Program Verification	SS 2023	LVA 703083+703084
Sheet 3		Deadline: March 29, 2023, 10am

- Prepare your solutions on paper.
- Mark the exercises in OLAT before the deadline.
- Marking an exercise means that a significant part of that exercise has been treated.

Exercise 1 *Type-Checking of Formulas* **6 p.**

Consider the type-checking algorithm for formulas from the (solution of the) previous exercise sheet. Prove soundness of the type-checking algorithm as in slides 2/40 – 2/41.

$$typeCheckFormula\ \Sigma\ \mathcal{V}\ \mathcal{P}\ \varphi = return\ () \longrightarrow \varphi \in \mathcal{F}(\Sigma, \mathcal{P}, \mathcal{V})$$

Be precise when applying induction: what kind of induction? on which property $P(\ldots)$? on which variables? which variables are arbitrary?

Exercise 2 *Data Type Definitions* **7 p.**

Consider slides 3/4 and 3/6.

1. Consider the following sequence of datatypes that define rose trees, i.e., trees where each node may have arbitrarily many children. (3 points)

   $$\mathsf{data}\ \mathsf{Nat} = \mathsf{Zero} : \mathsf{Nat} \mid \mathsf{Succ} : \mathsf{Nat} \to \mathsf{Nat}$$
   $$\mathsf{data}\ \mathsf{Tree} = \mathsf{Node} : \mathsf{Nat} \times \mathsf{Tree\_List} \to \mathsf{Tree}$$
   $$\mathsf{data}\ \mathsf{Tree\_List} = \mathsf{Nil} : \mathsf{Tree\_List} \mid \mathsf{Cons} : \mathsf{Tree} \times \mathsf{Tree\_List} \to \mathsf{Tree\_List}$$

   - Describe the universes of trees and tree-lists as inductive sets.
   - Are all universes non-empty? For each non-empty universe provide an element that is in the universe.
   - Why is the definition not allowed wrt. slide 3/4?

2. Adjust the handling of datatype definitions on slides 4 and 6, such that the above definition of rose trees is permitted. This might involve several modifications. Also formulate an alternative property that ensures non-empty universes. This property can be formulated via a mathematical description or via an algorithm. (You do not have to prove that the formulated property or algorithm is correct.) (4 points)

Exercise 3 *Functional Programming* **7 p.**

Consider slides 3/14 – 3/20.

1. Specify an algorithm for subtraction of two natural numbers within the functional programming language defined in the slides and evaluate "$3-2$" and "$2-3$" step-by-step on paper. (You do not have to explicitly compute or mention matching substitutions!) (2 points)

2. Specify an algorithm for the division of two natural numbers within the functional programming language defined in the slides. Evaluate "$2/2$" step-by-step on paper. How does your algorithm handle division-by-zero? How does your algorithm handle non-exact division, e.g., dividing 1 by 2. (2 points)

3. Function definitions on slide 3/15 are quite restricted, e.g., no mutual recursion, no if-then-else, no built-in integers, etc. (3 points)
   - Try to modify the definition of function definitions on slide 3/15 in a way that allows mutual recursion.
   - Ensure that the even-odd definitions on slide 3/17 are accepted.
   - Are there any adjustments of the operational semantics on slide slide 3/22 required? If so, which ones?