CS-205 Declarative Programming — October 10th 2023

Cécilia Pradic, Monika Seisenberger

# Lab 2: Types and typeclasses

Dear all,

- You can do your lab by editing a single file.
- The challenge tasks (from 2.4) do not need to be signed-off today; only attempt them once you are done with the main tasks.
- This week is the last opportunity to sign off lab 1 with full marks.
- Do not hesitate to look up the type of functions in `ghci` using `:type` and try them out on examples. Some functions that we will introduce in today's lab that you may want to play with are `(/=)`, `not`, `(==)` and `(&&)`.

Happy hacking!

**Task 2.1**: **(base types, list and tuples)**

(a) Consider the following expressions

```
('a','b','c')
['a','b','c']
[('1', False), ('0',True)]
```

What are their types? First try to guess them and then use the command `:type` in `ghci` to check your guess. Write down the answer as a comment in your file.

(b) Write definitions with the following names and types in your file. You can pick any value you like, as long as it typechecks.

```
someListOfDouble :: [Double]
some3DVector :: (Double, Double, Double)
some2DMatrix :: ((Double, Double), (Double,Double))
someListOfListOfListOfCharBool :: [[[(Char,Bool)]]]
```

**Task 2.2**: **(polymorphism)**

(a) Write the following equation[1] in your file and save.

```
  fourth (x, y, z, t) = t
```

Check the type of fourth in `ghci` using `:type`. What is the output of `fourth ('a',True,[[False]],2.2)`? Would it run if you additionally put in the type annotation below?

```
fourth :: (a, a, a, a) -> a
```

Write these answers as a comment in your file.

[1]There is a new syntactic construct in the definition of `fourth`, the tuple notation on the left-hand side of the equation, that we have not yet covered during the lectures. That construct allows to decompose tuple arguments before writing the body of the function. This is an instance of *pattern-matching* will be covered in detail in a later lecture.

(b) What is the most general type of the following functions?

```
third x y z t = z
swap (x,y) = (y,x)
```

Write the definition and the corresponding type annotation in your file.

**Task 2.3:** **(typeclasses)** In the lecture we have just seen the notion of *typeclass constraints*, which are a powerful construct which allows to restrict the scope of type variables.

For instance, the function `sum` has the following signature

```
sum :: Num a => [a] -> a
```

meaning that `sum` can be applied to lists of values of type `a` *as long as* `a` *is a type that belongs to the typeclass* `Num`. Whether or not a given type `t` belongs to a typeclass depends on whether the program contains what is called an *instance* of the typeclass for the type in question, which provides basic functionalities related to the constraint.

Defining instances is beyond the scope of this lab; but there are a few typeclasses and instances that are defined in the base library of Haskell that we are going to encounter, including:

- `Num a` which says that values of `a` can be adddded and multiplied
- `Ord a` which says that `a` is a type over which we can use operators <, <=, . . .
- `Eq a` which says that we can use the equality operator ==
- `Show a` which says that we can convert values `a` to a `String`

**Questions:**

(a) For each of the definitions below, which typeclasses constraints should you use?

```
addcubes  (x, y)   = x ^ 3 + y ^ 3
ordered x y z  = x <= y && y <= z
palindrome xs = xs == reverse xs
```

Write the corresponding typing annotations alongside the definitions in your file.

(b) Write a function

```
distinct :: Int -> Int -> Int -> Bool
```

which outputs `True` if all its inputs are disinct, `False` otherwise (hint: use /= and &&).

Once you have a definition, figure out a more general type for `distinct` than the one we just gave you above. Replace the type annotation in your file so that it mentions this more general type instead.

**Task 2.4** **Challenge task:** These tasks are optional; attempt them only if you have done all the others.

(a) Write definitions with the following names and types

```
listOfFunctions :: [Bool -> Bool]
weirdPair :: ([a],[[a]])
```

(b) Formally, a type `T` is *more general* than a type `T'` if, whenever we substitute the type variables in `T` and `T'` with ground types such as `Int` or `Bool` in a consistent way, every value of type `T` is also of type `T'`. We say they are incomparable if neither `T` is more general than `T'` nor `T'` is more general than `T`.

For every pair of types in the list below, determine if they are more general than one another or incomparable.

- `[Int]` and `[a]`
- `(a,b)` and `(a,a)`
- `(Bool,Bool)` and `[Bool]`

Write your answer as a comment in your file, and, for each pair of types in that list, define a value `separatorN` (for `N` = 1,2,3) that belongs to a type less general than one of the two types, but no types less general than the other.

(c) In case that could be useful, let's illustrate the syntax for branching using an if/then/else construct:

```haskell
doubleIfTrue :: Bool -> Int -> Int
doubleIfTrue b x = if b then
                     2 * x
                   else
                     x
```

Define a function

```haskell
median :: Int -> Int -> Int -> Int
```

so that `median x y z` corresponds to the middle value. What is the most general type you can give to median? Put that in your file.