universität innsbruck

Program Verification	SS 2024	LVA 703083+703084
Sheet 2		Deadline: March 19, 2024, 3pm

- 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.

1. Show
$$\mathcal{M} \models \forall xs, ys, zs.\, \mathsf{app}(xs, \mathsf{app}(ys, zs)) = \mathsf{app}(\mathsf{app}(xs, ys), zs)$$
by unfolding the definition of $\models$ on slides 2/25 step by step. You can use that $\mathsf{app}^{\mathcal{M}}$ is associative. (4 points)
2. Provide a different model $\mathcal{M}'$ such that all of the following formulas are valid in $\mathcal{M}'$.
$$\forall xs, ys, zs.\, \mathsf{app}(xs, \mathsf{app}(ys, zs)) = \mathsf{app}(\mathsf{app}(xs, ys), zs)$$
$$\forall xs.\, \mathsf{app}(xs, xs) = xs$$
$$\neg\forall xs, ys.\, xs = ys$$
You do not need to prove that $\mathcal{M}'$ has the desired property.
Hint: you only need to change $\mathsf{app}^{\mathcal{M}}$. (2 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 2/35.

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.
   - $t_1 := \mathsf{app}(\mathsf{Nil}, xs, ys)$
   - $t_2 := \mathsf{app}(\mathsf{app}(ys, \mathsf{Nil}), xs)$
   - $t_3 := \mathsf{plus}(\mathsf{Succ}(n), \mathsf{Zero})$

(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 type of this algorithm is fixed. The return value is `Maybe ()`, where `return ()` represents a type-correct formula and `Nothing` an ill-typed formula.

   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 2/36. This part can be done even if the algorithm has not been implemented. (1 point)

6. Perform one of the proofs (soundness or completeness) as in slide 2/36 – slide 2/42. (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)