

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

Exercise 1 *Models in Many-Sorted Logic*
6 p.

Consider \mathcal{M} , \mathcal{P} and Σ of [slide 2/26](#), where we further assume that \mathcal{M} contains symbols $\text{plus} : \text{Nat} \times \text{Nat} \rightarrow \text{Nat}$ and $\text{eq} \subseteq \text{Nat} \times \text{Nat}$ where $\text{plus}^{\mathcal{M}}(x, y) = x + y$ and $\text{eq}^{\mathcal{M}} = \{(n, n) \mid n \in \mathbb{N}\}$.

1. Show

$$\mathcal{M} \models_{\alpha} \forall x, y. \text{eq}(\text{plus}(x, y), \text{plus}(y, x))$$

by unfolding the definition of \models on [slides 2/25](#) step by step. You can use that addition on \mathbb{N} is commutative. (3 points)

2. Provide a different model \mathcal{M}' such that all of the following formulas are valid in \mathcal{M}' .

$$\begin{aligned} \forall xs, ys, zs. \text{app}(xs, \text{app}(ys, zs)) &= \text{app}(\text{app}(xs, ys), zs) \\ \forall xs. \text{app}(xs, xs) &= xs \\ \neg \forall xs, ys. xs &= ys \end{aligned}$$

You do not need to prove that \mathcal{M}' has the desired property. (3 points)

Exercise 2 *Error Monads*
4 p.

Prove that the implementations of `>>=` and `return` for the `Maybe`-type ([slide 2/31](#)) satisfy the three monad laws on [slide 2/32](#).

Hint: use equational reasoning in combination with case analysis on values of type `Maybe`. For each monad law, at most one case analysis is required. (4 points)

Exercise 3 *Type-Checking of Formulas*
10 p.

Consider the type-checking algorithm on [slide 2/37](#).

1. Encode the example signature Σ of [slide 2/26](#) in Haskell as a function of type `Sig`. Hint: Haskell has a predefined function `lookup`. (1 point)
2. Encode the following set \mathcal{V} in Haskell as a function of type `Vars`: whenever the name of the variable is just a single character, then it is of type `Nat`, and whenever the name is not a single character and ends with an `s`, (like `xs` or `foos`) then it is of type `List`. No other elements are in \mathcal{V} . (1 point)
3. Encode the following terms in Haskell and run the type-checking algorithm to test whether they are well-typed w.r.t. Σ and \mathcal{V} from the previous two subtasks.
 - $t_1 := \text{app}(\text{Nil}, xs, ys)$

- $t_2 := \text{app}(\text{app}(ys, \text{Nil}), xs)$
- $t_3 := \text{plus}(\text{Succ}(n), \text{Zero})$

(1 point)

4. Write a type-checking algorithm for formulas (cf. [slide 2/24](#)) in the style of the type-checking algorithm for terms. Here, the datatype for formulas is already provided in the template file. Also the Haskell-type of type-checking algorithm for formulas is fixed.

You can test your algorithm by running it on two example formulas.

(2 points)

5. Formulate the correctness statement of the type-checking algorithm for formulas (soundness + completeness), cf. [slide 2/38](#). This part can be done even if the algorithm has not been implemented. (1 point)

6. Perform the completeness proof as in [slide 2/39](#) – [slide 2/40](#). (If you did not implement the algorithm in Haskell, base your proof on an informal implementation of the algorithm). Here, you can shorten the proof by just considering the cases for negation, predicates and quantifiers, so you are allowed to omit true and conjunctions. (4 points)