{ = } e_2 \stackrel{\text{def}}{=} e_1 \texttt{ != } e_2$$

$$\texttt{not } e_1 \texttt{ < } e_2 \stackrel{\text{def}}{=} e_1 \texttt{ >= } e_2$$

$$\texttt{not } e_1 \texttt{ != } e_2 \stackrel{\text{def}}{=} e_1 \texttt{ = } e_2$$

$$\texttt{not } e_1 \texttt{ >= } e_2 \stackrel{\text{def}}{=} e_1 \texttt{ < } e_2$$

Boolean Expressions (6)

Also other arithmetic comparisons are expressed with syntactic sugar:

$$e_1 \texttt{ > } e_2 \stackrel{\text{def}}{=} e_2 \texttt{ < } e_1$$

$$e_1 \texttt{ <= } e_2 \stackrel{\text{def}}{=} e_2 \texttt{ >= } e_1$$

Statements (1)

$$\textbf{While} ::= x \texttt{ := } e \mid \texttt{ skip } \mid \{S\} \mid S_1 \texttt{ ; } S_2 \mid \texttt{ if } b \texttt{ then } S_1 \texttt{ else } S_2 \mid \texttt{ while } b \texttt{ do } S$$

$\mathcal{S}_{ds} : \mathit{While} \to \mathit{State} \hookrightarrow \mathit{State}$

$$\mathcal{S}_{ds}[\![x \texttt{ := } e]\!]\varphi = \begin{cases} \varphi'[x \mapsto a] & \mathcal{A}[\![e]\!]\varphi = (a, \varphi') \\ \uparrow & \text{otherwise} \end{cases}$$

$$\mathcal{S}_{ds}[\![\texttt{skip}]\!]\varphi = \varphi$$

$$\mathcal{S}_{ds}[\![\{S\}]\!]\varphi = \mathcal{S}_{ds}[\![S]\!]\varphi$$

Statements (2)

$\mathcal{S}_{ds} : \mathit{While} \to \mathit{State} \hookrightarrow \mathit{State}$

$$\begin{aligned}
\mathcal{S}_{ds}[\![S_1 \texttt{ ; } S_2]\!]\varphi &= (\mathcal{S}_{ds}[\![S_2]\!] \circ \mathcal{S}_{ds}[\![S_1]\!])\varphi \\
\mathcal{S}_{ds}[\![\texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2]\!]\varphi &= \mathit{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{ds}[\![S_1]\!], \mathcal{S}_{ds}[\![S_2]\!]) \\
\mathcal{S}_{ds}[\![\texttt{while } b \texttt{ do } S]\!]\varphi &= \mathsf{FIX}(\lambda g.\mathit{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{ds}[\![S]\!], \mathit{id}))
\end{aligned}$$

Where

$$\mathit{cond}(\mathit{pred}, g_1, g_2) = \begin{cases} g_1(\varphi') & \mathit{pred}(\varphi) = (\mathbf{tt}, \varphi') \\ g_2(\varphi') & \mathit{pred}(\varphi) = (\mathbf{ff}, \varphi') \\ \uparrow & \text{otherwise} \end{cases}$$

Abstract States (1)

We define for any abstract domain A the abstract state type $\mathbb{S}_A = \mathit{Var} \times A$.

Assumption

The absence of a variable in the abstract state is interpreted as $\top$. This is due to the fact that we assume that all referenced variables in the program are initialized.

Abstract States (2)

Moreover, $\bot_{\mathbb{S}_A}$ represents a runtime error, therefore no update operation can be performed over this state:

$$s(x) = \begin{cases} a & (x, a) \in s \\ \top_A & \textit{otherwise} \end{cases}$$

$$s[x \mapsto a] = \begin{cases} \bot_{\mathbb{S}_A} & s = \bot_{\mathbb{S}_A} \\ \{(k, v) \mid (k, v) \in s,\ k \neq x\} & a \neq \top,\ s \neq \bot_{\mathbb{S}_A} \\ \{(k, v) \mid (k, v) \in s,\ k \neq x\} & \textit{otherwise} \end{cases}$$

Abstract States (3)

$\mathbb{S}$ requires that A is a complete lattice, and it is a complete lattice as well:

$$
\begin{aligned}
\bot_{\mathbb{S}_A} &= \{(x, \bot_A) \mid x \in \mathit{Var}\} \\
\top_{\mathbb{S}_A} &= \emptyset \\
s_1 \vee_{\mathbb{S}_A} s_2 &= \{(\mathit{var}, a_1 \vee_A a_2) \mid (\mathit{var}, a_1) \in s_1, (\mathit{var}, a_2) \in s_2\} \\
s_1 \wedge_{\mathbb{S}_A} s_2 &= \{(\mathit{var}, a_1 \wedge_A a_2) \mid (\mathit{var}, a_1) \in s_1, (\mathit{var}, a_2) \in s_2\} \\
&\quad \cup \{e \mid e \in s_1, e \notin s_2\} \\
&\quad \cup \{e \mid e \in s_2, e \notin s_1\}
\end{aligned}
$$

Introduction

Etiam eu interdum ligula
Nunc mi eros, vulputate in ornare a, viverra eget quam

- Morbi **vitae lacus** porta neque tincidunt sodales
- Proin tincidunt, **neque** at tincidunt mollis
- Ut lacinia sem a nibh consequat porttitor

First section

Normal block

Fusce luctus venenatis felis quis semper

Alert block

$$E = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_4)$$

Example block

Proin tincidunt, neque at tincidunt mollis