Last Name: ______________________________

First Name: ______________________________

Matriculation Number: ______________________________

Exercise	Points	Score
Single Choice	6	
Well-Definedness of Functional Programs	31	
Verification of Functional Programs	36	
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 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. _✗_ 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.

Exercise 2: Well-Definedness of Functional Programs 31

Consider the following functional program where $\mathsf{shuffle}$ converts binary trees into lists and shuffles the order of the elements.

$$\begin{aligned}
\mathsf{append}(\mathsf{Nil}, xs) &= xs && (1)\\
\mathsf{append}(\mathsf{Cons}(x, xs), ys) &= \mathsf{Cons}(x, \mathsf{append}(xs, ys)) && (2)\\
\mathsf{mirror}(\mathsf{Leaf}) &= \mathsf{Leaf} && (3)\\
\mathsf{mirror}(\mathsf{Node}(\ell, x, r)) &= \mathsf{Node}(\mathsf{mirror}(r), x, \mathsf{mirror}(\ell)) && (4)\\
\mathsf{shuffle}(\mathsf{Node}(\ell, x, r)) &= \mathsf{append}(\mathsf{shuffle}(\mathsf{mirror}(r)), \mathsf{Cons}(x, \mathsf{shuffle}(\mathsf{mirror}(\ell)))) && (5)
\end{aligned}$$

(a) Turn the program into a well-defined functional program (without considering termination). (10)

- Add all missing data type definitions via data.
  Note: there is no unique solution.
- Provide a suitable type for the functions mirror and $\mathsf{shuffle}$, 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.

Solution:

$$\begin{aligned}
\mathsf{data\ Element} &= \mathsf{Elem} : \mathsf{Element} && \text{(this could also be Booleans, integers, \dots)}\\
\mathsf{data\ List} &= \mathsf{Nil} : \mathsf{List} \mid \mathsf{Cons} : \mathsf{Element} \times \mathsf{List} \to \mathsf{List}\\
\mathsf{data\ Tree} &= \mathsf{Leaf} : \mathsf{Tree} \mid \mathsf{Node} : \mathsf{Tree} \times \mathsf{Element} \times \mathsf{Tree} \to \mathsf{Tree}\\
\mathsf{mirror} &: \mathsf{Tree} \to \mathsf{Tree}\\
\mathsf{shuffle} &: \mathsf{Tree} \to \mathsf{List}\\
\mathsf{shuffle}(\mathsf{Leaf}) &= \mathsf{Nil} && \text{(added equation)}
\end{aligned}$$

(b) Compute all dependency pairs of mirror and $\mathsf{shuffle}$. Indicate which of these pairs can be removed by the subterm-criterion. (7)

Solution: The DPs are

$$\begin{aligned}
\mathsf{mirror}^\sharp(\mathsf{Node}(\ell, x, r)) &\to \mathsf{mirror}^\sharp(\ell) && (6)\\
\mathsf{mirror}^\sharp(\mathsf{Node}(\ell, x, r)) &\to \mathsf{mirror}^\sharp(r) && (7)\\
\mathsf{shuffle}^\sharp(\mathsf{Node}(\ell, x, r)) &\to \mathsf{shuffle}^\sharp(\mathsf{mirror}(\ell)) && (8)\\
\mathsf{shuffle}^\sharp(\mathsf{Node}(\ell, x, r)) &\to \mathsf{shuffle}^\sharp(\mathsf{mirror}(r)) && (9)
\end{aligned}$$

Only the dependency pairs of $\mathsf{mirror}^\sharp$ can be removed by the subterm criterion.

(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)

> **Solution:** Since the two dependency pairs invoke mirror, clearly the two mirror equations (3) and (4) are usable. However, no further equation is usable.

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

$$\begin{aligned} p_{\mathsf{shuffle}^\sharp}(t) &= \dots \\ p_{\mathsf{mirror}}(t) &= \dots \\ p_{\mathsf{Node}}(\ell, x, r) &= \dots \\ \dots &= \dots \end{aligned}$$

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.

