Lastname: ____________________

Firstname: ____________________

Matriculation Number: ____________________

Exercise	Points	Score
Well-Definedness of Functional Programs	26	
Verification of Functional Programs	35	
Single Choice	6	
Verification of Imperative Programs	33	
$\sum$	100	

- The duration of 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 and use extra blank sheets if more space is required.
- Your answers can be written in English or German.

Exercise 1: Well-Definedness of Functional Programs $\boxed{26}$

Consider the following functional program that computes the maximum of a list of natural numbers.

$$\begin{aligned}
\mathsf{data\ Nat} &= \mathsf{Zero} : \mathsf{Nat} && (1)\\
&\mid \mathsf{Succ} : \mathsf{Nat} \to \mathsf{Nat} && (2)\\
\mathsf{data\ List} &= \mathsf{Nil} : \mathsf{List} && (3)\\
&\mid \mathsf{Cons} : \mathsf{Nat} \times \mathsf{List} \to \mathsf{List} && (4)\\
\mathsf{max}(\mathsf{Succ}(x), \mathsf{Succ}(y)) &= \mathsf{Succ}(\mathsf{max}(y, x)) && (5)\\
\mathsf{max}(\mathsf{Zero}, y) &= y && (6)\\
\mathsf{max}(x, \mathsf{Zero}) &= x && (7)\\
\mathsf{maxlist}(\mathsf{Cons}(x, \mathsf{Nil})) &= x && (8)\\
\mathsf{maxlist}(\mathsf{Cons}(x, \mathsf{Cons}(y, xs))) &= \mathsf{maxlist}(\mathsf{Cons}(\mathsf{max}(x, y), xs)) && (9)
\end{aligned}$$

(a) Is the program pattern disjoint and/or pattern complete? If not, then modify it in a sensible way such that it becomes both pattern disjoint and pattern complete and briefly explain your modifications. (8)

Solution: max is not pattern disjoint and we modify the second equation to

$$\mathsf{max}(\mathsf{Zero}, \mathsf{Succ}(y)) = \mathsf{Succ}(y)$$

which obviously preserves the semantics.

maxlist is not pattern complete, since the case maxlist(Nil) is missing. Again, defining maxlist(Nil) = Zero is an equation that preserves the semantics, since Zero is the neutral element of the max-operation over natural numbers.

(b) First, compute all dependency pairs of the program. Second, indicate which of these pairs can be removed by the subterm-criterion and/or by the size-change principle with a brief justification why these pairs can be removed or why they cannot be removed. (8)

Solution: There are only two dependency pairs.

$$\mathsf{max}^\sharp(\mathsf{Succ}(x), \mathsf{Succ}(y)) = \mathsf{max}^\sharp(y, x) \tag{10}$$
$$\mathsf{maxlist}^\sharp(\mathsf{Cons}(x, \mathsf{Cons}(y, xs))) = \mathsf{maxlist}^\sharp(\mathsf{Cons}(\mathsf{max}(x, y), xs)) \tag{11}$$

The dependency pair for max can be removed by the size-change principle: there are only two multigraphs: $\{1 \succ 2, 2 \succ 1\}$ and $\{1 \succ 1, 2 \succ 2\}$ where only the latter one is idem-potent, and that contains a strict edge from an argument to itself.

The dependency pair for maxlist cannot be removed, as there is only one argument of $\mathsf{maxlist}^\sharp$, and $\mathsf{Cons}(x, \mathsf{Cons}(y, xs)) \rhd \mathsf{Cons}(\mathsf{max}(x, y), xs)$ does not hold (although the list in obviously getting shorter).

(c) Compute and write down the set of usable equations w.r.t. the dependency pairs of $\mathsf{maxlist}^\sharp$. (10)

Afterwards, prove termination of maxlist by a polynomial interpretation. You only have to provide the interpretations of the relevant function symbols, and do not have to argue why these interpretations satisfy the termination constraints.

$$\begin{aligned} p_{\mathsf{maxlist}^\sharp}(xs) &= \dots \\ \dots &= \dots \end{aligned}$$

Solution: The usable equations are the max-equations.

