Marius Furter
Institut für Mathematik
Universität Zürich

FS 2025

Logic and Foundation with Haskell

Exercise sheet 3

Exercise 1. The following defines a new datatype called `CoolBool` with two values `Nope` and `Yup`.

```
data CoolBool = Nope | Yup deriving (Show, Eq, Ord)
```

Types must start with capital letters. The `deriving` statement allows it to inherit default behavior for the typeclasses `Show`, `Eq` and `Ord`. We will learn more about how to define new types later in the course. After loading a script with the above definition, test the following:

(i) Check that `Yup` and `Nope` are defined and have type `CoolBool`.
(ii) Check how the values are ordered using `<`.

In the remainder of the sheet, we will implement basic Boolean functions for our new datatype.

Exercise 2. Write functions that convert between `Bool` and `CoolBool`. They should have type

```
boolToCool :: Bool -> CoolBool
coolToBool :: CoolBool -> Bool
```

You should **not** be using these for the following exercises.

Exercise 3. Implement logical operators for `CoolBool` with types

```
-- not
coolNot :: CoolBool -> CoolBool
-- && (and)
coolBoth :: CoolBool -> CoolBool -> CoolBool
-- || (or)
coolEither :: CoolBool -> CoolBool -> CoolBool
```

Exercise 4. Implement `coolAnd :: [CoolBool] -> CoolBool` which returns `Yup` iff all the entries in the list are `Yup`. Similarly, implement `coolOr :: [CoolBool] -> CoolBool` which returns `Yup` iff at least one entry is `Yup`.

Exercise 5. Implement `coolElem :: Eq a => a -> [a] -> CoolBool` which returns `Yup` iff the argument is an element of the list. Do not use `elem`.

Exercise 6. Implement

```
coolAll :: (a -> CoolBool) -> [a] -> CoolBool
coolAny :: (a -> CoolBool) -> [a] -> CoolBool
```

The first function takes a predicate `p :: a -> CoolBool` and a list `l`, and returns `Yup` iff all the entries of `l` satisfy `p`. The second function should return `Yup` iff at least one of the entries satisfies the predicate.