Program verification 2019

Lecture 2: Symbolic operators

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2019-01-08

Topic 2.1

Logical representation

Computing reachable states

- Proving safety is computing reachable states.
- states are infinite $\implies$ enumeration impossible
- To compute reachable states, we need
  - finite representations of transition relation and
  - ability to compute transitive closure of transition relation
- Idea: use logic for the above goals

Program statements as formulas (Notation)

- In logical representation, we add a new variable *err* in V to represent error state. Initially, $err = 0$ and $err = 1$ means error has occurred.
- V' be the vector of variables obtained by adding prime after each variable in V. We use V' to denote next value of variables.
- $$\text{For } U \subseteq V, \text{ let } frame(U) \triangleq \bigwedge_{x \in V \setminus U} (x' = x)$$

  In case of singleton U, we only write the element as parameter.

Program statements as formulas (contd.)

We define logical formula ρ for the data statements as follows.

- $\rho(\texttt{x := exp}) \triangleq \texttt{x}' = \texttt{exp} \land \mathit{frame}(\texttt{x})$
- $\rho(\texttt{x := havoc()}) \triangleq \mathit{frame}(\texttt{x})$
- $\rho(\texttt{assume(F)}) \triangleq \texttt{F} \land \mathit{frame}(\emptyset)$
- $\rho(\texttt{assert(F)}) \triangleq \texttt{F} \Rightarrow \mathit{frame}(\emptyset)$

Since control locations in a program are always finite, control statements need not be redefined.

Example 2.1

Let $V = [x, y, err]$.

- $\rho(\texttt{x := y} + 1) = (\texttt{x}' = \texttt{y} + 1 \land \texttt{y}' = \texttt{y} \land err' = err)$
- $\rho(\texttt{x := havoc()}) = (\texttt{y}' = \texttt{y} \land err' = err)$
- $\rho(\texttt{assume(x} > 0\texttt{)}) = (\texttt{x} > 0 \land \texttt{x}' = \texttt{x} \land \texttt{y}' = \texttt{y} \land err' = err)$
- $\rho(\texttt{assert(x} > 0\texttt{)}) = (\texttt{x} > 0 \Rightarrow (\texttt{x}' = \texttt{x} \land \texttt{y}' = \texttt{y} \land err' = err))$

Exercise 2.1

Show ρ correctly models the assert statement

Topic 2.2

Aggregated semantics

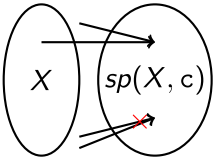
Strongest post: set of valuations to set of valuations

Definition 2.1

Strongest post operator $sp : \mathfrak{p}(\mathbb{Q}^{|V|}) \times \mathcal{P} \to \mathfrak{p}(\mathbb{Q}^{|V|})$ *is defined as follows.*

$$sp(X, \mathtt{c}) \triangleq \{v' | \exists v : v \in X \land (v', \mathtt{skip}) \in T^*((v, \mathtt{c}))\},$$

where $X \subseteq \mathbb{Q}^{|V|}$ *and* $\mathtt{c}$ *is a program.*



Example 2.2

Consider $V = [\mathrm{x}]$ *and* $X = \{[n] | n > 0\}$.
$sp(X, \mathrm{x} := \mathrm{x} + 1) = \{[n] | n > 1\}$

Exercise 2.2

Why use of word "strongest"?

