Secure and Dependable Systems
Jacobs University Bremen
Dr. Florian Rabe

Homework 5
given: 2017-03-28

Homework 5

You have to submit your solutions as announced in the lecture.
Unless mentioned otherwise, all problems are due 2017-04-06, before the lecture.
There will be no deadline extensions unless mentioned otherwise in the lecture.

Problem 5.1 *Verification: Class Invariants* Points: 2

Argue informally but rigorously why the formula in the stack example from the notes is a strong[1] class invariant.

Solution: For a fresh instance, the class invariant holds because both sides of the equation are 0.

Later, every assignment to the variable *elements* is immediately followed by an assignment to the variable *size*. In both cases, *size* changes by the same amount as the length of *elements*. Thus, the class invariant is preserved.

Problem 5.2 *Verification: Class Invariants* Points: 3

Solve one of the following (using pseudo-code, our example formal system, or a programming language):

1. Give a useful weak class invariant for the following class:

```
class Date(year : int, month : int, day : int)
    fun yesterday() : Date = {...}
    fun tomorrow() : Date = {...}
```

   You can assume that there are no leap years.

   Implement the methods such that the class invariant is preserved.

2. Give a useful strong class invariant for the following class:

```
abstract class PriorityQueue()
    private var data : List[int] = Nil
    fun dequeue() : Option[int] = {if (data == Nil) {None} else {Some(data.head)}}
    fun enqueue(x : int) : unit = {...}
```

   Implement *insert* such that the class invariant is preserved.

Solution:

1. The class invariant should be (or imply) $year! = 0 \wedge 1 \leq month \leq 12 \wedge 1 \leq day \leq numDays(month)$ for a function $numDays(d : int) : int$ that returns the number of days in the respective month.
   The implementation part is straightforward.
2. Assuming that the priority queue should always return the greatest value, the class invariant should be (or imply) **match** $data\ \{Nil \mapsto true \mid Cons(hd, tl) \mapsto \forall i.\ 0 \leq i < length(tl) \Rightarrow hd \geq get(tl, i)\}$, where *get* retrieves an element from a list. The solution that always returns the least value is also correct.
   The implementation part is straightforward.

Other class invariants were possible. Their usefulness is judged based on whether invalid instances are eliminated. Depending on the usefulness, partial credit was given.

[1] I had forgotten to put the word *strong* here. So that part was not required for the solution.

Problem 5.3 *Verification: Pure Functions* Points: 3

Using the definitions from the notes, formally prove $m + n == n + m$ by induction.

To discharge the subgoals, you may use the theorems $zero\_left$ etc.

Solution: We apply the induction rule to the proof goal $m + n == n + m$ to the variable n, which yields two subgoals:

$$\frac{m : nat \vdash m + zero == zero + m \qquad m : nat, n : nat, p : \textbf{proof}\ m + n == n + m \vdash m + succ(n) == succ(n) + m}{m : nat, n : nat \vdash m + n == n + m}$$

The left subgoal can be discharged by using $zero\_right$, $zero\_left$, and reflexivity of $==$ to evaluate

$$m + zero == zero + m \quad \rightsquigarrow \quad m == m \quad \rightsquigarrow \quad true.$$

The right subgoal can be discharged by using $succ\_right$, $succ\_left$, p, and reflexivity of $==$ to evaluate

$$m + succ(n) == succ(n) + m \quad \rightsquigarrow \quad succ(m + n) == succ(n + m) \quad \rightsquigarrow$$

$$succ(m + n) == succ(m + n) \quad \rightsquigarrow \quad true.$$

Problem 5.4 *Proof Assistants: Practice* Points: 4

Install Isabelle or Coq (see the links in the lecture notes).

Write a simple pure recursive function and verify that it meets its specification using the tool. Generate executable code from your function.

Submit a reasonable combination of screen shots, shell logs, system output etc. that demonstrates you completed the task.

You may use any example that is already part of the available documentation or tutorials. But you have to prove that you actually installed the system and ran the verification. For example, you can copy an example from the tutorial, rename the function to your name, and then run the verification.