Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A. Lochbihler and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 12: Axiomatic Semantics

Submission deadline: May 19th, 2014

Assignment 1 (Loop Invariants in Dafny)

Dafny is a verification tool which can be used to prove functional properties (and termination) of imperative programs. In particular, Dafny supports verification of loops via loop invariants (and variants, when proving termination). In this exercise, you can try out Dafny as a tool for helping experiment with the important skill of finding loop invariants. A version of Dafny can be used via a web interface, at: `http://rise4fun.com/Dafny`

The syntax of Dafny programs is slightly different to that of IMP, but mostly quite similar. Syntax for assignments is the same (although local variables must be declared, using the syntax `var x;` (with a default type of `int`)). There is no `skip` command, but statement blocks (e.g., branches of conditionals) may be left empty. Dafny 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 Dafny as:

```
if (x>0) {
  x := x-1;
  while (x>0) {
    x := x-1;
  }
} else {
  // leave blank for skip - or omit the else-branch
}
```

Loop invariants can be written between the while-condition and the open brace {.
For example:

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

Consider now the following IMP program (which we call s) below:

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

This program computes n^{k} and stores it in the variable r, provided that $k > 0$. We want to find an invariant for the while loop in this program. Recall that a loop invariant is a formula that holds before the loop, and that is preserved by the loop body.

You can find a pre-prepared version of the corresponding Dafny program at, `http://rise4fun.com/Dafny/fpSJ` In this pre-prepared version, the translated program is the body of a method, whose pre-condition (`requires`) is $\mathtt{k} \geq 0$ and whose post-condition is that $\mathtt{r} = \mathtt{n}^{\mathrm{k}}$. Since Dafny doesn't support "power" as a built in operator, we have defined it as a function.

(a) The loop invariant is initially set to be `true`. Try running the verifier by clicking the "play" button. As you would expect, this loop invariant is not strong enough such that, along with the negation of the while-condition, we can prove the post-condition.

(b) Try changing the loop invariant to `i > 0` and run Dafny again. What is the problem now?

(c) Try changing the loop invariant to `i < k` and run Dafny again. What is the problem now?

(d) Try changing the loop invariant to `r == pow(n,i)` and run Dafny. Why does the post-condition not verify?

(e) Find a suitable loop invariant, such that the verifier succeeds with 0 errors.

(f) Using your discovered loop invariant, prove in axiomatic semantics that the above program s computes n^{k}. More formally, show that

$$\vdash \{\, \mathtt{k} \geq 1 \ \wedge \mathtt{k} = K \wedge \mathtt{n} = N \,\}\ s\ \{\, \mathtt{r} = N^{K} \,\}$$

(g) Find a suitable loop invariant for the program found at `http://rise4fun.com/Dafny/cucW`. Your invariant should be strong enough to prove the assertion in the code (i.e., the verifier should succeed with no errors).

(h) Do the same for the program found at `http://rise4fun.com/Dafny/K32F`.

Assignment 2 (Program Correctness)

Let s be the following IMP program:

```
a := 1;
b := 0;
while a<n do
  a := a * 10;
  if (a <= n) then
    b := b + 1
  else
    skip
  end
end
```

You could practice trying to find a loop invariant for this program.

Similar to the "square root" example seen in your session exercises, a suitable loop invariant for the program is:

$(\mathtt{a} \leq \mathtt{n} \Rightarrow \mathtt{a} = 10^{\mathtt{b}}) \wedge (\mathtt{a} > \mathtt{n} \Rightarrow \mathtt{a} = 10^{\mathtt{b}+1}) \wedge 10^{\mathtt{b}} \leq \mathtt{n} \wedge \mathtt{n} = N$

Using this loop invariant (or an alternative one if you prefer), prove that the program computes the floor of $log_{10}(\mathtt{n})$; i.e., that:

$\vdash \{\mathtt{n} = N \wedge \mathtt{n} \geq 1\}\ s\ \{10^{\mathtt{b}} \leq N \wedge N < 10^{\mathtt{b}+1}\}$

Assignment 3 (Sequential Composition)

Show that (for all statements s_1,s_2, and for all predicates P and Q):

$$\vdash \{P\}\ s_1; s_2\ \{Q\} \quad \Leftrightarrow \quad \exists R \cdot \vdash \{P\}\ s_1\ \{R\} \wedge \ \vdash \{R\}\ s_2\ \{Q\}$$

Assignment 4 (Associativity of Sequential Composition)

Show that, for all statements s_1,s_2 and s_3, and for all predicates P and Q:

$$\vdash \{P\}\ (s_1; s_2); s_3\ \{Q\} \quad \Rightarrow \quad \vdash \{P\}\ s_1; (s_2; s_3)\ \{Q\}$$

Assignment 5 (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
```

Convince yourself that the program terminates when `x` and `y` store positive integers (you do not need to prove this).

Tasks: For these tasks, you can write *gcd*(m, n) in assertions, to denote the actual greatest common divisor of two positive integers m and n. You may assume that $\forall n.\textit{gcd}(n, n) = n$ and $\forall m, n.\textit{gcd}(m + n, n) = \textit{gcd}(m, n) = \textit{gcd}(m, m + n)$.

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

(b) Find an invariant for the loop.

(c) Show that $\vdash \{\mathbf{P}\}\ s\ \{\mathbf{Q}\}$.

In case it helps you to think about the questions, you might want to recall the definition of the gcd:
Let x, y be positive integers. 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 $k \in \mathbb{N}$.

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

Note: This exercise is a bit more involved.