> **Solution:** The interpretation of $\mathsf{shuffle}^\sharp$ can just be the identity, since there is only one argument. Consequently, Node must be large enough to orient both dependency pairs. Hence, the interpretation of $\mathsf{Node}(\ell, x, r)$ must be at least $\ell + r$. In order to obtain a strict decrease, it actually must be at least $1 + \ell + r$. If we additionally set the interpretation of leafs to 0, then the interpretation counts the number of nodes in a tree. Since mirror does neither increase nor decrease the number of nodes, we assign it the identity function. In total this gives rise to:
>
> $$\begin{aligned} p_{\mathsf{shuffle}^\sharp}(t) &= t \\ p_{\mathsf{mirror}}(t) &= t \\ p_{\mathsf{Node}}(\ell, x, r) &= 1 + \ell + r \\ p_{\mathsf{Leaf}} &= 0 \end{aligned}$$
>
> For this interpretation indeed all dependency pairs of $\mathsf{shuffle}^\sharp$ are oriented strictly and the usable equations weakly. Hence, termination is proven.

Exercise 3: Verification of Functional Programs $\boxed{36}$

Consider the following functional program on natural numbers and lists of natural numbers, where the well-known data-type definitions for Nat and List have been omitted. Observe that the definition of plus is not the standard one.

$$\begin{aligned}
\mathsf{plus}(x, \mathsf{Zero}) &= x \\
\mathsf{plus}(x, \mathsf{Succ}(y)) &= \mathsf{plus}(\mathsf{Succ}(x), y) \\
\mathsf{sumlist}(\mathsf{Nil}) &= \mathsf{Zero} \\
\mathsf{sumlist}(\mathsf{Cons}(x, xs)) &= \mathsf{plus}(x, \mathsf{sumlist}(xs)) \\
\mathsf{listsum}(\mathsf{Nil}) &= \mathsf{Zero} \\
\mathsf{listsum}(\mathsf{Cons}(x, \mathsf{Nil})) &= x \\
\mathsf{listsum}(\mathsf{Cons}(x, \mathsf{Cons}(y, xs))) &= \mathsf{listsum}(\mathsf{Cons}(\mathsf{plus}(x, y), xs))
\end{aligned}$$

Prove that the formula

$$\forall xs.\ \mathsf{listsum}(xs) =_{\mathsf{Nat}} \mathsf{sumlist}(xs) \tag{A}$$

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 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.

Solution:

We first prove associativity of addition which is the mentioned auxiliary property (B).

$$\forall x, y, z.\ \mathsf{plus}(\mathsf{plus}(x, y), z) =_{\mathsf{Nat}} \mathsf{plus}(x, \mathsf{plus}(y, z)) \tag{B}$$

Here, we perform structural induction on z for arbitrary x and y.

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

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

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

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

and here we get stuck, since we have to show that a Succ can be moved outside a plus. To this end we use the auxiliary property

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

to continue this case of the inductive proof

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

We finally prove (A) by induction on xs using induction w.r.t. the algorithm $\mathsf{listsum}$.

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

$$\begin{aligned}
& \mathsf{listsum}(\mathsf{Nil}) =_{\mathsf{Nat}} \mathsf{sumlist}(\mathsf{Nil}) \\
\leadsto\ & \mathsf{Zero} =_{\mathsf{Nat}} \mathsf{sumlist}(\mathsf{Nil}) \\
\leadsto\ & \mathsf{Zero} =_{\mathsf{Nat}} \mathsf{Zero} \\
\leadsto\ & \mathsf{true}
\end{aligned}$$

- case $\mathsf{Cons}(x, \mathsf{Nil})$:
  There is no IH and we derive:

$$\begin{aligned}
& \mathsf{listsum}(\mathsf{Cons}(x, \mathsf{Nil})) =_{\mathsf{Nat}} \mathsf{sumlist}(\mathsf{Cons}(x, \mathsf{Nil})) \\
\leadsto\ & x =_{\mathsf{Nat}} \mathsf{sumlist}(\mathsf{Cons}(x, \mathsf{Nil})) \\
\leadsto\ & x =_{\mathsf{Nat}} \mathsf{plus}(x, \mathsf{sumlist}(\mathsf{Nil})) \\
\leadsto\ & x =_{\mathsf{Nat}} \mathsf{plus}(x, \mathsf{Zero}) \\
\leadsto\ & x =_{\mathsf{Nat}} x \\
\leadsto\ & \mathsf{true}
\end{aligned}$$

