CS766: Analysis of concurrent programs (first half) 2021

Lecture 2: Symbolic operator: strongest post

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2021-01-15

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 set of states and
    - For example, $x > 0$ represents infinite set $\{1, 2, 3, ...\}$
  - ability to compute transitive closure of transition relation
- Idea: use logic for the above goals

Topic 2.1

Program statements as formulas

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.
  - V denote the current value of the variables
  - V' denote the next value of the variables

Example 2.1

Let $V = [\mathrm{x}, \mathrm{y}, err]$. *Therefore,* $V' = [\mathrm{x}', \mathrm{y}', err']$.

Notation : frame

Definition 2.1

$$\text{For } U \subseteq V, \text{ let } frame(U) \triangleq \bigwedge_{x \in V \setminus U} (x' = x)$$

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

Exercise 2.1

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

- $frame(\mathrm{x}) :=$
- $frame(\mathrm{y}) :=$
- $frame(\emptyset) :=$
- $frame([\mathrm{x}, \mathrm{y}]) :=$
- $frame(V) :=$

Program statements as formulas (contd.)

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

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

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

Example 2.2

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

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

Exercise 2.2

Show ρ *correctly models the assert statement*

Executing as satisfaction

We can use ρ to execute the commands.

Give the values for the current state, get the values for the next state.

Example 2.3

Consider command $\rho(\mathtt{x} \ := \ \mathtt{y}+1) = (\mathtt{x}' = \mathtt{y}+1 \land \mathtt{y}' = \mathtt{y} \land \mathit{err}' = \mathit{err})$
Consider current state: $\{\mathtt{x} = 1, \mathtt{y} = 1, \mathit{err} = 0\}$
To execute the command, we solve the following constraints

$$(\mathtt{x}' = 1+1 \land \mathtt{y}' = 1 \land \mathit{err}' = 0)$$

We obtain

$$\{\mathtt{x}' = 2 \land \mathtt{y}' = 1 \land \mathit{err}' = 0\}$$

Commentary: In the case, we have a unique solution for the primed variables. However, that may not be necessary. For some commands, we may have multiple solutions or none.

Example: executing as satisfaction

Example 2.4

Consider $\rho(\texttt{assert}(\mathrm{x} > 0)) = (\mathrm{x} > 0 \Rightarrow (\mathrm{x}' = \mathrm{x} \wedge \mathrm{y}' = \mathrm{y} \wedge \mathit{err}' = \mathit{err}))$ *and current state* $\{\mathrm{x} = -1, \mathrm{y} = 1, \mathit{err} = 0\}$.

To execute the command, we solve the following constraints

$$(-1 > 0 \Rightarrow (\mathrm{x}' = -1 \wedge \mathrm{y}' = 1 \wedge \mathit{err}' = 0))$$

If we simplify the above formula, we obtain

$$\top$$

Any state can be the next state, let us choose the following.

$$\{\mathrm{x} = 12345, \mathrm{y} = 100000, \mathit{err} = 1\}$$

Exercise 2.3

What happens if current state is $\{\mathrm{x} = 2, \mathrm{y} = 1, \mathit{err} = 0\}$*?*

Topic 2.2

Aggregated semantics

Aggregate

Another view of executions

$$\text{sets of valuations} \rightarrow \text{sets of valuations}$$

Notation

- valuation : $\mathbb{Q}^{|V|}$
- set of valuations : $\mathfrak{p}(\mathbb{Q}^{|V|})$
- set of valuations $\rightarrow$ set of valuations : $\mathfrak{p}(\mathbb{Q}^{|V|}) \rightarrow \mathfrak{p}(\mathbb{Q}^{|V|})$

We will only refer to the set of reachable valuations/states at a location, not at the whole program.

