Exercise sheet for lecture 08— Applicatives

In this exercise we look at applicative functors. You can find the signatures of the methods shown below, as well as predefined implementations, in the Git repository at https://gitlab2.informatik.uni-wuerzburg.de/intro-to-fp/tasksheets.

`Applicative` in Cats

For the following exercises, we use the `Applicative` type class from Cats. It basically works like the one from the lecture, but `ap` is defined `abstract`, i.e. an implementation always has to define `ap` instead of choosing between it and `map2`. The methods also aren't defined as extensions, to use them as such you have to import the respective syntax package or simply `cats.implicits.given`.

For the function `ap` Cats provides the operator `<*>`, so instead of `Applicative[F].ap(ff)(fa)` we can also write `ff <*> fa`, where `ff: F[A => B]` and `fa: F[A]`.

1 Tuple composition for Applicative

Implement an `Applicative` instance for tuples of `Applicative`s, as defined in the lecture. The tuple's elements should interact with their respective `Applicative` instance separately and the result be returned as a tuple.

```
def applicativeProduct[F[_], G[_]](
  using F: Applicative[F], G: Applicative[G]
) : Applicative[[a] =>> (F[a],G[a])] = ???
```

2 Applicative Combinators

In the lecture, we defined `Applicative` via the functions `pure` and either `ap` or `map2`. An alternative definition of `Applicative` is possible using the functions `pure`, `map` and `product`. The function `product` takes two values inside an `Applicative`, `F[A]` and `F[B]`, and returns an `F[(A,B)]` with the tuple of both values.

Proof that these definitions are equaly powerful, by:

a) implementing `ap` only using `product` and `map`.

b) implementing `product` only using `pure` and `ap`.

Hint: It may be easier, to first implement `product` with help from the `map` function, which we have shown in the lecture to be implementable with `pure` and `ap`, and then substituting it accordingly.

3 Applicative instance for binary trees

The binary trees presented on the previous exercise sheet are not only `Functor`s but also `Applicative`s.

a) Implement an `Applicative` instance for the binary trees from the previous exercise sheet with `ap` and `pure`.

b) Describe in your own words, what `map2` does on our binary trees.

Hints:

- `Applicative` defines `map` via `ap` and `pure`. To use `map` in your `ap` implementation, you first have to provide an alternative `map` implementation (e.g. the one from the `Functor` instance).
- Think about how `ap` and `map2` work on lists and try to find parallels to `Tree`.
- In Git you can find a main method, which creates two example trees and combines them with `map2`. You can also find an example for each of the applicative laws known from the lecture, to test your implementation with.