For the interpretation we interpret natural numbers by themselves, we encode the list length. max is interpreted as sum.

$$\begin{aligned} p_{\mathsf{maxlist}^\sharp}(xs) &= xs \\ p_{\mathsf{Succ}}(x) &= 1 + x \\ p_{\mathsf{Zero}} = p_{\mathsf{Nil}} &= 0 \\ p_{\mathsf{max}}(x, y) &= x + y \\ p_{\mathsf{Cons}}(x, xs) &= 1 + xs \end{aligned}$$

Exercise 2: Verification of Functional Programs $\boxed{35}$

Consider the following functional program on lists over some element type E and Booleans, where $\mathsf{f} : \mathsf{E} \to \mathsf{Bool}$ is some defined function where the defining equations are not of interest for this exam question.

$$\begin{aligned}
\mathsf{f}(\dots) &= \dots\\
\mathsf{if}(\mathsf{True}, xs, ys) &= xs\\
\mathsf{if}(\mathsf{False}, xs, ys) &= ys\\
\mathsf{filter}(\mathsf{Nil}) &= \mathsf{Nil}\\
\mathsf{filter}(\mathsf{Cons}(x, xs)) &= \mathsf{if}(\mathsf{f}(x), \mathsf{Cons}(x, \mathsf{filter}(xs)), \mathsf{filter}(xs))
\end{aligned}$$

(a) Specify the property that $\mathsf{filter}(xs)$ is always shorter than xs. To this end, you should define a recursive function $\mathsf{shorter} : \mathsf{List} \times \mathsf{List} \to \mathsf{Bool}$ such that (5)

- shorter checks that the second argument is at least as long as the first argument, and
- shorter does not invoke any other auxiliary algorithm; in particular, there should not be any definition of length or lessOrEqual or similar functions.

and then write down the intended property with the help of shorter.

(b) Provide a proof of the specified property by using induction and equational reasoning via $\rightsquigarrow$. (30)

- 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 any IH that you get, including quantifiers.
- Write down each single $\rightsquigarrow$-step in your proof.
- You will require a case-analysis within the inductive proof for the if-then-else, e.g., you might have to consider two cases $\mathsf{f}(x) = \mathsf{True}$ and $\mathsf{f}(x) = \mathsf{False}$.
- You will need at least one further auxiliary property, which is a monotonicity property of the shorter-function of the form $\mathsf{shorter}(\dots,\dots) \longrightarrow \mathsf{shorter}(\dots,\dots)$. Write down this property and prove it in the same way in that you have to prove the main property.
  If you require further auxiliary properties, just state them without giving a proof.
- You may write just b instead of $b =_{\mathsf{Bool}} \mathsf{True}$ within your proofs.
- In case you did not solve part (a), you can ask the instructor for the solution to part (a) and continue here, but then part (a) will be graded with 0 points.

Solution:

1. We implement $\mathsf{shorter}$ as follows:

$$\begin{aligned}\mathsf{shorter}(\mathsf{Nil}, xs) &= \mathsf{True}\\ \mathsf{shorter}(\mathsf{Cons}(y, ys), \mathsf{Nil}) &= \mathsf{False}\\ \mathsf{shorter}(\mathsf{Cons}(y, ys), \mathsf{Cons}(x, xs)) &= \mathsf{shorter}(ys, xs)\end{aligned}$$

   and then the desired property becomes

$$\forall xs.\ \mathsf{shorter}(\mathsf{filter}(xs), xs)$$

2. We prove the property by structural induction on xs.

   - case Nil:
     There is no IH and we derive:

$$\begin{aligned}&\mathsf{shorter}(\mathsf{filter}(\mathsf{Nil}), \mathsf{Nil})\\ \rightsquigarrow\ &\mathsf{shorter}(\mathsf{Nil}, \mathsf{Nil})\\ \rightsquigarrow\ &\mathsf{True}\\ \rightsquigarrow\ &\mathsf{true}\end{aligned}$$

   - case $\mathsf{Cons}(x, xs)$:
     The IH is $\mathsf{shorter}(\mathsf{filter}(xs), xs)$ and we derive:

$$\begin{aligned}&\mathsf{shorter}(\mathsf{filter}(\mathsf{Cons}(x, xs)), \mathsf{Cons}(x, xs))\\ \rightsquigarrow\ &\mathsf{shorter}(\mathsf{if}(\mathsf{f}(x), \mathsf{Cons}(x, \mathsf{filter}(xs)), \mathsf{filter}(xs)), \mathsf{Cons}(x, xs))\end{aligned}$$

     and here we get stuck with pure simplification. In order to proceed we perform a case analysis on $\mathsf{f}(x)$.

