Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A. Lochbihler and P. Müller

Formal Methods and Functional Programming

Exercise Sheet 13: Axiomatic Semantics (Total Correctness) and Modelling

Submission deadline: May 26th, 2014

Assignment 1

Let s be the following statement:

```
y := 1;
z := 0;
while z<x do
  y := y * 2;
  z := z + 1
end
```

Our goal is to prove that $\vdash \{\mathrm{x} = X \ \wedge \ X \geq 0\}\ s\ \{\Downarrow \mathrm{y} = 2^X\}$

(a) Find a suitable loop invariant. You may use Dafny for help. **Hint:** Mention all of the variables used in the loop.

(b) Find a suitable loop variant. Again, you may use Dafny. A loop variant is given in Dafny in a `decreases` clause, as in the following example:

```
x := 100;
while x>0
  invariant x>=0;
  decreases x;
{ x:=x-1; }
```

(c) Give a complete proof outline for $\vdash \{\mathrm{x} = X \ \wedge \ X \geq 0\}\ s\ \{\Downarrow \mathrm{y} = 2^X\}$.

Assignment 2

This question concerns termination and the Zune bug, as discussed in the lectures.

(a) Suppose that, for some statement s, the triple $\{\textit{true}\}\ s\ \{\Downarrow\ \textit{true}\}$ can be derived. What does this tell us about s?

(b) Let s be the following (corrected) IMP statement:

```
while (L(year) and 366 < days or not L(year) and 365 < days) do
  if (L(year)) then
      days = days - 366
  else
      days = days - 365;
  end;
  year += 1
end
```

Show that $\vdash \{\textit{true}\}\ s\ \{\Downarrow\ \textit{true}\}$.

Assignment 3

Implement Dekker's algorithm – which is said to be the first known algorithm that (really) solves the mutual exclusion problem for two concurrent processes – in Promela and verify exclusivity using Spin.

In Dekker's algorithm, the critical section is protected by two bit-valued flags. The first one is actually a pair that is used to signal that the first and second, respectively, process is interested in entering the critical section. The other flag alternates and is used to decide which process may enter the critical section in case both are interested.

The algorithm guarantees mutual exclusion to the critical section as well as deadlock and starvation freedom. Instead of relying on low-level test-and-set instructions or interrupts, or on signal/wait thread operations, each process uses busy waiting to detect when it may enter the critical section. This makes the algorithm highly portable between different languages and hardware architectures, but also less efficient in case of lots of contention.

Hint: In order to verify the exclusivity of the critical section, let `mutex` count the number of processes that are currently in the critical section. Then, use the following *supervisor* process (also known as *monitor* or *watchdog*) to assert exclusivity:

```
proctype supervisor() {
  assert(mutex != 2);
}
```

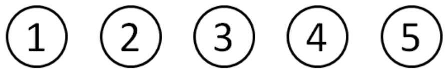
Assignment 4

The problem of the dining philosophers illustrates common issues for concurrent programs. N philosophers sit around a circular table. Each philosopher spends her/his life thinking and eating. In the center of the table is a large platter of spaghetti. Because the spaghetti are long and tangled a philosopher must use two forks to eat them. Since the philosophers can only afford N forks, a single fork is placed between each pair of philosophers, which they have to share. Each philosopher can only reach the forks to her/his immediate left and right with her/his left and right hand, respectively.

(a) You find a skeleton (`philosopher_skeleton.pml`) of a Promela model of the dining philosophers at the course webpage. Complete the model such that each philosopher chooses non-determinstically to pick up her/his left or right fork first. Use Spin to find a deadlock and explain its source. How does the falsification time change when increasing the number N of philosophers?

(b) The deadlock can be avoided when each philosopher behaves more deterministically to pick up the forks (not all philosophers have to behave the same). Model your solution in Promela and use Spin to check that it is indeed deadlock free. Is your model also starvation free (i.e., each philosopher will eventually eat)? (You will learn how to express and check this property by next week).

Assignment 5



Consider the following game between a mole (*Maulwurf*) and a hunter. The mole has five holes as in the above figure. At the beginning of the game, the mole hides in one of these holes. Now the game proceeds in rounds of the following form: the hunter checks one hole to see whether the mole is inside. If it is, the hunter wins the game. If not, the mole must move one hole to the left or right (if it is in the leftmost hole already, it must move to the right and conversely for the rightmost hole). After the mole has moved, the next round starts.

It turns out that the hunter has a strategy to win the game, no matter how the mole moves. One sure winning strategy for the hunter is to check holes in the sequence 2-3-4-2-3-4.

Your task is to model this game in Promela. Your model has to contain the data structure for the holes, the set-up of the initial state (the hiding of the mole), the behaviour of the mole, the implementation of the hunter's winning strategy, and an assertion that ensures that after executing the strategy, the hunter has definitely caught the mole.