Lastname: _____

Firstname: _____

Matriculation Number: _____

Exercise	Points	Score
Single Choice (20 minutes)	12	
Well-Definedness of Functional Programs (45 minutes)	34	
Verification of Functional Programs (40 minutes)	30	
Verification of Imperative Programs (35 minutes)	24	
$\sum$	100	

- The intended time for a regular paper exam would be 100 minutes, so 1 point = 1 minute. Because of the sequentiality of the exam, extra-time was added for each exercise.
- 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.
- Upload the solution for each exercise as a single PDF-file into OLAT. Use `convert` or similar programs to combine multiple images into one PDF, if required.

Exercise 1: Single Choice (20 minutes) 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 (45 minutes) 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.

Solution:

$$\begin{aligned}
\mathsf{data\ Bool} &= \mathsf{True} : \mathsf{Bool} \mid \mathsf{False} : \mathsf{Bool}\\
\mathsf{data\ Pair} &= \mathsf{Pair} : \mathsf{List} \times \mathsf{List} \to \mathsf{Pair}\\
\mathsf{first} &: \mathsf{Pair} \to \mathsf{List}\\
\mathsf{add_pair} &: \mathsf{Nat} \times \mathsf{Bool} \times \mathsf{Pair} \to \mathsf{Pair}\\
\mathsf{partition} &: \mathsf{Nat} \times \mathsf{List} \to \mathsf{Pair}\\
\mathsf{q_sort} &: \mathsf{List} \to \mathsf{List}
\end{aligned}$$

(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)

Solution: For add_pair there are no dependency pairs. The other dependency pairs are

$$\mathsf{partition}^\sharp(x, \mathsf{Cons}(y, ys)) \to \mathsf{partition}^\sharp(x, ys) \tag{18}$$

$$\mathsf{q\_sort}^\sharp(\mathsf{Cons}(x, xs)) \to \mathsf{q\_sort}^\sharp(\mathsf{first}(\mathsf{partition}(x, xs))) \tag{19}$$

$$\mathsf{q\_sort}^\sharp(\mathsf{Cons}(x, xs)) \to \mathsf{q\_sort}^\sharp(\mathsf{second}(\mathsf{partition}(x, xs))) \tag{20}$$

Only dependency pair (18) can be removed by the subterm criterion.

(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)

Solution: Since the two dependency pairs invoke first, second, and partition, clearly the corresponding equations are usable. Since partition invokes le and add_pair, also these equations become usable. Hence, in total we know that equations (7) – (15) are usable.

(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.

Solution: The interpretations of partition, first, and second must be quite small, since otherwise there is no strict decrease for the dependency pairs. However, the last argument also cannot be ignored since otherwise the usable equations are not satisfiable. This gives rise to:

$$p_{\mathsf{partition}}(x, xs) = xs$$
$$p_{\mathsf{first}}(q) = q$$
$$p_{\mathsf{second}}(q) = q$$

Looking at the equations of add_pair and partition one further figures out the following two interpretations:

$$p_{\mathsf{add\_pair}}(x, b, q) = 1 + q$$
$$p_{\mathsf{Pair}}(xs, ys) = xs + ys$$

Finally, the le-constraints need to oriented, where here we just ignore natural numbers and Booleans.

$$p_{\mathsf{le}}(x, y) = 0$$
$$p_{\mathsf{True}} = 0$$
$$p_{\mathsf{False}} = 0$$
$$p_{\mathsf{Zero}} = 0$$
$$p_{\mathsf{Succ}}(x) = 0$$

Exercise 3: Verification of Functional Programs (40 minutes) 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 $\forall x.\ \mathsf{even}(\mathsf{plus}(x, x))$.

Solution: We prove the property by induction on x.

