universität innsbruck

Program Verification	SS 2026	LVA 703083+703084
Sheet 5		Deadline: April 20, 2026, 5pm

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

Exercise 1 *Processing Function Definitions* **9 p.**

Slide 4/13 contains a Haskell function to process data definitions. The task of this exercise is to implement a similar function for checking and processing function definitions w.r.t. slide 3/15.

1. Implement a Haskell function `linear :: Term -> Bool` which decides whether a term is linear or not, cf. slide 3/14. (2 points)
2. Implement a Haskell function

```
checkEquation ::
  SigList ->        -- defined symbols, including f
  SigList ->        -- constructors
  FSym ->           -- f
  FSymInfo ->       -- type of f
  (Term, Term) ->   -- equation (l,r)
  Check ()
```

   that checks whether a single equation satisfies the conditions that are mentioned on slide 3/15. Of course, you should use the provided functions for type-checking, type-inference, etc., as much as possible. (4 points)
3. Implement the Haskell function `processFunctionDefinition` mentioned on Slide 4/15. (3 points)

Once you have completed your implementation, you can test it via `test`, which processes some example program, which should be accepted.
By manually inserting errors into the example program, you can run `test` again, to see whether these errors are detected by your implementation.

Exercise 2 *Implementation of Pattern Disjointness* **5 p.**

1. Extend `processFunctionDefinition` in a way that also pattern disjointness is ensured. In the case that the program is not pattern disjoint, a counterexample must be generated: a term t should be printed that is matched by the lhss of two different rules. (3 points)
2. Modify the counterexample generation: it should now produce *ground* terms that are type-correct. You may assume that unification preserves well-typedness, i.e., it suffices to take term t of the previous task and instantiate it to a ground term $t\sigma$ by a type-correct ground substitution σ. (2 points)

Note that the example program is not pattern disjoint and you can run `test` to see whether the counter-example generation works correctly in your implementation. If you remove the second defining equation of `add`, then the example program should be accepted again.

Exercise 3 *Correctness of Implementation of Unification* **6 p.**

Study the proof given on slides 4/28–31.

1. Perform the proof of case 3, i.e., where the arguments are $(f(ts), x) : u$ and v. (2 points)

2. In case 4 with arguments $(x, t) : u$ and v the algorithm deviates from the abstract algorithm in the following sense: the abstract algorithm only applies (eliminate) if x occurs in U, but such a condition is not tested in the implementation.

   Prove that this difference does not cause a problem, i.e., prove $P((x, t) : u, v, U)$ where $x \neq t$, $x \notin \mathcal{V}ars(t)$ and x does not occur in $set\ u \cup set\ v$, where of course you may assume an IH for the recursive invocation of *unifyMain*. (4 points)