CS520 Lecture 4
Program specifications and Their Proofs

January 12, 2020

4.1 Motivation

1. Are there any methods that let us specify desired properties or intended behaviours of a program and prove that the specified properties indeed hold? For instance, consider the program

$$C_{div3} \equiv a := 0; b := x; \text{while } (b \geq 3) \text{ do } b := b - 3; a := a + 1$$

We want to express formally that the program divides x by 3, and stores the quotient in a and the remainder in b. We also want to prove this formal specification.
2. We will study Hoare logic and its variant for total correctness. They provide the kind of methods that we are looking for.
3. Hoare logic and its total-correctness variant are the basis of modern automatic software verifiers, such as Facebook's infer.
4. Another reason for studying Hoare logic is that it shows why we need or where we use denotational semantics that we studied.

4.2 Syntax and Semantics for specifications

1. Specifications are a new type of phrases that formally express properties of programs.
2. Syntax in terms of abstract grammar:

$$\begin{aligned} \langle \text{spec} \rangle ::= & \{\langle \text{assert} \rangle\} \langle \text{comm} \rangle \{\langle \text{assert} \rangle\} \\ & |[\langle \text{assert} \rangle] \langle \text{comm} \rangle [\langle \text{assert} \rangle] \end{aligned}$$

Examples :

$$c_{fib} \equiv \begin{array}{c} k := 1; y := 0; x := 1; \\ \text{while } (k \neq n) \text{ do } t := y; y := x; x := x + t; k := k + 1 \end{array}$$

$$\{n \geq 0\} c_{fib} \{x = fib(n)\} \{true\} c_{fib} \{x = fib(n)\}$$
$$[n \geq 0] c_{fib} [x = fib(n)] [true] c_{fib} [x = fib(n)]$$

$$\{x \geq 0\} c_{div3} \{x = 3 \times a + b \wedge 0 \leq b < 3 \wedge a \geq 0\}$$
$$[x \geq 0] c_{div3} [x = 3 \times a + b \wedge 0 \leq b < 3 \wedge a \geq 0]$$
$$\{true\} c_{div3} \{x = 3 \times a + b \wedge 0 \leq b < 3 \wedge a \geq 0\}$$
$$[true] c_{div3} [x = 3 \times a + b \wedge 0 \leq b < 3 \wedge a \geq 0]$$

3. Intuitive reading :
$\{p\}c\{q\}$ holds iff when c is run in a state satisfying p and it terminates normally[1], then the final state satisfies q.
$[p]c[q]$ holds iff when c is run in a state satisfying p, then it terminates normally[2] and the final state satisfies q.

[1] condition
[2] conclusion

Note that $[p]c[q]$ expresses a stronger property than $\{p\}c\{q\}$. The former is called total correctness specification[3] and the latter partial correctness specification[4].

$$p \cdots \text{precondition or precedent}$$
$$q \cdots \text{postcondition or consequent}$$

Exercise. Among all the partial correctness and total correctness specification from above, pick those that hold.

4. Formal semantics:

$$[\![-]\!]_{spec} \in [\langle \text{spec} \rangle \to \mathbb{B}]$$

$$[\![\{p\}c\{q\}]\!] = tt \text{ iff } \forall \sigma [\![p]\!]\sigma = tt \wedge [\![c]\!]\sigma \neq \bot \Rightarrow [\![q]\!]([\![c]\!]\sigma) = tt$$

$$[\![[p]c[q]]\!] = tt \text{ iff } \forall \sigma [\![p]\!]\sigma = tt \Rightarrow [\![c]\!]\sigma \neq \bot \wedge [\![q]\!]([\![c]\!]\sigma) = tt$$

4.3 Inference Rules

1. Methods or rules for proving or deriving partial or total correctness triples.

2. Have the inference rule form :

$$\frac{premises \quad \phi_1 \quad \phi_2 \quad \cdots \quad \phi_n}{conclusion \qquad \psi}$$

(If $\phi_1, \ldots, \phi_n$ are all true, then ψ is true.)

