Last Name: ______________________________

First Name: ______________________________

Matriculation Number: ______________________________

Exercise	Points	Score
Single Choice	15	
Well-Definedness of Functional Programs	27	
Verification of Functional Programs	31	
Verification of Imperative Programs	27	
$\sum$	100	

- The time limit for the exam is 100 minutes, so 1 point = 1 minute.
- The available points per exercise are written in the margin.
- Write on the printed exam for Exercises 1 and 4 and use blank sheets for the rest.
- Your answers can be written in English or German.

Exercise 1: Single Choice 15

For each statement indicate whether it is true (✓) or false (✗). Giving the correct answer is worth 3 points, giving no answer counts 1 point, and giving the wrong answer counts 0 points (for that statement).

1. ____ The property that a functional program P is well-defined is a necessary criterion to ensure that the semantics of P is well-defined.
2. ____ Whenever termination of a functional program can be proven solely by the subterm criterion, then termination can also be proven solely by the size-change principle.
3. ____ Given a well-defined and finite functional program, the set of extracted axioms (equality of constructors, defining equations, induction principle for datatypes) is also finite.
4. ____ Ramsey's theorem was used to prove the following statement: whenever there are no maximal multigraphs, then size-change termination is satisfied.
5. ____ The substitution lemma is always satisfied, i.e., in particular it is not restricted to the standard model $\mathcal{M}$ of a well-defined functional program.

Exercise 2: Well-Definedness of Functional Programs 27

Consider the following functional program where shuffle shuffles the order of list elements.

$$\mathsf{append}(\mathsf{Cons}(x, xs), ys) = \mathsf{Cons}(x, \mathsf{append}(xs, ys)) \tag{1}$$
$$\mathsf{rev}(\mathsf{Nil}, ys) = ys \tag{2}$$
$$\mathsf{rev}(\mathsf{Cons}(x, xs), ys) = \mathsf{rev}(xs, \mathsf{Cons}(x, ys)) \tag{3}$$
$$\mathsf{shuffle}(\mathsf{Cons}(x, xs)) = \mathsf{Cons}(x, \mathsf{shuffle}(\mathsf{rev}(\mathsf{shuffle}(xs), \mathsf{Nil}))) \tag{4}$$

(a) Turn the program into a well-defined functional program (without considering termination). (8)

- Add all missing data type definitions via data.
  Note: there is no unique solution.
- Provide a suitable type for the functions rev and shuffle, assuming a suitable type for append.
- If the program is not pattern-disjoint or not pattern-complete, then modify the equations and/or add new equations to obtain a pattern-disjoint and pattern-complete program.

(b) Compute all dependency pairs of rev and shuffle. Indicate which of these pairs can be removed by the subterm-criterion. (5)

(c) Compute the set of usable equations w.r.t. the dependency pairs of $\mathsf{shuffle}^\sharp$. It suffices to mention the indices of the equations. (4)

(d) Prove termination of shuffle by completing the following polynomial interpretation p. (10)

$$p_{\mathsf{shuffle}^\sharp}(t) = \dots$$
$$p_{\mathsf{rev}}(xs, ys) = \dots$$
$$p_{\mathsf{Cons}}(x, xs) = \dots$$
$$\dots = \dots$$

Hints:

- You only need numbers 0 and 1 in the polynomial interpretation.
- Use intuition and don't try to compute the constraints symbolically.
- It makes sense to start filling in suitable interpretations by looking at the constraints of the dependency pairs for $\mathsf{shuffle}^\sharp$ first, and then look at the constraints of the usable equations from the previous part.

Exercise 3: Verification of Functional Programs 31

Consider the following functional program on binary trees and lists of natural numbers, where the well-known data-type definitions for Nat, List, and Tree have been omitted. We further assume that addition on natural numbers has been defined and several properties on addition are already known, e.g., addition is associative, commutative, and has 0 as left- and right-neutral element. We now define functions to flatten a tree to a list ($\mathsf{flatten}$), to append two lists (append), to compute the sum of elements in a tree ($\mathsf{sumtree}$), and to compute the sum of elements in a list ($\mathsf{sumlist}$).

$$
\begin{aligned}
\mathsf{flatten}(\mathsf{Leaf}) &= \mathsf{Nil}\\
\mathsf{flatten}(\mathsf{Node}(\ell, x, r)) &= \mathsf{append}(\mathsf{flatten}(\ell), \mathsf{Cons}(x, \mathsf{flatten}(r)))\\
\mathsf{sumtree}(\mathsf{Leaf}) &= 0\\
\mathsf{sumtree}(\mathsf{Node}(\ell, x, r)) &= +(\mathsf{sumtree}(\ell), +(x, \mathsf{sumtree}(r)))\\
\mathsf{append}(\mathsf{Nil}, ys) &= ys\\
\mathsf{append}(\mathsf{Cons}(x, xs), ys) &= \mathsf{Cons}(x, \mathsf{append}(xs, ys))\\
\mathsf{sumlist}(\mathsf{Nil}) &= 0\\
\mathsf{sumlist}(\mathsf{Cons}(x, xs)) &= +(x, \mathsf{sumlist}(xs))
\end{aligned}
$$

Prove that the formula

$$\forall t.\ \mathsf{sumlist}(\mathsf{flatten}(t)) =_{\mathsf{Nat}} \mathsf{sumtree}(t) \tag{A}$$

is a theorem in the standard model by using induction and equational reasoning via $\rightsquigarrow$.

- Briefly state on which variable(s) you perform induction, and which induction scheme you are using.
- Write down each case explicitly and also write down the IH that you get, including quantifiers.
- Write down each single $\rightsquigarrow$-step in your proof.
- You may assume all usual properties on arithmetic on natural numbers, i.e., properties about $+$ and 0.
- You will need one further auxiliary property (B). Write down this property and prove it in the same way as it is required for formula (A). Only exception: if you need further auxiliary properties for proving (B), then just state these properties without proving them.

Exercise 4: Verification of Imperative Programs 27

Consider the following program P. As input it takes an array $a[0], \ldots, a[n-1]$ of length $n \geq 0$ and some value x. Either it computes an array index i such that $a[i] = x$, or it detects that x does not occur in the array, and then $i = -1$.

```
i := -1;
m := n;
while (m != 0 && i = -1) {
  m := m - 1;
  if a[m] = x then i := m else m := m    (* else branch is skip-step *)
}
```

(a) Formulate pre- and post-conditions that state partial correctness of P. Here, both cases should be covered, i.e., x occurs in the array or x does not occur in the array. (4)
You do not need to prove the property!

(b) Construct a proof tableau for proving termination. (10)

```
i := -1;

m := n;

while (m != 0 && i = -1) {

  m := m - 1;

  if a[m] = x then

    i := m

    else

    m := m

}
```

(c) Construct a proof tableau for proving one part of the partial correctness property: whenever the resulting i differs from -1 then i is a proper index of the array and $a[i] = x$. (13)
To this end, define a suitable invariant `Inv(i,m)` and use this abbreviation in the tablau.

```
i := -1;

m := n;

while (m != 0 && i = -1) {

  m := m - 1;

  if a[m] = x then

    i := m

    else

    m := m

}
```

Here is another blank template that can be used for a second attempt of either (b) or (c). If you use this template, please clearly indicate which of your solutions should (not) be graded.

```
i := -1;

m := n;

while (m != 0 && i = -1) {

  m := m - 1;

  if a[m] = x then

    i := m

    else

    m := m

}
```