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

$$\begin{aligned} & \mathsf{even}(\mathsf{plus}(\mathsf{Zero}, \mathsf{Zero})) \\ \rightsquigarrow\ & \mathsf{even}(\mathsf{Zero}) \\ \rightsquigarrow\ & \mathsf{True} \\ \rightsquigarrow\ & \mathsf{true} \end{aligned}$$

- case $\mathsf{Succ}(x)$:
  The IH is $\mathsf{even}(\mathsf{plus}(x, x))$ and we derive:

$$\begin{aligned} & \mathsf{even}(\mathsf{plus}(\mathsf{Succ}(x), \mathsf{Succ}(x))) \\ \rightsquigarrow\ & \mathsf{even}(\mathsf{plus}(x, \mathsf{Succ}(\mathsf{Succ}(x)))) \end{aligned}$$

  and here we get stuck, since we have to move the Succ's outside the plus. Hence, we use the auxiliary property

$$\forall x, y.\ \mathsf{plus}(x, \mathsf{Succ}(y)) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(x, y)) \qquad (*)$$

  to continue this case of the inductive proof

$$\begin{aligned} & \mathsf{even}(\mathsf{plus}(x, \mathsf{Succ}(\mathsf{Succ}(x)))) \\ \rightsquigarrow\ & \mathsf{even}(\mathsf{Succ}(\mathsf{plus}(x, \mathsf{Succ}(x)))) \\ \rightsquigarrow\ & \mathsf{even}(\mathsf{Succ}(\mathsf{Succ}(\mathsf{plus}(x, x)))) \\ \rightsquigarrow\ & \mathsf{even}(\mathsf{plus}(x, x)) \\ \overset{IH}{\rightsquigarrow}\ & \mathsf{True} \\ \rightsquigarrow\ & \mathsf{true} \end{aligned}$$

We still need to prove $(*)$ which we do by induction on x for arbitrary y.

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

$$\begin{aligned} & \mathsf{plus}(\mathsf{Zero}, \mathsf{Succ}(y)) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(\mathsf{Zero}, y)) \\ \rightsquigarrow\ & \mathsf{Succ}(y) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(\mathsf{Zero}, y)) \\ \rightsquigarrow\ & \mathsf{Succ}(y) =_{\mathsf{Nat}} \mathsf{Succ}(y) \\ \rightsquigarrow\ & \mathsf{true} \end{aligned}$$

- case $\mathsf{Succ}(x)$:
  The IH is $\forall y.\ \mathsf{plus}(x, \mathsf{Succ}(y)) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(x, y))$ and we derive:

$$\begin{aligned} & \mathsf{plus}(\mathsf{Succ}(x), \mathsf{Succ}(y)) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(\mathsf{Succ}(x), y)) \\ \rightsquigarrow\ & \mathsf{plus}(x, \mathsf{Succ}(\mathsf{Succ}(y))) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(\mathsf{Succ}(x), y)) \\ \overset{IH}{\rightsquigarrow}\ & \mathsf{Succ}(\mathsf{plus}(x, \mathsf{Succ}(y))) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(\mathsf{Succ}(x), y)) \\ \rightsquigarrow\ & \mathsf{Succ}(\mathsf{plus}(x, \mathsf{Succ}(y))) =_{\mathsf{Nat}} \mathsf{Succ}(\mathsf{plus}(x, \mathsf{Succ}(y)) \\ \rightsquigarrow\ & \mathsf{true} \end{aligned}$$

Exercise 4: Verification of Imperative Programs (35 minutes) 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 |)

    (| 1 = b^0 |)

x = 0;

    (| 1 = b^x |)

y = 1;

    (| y = b^x |)

while (y < z) {

    (| y = b^x ∧ y < z |)
    (| y * b = b^(x + 1) |)

  x := x + 1;

    (| y * b = b^x |)

  y : = y * b;

    (| y = b^x |)

}

    (| y = b^x ∧ y >= z |)

    (| 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 |)

    (| 1 >= 1 ∧ b > 1 ∧ max(z - 1, 0) >= 0 |)

x = 0;

    (| 1 >= 1 ∧ b > 1 ∧ max(z - 1, 0) >= 0 |)

y = 1;

    (| y >= 1 ∧ b > 1 ∧ max(z - y, 0) >= 0 |)

while (y < z) {

    (| y >= 1 ∧ b > 1 ∧ y < z ∧ e0 = max(z - y, 0) >= 0 |)
    (| y * b >= 1 ∧ b > 1 ∧ e0 > max(z - y * b, 0) >= 0 |)

  x := x + 1;

    (| y * b >= 1 ∧ b > 1 ∧ e0 > max(z - y * b, 0) >= 0 |)

  y : = y * b;

    (| y >= 1 ∧ b > 1 ∧ e0 > max(z - y, 0) >= 0 |)

}
```