Instructor: Fei He

TA: Jianhui Chen
Fengmin Zhu

Due date: 05/12/2019

Assignment 7

Zheng Zeng
2016013263

Instructions: Write your answers in the corresponding `hw7.tex` file, compile it to a pdf, and hand the pdf file (or a `.zip` archive if you have other files) to *Tsinghua Web Learning* by the due date. Be sure to add your **student ID** and **full name** in the `stuid` and `stuname` macros at the top of `hw7.tex`. Problems marked with "(Optional)" are NOT mandatory, to receive full credits, you do NOT need to solve them. You will obtain extra points if you correctly solve them.

Acamedic Honesty: Any kind of plagiarism is strictly prohibited in the full semester for this course. Students who are suspected to copy other's work and is confirmed through investigation will receive no credits (i.e, zero) for this assignment. If you asked other students for help, or your referred to any material that is not provided by us (e.g. websites, blogs, articles, papers, etc., both online and offline), please mention them in your assignment (e.g. writing an acknowledgment, adding a reference).

Note: In problems 1, 2, 3 and 5 for this assignment, we will continue reasoning about IMP programs, which we have seen in Assignment 6. Above all, to make our life easier, we extend the IMP boolean expressions to include a full set of comparison expressions (with 6 operators) and logical expressions (with logical disjunction):

$$b ::= \mathsf{true} \mid \mathsf{false} \mid a_1 \ op \ a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2$$

where the comparison operator $op \in \{=, \neq, <, \leq, >, \geq\}$. Remember in logical formulae (and also assertions), we write $\top$ for true and $\bot$ for false.

We use the same notations from lectures for Hoare triples, standard Hoare rules (say Asgn, Seq, etc.), and weakest preconditions.

1 Hoare Triples

Determine the validity of the following Hoare triples and briefly explain why. Be careful to distinguish partial correctness and total correctness.

1. $\{X = 0 \wedge Y = 1\}\ X := X + 1; Y := Y + 1\ \{X = 1 \wedge Y = 2\}$
2. $\{\top\}$ while $X \leq 0$ do $X := X + 1$ end $\{X \geq 0\}$
3. $[\top]$ while $0 < X$ do $X := X - 1$ end $[X \leq 0]$
4. $[X < 0]$ while $0 < X$ do $X := X + 1$ end $[X \leq 0]$
5. $\{\top\}$ while $0 < X$ do $X := X - 1$ end $\{X = 0\}$

Solution

1. Valid. $X = old(X) + 1, Y = old(Y) + 1$ after executing c, and c terminates.
2. Valid. If c terminates, $X \leq 0$ must not hold, i.e. $X > 0$ must hold, therefore $X \geq 0$ in the final environment.
3. Valid. If c terminates, $\neg(0 < X)$, i.e. $X \leq 0$ must hold. And c terminates since X decreases.
4. Valid. If $X < 0$, c terminates immediately because it won't get into the loop, then $X < 0$, so $X \leq 0$ holds.
5. Invalid. If $X = -1$ in the beginning environment, $X = -1$ holds in the final environment, $X = 0$ not holds.

2 Decorated Programs

The beauty of Hoare logic is that it is *compositional*: the structure of proofs exactly follows the structure of programs. This suggests that we can record the essential ideas of a proof (leaving out some low-level calculational details) by "decorating" a program with appropriate assertions on each of its commands. Such a *decorated program* carries within it an argument for its own correctness.

For example, consider the program (where m and p denote two constant integers):

$$
\begin{array}{l}
X := m; \\
Z := p; \\
\textsf{while } \neg(X = 0) \textsf{ do} \\
\quad Z := Z - 1; \\
\quad X := X - 1 \\
\textsf{end}
\end{array}
$$

To show that the above program satisfies precondition $\top$ and postcondition $Z = p - m$, we decorate the above as the following:

$$
\begin{array}{l}
\{\top\} \twoheadrightarrow \{m = m\} \\
X := m; \\
\{X = m\} \twoheadrightarrow \{X = m \wedge p = p\} \\
Z := p; \\
\{X = m \wedge Z = p\} \twoheadrightarrow \{Z - X = p - m\} \\
\textsf{while } \neg(X = 0) \textsf{ do} \\
\{Z - X = p - m \wedge X \neq 0\} \twoheadrightarrow \{(Z - 1) - (X - 1) = p - m\} \\
\quad Z := Z - 1; \\
\quad \{Z - (X - 1) = p - m\} \\
\quad X := X - 1 \\
\quad \{Z - X = p - m\} \\
\textsf{end} \\
\{Z - X = p - m \wedge \neg(X \neq 0)\} \twoheadrightarrow \{Z = p - m\}
\end{array}
$$

