Secure and Dependable Systems
Jacobs University Bremen
Dr. Florian Rabe

Homework 4
given: 2017-03-20

# Homework 4

You have to submit your solutions as announced in the lecture.
**Unless mentioned otherwise, all problems are due 2017-03-30, before the lecture.**
There will be no deadline extensions unless mentioned otherwise in the lecture.

---

## Problem 4.1 *Verification*

Points: 4

Choose two of the following functions, and give appropriate function specifications, loop invariants, and termination orderings for them.

```
fun $EuclideanAlgorithm(m : int, n : int) : int$ =
    $x := m$
    $y := n$
    while $x \neq y$
        if $x < y$
            $y := y - x$
        else
            $x := x - y$
    return $x$
```

---

**Solution:**

- precondition: $m > 0 \wedge n > 0$
- postcondition (for output $z$): $z == \gcd(m, n)$
- loop invariant: $\gcd(m, n) == \gcd(x, y)$
- termination ordering: $x + y$

---

```
fun $factorial(n : \mathbb{N}) : \mathbb{N}$ =
    $product := 1$
    $factor := 1$
    while $factor \leq n$
        $product := product \cdot factor$
        $factor := factor + 1$
    return $product$
```

---

**Solution:**

- precondition: *true* (nothing)
- postcondition (for output $z$): $z == n!$
- loop invariant: $product \cdot factor \cdot (factor + 1) \cdot \ldots \cdot n == n!$
- termination ordering: $n - factor$

---

```
fun $linearFibonacci(n : \mathbb{N}) : \mathbb{N}$ =
    if $n \leq 1$
        $n$
    else
        $prev := 0$
        $current := 1$
        $i = 1$
        while $i < n$
            $next := current + prev$
            $prev := current$
            $current := next$
            $i := i + 1$
        **return** $current$
```

**Solution:** Let $F_n$ be the $n$-th Fibonacci number with $F(0) = 0$ and $F(1) = 1$.

- precondition: *true* (nothing)
- postcondition (for output $z$): $z == F_n$
- loop invariant: $prev == F_{i-1} \wedge current == F_i$
- termination ordering: $n - i$

```
fun $revertImmutable[A](x : List[A]) : List[A]$ =
    $rest := x$
    $rev := []$
    while $rest \neq []$
        $rev := cons(rest.head, rev)$
        $rest := rest.tail$
    return $rev$
```

**Solution:** Let $revert(x)$ and $length(x)$ be the reversal and length of the list $x$, and let $x+y$ be the concatenation of the lists $x$ and $y$.

- precondition: *true* (nothing)
- postcondition (for output $z$): $z == revert(x)$
- loop invariant: $x == revert(rev) + rest$
- termination ordering: $length(rest)$

## Problem 4.2 *Dynamic Logic: Practice* Points: 3

Install either Why3 or KeY (see the links in the lecture notes).

Write a simple function that contains a while-loop in it. Annotate it with pre/postcondition and loop invariant and use the tool to verify that it meets the specification.

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.

## Problem 4.3 *Dynamic Logic: Theory* Points: 3

Prove the soundness of the rules for if and while.

**Solution:** For a state $q$ that supplies values for all undefined variables in $\Gamma$, we write $\Gamma; q \vdash t \rightsquigarrow t'$ if $t$ evaluates to $t'$ in state $q$.

**if** The rule is:

$$\frac{\Gamma \vdash C \Rightarrow [t]F \qquad \Gamma \vdash \neg C \Rightarrow [t']F}{\Gamma \vdash [\textbf{if } (C)\ \{t\} \textbf{ else } \{t'\}]F}$$

We assume the premises are theorems:

$$(1) \qquad \Gamma; q \vdash C \Rightarrow [t]F \rightsquigarrow \mathit{true} \qquad \text{for all } q$$

$$(2) \qquad \Gamma; q \vdash \neg C \Rightarrow [t']F \rightsquigarrow \mathit{true} \qquad \text{for all } q$$

Our goal is to show that the conclusion is a theorem:

$$\Gamma; q \vdash [\textbf{if } (C)\ \{t\} \textbf{ else } \{t'\}]F \rightsquigarrow \mathit{true} \text{ for all } q$$

To prove the goal, we assume an arbitrary state $q$ and distinguish two cases:

- $\Gamma; q \vdash C \ \rightsquigarrow \ \mathit{true}$: The evaluation rule for **if** yields

  $$\Gamma \vdash [\textbf{if } (C) \ \{t\} \textbf{ else } \{t'\}]F \ \rightsquigarrow \ t$$

  Because $C$ is pure, this is independent of $q$ and can be substituted anywhere. So our goal becomes

  $$\Gamma; q \vdash [t]F \ \rightsquigarrow \ \mathit{true}$$

  By applying modus ponens to (1), we obtain

  $$\Gamma; q \vdash [t]F$$

  which concludes the proof.

- $\Gamma; q \vdash C \ \rightsquigarrow \ \mathit{false}$: This case proceeds accordingly using (2) instead of (1).

**while** The rule is:

$$\frac{\Gamma \vdash I \qquad \Gamma^* \vdash (I \wedge C) \Rightarrow [t]I \qquad \Gamma^* \vdash (I \wedge \neg C) \Rightarrow F}{\Gamma \vdash [\textbf{while } C \ \{t\}]F}$$

We assume the premises are theorems:

$$(1) \qquad \Gamma; q \vdash I \ \rightsquigarrow \ \mathit{true} \qquad \text{for all } q$$

$$(2) \qquad \Gamma^*; q \vdash (I \wedge C) \Rightarrow [t]I \ \rightsquigarrow \ \mathit{true} \qquad \text{for all } q$$

$$(3) \qquad \Gamma^*; q \vdash (I \wedge \neg C) \Rightarrow F \ \rightsquigarrow \ \mathit{true} \qquad \text{for all } q$$

Our goal is to show that the conclusion is a theorem:

$$\Gamma; q \vdash [\textbf{while } C \ \{t\}]F \ \rightsquigarrow \ \mathit{true} \text{ for all } q$$

If the while-loop never terminates, this holds trivially. So we can assume it terminates after $N$ iterations. Let $q_0 = q$ and let $q_{n+1}$ be the state after evaluating $t$ in state $q_n$ We prove by induction on $n$ that

$$(*) \qquad \Gamma^*; q_n \vdash I \ \rightsquigarrow \ \mathit{true} \qquad \text{for all } n = 0, \ldots, N$$

- Case for 0: This follows immediately from (1).
- Case for $n+1 \leq N$: This follows from (2) after observing that $n+1 \leq N$ is only possible if $C$ holds in $q_n$.

Our goal becomes

$$\Gamma^*; q_N \vdash F \ \rightsquigarrow \ \mathit{true}$$

Moreover, we know that $(**)$ $\neg C$ must hold in $q_N$. Applying $(*)$ at $N$ and $(**)$ to (3) yields the goal.