Marius Furter
Institut für Mathematik
Universität Zürich

FS 2025

Logic and Foundation with Haskell

Exercise sheet 7

Recall the definition of the `Set` type from sheet 5:

```
data Set a = Set [a] deriving Show
```

In this sheet, we will use this type to implement functions. If you want, you may alternatively use the sets defined in `Data.Set`, which you can import by adding

```
import Data.Set (...)
```

at the start of your code, where ... is replaced by a comma-separated list of functions.

Exercise 1. Define a datatype `Fun a b` that implements functions. A value $f\colon A \to B$ of this type should consist of

(i) a *domain set* A of type `Set a`,
(ii) a *co-domain set* B of type `Set b`,
(iii) a set of pairs `Set (a,b)` defining the function.

It should follow the pattern

```
data Fun a b = Fun (..) (..) (..) deriving Show
```

We think of a function as a special type of relation satisfying: "For all $a \in A$, there is exactly one $b \in B$ such that $(a, b) \in f$". However, we cannot enforce this condition directly on the level of types in Haskell.

Exercise 2. Write code that checks whether a value of type `Fun a b` satisfies the condition for being a function: "For all $a \in A$, there is exactly one $b \in B$ such that $(a, b) \in f$".

Exercise 3. Implement composition of functions.

Exercise 4. Write code that computes the image $f(A) := \{b \in B \mid \exists a \in A : f(a) = b\}$ of a function.

Exercise 5. Write code that checks whether a function is injective / surjective.

Exercise 6. Write code that interconverts between objects `f :: Fun a b` and standard Haskell functions `f :: a -> b`.