     – if $\mathsf{f}(x) = \mathsf{True}$ then we continue as follows

$$\begin{aligned}&\mathsf{shorter}(\mathsf{if}(\mathsf{f}(x), \mathsf{Cons}(x, \mathsf{filter}(xs)), \mathsf{filter}(xs)), \mathsf{Cons}(x, xs))\\ =\ &\mathsf{shorter}(\mathsf{if}(\mathsf{True}, \mathsf{Cons}(x, \mathsf{filter}(xs)), \mathsf{filter}(xs)), \mathsf{Cons}(x, xs))\\ \rightsquigarrow\ &\mathsf{shorter}(\mathsf{Cons}(x, \mathsf{filter}(xs)), \mathsf{Cons}(x, xs))\\ \rightsquigarrow\ &\mathsf{shorter}(\mathsf{filter}(xs), xs)\\ \rightsquigarrow\ &\mathsf{True}\\ \rightsquigarrow\ &\mathsf{true}\end{aligned}$$

     – if $\mathsf{f}(x) = \mathsf{False}$ then we continue as follows

$$\begin{aligned}&\mathsf{shorter}(\mathsf{if}(\mathsf{f}(x), \mathsf{Cons}(x, \mathsf{filter}(xs)), \mathsf{filter}(xs)), \mathsf{Cons}(x, xs))\\ =\ &\mathsf{shorter}(\mathsf{if}(\mathsf{False}, \mathsf{Cons}(x, \mathsf{filter}(xs)), \mathsf{filter}(xs)), \mathsf{Cons}(x, xs))\\ \rightsquigarrow\ &\mathsf{shorter}(\mathsf{filter}(xs), \mathsf{Cons}(x, xs))\end{aligned}$$

     We again get stuck, since the IH has the stronger property $\mathsf{shorter}(\mathsf{filter}(xs), xs)$; hence we need an auxiliary property that tells us that adding a Cons in the second argument preserves being shorter. Our auxiliary property is

$$\forall xs, ys, z.\ \mathsf{shorter}(ys, xs) \longrightarrow \mathsf{shorter}(ys, \mathsf{Cons}(z, xs))$$

     and using this lemma and the IH with conditional simplification leads to

$$\mathsf{shorter}(\mathsf{filter}(xs), \mathsf{Cons}(x, xs)) \rightsquigarrow \mathsf{True} \rightsquigarrow \mathsf{true}$$

In order to show the auxiliary property we use induction on xs and ys w.r.t. algorithm $\mathsf{shorter}$.

- first equation:

$$\begin{aligned} & \mathsf{shorter}(\mathsf{Nil}, xs) \longrightarrow \mathsf{shorter}(\mathsf{Nil}, \mathsf{Cons}(z, xs)) \\ \rightsquigarrow\ & \mathsf{shorter}(\mathsf{Nil}, xs) \longrightarrow \mathsf{True} \\ \rightsquigarrow\ & \mathsf{shorter}(\mathsf{Nil}, xs) \longrightarrow \mathsf{true} \\ \rightsquigarrow\ & \mathsf{true} \end{aligned}$$

- second equation:

$$\begin{aligned} & \mathsf{shorter}(\mathsf{Cons}(y, ys), \mathsf{Nil}) \longrightarrow \mathsf{shorter}(\mathsf{Cons}(y, ys), \mathsf{Cons}(z, \mathsf{Nil})) \\ \rightsquigarrow\ & \mathsf{False} \longrightarrow \mathsf{shorter}(\mathsf{Cons}(y, ys), \mathsf{Cons}(z, \mathsf{Nil})) \\ \rightsquigarrow\ & \mathsf{false} \longrightarrow \mathsf{shorter}(\mathsf{Cons}(y, ys), \mathsf{Cons}(z, \mathsf{Nil})) \\ \rightsquigarrow\ & \mathsf{true} \end{aligned}$$

