Übungsblatt zu Vorlesung 08— Applicatives

In dieser Übung beschäftigen wir uns mit applikativen Funktoren. Die Signaturen der in den Aufgaben geforderten Methoden sowie die vorgegebenen Implementierungen finden Sie online unter `https://gitlab2.informatik.uni-wuerzburg.de/intro-to-fp/tasksheets` als Git-Repository.

`Applicative` in Cats

Für die folgenden Aufgaben nutzen wir die `Applicative` Typklasse aus Cats. Diese funktioniert im Prinzip genauso, wie die in der Vorlesung vorgestellte, jedoch ist `ap` als `abstract` definiert, d.h. zum Implementieren muss immer eine Definition für `ap` angegeben werden, anstatt zwischen diesem und `map2` wählen zu können. Außerdem sind die Methoden nicht als Extension definiert, um sie so zu verwenden, muss das entsprechdende Syntaxpackage oder einfach `cats.implicits.given` importiert werden.

Für die Funktion `ap` existiert mit `<*>` weiter ein eigener Operator, sodass wir anstatt `Applicative[F].ap(ff)(fa)` auch `ff <*> fa` schreiben können, wobei gilt `ff: F[A => B]` und `fa: F[A]`.

1 Tupel-Komposition für Applicative

Implementieren Sie eine `Applicative`-Instanz für Tupel von `Applicative`s, wie in der Vorlesung definiert. Die Elemente des Tupels sollen getrennt mit ihreren jeweiligen `Applicative`-Instanzen interagieren und das Ergebnis dann wieder als Tupel ausgegeben werden.

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

2 Applicative Combinators

In der Vorlesung haben wir `Applicative` über die Funktionen `pure` und `ap` bzw. `map2` definiert. Alternativ hätten wir `Applicative` über die Funktionen `pure`, `map` und `product` definieren können. Die Funktion `product` bekommt zwei Werte im `Applicative`, `F[A]` und `F[B]`, und gibt ein `F[(A,B)]` mit dem Tupel der beiden Werte zurück.

Zeigen Sie, dass diese Definitionen gleich mächtig sind, indem Sie

a) `ap` nur mit Hilfe von `product` und `map` implementieren.

b) `product` nur mit Hilfe von `pure` und `ap` implementieren.

Hinweis: Es ist möglicherweise einfacher, zur Implementierung von `product` zuerst einmal die Funktion `map` zur Hilfe zu nehmen, von der wir schon in der Vorlesung gezeigt haben, dass sie sich nur mit `pure` und `ap` implementieren lässt, und das dann entsprechend umzuformen.

3 Applicative-Instanz für Binärbäume

Die auf dem letzten Übungsblatt vorgestellten Binärbäume sind nicht nur `Functor`s sondern auch `Applicative`s.

a) Implementieren Sie für die Binärbäume vom letzten Übungsblatt eine `Applicative`-Instanz mit `ap` und `pure`.

b) Beschreiben Sie in eigenen Worten, was die `map2`-Funktion auf unseren Binärbäumen tut.

Hinweise:

- `Applicative` definiert `map` über `ap` und `pure`. Um in Ihrer `ap` Implementierung `map` verwenden zu können, müssen Sie zuerst eine alternative Implementierung für `map` (z.B. die aus der `Functor`-Instanz) bereitstellen.
- Überlegen Sie sich, wie `ap` und `map2` auf Listen arbeiten und versuchen Sie Parallelen zu `Tree` zu finden.
- Im Git finden Sie eine Main-Methode, die zwei Beispielbäume baut und diese per `map2` verknüpft. Außerdem können Sie hier die Applicative-Laws aus der Vorlesung anhand je eines Beispiels für Ihre Implementierung überprüfen.