3. Rules associated with program constructs:

$$\frac{}{\{p\}skip\{p\}} \quad \frac{}{[p]skip[p]}$$

$$\frac{\{p\}c_1\{r\} \quad \{r\}c_2\{q\}}{\{p\}c_1; c_2\{q\}} \quad \frac{[p]]c_1[r] \quad [r]]c_2[q]}{[p]c_1; c_2[q]}$$

$$\frac{\{p \wedge b\}c_1\{q\} \quad \{p \wedge \neg b\}c_2\{q\}}{\{p\}\text{if } b \text{ then } c_1 \text{ else } c_2\{q\}} \quad \frac{[p \wedge b]c_1[q] \quad [p \wedge \neg b]c_2[q]]}{[p]]\text{if } b \text{ then } c_1 \text{ else } c_2[q]]}$$

$$\frac{}{\{^q/_{x \to e}\}x := e\{q\}} \quad \frac{}{[^q/_{x \to e}]x := e[q]}$$

[3] also total correctness triple

[4] also partial correctness triple or Hoare triple or triple

$$\frac{\{i \wedge b\}c\{i\}}{\{i\}\text{while } b \text{ do } c\{i \wedge \neg b\}}$$

$$\frac{i \wedge b \Rightarrow e \geq 0 \qquad [i \wedge b \wedge e = v_0]c[i \wedge c < v_0]}{[i]\text{while } b \text{ do } c[i \wedge \neg b]}$$

(1) Note that rules for total correctness and the corresponding ones for partial correctness are identical except for the case of loop. This is expected because these two notions differ only in their treatment of non-termination.
(2) The rules for while require that i should be preserved by the body of the loop. The one for total correctness additionally requires that the value of e should decrease whenever we run the loop body c once, but it cannot be negative. All these requirements together give the conclusions of the rules.
(3) The rules for assignment also deserve some thoughts. They in a sense say that running an assignment backward symbolically is the same as doing substitution. It holds because of the substitution theorem (Prop 1.3, Prop 1.4 in the textbook).

Reminder of the theorem : specialised to our case here $\begin{array}{ccc} \Sigma & \xrightarrow{\lambda\sigma.[\sigma|x:[\![e]\!]\sigma]} & \Sigma \\ \downarrow{\scriptstyle [\![q/_{x \to e}]\!]} & & \downarrow{\scriptstyle [\![q]\!]} \\ \mathbb{B} & = & \mathbb{B} \end{array}$ 4.

Rules not associated with any specific program constructs (sometimes called structural rules or adaption rules)

$$\frac{p \Rightarrow p' \qquad \{p'\}c\{q'\} \qquad q' \Rightarrow q}{\{p\}c\{q\}} \qquad \frac{p \Rightarrow p' \qquad [p']c[q'] \qquad q' \Rightarrow q}{[p]c[q]}$$

They are called the rule of consequence. They often enable us to use the other rules, in particular, those for loop and if.
5. I omit many structural rules and the rules for newvar. Look at the textbook if you are interested.

4.4 Example Proof

$$\{x \geq 0\}c_{div3}\{x = 3 \times a + b \wedge 0 \leq b < 3\}$$
$$q_1 := x = 3 \times a + b, q_2 := 0 \leq b < 3$$

$$\frac{\{x \geq 0\}a := 0; b := x\{q_1 \wedge b \geq 0\} \qquad \{q_1 \wedge b \geq 0\}\text{while } \ldots\{q_1\}}{\{x \geq 0\}c_{div3}\{q_1 \wedge q_2\}}$$