Concretely, a decorated program consists of the program commands interleaved with assertions – either a single assertion, or possibly two assertions separated by an implication "$\twoheadrightarrow$" (we use this strange notation to distinguish from our reduction relation $\rightarrow$). To check that a decorated program represents a valid proof, we check that each individual command is *locally consistent* with its nearby assertions, following the standard Hoare rules, e.g.

$$\{m = m\}\ X := m\ \{X = m\}$$

is valid by rule Asgn. Note that this hoare triple must not only be valid, but also be an **instantiation** of some Hoare rule. To convince yourself, carefully check that every such triples are indeed valid, and they are instantiations of the standard Hoare rules.

Now, it's your turn. Here is a program that squares X by repeated addition:

$$
\begin{array}{l}
Y := 0; \\
Z := 0; \\
\textsf{while } \neg(Y = X) \textsf{ do} \\
\quad Z := Z + X; \\
\quad Y := Y + 1 \\
\textsf{end}
\end{array}
$$

(1) We expect the above to satisfy precondition $X = m$ and postcondition $Z = m \times m$ for an arbitrary **natural number** m. Please write the corresponding decorated program which represents a valid proof of it. *Hint: you may need a "good" loop invariant.*

(2) (Optional) Translate your decorated program into a verified Dafny program. All assertions shall be translated properly. If you do this optional exercise, please attach the source file (name it `square.dfy`) in your `.zip` archive.

Solution

(1)

$\{X = m\} \twoheadrightarrow \{X = m \wedge 0 = 0\}$
$Y := 0;$
$\{X = m \wedge Y = 0\} \twoheadrightarrow \{X = m \wedge Y = 0 \wedge Y \leq X \wedge 0 = 0\}$
$Z := 0;$
$\{X = m \wedge Y = 0 \wedge Y \leq X \wedge Z = 0\} \twoheadrightarrow \{X = m \wedge Y \leq X \wedge Z = Y * m\}$
while $\neg(Y = X)$ do
$\{X = m \wedge Y \leq X \wedge Z = Y * m \wedge Y \neq X\} \twoheadrightarrow \{X = m \wedge Z + X = (Y + 1) * m \wedge Y + 1 \leq X\}$
$\quad Z := Z + X;$
$\quad \{X = m \wedge Z = (Y + 1) * m \wedge Y + 1 \leq X\}$
$\quad Y := Y + 1;$
$\quad \{X = m \wedge Z = Y * m \wedge Y \leq X\}$
end
$\{X = m \wedge Z = Y * m \wedge \neg(Y = X)\} \twoheadrightarrow \{Z = Y * m\}$

3 Deductive Verification

In this part, we extend our IMP language with an *assumption statement* introduced in the lecture, which has the form "assume b" where b is a boolean expression.

1. Compute $\mathsf{wlp}(X := X + 1; \text{assume } X > 0; Y := Y + X, X + Y + Y \geq 3)$.
2. Generate the verification condition for basic path

$$\begin{array}{l}\{\top\}\\ X := X - K;\\ \text{assume } K \leq X\\ \{X \geq 0\}\end{array}$$

Solution

1.

$$\begin{aligned}
& \mathsf{wlp}(X := X + 1; \text{assume } X > 0; Y := Y + X, X + Y + Y \geq 3)\\
= \ & \mathsf{wlp}(X := X + 1; \text{assume } X > 0, \mathsf{wlp}(Y := Y + X, X + Y + Y \geq 3))\\
= \ & \mathsf{wlp}(X := X + 1; \text{assume } X > 0, X + Y + X + Y + X \geq 3)\\
= \ & \mathsf{wlp}(X := X + 1, \mathsf{wlp}(\text{assume } X > 0, X + Y + X + Y + X \geq 3))\\
= \ & \mathsf{wlp}(X := X + 1, X > 0 \to X + Y + X + Y + X \geq 3)\\
= \ & (X + 1) > 0 \to X + 1 + Y + X + 1 + Y + X + 1 \geq 3\\
= \ & \top
\end{aligned}$$

2. The VC is

$$\{\top\}X := X - K; \text{assume } K \leq X\{X \geq 0\} : \top \to \mathsf{wlp}(X := X - K; \text{assume } K, X \geq 0)$$

First calculate the weakest literal precondition:

$$\begin{aligned}
& \mathsf{wlp}(X := X - K; \text{assume } K \leq X, X \geq 0)\\
= \ & \mathsf{wlp}(X := X - K, \mathsf{wlp}(\text{assume } K \leq X, X \geq 0))\\
= \ & \mathsf{wlp}(X := X - K, K \leq X \to X \geq 0)\\
= \ & K \leq X \to X - K \geq 0\\
= \ & \top
\end{aligned}$$

Simplifying the VC based on this computation produces $\top \to \top$, which is clearly $T_{\mathbb{Z}}$-valid.