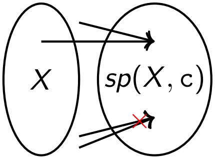
Strongest post: set of valuations to set of valuations

Definition 2.2

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 2.5

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

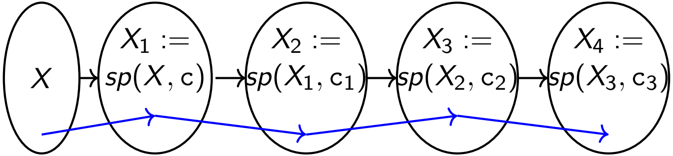
$sp(X, \mathtt{x} := \mathtt{x} + 1) = \{[n] | n > 1\}$

Exercise 2.4

Why use of word "strongest"?

Reachability and strongest post

No reachable state will escape the strongest post.



On the other hand, if we do not track all the states in strongest post, we may miss some reachable states.

Symbolic sp

We have discussed that a formula in $\Sigma(V)$ represents a set of valuations.

Hence, we declare symbolic sp that transforms formulas.

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

For data statements, the equivalent definition of symbolic sp is

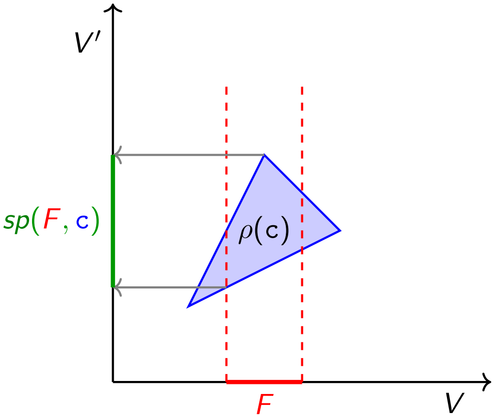
$$sp(F, c) \triangleq (\exists V : F \land \rho(c))[V/V'].$$

Example 2.6

Let $V = [\mathrm{x}, \mathrm{y}, err]$ *and* $c = \mathrm{x} := \mathrm{y} + 1$. $\rho(c) = \mathrm{x}' = \mathrm{y} + 1 \land \mathrm{y}' = \mathrm{y} \land err' = err$

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

Existence == projection



Exercise : symbolic sp

Example 2.7

- $sp(\mathtt{y} > 2 \land err = 0, \mathtt{y} := \mathtt{havoc()}) = \ (err = 0)$
- $sp(\mathtt{y} > 2 \land err = 0, \mathtt{assume}(\mathtt{y} > \mathtt{0})) = \ (\mathtt{y} > 2 \land err = 0)$

Exercise 2.5

- $sp(\mathtt{y} > 2 \land err = 0, \mathtt{x} := \mathtt{havoc()}) =$
- $sp(\mathtt{y} > 2 \land err = 0, \mathtt{assume}(\mathtt{y} < \mathtt{10})) =$
- $sp(\mathtt{y} > 2 \land err = 0, \mathtt{assert}(\mathtt{y} < \mathtt{0})) =$
- $sp(\bot, c) =$

Exercise: simplfy sp

Exercise 2.6

Show that

- $sp(F, \texttt{x := havoc()}) = \exists x.F$
- $sp(F, \texttt{assume(G)}) = F \wedge G$
- $sp(F, \texttt{assert(G)}) = F \vee \underbrace{\exists V.(F \wedge \neg G)}_{\textit{No free variables}}$

Exercise 2.7

Why not simplify $sp(F, \texttt{x := exp})$ *like above?*

Symbolic sp for control statements (other than while)

For control statements, the equivalent definitions of symbolic sp are

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

Example 2.8

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

Exercise 2.8

1. $sp(x + y > 0, \texttt{assume}(\texttt{x} > \texttt{0}); \texttt{y} := \texttt{y} + 1)$
2. $sp(x + y > 0, \texttt{assume}(\texttt{x} > \texttt{0})[]\texttt{y} := \texttt{y} + 1)$

Topic 2.3

Some math: least fixed point

Least fixed point (lfp)

Definition 2.3

For a function f, x is a fixed point of f if $f(x) = x$.

