ETH zürich

P. Müller and D. Basin

Formal Methods and Functional Programming

Session Sheet 13: Axiomatic Semantics (Hoare Logic)

Viper

Viper (`viper.ethz.ch`) is a verification project, providing a language and tools for verifying functional properties of programs in that language. In particular, Viper supports verification of loops via loop invariants, as you have learned in the course. In this exercise session, you can try out Viper as a tool for helping to practice the important skill of finding loop invariants. There are two simple ways of running Viper: the recommended way is to download the VSCode plugin for Viper, as explained here: `http://www.pm.inf.ethz.ch/research/viper/downloads.html` (make sure to install the "Viper" plugin, and that you have the right dependencies, as explained in the instructions). Alternatively, you can run the Viper tools through the (slower) web interface, available at `http://viper.ethz.ch/examples/blank-example.html`.

The syntax of Viper programs is slightly different to that of IMP, but very similar. Syntax for assignments is the same (although local variables must be declared with an explicit type, using the syntax `var x: Int`. There is no `skip` command, but statement blocks (e.g., branches of conditionals) may be left empty. Viper uses C/Java-like syntax for conditionals and loops. For example, the IMP program

```
if x > 0 then x := x - 1; while x > 0 do x := x - 1 end else skip end
```

could be translated into Viper as follows:

```
if (x > 0) {
  x := x - 1
  while (x > 0) {
    x := x - 1
  }
} else { /* else branch can be omitted */ }
```

Loop invariants can be written between the while-condition and the open brace `{`, e.g.:

```
while (x > 0)
  invariant x >= 0
{
  x := x - 1
}
```

Assignment 1 (Adding the first n natural numbers)

Consider the following IMP program s_1:

```
i := 0;
s := 0;
while i <= x do
  s := s + i;
  i := i + 1
end
```

You can find the corresponding Viper program on the course website: `program_1_adding_numbers.vpr`.

Task 1.1. Assume $\mathtt{x} \geq 0$ initially (i.e. as a pre-condition). What is the value of `s` at the end of the function? Write a post-condition to ensure that `s` has the right value after completion (keyword `ensures` in Viper). Also add the pre-condition to the program (keyword `requires` in Viper).

Solution. Mathematics tells us that $\mathtt{s} = \frac{(\mathtt{x}+1)\mathtt{x}}{2}$. See the solution program for the pre-condition and post-condition in Viper.

Task 1.2. Find a suitable loop invariant which enables Viper to prove the post-condition given the pre-condition.

Solution. A suitable loop invariant is:

$$\mathtt{s} = \frac{(\mathtt{i}-1)\mathtt{i}}{2} \wedge \mathtt{i} \leq \mathtt{x}+1$$

Observe that after the execution `i` equals $\mathtt{x}+1$ to guess the first conjunct. The second conjunct is needed to actually prove $\mathtt{i} = \mathtt{x}+1$ after the execution (we get $\mathtt{i} \geq \mathtt{x}+1$ from the negated loop condition).

Task 1.3. Prove that

$$\{\ \mathtt{x} = X \wedge \mathtt{x} \geq 0\ \}\ s_1\ \{\ \mathtt{s} = \frac{(X+1)X}{2}\ \}$$

Solution. We choose the invariant to be

$$\mathtt{i} \leq \mathtt{x} + 1 \wedge \mathtt{s} = \frac{(\mathtt{i}-1)\mathtt{i}}{2} \wedge \mathtt{x} = X$$

The last conjunct expresses that x remains constant throughout the execution. It is not needed in Viper as the method parameters cannot be modified.