4 The div function

Integer division, even by a constant, is not a function of $T_{\mathbb{Z}}$; however, it is useful for reasoning about programs like BinarySearch, shown in Fig. 1 (this is just pseudo-code, don't complain about the syntax, if it looks strange to you). Show how basic paths that include the div function can be altered to use only standard linear arithmetic. *Hint: Use an additional assumption statement. How does this change affect the resulting verification conditions?*

```
@pre 0 ≤ ℓ ∧ u < |a| ∧ sorted(a, ℓ, u)
@post rv ↔ ∃i. ℓ ≤ i ≤ u ∧ a[i] = e
bool BinarySearch(int[] a, int ℓ, int u, int e) {
  if (ℓ > u) return false;
  else {
    int m := (ℓ + u) div 2;
    if (a[m] = e) return true;
    else if (a[m] < e) return BinarySearch(a, m + 1, u, e);
    else return BinarySearch(a, ℓ, m − 1, e);
  }
}
```

Figure 1: Binary Search

Solution

We only need to alter line $m := (l + u)$ div 2 to assume $l + u - 1 \leq 2 * m \leq l + u$ in each basic path of the programs. For instance, the basic path

@pre $0 \leq l \wedge u \leq |a| \wedge \mathsf{sorted}(a, l, u)$
assume $l \leq u$;
$m := (l + u)$ div 2;
assume $a[m] = e$;
$rv := true$;
@post $rv \leftrightarrow \exists i.l \leq i \leq u \wedge a[i] = e$

will be altered to

@pre $0 \leq l \wedge u \leq |a| \wedge \mathsf{sorted}(a, l, u)$
assume $l \leq u$;
assume $l + u - 1 \leq 2 * m \leq l + u$
assume $a[m] = e$;
$rv := true$;
@post $rv \leftrightarrow \exists i.l \leq i \leq u \wedge a[i] = e$

5 Hoare Rules (Optional)

The Hoare rules we have seen from lectures are *verification-friendly*, say they are strong enough to reason about almost all IMP programs. Suppose we add more extensions to IMP, which is our favorite thing, new Hoare rules need be specified to support these features. Again, these new rules should be verification-friendly. In this problem, you are asked to design a couple of new Hoare rules that describe partial correctness.

(1) Design a new Hoare rule for havoc, which we did in Assignment 6.

(2) Design a new Hoare rule for the repeat-loop:

$$\textsf{repeat } c \textsf{ until } b \textsf{ end},$$

which behaves like a while-loop, except that the loop guard b is checked after each execution of the body c, with the loop repeating as long as the guard stays false. Because of this, the body will always execute at least once. Formally, the evaluation rules are given by:

$$\text{(RepeatTrue)}\frac{\langle \sigma, c\rangle \Downarrow \sigma' \qquad \mathcal{B}[\![b]\!]_{\sigma'} = \top}{\langle \sigma, \textsf{repeat } c \textsf{ until } b \textsf{ end}\rangle \Downarrow \sigma'}$$

$$\text{(RepeatFalse)}\frac{\langle \sigma, c\rangle \Downarrow \sigma' \qquad \mathcal{B}[\![b]\!]_{\sigma'} = \bot \qquad \langle \sigma', \textsf{repeat } c \textsf{ until } b \textsf{ end}\rangle \Downarrow \sigma''}{\langle \sigma, \textsf{repeat } c \textsf{ until } b \textsf{ end}\rangle \Downarrow \sigma''}$$

(3) Finally, we consider two kinds of commands which indicate a certain statement should hold any time this part of the program is reached – the assumption statement "assume b", and the assertion statement "assert b". The two differ as follows:

- If an assertion statement fails, it causes the program to go into an *error state* and exit.
- If an assumption statement fails, the program fails to evaluate at all. In other words, the program gets *stuck* and has no final state.