Symbolic sp

A formula in $\Sigma(V)$ represents a set of valuations.
Hence, we define symbolic sp that transforms formulas.

$$sp : \Sigma(V) \times \mathcal{P} \rightarrow \Sigma(V)$$

For data statements, the equivalent definition of symbolic sp is

$$sp(F, \mathrm{c}) \triangleq (\exists V : F \wedge \rho(\mathrm{c}))[V/V'].$$

Example 2.3

Let $V = [\mathrm{x}, \mathrm{y}, err]$ *and* $\mathrm{c} = \mathrm{x} := \mathrm{y} + 1$.
$sp(\mathrm{y} > 2, \mathrm{c}) = (\exists \mathrm{x}, \mathrm{y}, err.\ (\mathrm{y} > 2 \wedge \mathrm{x}' = \mathrm{y} + 1 \wedge \mathrm{y}' = \mathrm{y} \wedge err' = err))[V/V']$
$= (\mathrm{y}' > 2 \wedge \mathrm{x}' > 3)[V/V'] = (\mathrm{y} > 2 \wedge \mathrm{x} > 3)$

Exercise 2.3

- $sp(\mathrm{y} > 2 \wedge err = 0, \mathrm{x} := \mathtt{havoc}()) =\ (\mathrm{y} > 2 \wedge err = 0)$
- $sp(\mathrm{y} > 2 \wedge err = 0, \mathtt{assume}(\mathrm{y} < 10)) =\ (10 > \mathrm{y} > 2 \wedge err = 0)$
- $sp(\mathrm{y} > 2 \wedge err = 0, \mathtt{assert}(\mathrm{y} < 0)) =\ \top$

Symbolic sp for control statements

For control statements, the equivalent definitions of symbolic sp are

$$sp(F, c_1; c_2) \triangleq sp(sp(F, c_1), c_2)$$
$$sp(F, c_1[]c_2) \triangleq sp(F, c_1) \vee sp(F, c_2)$$
$$sp(F, \texttt{if(F}_1\texttt{) c}_1\texttt{ else c}_2) \triangleq sp(F, \texttt{assume(F}_1\texttt{)}; c_1) \vee sp(F, \texttt{assume(}\neg\texttt{F}_1\texttt{)}; c_2)$$
$$sp(F, \texttt{while(G) c}) \triangleq sp(lfp_{F'}(F \vee sp(F' \wedge \texttt{G}, \texttt{c})), \texttt{assume(}\neg\texttt{G))}$$

Example 2.4

$sp(x = 0, \texttt{if}(\texttt{y} > 0)\ \texttt{x := x} + 1\ \texttt{else x := x} - 1)$
$= (y > 0 \wedge x = 1 \vee y \leq 0 \wedge x = -1)$

Exercise 2.4

1. $sp(x + y > 0, \texttt{assume}(\texttt{x} > 0); \texttt{y} := \texttt{y} + 1)$
2. $sp(y < 2, \texttt{while}(y < 10)\ \texttt{y} := \texttt{y} + 1)$
3. $sp(y > 2, \texttt{while}(y < 10)\ \texttt{y} := \texttt{y} + 1)$
4. $sp(y = 0, \texttt{while}(\top)\ \texttt{y} := \texttt{y} + 1)$

Safety and symbolic sp

Theorem 2.1

For a program c*, if* $\not\models sp(err = 0, c) \wedge err = 1$ *then* c *is safe.*

Exercise 2.5

Prove the above lemma.

We need two key tools from logic to use *sp* as verification engine.

- quantifier elimination (for data statements)
- *lfp* computation (for loop statement)

There are quantifier elimination algorithms for many logical theories, e.g., integer arithmetic.

However, there is no general algorithm for computing *lfp*. Otherwise, the halting problem is decidable.

Field of verification

This course is all about developing

incomplete but sound methods for lfp

that work for

some of the programs of our interest.

Weakest pre — dual of sp

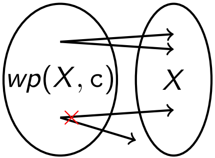
Now we define a an operator that executes the programs backwards!