$\{\mathtt{x} = X \wedge \mathtt{x} \geq 0\}$
$\models$
$\{0 \leq \mathtt{x} + 1 \wedge 0 = \frac{0(0-1)}{2} \wedge \mathtt{x} = X\}$
`i := 0;`
$\{\mathtt{i} \leq \mathtt{x} + 1 \wedge 0 = \frac{\mathtt{i}(\mathtt{i}-1)}{2} \wedge \mathtt{x} = X\}$
`s := 0;`
$\{\mathtt{i} \leq \mathtt{x} + 1 \wedge \mathtt{s} = \frac{\mathtt{i}(\mathtt{i}-1)}{2} \wedge \mathtt{x} = X\}$
`while i <= x do`
$\{\mathtt{i} \leq \mathtt{x} \wedge \mathtt{i} \leq \mathtt{x} + 1 \wedge \mathtt{s} = \frac{\mathtt{i}(\mathtt{i}-1)}{2} \wedge \mathtt{x} = X\}$
$\models$
$\{\mathtt{i} \leq \mathtt{x} \wedge \mathtt{s} + \mathtt{i} = \frac{\mathtt{i}(\mathtt{i}+1)}{2} \wedge \mathtt{x} = X\}$
`s := s + i;`
$\{\mathtt{i} \leq \mathtt{x} \wedge \mathtt{s} = \frac{\mathtt{i}(\mathtt{i}+1)}{2} \wedge \mathtt{x} = X\}$
$\models$
$\{\mathtt{i} + 1 \leq \mathtt{x} + 1 \wedge \mathtt{s} = \frac{(\mathtt{i}+1)(\mathtt{i}+1-1)}{2} \wedge \mathtt{x} = X\}$
`i := i + 1`
$\{\mathtt{i} \leq \mathtt{x} + 1 \wedge \mathtt{s} = \frac{\mathtt{i}(\mathtt{i}-1)}{2} \wedge \mathtt{x} = X\}$
`end`
$\{\neg(\mathtt{i} \leq \mathtt{x}) \wedge \mathtt{i} \leq \mathtt{x} + 1 \wedge \mathtt{s} = \frac{\mathtt{i}(\mathtt{i}-1)}{2} \wedge \mathtt{x} = X\}$
$\models$
$\{\mathtt{s} = \frac{X(X+1)}{2}\}$

Task 1.4. So far, we have ignored termination. Does the program terminate? If so, how should we adapt the proof outline from Task 1.3 to prove it?

Solution. The program terminates, because the loop body will be executed exactly $x + 1$ times. We can adapt the proof from Task 1.3 with the loop variant $x - i$, and the rules for total correctness. Recall that the variant must decrease at each iteration, and be at least 0 (provided that the loop condition holds). Thus, i, x, or $-i$ are incorrect variants.

Assignment 2 (Greatest Common Divisor)

Consider the following program s computing the greatest common divisor (gcd) of two given positive integers:

```
b := x;
c := y;
while b # c do
  if b < c then
    c := c - b
  else
    b := b - c
  end
end;
z := b
```

You can find the corresponding Viper program on the course website: `program_2_gcd.vpr`.

Note: In case it helps you to think about the questions, you might want to recall the definition of the *total* function gcd: For *positive* integers x and y, the number z is the greatest common divisor of x and y iff $z|x$ and $z|y$ and there is no z', with $z' > z$, such that $z'|x$ and $z'|y$. Here, $z|x$ means that z divides x, i.e., $z \cdot k = x$, for some $k \in \mathbb{N}$.

Note: For the tasks below, you can write $\mathit{gcd}(m, n)$ in assertions, to denote the actual greatest common divisor of two positive integers m and n. You may assume that $\forall n \cdot \mathit{gcd}(n, n) = n$ and $\forall m, n \in \mathbb{N}^+ \cdot \mathit{gcd}(m + n, n) = \mathit{gcd}(m, n) = \mathit{gcd}(m, m + n)$ (equivalently - $\forall m, n \in \mathbb{N}^+ \cdot m > n \Rightarrow \mathit{gcd}(m - n, n) = \mathit{gcd}(m, n) = \mathit{gcd}(m, m - n)$).