Definition 2.4

For a function f, $\ell = lfp_x(f(x))$ is the least fixed point of f if

- $f(\ell) = \ell$ *and*
- $\forall y < \ell.\ f(y) \neq y$.

Definition 2.5

For a function f, $\ell = gfp_x(f(x))$ is the greatest fixed point of f if

- $f(\ell) = \ell$ *and*
- $\forall y > \ell.\ f(y) \neq y$.

Example 2.9

Consider function $f(x) = 2/x$. $\sqrt{2}$ and $-\sqrt{2}$ are the fixed points of f. Therefore,

$$lfp_x(2/x) = -\sqrt{2} \qquad gfp_x(2/x) = \sqrt{2}$$

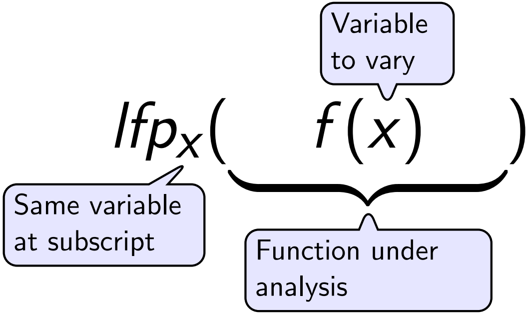
Example: fixed-points

Exercise 2.9

Give least fixed point and greatest fixed point of the following functions.

- $f(x) = x + 1$
- $f(x) = x$
- $f(x) = x^2$
- $f(x) = x^2 + x - 1$

Notation: least/greatest fixed point



There can be other variables in the function that are assumed to be fixed with respect to the analysis and the answer is parameterized by the free variable.

Example 2.10

Consider

$$lfp_x(x^2 + y) = \frac{-1 - \sqrt{1 - 4y}}{2}$$

Functions for formula

Consider a function like the following that takes a formula as input and returns another.

$$f : \Sigma \to \Sigma$$

Example 2.11

Strongest post $sp(F, \texttt{c})$ *takes two parameters. If we fix* c*, the function takes a formula as input and returns an output.*

- $sp(x = 0, \mathtt{x} := \mathtt{havoc}()) = \top$
- $sp(\mathrm{y} > 2, \mathtt{x} := \mathtt{havoc}()) = \mathrm{y} > 2$ *(fixed point!!)*
- $sp(\mathrm{y} + \mathrm{x} > 2, \mathtt{x} := \mathtt{havoc}()) = \top$

Exercise 2.10

a. *What is the greatest fixed point for* $gfp_F(sp(F, \mathtt{x} := \mathtt{havoc}()))$*?*
b. *What is the least fixed point for* $lfp_F(sp(F, \mathtt{x} := \mathtt{havoc}()))$*?*

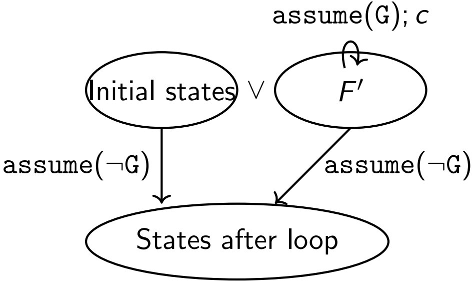
Topic 2.4

sp for loops

Handling while loop

`while(G) c`

F' are set of reachable states at loop head after some number of iterations.



Symbolic sp for control statements (while)

$$sp(F, \texttt{while(G) c}) \triangleq sp(lfp_{F'}(F \lor sp(F' \land \texttt{G}, \texttt{c})), \texttt{assume}(\neg\texttt{G}))$$

Exercise 2.11

a. *What is the return type of lfp in the above?*

b. *What is the meaning of sp in the lfp?*

c. *What is the meaning of the whole function in the lfp?*

c. *What will happen if we remove '$F \lor$' inside the lfp?*

e. *What is the purpose of outside sp?*

Exercise: symbolic sp for control statements

Exercise 2.12 (Give intuitive answers!)

We have not yet learned an algorithm for *sp*

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 2.13
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.

End of Lecture 2