Lastname: ____

Firstname: ____

Matriculation Number: __________________________________

Exercise	Points	Score
Single Choice	12	
Well-Definedness of Functional Programs	34	
Verification of Functional Programs	30	
Verification of Imperative Programs	24	
$\sum$	100	

- You have 100 minutes to solve this exam, so 1 point = 1 minute.
- The available points per exercise are written in the margin.
- Write on the printed exam and use extra blank sheets if more space is required.
- Your answers can be written in English or German.

Exercise 1: Single Choice 12

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. ____ Well-definedness of functional programs is undecidable.
2. ____ A calculus $\vdash$ is complete w.r.t. some semantic property $\models$ if and only if it is satisfied, that for all formulas φ, whenever $\vdash \varphi$ then $\models \varphi$.
3. ____ Consider a functional program and let P be a set of dependency pairs, all having the shape $f^\sharp(\ldots) \to f^\sharp(\ldots)$. Whenever the set of usable equations of P is non-empty, then the subterm-criterion cannot be applied on P, i.e., it will not be possible to delete any pair of P.
4. ____ The algorithm for pattern disjointness invokes the unification algorithm.

Exercise 2: Well-Definedness of Functional Programs 34

Consider the following functional program that implements quick-sort.

$$\begin{aligned}
\mathsf{data\ Nat} &= \mathsf{Zero} : \mathsf{Nat} && (1)\\
&\mid \mathsf{Succ} : \mathsf{Nat} \to \mathsf{Nat} && (2)\\
\mathsf{data\ List} &= \mathsf{Nil} : \mathsf{List} && (3)\\
&\mid \mathsf{Cons} : \mathsf{Nat} \times \mathsf{List} \to \mathsf{List} && (4)\\
\mathsf{append}(\mathsf{Nil}, xs) &= xs && (5)\\
\mathsf{append}(\mathsf{Cons}(x, xs), ys) &= \mathsf{Cons}(x, \mathsf{append}(xs, ys)) && (6)\\
\mathsf{le}(\mathsf{Zero}, y) &= \mathsf{True} && (7)\\
\mathsf{le}(\mathsf{Succ}(x), \mathsf{Zero}) &= \mathsf{False} && (8)\\
\mathsf{le}(\mathsf{Succ}(x), \mathsf{Succ}(y)) &= \mathsf{le}(x, y) && (9)\\
\mathsf{first}(\mathsf{Pair}(xs, ys)) &= xs && (10)\\
\mathsf{second}(\mathsf{Pair}(xs, ys)) &= ys && (11)\\
\mathsf{add_pair}(y, \mathsf{True}, \mathsf{Pair}(ls, hs)) &= \mathsf{Pair}(\mathsf{Cons}(y, ls), hs) && (12)\\
\mathsf{add_pair}(y, \mathsf{False}, \mathsf{Pair}(ls, hs)) &= \mathsf{Pair}(ls, \mathsf{Cons}(y, hs)) && (13)\\
\mathsf{partition}(x, \mathsf{Nil}) &= \mathsf{Pair}(\mathsf{Nil}, \mathsf{Nil}) && (14)\\
\mathsf{partition}(x, \mathsf{Cons}(y, ys)) &= \mathsf{add_pair}(y, \mathsf{le}(y, x), \mathsf{partition}(x, ys)) && (15)\\
\mathsf{q_sort}(\mathsf{Nil}) &= \mathsf{Nil} && (16)\\
\mathsf{q_sort}(\mathsf{Cons}(x, xs)) &= \mathsf{append}(\mathsf{q_sort}(\mathsf{first}(\mathsf{partition}(x, xs))), \mathsf{Cons}(x, \mathsf{q_sort}(\mathsf{second}(\mathsf{partition}(x, xs))))) && (17)
\end{aligned}$$

(a) Complete missing type informations in the program: (10)

- Add missing data type definitions via data.
- Provide a suitable type for each of the functions first, add_pair, partition, and q_sort.

The result should be a well-defined functional program – assuming suitable types for the other functions le, append, second in the program.

(b) Compute all dependency pairs of add_pair, partition and q_sort. Indicate which of these pairs can be removed by the subterm-criterion. (8)

(c) Compute the set of usable equations w.r.t. the dependency pairs of $\mathsf{q\_sort}^\sharp$. It suffices to mention the indices of the equations. (6)

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

$$p_{\mathsf{q\_sort}^\sharp}(xs) = xs$$
$$p_{\mathsf{Cons}}(x, xs) = 1 + xs$$
$$p_{\mathsf{Nil}} = 0$$

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 $\mathsf{q\_sort}^\sharp$ first, and then look at the constraints of the usable equations from the previous part.

Exercise 3: Verification of Functional Programs 30

Consider the following functional program on natural numbers and Booleans.

$$\begin{aligned}
\mathsf{plus}(\mathsf{Zero}, y) &= y \\
\mathsf{plus}(\mathsf{Succ}(x), y) &= \mathsf{plus}(x, \mathsf{Succ}(y)) \\
\mathsf{even}(\mathsf{Zero}) &= \mathsf{True} \\
\mathsf{even}(\mathsf{Succ}(\mathsf{Zero})) &= \mathsf{False} \\
\mathsf{even}(\mathsf{Succ}(\mathsf{Succ}(x))) &= \mathsf{even}(x)
\end{aligned}$$

Prove that the formula

$$\forall x.\ \mathsf{even}(\mathsf{plus}(x, x)) =_{\mathsf{Bool}} \mathsf{True}$$

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 will need at least one further auxiliary property. Write down this property and prove it in the same way in that you have to prove the main property.
- You may write just b instead of $b =_{\mathsf{Bool}} \mathsf{True}$ within your proofs. For example, the property you have to prove can be written just as $\forall x.\ \mathsf{even}(\mathsf{plus}(x, x))$.

Exercise 4: Verification of Imperative Programs 24

Consider the following program P where at the end x will store the logarithm of z w.r.t. basis b.

```
x := 0;
y := 1;
while (y < z) {
  x := x + 1;
  y := y * b;
}
```

(a) Construct a proof tableau for proving partial correctness. Here, we only consider that an upper-bound of the logarithm is computed: $b^x \geq z$. (12)

```
    (| b > 0 |)

x = 0;

y = 1;

while (y < z) {

  x := x + 1;

  y : = y * b;

}

    (| b^x >= z |)
```

(b) The program terminates whenever $b > 1$ and a suitable variant e to prove termination is $max(z - y, 0)$. Complete the proof tableau below to prove termination formally. Hint: In order to prove that the variant decreases in every loop iteration, you will have to find an invariant on b and y such that $y < y \cdot b$. (12)

```
(| b > 1 |)

x = 0;

y = 1;

while (y < z) {

  x := x + 1;

  y : = y * b;

}
```