Task 2.1. Formalise the claim that the above program computes the gcd of `x` and `y` as pre- and postcondition $\mathbf{P}$ and $\mathbf{Q}$, respectively.

Solution. The pre- and postconditions are:

$$\mathbf{P} \equiv \mathtt{x} = X \wedge \mathtt{y} = Y \wedge X > 0 \wedge Y > 0$$
$$\mathbf{Q} \equiv \mathtt{z} = \mathit{gcd}(X, Y)$$

Task 2.2. Find a suitable invariant for the loop.

Hint: Consider using a relationship between the input variables `x`, `y` and the 'loop' variables `b` and `c` as part of your loop invariant.

Solution. A suitable loop invariant is:

$$\mathit{gcd}(\mathtt{x}, \mathtt{y}) = \mathit{gcd}(\mathtt{b}, \mathtt{c}) \wedge \mathtt{b} > 0 \wedge \mathtt{c} > 0 \wedge \mathtt{x} = X \wedge \mathtt{y} = Y$$

Task 2.3. Give a suitable loop variant to prove that the program terminates.

Solution. $b + c$: Either b or c decreases, while the other stays constant.

Task 2.4. Prove that

$$\{\ \mathtt{x} = X_0\ \wedge\ \mathtt{y} = Y_0\ \wedge\ X_0 > 0\ \wedge\ Y_0 > 0\ \}\ s\ \{\ \Downarrow \mathtt{z} = \mathit{gcd}(X_0, Y_0)\ \}.$$

Solution.

$\{x = X_0 \wedge y = Y_0 \wedge X_0 > 0 \wedge Y_0 > 0\}$
$\models$
$\{\mathit{gcd}(X_0, Y_0) = \mathit{gcd}(x, y) \wedge x > 0 \wedge y > 0\}$
`b := x;`
$\{\mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b, y) \wedge b > 0 \wedge y > 0\}$
`c := y;`
$\{\mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b, c) \wedge b > 0 \wedge c > 0\}$
`while b#c do`*
$\{b \neq c \wedge \mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b, c) \wedge b > 0 \wedge c > 0 \wedge b + c = V\}$
`if b < c then`
$\{b < c \wedge b \neq c \wedge \mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b, c) \wedge b > 0 \wedge c > 0 \wedge b + c = V\}$
$\models^{(1)}$
$\{\mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b, c - b) \wedge b > 0 \wedge c - b > 0 \wedge b + (c - b) < V\}$
`c := c - b`
$\{\Downarrow\ \mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b, c) \wedge b > 0 \wedge c > 0 \wedge b + c < V\}$
`else`
$\{\neg(b < c) \wedge b \neq c \wedge \mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b, c) \wedge b > 0 \wedge c > 0 \wedge b + c = V\}$
$\models^{(2)}$
$\{\mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b - c, c) \wedge (b - c) > 0 \wedge c > 0 \wedge (b - c) + c < V\}$
`b := b - c`
$\{\Downarrow\ \mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b, c) \wedge b > 0 \wedge c > 0 \wedge b + c < V\}$
`end`
$\{\Downarrow\ \mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b, c) \wedge b > 0 \wedge c > 0 \wedge b + c < V\}$
`end;`
$\{\Downarrow\ \neg(b \neq c) \wedge \mathit{gcd}(X_0, Y_0) = \mathit{gcd}(b, c) \wedge b > 0 \wedge c > 0\}$
$\models$
$\{\Downarrow\ b = \mathit{gcd}(X_0, Y_0)\}$
`z := b`
$\{\Downarrow\ z = \mathit{gcd}(X_0, Y_0)\}$

(*) The side condition holds: $b > 0 \wedge c > 0$ entails $b + c \geq 0$

(1) $0 < b < c$ implies $\mathit{gcd}(b, c) = \mathit{gcd}(b, c - b)$

(2) First, we have $\neg(b < c)$ and $b \neq c$ implies $c < b$. Second, we have $0 < c < b$ implies $\mathit{gcd}(b, c) = \mathit{gcd}(b - c, c)$.