$$\frac{\dfrac{x \geq 0 \Rightarrow x = x \wedge x \geq 0 \qquad \overline{\{x = x \wedge x \geq 0\}a := 0\{x = 3a + x \wedge x \geq 0\}}}{\{x \geq 0\}a := 0\{x = 3a + x \wedge x \geq 0\}} \qquad \overline{\{x = 3a + x \wedge x \geq 0\}b := x\{q_1 \wedge b \geq 0\}}}{\{x \geq 0\}a := 0; b := x\{q_1 \wedge b \geq 0\}}$$

$$\dfrac{\dfrac{}{\{q_1 \wedge b \geq 0 \wedge b \geq 3\}\ b:=b-3\ \{x=3(a+1)+b \wedge b \geq 0\}} \quad \dfrac{}{\{x=3(a+1)+b \wedge b \geq 0\}\ a:=a+1\ \{q_1 \wedge b \geq 0\}}}{\dfrac{\{q_1 \wedge b \geq 0 \wedge b \geq 3\}\ b:=b-3;a:=a+1\ \{q_1 \wedge b \geq 0\}}{\{q_1 \wedge b \geq 0\}\ \text{while } (b \geq 3) \text{ do } b:=b-3;a:=a+1\ \{q_1 \wedge q_2\}}}$$

1. In practice, people use the rule of consequence without mentioning it explicitly. Also, they use many derived rules.
2. This proof has the flavour of running a program backward symbolically because of its heavy use of the assignment rule and the fact that the rule of consequence is used only when it is necessary.

Exercise 1. Prove

$$\{n \geq 1\} C_{fib} \{x = fib(n)\}$$

$$\{a \geq 1 \wedge b \geq 1 \wedge a = a_0 \wedge b = b_0\} Euclid \{a = gcd(a_0, b_0)\}$$

where $Euclid \equiv$ while $(b \neq a)$ do if $a > b$ then $a := a - b$ else $b := b - 1$

Exercise 2. Find a forward rule for assignment. That is, for all p and x, e find q s.t. $\dfrac{}{\{p\}x := e\{q\}}$

4.5 Soundness

Theorem 4.5.1 (Soundness Theorem)
If $\{p\}c\{q\}$ *is derivable using the rules that we studied*[5]*, then* $[\![\{p\}c\{q\}]\!] = tt$*, i.e. the triple* $\{p\}c\{q\}$ *holds.*
If $[p]c[q]$ *is derivable, then* $[\![[p]c[q]]\!] = tt$*.*

Intuitively, the theorem says that all rules are correct. In fact, typical proofs of the theorem show the correctness of the rules in the following sense:
If $\dfrac{\phi_1 \quad \ldots \quad \phi_n}{\psi}$ then $[\![\phi_1]\!] = tt \wedge \ldots \wedge [\![\phi_n]\!] = tt \Rightarrow [\![\psi]\!] = tt$
Tue rules for loop(or while) are the most important cases. We will consider only the one for partial correctness.

$$\dfrac{\{i \wedge b\}c\{i\}}{\{i\}\text{while } b \text{ do } c\{i \wedge \neg b\}}$$

We first do a bit of recoriting for the semantics of specifications.

$$[\![\{p\}c\{q\}]\!] \text{ iff } \forall \sigma \in \Sigma. [\![p]\!]\sigma = tt \Rightarrow [\![q]\!]_\perp([\![c]\!]\sigma) \sqsubseteq tt$$

where $[\![q]\!]_\perp \in [\Sigma_\perp \to \mathbb{B}_\perp]$ s.t. $[\![q]\!]_\perp \sigma = [\![q]\!]\sigma$for all $\sigma \in \Sigma$ and $[\![q]\!]_\perp(\perp) = \perp$.
We need to prove that if

$$\forall \sigma. [\![i \wedge b]\!]\sigma = tt \Rightarrow [\![i]\!]_\perp([\![c]\!]\sigma) \sqsubseteq tt \qquad (*)$$

then

$$\forall \sigma [\![i]\!]\sigma = tt \Rightarrow [\![i \wedge \neg b]\!]([\![\text{while } b \text{ do } c]\!]\sigma) \sqsubseteq tt$$