Definition 2.2

Weakest pre operator wp : $\mathfrak{p}(\mathbb{Q}^{|V|}) \times \mathcal{P} \to \mathfrak{p}(\mathbb{Q}^{|V|})$ *is defined as follows.*

$$wp(X, \mathtt{c}) \triangleq \{v | \forall v' : (v', \mathtt{skip}) \in T^*((v, \mathtt{c})) \Rightarrow v' \in X\},$$

where $X \subseteq \mathbb{Q}^{|V|}$ *and* $\mathtt{c}$ *is a program.*



Example 2.5

Consider $V = [\mathtt{x}]$ *and* $X = \{[n] | 5 > n > 0\}$.
$wp(X, \mathtt{x} := \mathtt{x} + 1 [] \mathtt{x} := \mathtt{x} - 1) = \{[n] | 4 > n > 1\}$

Exercise 2.6

Why use of word "weakest"?

Logical weakest pre

We define symbolic wp that transforms formulas.

$$wp : \Sigma(V) \times \mathcal{P} \rightarrow \Sigma(V)$$

The equivalent definition of symbolic wp for data statements are

$$\begin{aligned}
wp(F, \texttt{x := exp}) &\triangleq F[\texttt{exp}/\texttt{x}] \\
wp(F, \texttt{x := havoc()}) &\triangleq \forall x.F \\
wp(F, \texttt{assume(G)}) &\triangleq G \Rightarrow F \\
wp(F, \texttt{assert(G)}) &\triangleq G \wedge F
\end{aligned}$$

Example 2.6

- $wp((i \leq 3 \wedge r = (i-1)z + 1), \texttt{i := 1}) =$
- $wp((i < 3 \wedge r = iz + 1), \texttt{r := r + z}) =$
- $wp(x < 0, \texttt{assume}(\texttt{x} > \texttt{0})) =$

Logical weakest pre

The equivalent definition of symbolic wp for control statements are

$$wp(F, c_1; c_2) \triangleq wp(wp(F, c_2), c_1)$$
$$wp(F, c_1[]c_2) \triangleq wp(F, c_1) \wedge wp(F, c_2)$$
$$wp(F, \texttt{if}(\texttt{F}_1)\ \texttt{c}_1\ \texttt{else}\ \texttt{c}_2) \triangleq wp(F, \texttt{assume}(\texttt{F}_1); c_1) \wedge wp(F, \texttt{assume}(\neg\texttt{F}_1); c_2)$$
$$wp(F, \texttt{while}(G)c) \triangleq gfp_{F'}((G \vee F) \wedge wp(F', \texttt{assume}(\texttt{G}); c))$$

Lemma 2.1

For a program c, if $err = 0 \Rightarrow wp(err = 0, c)$ is valid then c *is safe.*

Exercise 2.7

Prove the above lemma.

Note: Our definition of wp is usually called weakest liberal precondition(wlp)

Topic 2.3

Problems

Assignment

Exercise 2.8 (Assignment 1)

1. *(.5) Example 1.10*
2. *(.5) Discuss weakest precondition(wp) vs. weakest liberal precondition(wlp)*
3. *(1) Exercise 1.4*
4. *(1) Show* $sp(wp(F, \mathrm{c}), \mathrm{c}) \subseteq F \subseteq wp(sp(F, \mathrm{c}), \mathrm{c})$
5. *(1) Write a C++ program that reads a SMT2 formula from command line and performs quantifier elimination using Z3 for the variables that do not end with '*

Strength complete

Exercise 2.9

Stregthening is complete strengthen(IM) 1 if $0 = $ IM(l 0) 2 then return failed 3 elseif IM(l 1) $=$ IM(l 2) for some transition hl 1 , , , l 2 i 4 then construct such that IM(l 1) $=$ IM(l 2) 5 return strengthen(IM[l 1 7 IM(l 1)]) 6 else return IM IM is inductive

Algo is complete is ψ is learned using weakest pre-condition. Otherwise, give counter example for pre. (If the input is an invariant, then it should terminate declaring so, as well as produce an inductive invariant map (completeness).) Source: when is a Formula a Loop Invariant, Stephan Falke and Deepak Kapur

End of Lecture 2