- case $\mathsf{Cons}(x, \mathsf{Cons}(y, xs))$:
  The IH is $\mathsf{listsum}(\mathsf{Cons}(\mathsf{plus}(x, y), xs)) =_{\mathsf{Nat}} \mathsf{sumlist}(\mathsf{Cons}(\mathsf{plus}(x, y), xs))$ and we derive:

$$\begin{aligned}
& \mathsf{listsum}(\mathsf{Cons}(x, \mathsf{Cons}(y, xs))) =_{\mathsf{Nat}} \mathsf{sumlist}(\mathsf{Cons}(x, \mathsf{Cons}(y, xs))) \\
\leadsto\ & \mathsf{listsum}(\mathsf{Cons}(\mathsf{plus}(x, y), xs)) =_{\mathsf{Nat}} \mathsf{sumlist}(\mathsf{Cons}(x, \mathsf{Cons}(y, xs))) \\
\overset{IH}{\leadsto}\ & \mathsf{sumlist}(\mathsf{Cons}(\mathsf{plus}(x, y), xs)) =_{\mathsf{Nat}} \mathsf{sumlist}(\mathsf{Cons}(x, \mathsf{Cons}(y, xs))) \\
\leadsto\ & \mathsf{plus}(\mathsf{plus}(x, y), \mathsf{sumlist}(xs)) =_{\mathsf{Nat}} \mathsf{sumlist}(\mathsf{Cons}(x, \mathsf{Cons}(y, xs))) \\
\leadsto\ & \mathsf{plus}(\mathsf{plus}(x, y), \mathsf{sumlist}(xs)) =_{\mathsf{Nat}} \mathsf{plus}(x, \mathsf{sumlist}(\mathsf{Cons}(y, xs))) \\
\leadsto\ & \mathsf{plus}(\mathsf{plus}(x, y), \mathsf{sumlist}(xs)) =_{\mathsf{Nat}} \mathsf{plus}(x, \mathsf{plus}(y, \mathsf{sumlist}(xs))) \\
\overset{(B)}{\leadsto}\ & \mathsf{plus}(x, \mathsf{plus}(y, \mathsf{sumlist}(xs))) =_{\mathsf{Nat}} \mathsf{plus}(x, \mathsf{plus}(y, \mathsf{sumlist}(xs))) \\
\leadsto\ & \mathsf{true}
\end{aligned}$$

Here, without property (B) we would get stuck and could not apply the $\overset{(B)}{\leadsto}$-step.

Exercise 4: Verification of Imperative Programs 27

Consider the following program P that computes the division of x by y, i.e., the quotient q and the remainder r is computed such that $x = q \cdot y + r \wedge r < y$ should be satisfied.

```
q := 0;
while (x >= y) {
  q := q + 1;
  x := x - y;
}
r := x;
```

(a) Formulate pre- and post-conditions that state partial correctness of P. (3)

> **Solution:** Since x is modified during the execution, we have to store the initial value of x in a logical variable, here: x_0.
>
> `(| x0 = x |) P (| x0 = q * y + r ∧ r < y |)`

(b) Construct a proof tableau for proving partial correctness. (12)

```
    (| x0 = x |)
    (| x0 = 0 * y + x |)

q := 0;

    (| x0 = q * y + x |)

while (x >= y) {

    (| x0 = q * y + x ∧ x >= y |)
    (| x0 = (q + 1) * y + (x - y) |)

  q := q + 1;

    (| x0 = q * y + (x - y) |)

  x := x - y;

    (| x0 = q * y + x |)

}

    (| x0 = q * y + x ∧ ! (x >= y) |)
    (| x0 = q * y + x ∧ x < y  |)

r := x;

    (| x0 = q * y + r ∧ r < y |)
```

(c) Find a reasonable precondition that ensures termination and complete the proof tableau for proving termination formally. (12)

```
    (| y > 0 |)
    (| y > 0 ∧ max(x, 0) >= 0 |)

q = 0;

    (| y > 0 ∧ max(x, 0) >= 0|)

while (x >= y) {

    (| y > 0 ∧ x >= y ∧ e0 = max(x, 0) >= 0 |)
    (| y > 0 ∧ e0 > max(x - y, 0) >= 0 |)

  q := q + 1;

    (| y > 0 ∧ e0 > max(x - y, 0) >= 0 |)

  x : = x - y;

    (| y > 0 ∧ e0 > max(x, 0) >= 0 |)

}

(* the part after the while-loop should be omitted *)
```

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.

```
q := 0;

while (x >= y) {

  q := q + 1;

  x := x - y;

}

r := x;
```