To formally express the program may go into an error state, we have to change the evaluation relation from "$\langle \sigma, c\rangle \Downarrow \sigma'$" into "$\langle \sigma, c\rangle \Downarrow r$", where the evaluation *result*

$$r ::= \mathsf{norm}(\sigma) \mid \mathsf{err}$$

can state two possible cases: $\mathsf{norm}(\sigma)$ for normally execution with ending state σ, or err for reaching the error state. The inference rules for the original IMP commands need be modified and we should handle errors carefully (read and think about the differences):

$$\text{(Skip)}\frac{}{\langle \sigma, \textsf{skip}\rangle \Downarrow \mathsf{norm}(\sigma)} \qquad \text{(Ass)}\frac{\mathcal{A}[\![a]\!]_\sigma = n}{\langle \sigma, x := a\rangle \Downarrow \mathsf{norm}(\sigma[x \mapsto n])}$$

$$\text{(Seq)}\frac{\langle \sigma, c_1\rangle \Downarrow \mathsf{norm}(\sigma') \qquad \langle \sigma', c_2\rangle \Downarrow r}{\langle \sigma, c_1; c_2\rangle \Downarrow r} \qquad \text{(SeqErr)}\frac{\langle \sigma, c_1\rangle \Downarrow \mathsf{err}}{\langle \sigma, c_1; c_2\rangle \Downarrow \mathsf{err}}$$

$$\text{(IfTrue)}\frac{\mathcal{B}[\![b]\!]_\sigma = \top \qquad \langle \sigma, c_1\rangle \Downarrow r}{\langle \sigma, \textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2 \textsf{ fi}\rangle \Downarrow r} \qquad \text{(IfFalse)}\frac{\mathcal{B}[\![b]\!]_\sigma = \bot \qquad \langle \sigma, c_2\rangle \Downarrow r}{\langle \sigma, \textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2 \textsf{ fi}\rangle \Downarrow r}$$

$$\text{(WhileFalse)}\frac{\mathcal{B}[\![b]\!]_\sigma = \bot}{\langle \sigma, \textsf{while } b \textsf{ do } c \textsf{ end}\rangle \Downarrow \mathsf{norm}(\sigma)} \qquad \text{(WhileTrue)}\frac{\mathcal{B}[\![b]\!]_\sigma = \top \qquad \langle \sigma, c\rangle \Downarrow \mathsf{norm}(\sigma') \qquad \langle \sigma', \textsf{while } b \textsf{ do } c \textsf{ end}\rangle \Downarrow r}{\langle \sigma, \textsf{while } b \textsf{ do } c \textsf{ end}\rangle \Downarrow r}$$

$$\text{(WhileTrueErr)}\frac{\mathcal{B}[\![b]\!]_\sigma = \top \qquad \langle \sigma, c\rangle \Downarrow \mathsf{err}}{\langle \sigma, \textsf{while } b \textsf{ do } c \textsf{ end}\rangle \Downarrow \mathsf{err}}$$

We redefine hoare triples $\{P\}\ c\ \{Q\}$ to mean that, whenever c is started in a state satisfying P, and terminates with result r, then $r = \mathsf{norm}(\sigma)$ (and hence $r \neq \mathsf{err}$) where the state σ satisfies Q.

Your tasks: (a) give the evaluation rules for assumption and assertion statements; (b) state Hoare rules for them.

Solution

(1)

$$\text{Havoc}\ \frac{}{\{P\}\ \mathsf{havoc}\ x\ \{P[n/x]\}}$$

(2)

$$\text{Repeat}\frac{\{P \wedge \neg b\}c\{P\}}{\{P\}\mathsf{repeat}\ c\ \mathsf{until}\ b\ \mathsf{end}\{P \wedge b\}}$$

(3) The evaluation rules should be:

$$(\text{Assume})\frac{\mathcal{B}[\![b]\!]_\sigma = \top}{\langle \sigma, \mathsf{assume}\ b\rangle \Downarrow \mathsf{norm}(\sigma)}$$

$$(\text{AssertTrue})\frac{\mathcal{B}[\![b]\!]_\sigma = \top}{\langle \sigma, \mathsf{assert}\ b\rangle \Downarrow \mathsf{norm}(\sigma)} \qquad (\text{AssertFalse})\frac{\mathcal{B}[\![b]\!]_\sigma = \bot}{\langle \sigma, \mathsf{assert}\ b\rangle \Downarrow \mathsf{err}}$$

The Hoare rules for them are:

$$\text{Assume}\frac{P \wedge b \Rightarrow \top}{\{P\}\mathsf{assume}\ b\{P \wedge b\}}$$

$$\text{Assert}\frac{P \Rightarrow b}{\{P\}\mathsf{assert}\ b\{P\}}$$