Marius Furter
Institut für Mathematik
Universität Zürich

FS 2025

Logic and Foundation with Haskell

Exercise sheet 8

Recall the definition of natural numbers from the lecture:

```
data Nat = Z | S Nat deriving Show
```

In this sheet, we will use this type to implement integers. For convenience, you can download the script `Nats.hs` from the repo, place it in the same folder as the exercise script, and import it using

```
import Nats
```

One way to define integers is as signed natural numbers:

$$\mathbb{Z} := \{(+, n) \mid n \in \mathbb{N}\} \cup \{(-, n) \mid n \in \mathbb{N}\} / \sim,$$

where we identify $(+, 0) \sim (-, 0)$.

Exercise 1. Define a new type `SignInt` has two data constructors `Plus` and `Minus` that each take a `Nat` as an argument.

Exercise 2. Define an instance of `Eq` for `SignInt` by implementing `(==)`. Your definition should encode the stated equivalence relation.

Exercise 3. Define addition and multiplication for `SignInt`s.

Exercise 4 (Optional)**.** Define instances of `Ord` and `Num` for `SignInt`.

An alternate way to define integers is as difference classes of natural numbers. In other words, an integer is a pair of natural numbers (a, b) which we interpret as $a - b$. Hence $(a, b) = (c, d)$ if and only if $a + d = b + c$. Note that this definition gives us entities that behave like negative numbers but is stated only in terms of addition. Formally, we define

$$\mathbb{Z} := \mathbb{N} \times \mathbb{N} / \sim,$$

where we identify $(a, b) \sim (c, d)$ if and only if $a + d = b + c$.

Exercise 5. Define a new type `DiffInt` has a single data constructor that takes two `Nat`s as arguments.

Exercise 6. Define an instance of `Eq` for `DiffInt` by implementing `(==)`. Your definition should encode the stated equivalence relation.

Exercise 7. Define addition and multiplication for `DiffInt`s.

Exercise 8 (Optional)**.** Define instances of `Ord` and `Num` for `DiffInt`.