- third equation with IH $\forall z.\ \mathsf{shorter}(ys, xs) \longrightarrow \mathsf{shorter}(ys, \mathsf{Cons}(z, xs))$

$$\begin{aligned} & \mathsf{shorter}(\mathsf{Cons}(x, xs), \mathsf{Cons}(y, ys)) \longrightarrow \mathsf{shorter}(\mathsf{Cons}(y, ys), \mathsf{Cons}(z, \mathsf{Cons}(x, xs))) \\ \rightsquigarrow\ & \mathsf{shorter}(ys, xs) \longrightarrow \mathsf{shorter}(\mathsf{Cons}(y, ys), \mathsf{Cons}(z, \mathsf{Cons}(x, xs))) \\ \rightsquigarrow\ & \mathsf{shorter}(ys, xs) \longrightarrow \mathsf{shorter}(ys, \mathsf{Cons}(x, xs)) \\ \rightsquigarrow\ & \mathsf{True} \\ \rightsquigarrow\ & \mathsf{true} \end{aligned}$$

Exercise 3: Single Choice $\boxed{6}$

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. _✓_ Statement: A correctness proof via refinement roughly works as follows: one shows that an abstract algorithm satisfies some property P, and that a concrete algorithm is a correct implementation of the abstract algorithm in order to show that the concrete algorithm has property P.
2. _✗_ Assume there is some algorithm A computing function f with certificate generation. Further assume that there is a certificate checking algorithm C for function f, in particular supporting the certificates that are generated by A.

   Statement: If algorithm A is indeed a correct implementation of f, then all certificates generated by A will be accepted by C.

Exercise 4: Verification of Imperative Programs $\boxed{33}$

Consider the following program P. You can assume that array a has a length of n, i.e., $a = [a[0], ..., a[n-1]]$ and $n \geq 0$.

```
b := false;
i := n;
while (i > 0) {
  i := i - 1;
  b := (b || (a[i] == x));
}
```

(a) Figure out what P computes in variable b. Write this down informally, and also provide a specification in form of a Hoare triple. (3)

Solution: b stores whether x occurs in a. Since a and x are not modified by the program, we do not introduce logical variables in this specification.
Formally: $(\!|n \geq 0|\!)\, P\, (\!|b = (\exists 0 \leq k < n.\ a[k] = x)|\!)$

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

If required, it is allowed to add preconditions which are essential to ensure termination.
Clearly specify the variant e.
$e = i$

```
    (| n >= 0 |)

b := false;

    (| n >= 0 |)

i := n;

    (| i >= 0 |)

while (i > 0) {

    (| e0 = i >= 0 ∧ i > 0 |)
    (| e0 > i - 1 >= 0 |)

  i := i - 1;

    (| e0 > i >= 0 |)

  b := b || a[i] == x;

    (| e0 > i >= 0 |)

}
```

(c) Prove partial correctness of the Hoare triple that you specified in the first part. (20)

```
    (| n >= 0 |)
    (| n >= 0 ∧ false = (∃ n <= k < n. a[k] = x) |)

b := false;

    (| n >= 0 ∧ b = (∃ n <= k < n. a[k] = x) |)

i := n;

    (| i >= 0 ∧ b = (∃ i <= k < n. a[k] = x) |)

while (i > 0) {

    (| i > 0 ∧ i >= 0 ∧ b = (∃ i <= k < n. a[k] = x) |)
    (| i - 1 >= 0 ∧ (b || a[i-1] == x) = (∃ i-1 <= k < n. a[k] = x) |)

  i := i - 1;

    (| i >= 0 ∧ (b || a[i] == x) = (∃ i <= k < n. a[k] = x) |)

  b := b || a[i] == x;

    (| i >= 0 ∧ b = (∃ i <= k < n. a[k] = x) |)

}

    (| !i > 0 ∧ i >= 0 ∧ b = (∃ i <= k < n. a[k] = x) |)
    (| b = (∃ 0 <= k < n. a[k] = x) |)
```

Fully formally, one should add $i \leq n$ to the invariant, in order to verify the implication within the while-loop, but it is not expected to add this condition when solving the exam.