**Technische Universität München**
**Fakultät für Informatik**
**Dr. Peter Lammich**
**Simon Wimmer**

**WS 2018/19**
**04.12.2018**

# Semantics of Programming Languages

**Exercise Sheet 8**

## Exercise 8.1 Exponentiation by Squaring

A classic algorithm for computing $x^n$ works by repeated squaring, using the following recurrence:

- $x^n = x * \overset{2}{x}^{(n-1)/2}$ if $n$ is odd
- $x^n = \overset{2}{x}^{n/2}$ if $n$ is even

Below you find an implementation of this algorithm. Prove that it fulfills the specification!

```
program_spec ex_by_sq
  assumes "n≥0"
  ensures "r = x₀ ^ nat n₀"
defines ‹
    r = 1;
    while (n≠0)
      @invariant‹undefined :: bool›
      @variant‹nat undefined›
    {
      if (n mod 2 == 1) {
        r = r * x
      };
      x = x * x;
      n = n / 2
    }
  ›
```

## Exercise 8.2 Finding the Least Index of an Element

Write a program that checks whether a certain element $x$ is contained in an array $a$. If $a$ contains $x$, your program should "return" the *least* index at which $x$ can be found in some variable. Otherwise your program should set the same variable to the array upper bound. Specify this property formally and prove that your program fulfills this property.

## Exercise 8.3 Right Rotation

Write down and verify a program that rotates an array to the right. That is, your program should fulfill the following specification:

```
program_spec rotate_right
  assumes "0<n"
  ensures "∀ i∈{0..<n}. a i = a₀ ((i−1) mod n)"
```

Hint: $0 < b \Longrightarrow - 1 \text{ mod } b = b - 1$ (*zmod_minus1*)!

## Homework 8.1 Extended Euclidean Algorithm

*Submission until Tuesday, December 11, 2018, 10:00am.*

The following is a direct translation of the pseudocode for the Extended Euclidean algorithm as described by the English version of Wikipedia (https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm):

```
program_spec euclid_extended
  assumes "a>0 ∧ b>0"
  ensures "old_r = gcd a₀ b₀ ∧ gcd a₀ b₀ = a₀ * old_s + b₀ * old_t"
defines ‹
    s = 0;    old_s = 1;
    t = 1;    old_t = 0;
    r = b;    old_r = a;
    while (r≠0)
      @invariant‹undefined :: bool›
      @variant‹nat undefined›
    {
      quotient = old_r / r;
      temp = old_r;
      old_r = r;
      r = temp − quotient * r;
      temp = old_s;
      old_s = s;
      s = temp − quotient * s;
      temp = old_t;
      old_t = t;
      t = temp − quotient * t
    }
```

```
⟩
```

Prove that the algorithm fulfills the specification! Have a look at the example in the Wikipedia article to understand which invariants you need. You may need to prove some additional facts about integer arithmetic. Remember to use the theorems *algebra_simps* and *gcd_red_int*.

## **Homework 8.2** Balanced Sum

*Submission until Tuesday, December 11, 2018, 10:00am.*

On sheet three we have considered the task of determining whether a list of integers can split in two parts that have the same sum. We now want to consider the same task again on arrays. We say that a $i$ is an *equilibrium index* of an array if it splits the array such that the sum of the elements to left of it and the sum of the elements starting with $i$ and to its right have the same sum:

**definition** *"is_equil a l h i ≡ l≤i ∧ i<h ∧ (∑j=l..<i. a j) = (∑j=i..<h. a j)"*

Prove that the following program fulfills its specification, i.e. that it determines an equilibrium index by either returning such an index in the variable $i$ or by setting $i$ to the array upper bound $h$ to signal that no such index exists.

```
program_spec equilibrium
  assumes ⟨l≤h⟩
  ensures ⟨is_equil a l h i ∨ i=h ∧ (∄i. is_equil a l h i)⟩
  defines ⟨
    usum=0; i=l;
    while (i<h)
      @variant⟨nat undefined⟩
      @invariant⟨undefined :: bool⟩
    {
      usum = usum + a[i]; i=i+1
    };
    i=l; lsum=0;
    while (usum ≠ lsum ∧ i<h)
      @variant⟨nat undefined⟩
      @invariant⟨undefined :: bool⟩
    {
      lsum = lsum + a[i];
      usum = usum − a[i];
      i=i+1
    }
  ⟩
```