[5] called rules in Hoare logic

Assume $(*)$.
Let

$$F \in [(\Sigma \to \Sigma_\perp) \to_c (\Sigma \to \Sigma_\perp)]$$
$$F(f)(b) = \text{if } [\![b]\!]\sigma = tt \text{ then } (f_{\perp\!\!\perp} \circ [\![c]\!])(\sigma) \text{ else } \sigma$$

Define $f_n = F^n(\perp)$ for $n \geq 0$.
Then, $[\![\text{while } b \text{ do } c]\!] = \sqcup_{n=0}^{\infty} f_n$.
We will show that for all $n \geq 0$,

$$\forall \sigma [\![i]\!]\sigma = tt \Rightarrow [\![i \wedge \neg b]\!]_\perp(f_n(\sigma)) \sqsubseteq tt \qquad (**)$$

This is sufficient because for all $\sigma \in \Sigma$ s.t. $[\![i]\!]\sigma = tt$,

$$\begin{aligned}
&[\![i \wedge \neg b]\!]_\perp([\![\text{while } b \text{ do } c]\!]) \\
&= [\![i \wedge \neg b]\!]_\perp((\sqcup_{n=0}^{\infty} f_n)\sigma) \\
&= [\![i \wedge \neg b]\!]_\perp(\sqcup_{n=0}^{\infty} f_n(\sigma)) \\
&= \sqcup_{n=0}^{\infty}([\![i \wedge \neg b]\!]_\perp(f_n(\sigma))) && \text{(because } [\![i \wedge \neg b]\!]_\perp \text{ is continuous)} \\
&\sqsubseteq tt && \text{(because of } (**))
\end{aligned}$$

Our proof of $(**)$ uses induction on n.
Base case $n = 0$: $f_0 = \perp$, $\therefore [\![i \wedge \neg b]\!]_\perp(\perp(\sigma) = [\![i \wedge \neg b]\!]_\perp(\perp) = \perp$.
Inductive case $n = m + 1$:

Pick σ s.t. $[\![i]\!]\sigma = tt$.
$[\![i \wedge \neg b]\!]_\perp(f_n(\sigma)) = [\![i \wedge \neg b]\!]_\perp(f_{m+1}(\sigma)) = [\![i \wedge \neg b]\!]_\perp(F(f_n)(\sigma))$
$= [\![i \wedge \neg b]\!]_\perp(\text{if } [\![b]\!]\sigma = tt \text{ then } (f_{m\perp\!\!\perp} \circ [\![c]\!])(\sigma) \text{ else } \sigma)$
$= \text{if } [\![b]\!]\sigma = tt \text{ then } ([\![i \wedge \neg b]\!]_\perp \circ f_{m\perp\!\!\perp})([\![c]\!]\sigma) \text{ else } [\![i \wedge \neg b]\!]_\perp\sigma$

Since $[\![i]\!]\sigma = tt$, if $[\![b]\!]\sigma \neq tt$, then $[\![i \wedge \neg b]\!]_\perp\sigma = tt \sqsubseteq tt$.
If $[\![b]\!]\sigma = tt$ and $[\![c]\!]\sigma = \perp$, then $([\![i \wedge \neg b]\!]_\perp \circ f_{m\perp\!\!\perp})([\![c]\!]\sigma) = \perp \sqsubseteq tt$.
If $[\![b]\!]\sigma = tt$ and $[\![c]\!]\sigma \neq \perp$, then $[\![i]\!]([\![c]\!]\sigma) = tt$ by $(*)$.
Thus, $([\![i \wedge \neg b]\!]_\perp \circ f_{m\perp\!\!\perp})([\![c]\!]\sigma) = [\![i \wedge \neg b]\!]_\perp(f_m([\![c]\!]\sigma)) \sqsubseteq tt$ by induction hypothesis.