Assignment 3

Consider the following IMP program `s`:

```
i := 0;
r := 1;
while i < k do
  i := i + 1;
  r := r * n
end
```

The following *incorrect* total correctness proof outline contains various errors.

$$
\begin{array}{l}
\{\mathtt{k} \geq 1 \wedge \mathtt{k} = K \wedge \mathtt{n} \geq 1 \wedge \mathtt{n} = N\} \\
\boxed{\texttt{i := 0;}} \\
\{\mathtt{k} \geq 1 \wedge \mathtt{k} = K \wedge \mathtt{n} \geq 1 \wedge \mathtt{n} = N \wedge \mathtt{i} = 0\} \\
\models \\
\{1 = \mathtt{n}^{\mathtt{i}} \wedge \mathtt{n} = N \wedge \mathtt{k} = K\} \\
\boxed{\texttt{r := 1;}} \\
\{\mathtt{r} = \mathtt{n}^{\mathtt{i}} \wedge \mathtt{n} = N \wedge \mathtt{k} = K\} \\
\boxed{\texttt{while i < k do}} \\
\quad \{\mathtt{i} < \mathtt{k} \wedge \mathtt{r} = \mathtt{n}^{\mathtt{i}} \wedge \mathtt{k} - \mathtt{i} = V\} \\
\quad \models \\
\quad \{\mathtt{r} = \mathtt{n}^{\mathtt{i}}\} \\
\quad \boxed{\texttt{i := i + 1;}} \\
\quad \{\mathtt{r} = \mathtt{n}^{\mathtt{i}-1}\} \\
\quad \models \\
\quad \{\mathtt{r} * \mathtt{n} = \mathtt{n}^{\mathtt{i}}\} \\
\quad \boxed{\texttt{r := r * n}} \\
\quad \{\Downarrow\ \mathtt{r} = \mathtt{n}^{\mathtt{i}}\} \\
\boxed{\texttt{end}} \\
\{\Downarrow\ \mathtt{i} \geq \mathtt{k} \wedge \mathtt{r} = \mathtt{n}^{\mathtt{i}} \wedge \mathtt{k} = K \wedge \mathtt{n} = N\} \\
\models \\
\{\Downarrow\ \mathtt{r} = N^K\}
\end{array}
$$

Task 3.1. Find the errors in the above proof outline. For each error, include a short explanation of what is missing or incorrect. Your explanation should include the name of the rule which is wrongly applied. You do not need to correct the mistakes.

Solution. (a) First $\textsc{Ass}_{AX}$: Textual substitution is not correctly applied, $0 = 0$ is needed in the precondition

(b) $\textsc{WhTot}_{AX}$: The loop invariant does not match up in all four relevant positions, i.e. before the loop, at the beginning of the loop body, at the end of the loop body and after the loop

(c) Third $\textsc{Ass}_{AX}$: Textual substitution is not correctly applied, $\mathtt{r} = \mathtt{n}^{\mathtt{i}+1-1}$ is needed in the precondition

(d) Last $\textsc{Cons}_{AX}$: The postcondition of the loop does not entail the postcondition of the program, $\mathtt{i} = \mathtt{k}$ cannot be derived

(e) $\textsc{WhTot}_{AX}$: Textual substitution is not correctly applied in the postcondition of the loop, $\neg(\mathtt{i} < \mathtt{k})$ is needed instead of $\mathtt{i} \geq \mathtt{k}$

(f) $\textsc{WhTot}_{AX}$: The side condition, i.e. $b \wedge \mathbf{P} \models 0 \leq e$, is not checked

(g) $\textsc{WhTot}_{AX}$: The loop variant is not shown to decrease, i.e. $e < Z$ is missing from the postcondition of the body of the loop

Task 3.2. Which of the errors that you identified in part (a) would *also* be errors in a partial correctness proof outline?

Solution. Errors (a) up to and including (e).