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 $\Rightarrow$ 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.

- For $U \subseteq V$, 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 $\rho$ for the data statements as follows.

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

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

Example 2.1

Let $V = [x, y, \text{err}]$.

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

Exercise 2.1

Show $\rho$ 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* $sp : p(Q|V|) \times \mathcal{P} \rightarrow p(Q|V|)$ *is defined as follows.*

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

*where* $X \subseteq Q|V|$ *and* $c$ *is a program.*

**Example 2.2**

Consider $V = [x]$ and $X = \{ [n] | n > 0 \}$.  

\[ sp(X, x := x + 1) = \{ [n] | n > 1 \} \]

**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.

$$sp : \Sigma(V) \times P \rightarrow \Sigma(V)$$

For data statements, the equivalent definition of symbolic sp is

$$sp(F, c) \triangleq (\exists V : F \land \rho(c))[V/V'].$$ 

Example 2.3

Let $V = [x, y, err]$ and $c = x := y + 1$.

$$sp(y > 2, c) = (\exists x, y, err. (y > 2 \land x' = y + 1 \land y' = y \land err' = err))[V/V'] = (y' > 2 \land x' > 3)[V/V'] = (y > 2 \land x > 3)$$

Exercise 2.3

- $sp(y > 2 \land err = 0, x := \text{havoc}()) = (y > 2 \land err = 0)$
- $sp(y > 2 \land err = 0, \text{assume}(y < 10)) = (10 > y > 2 \land err = 0)$
- $sp(y > 2 \land err = 0, \text{assert}(y < 0)) = \top$
Symbolic sp for control statements

For control statements, the equivalent definitions of symbolic sp are

\[
\begin{align*}
sp(F, c_1; c_2) & \triangleq sp(sp(F, c_1), c_2) \\
sp(F, c_1 [] c_2) & \triangleq sp(F, c_1) \lor sp(F, c_2) \\
sp(F, if(F_1) c_1 \text{ else } c_2) & \triangleq sp(F, assume(F_1); c_1) \lor sp(F, assume(\neg F_1); c_2) \\
sp(F, while(G) c) & \triangleq sp(lfp_F(F \lor sp(F' \land G, c)), assume(\neg G))
\end{align*}
\]

Example 2.4

\[
sp(x = 0, if(y > 0) x := x + 1 \text{ else } x := x - 1)
= (y > 0 \land x = 1 \lor y \leq 0 \land x = -1)
\]

Exercise 2.4

1. \(sp(x + y > 0, assume(x > 0); y := y + 1)\)
2. \(sp(y < 2, while(y < 10) y := y + 1)\)
3. \(sp(y > 2, while(y < 10) y := y + 1)\)
4. \(sp(y = 0, while(\top) y := y + 1)\)
Safety and symbolic sp

Theorem 2.1
For a program $c$, if $\neg sp(err = 0, c) \land err = 1$ 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.
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* \( wp : p(\mathbb{Q} | V |) \times P \rightarrow p(\mathbb{Q} | V |) \) *is defined as follows.*

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

where \( X \subseteq \mathbb{Q} | V | \) and \( c \) is a program.

**Example 2.5**

Consider \( V = [x] \) and \( X = \{ [n] | 5 > n > 0 \} \).

\[
wp(X, x := x + 1[\] x := 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

\[
\begin{align*}
wp(\, F, \, x := \text{exp}) & \triangleq F[\text{exp}/x] \\
wp(\, F, \, x := \text{havoc()} & \triangleq \forall x. F \\
wp(\, F, \text{assume}(G) & \triangleq G \Rightarrow F \\
wp(\, F, \text{assert}(G) & \triangleq G \land F
\end{align*}
\]

Example 2.6

\[
\begin{align*}
\text{wp}((\, i \leq 3 \land r = (i - 1)z + 1), \, i := 1) = \\
\text{wp}((\, i < 3 \land r = iz + 1), \, r := r + z) = \\
\text{wp}(x < 0, \text{assume}(x > 0)) =
\end{align*}
\]
Logical weakest pre

The equivalent definition of symbolic wp for control statements are

\[ wp(F, c_1; c_2) \triangleq wp(wp(F, c_2), c_1) \]
\[ wp(F, c_1[]c_2) \triangleq wp(F, c_1) \land wp(F, c_2) \]
\[ wp(F, if(F_1) c_1 else c_2) \triangleq wp(F, assume(F_1); c_1) \land wp(F, assume(\neg F_1); c_2) \]
\[ wp(F, while(G)c) \triangleq gfp_{F'}((G \lor F) \land wp(F', assume(G); c)) \]

Lemma 2.1
For a program \( c \), if \( \text{err} = 0 \Rightarrow wp(\text{err} = 0, c) \) 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 $sp(wp(F, c), c) \subseteq F \subseteq wp(sp(F, c), c)$
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

Strengthening is complete 

\begin{align*}
\text{strengthen}(\text{IM}) & 1 \text{ if } 0 = \text{IM}(l_0) \text{ then return failed } \\
& 2 \text{ elseif } \text{IM}(l_1) = \text{IM}(l_2) \text{ for some transition } h l_1, \ldots, l_2 i \text{ then construct such that } \\
& \quad \text{IM}(l_1) = \text{IM}(l_2) \text{ then return strengthen(} \text{IM}[l_1 \cup \text{IM}(l_1)]) \\
& 6 \text{ else return IM IM is inductive} \\
\end{align*}

Algo is complete is $\psi$ 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