Übungsblatt zu Vorlesung 10— Traversable Functors

1 map via traverse

Implementieren Sie `map` via `traverse` für beliebige Traversable Functors. Dies belegt, dass `Traverse` eine Erweiterung von `Functor` ist und die Funktion `traverse` eine Verallgemeinerung von `map`.

```
def mapViaTraverse[F[_],A,B](fa: F[A])(f: A => B)(using Traverse[F]): F[B] = ???
```

Hinweise:

- Benutzen Sie die `Traverse` Type Class aus Cats.
- Beachten Sie, dass `map` keinen `Applicative` als `using`-Parameter übergeben bekommt. Um `traverse` innerhalb von `map` zu benutzen, wird jedoch ein `Applicative` benötigt. Wählen Sie selbstständig eine passende `Applicative`-Instanz aus. Sie dürfen sowohl die in der Vorlesung und den vorangegangenen Übungsblättern vorgestellten Instanzen als auch solche aus Cats benutzen.
- Beachten Sie, dass jeder Monad auch ein Applicative Functor ist.

2 Traverse-Instanz für Binärbäume

Die Binärbäume von den vorherigen Übungsblättern sind, welch Überraschung, Traversable Functors.

a) Implementieren Sie eine `Traverse`-Instanz für Binärbäume. Cats fordert für `Traverse` die Implementierung von `foldLeft`, `foldRight` und `traverse`. Die ersten beiden können Sie aus der `Foldable`-Instanz übernehmen (Achten Sie auf den korrekten Aufruf, sonst bekommen Sie eine Endlosrekursion!). Für `traverse` genügt eine *nicht* tail-rekursive Lösung.

b) Überlegen sie, wie in der Vorlesung für diverse andere Typen gesehen, wie sich `sequence` auf Bäumen verhält.

3 Akkumulieren mit State

a) Mit der in der Vorlesung eingeführten Funktion `mapAccum` kann nun endgültig eine Funktion `reverse` geschrieben werden, die jeden Traversable Functor umdrehen kann. Implementieren Sie diese Funktion für beliebige `Traverse`.

Hinweise: Für diese Funktion wird ein Stack benötigt. Praktischerweise ist eine `List` ein Stack und wie man jeden `Traversable` zu einer Liste umwandeln kann wurde in der Vorlesung mit `toList` gezeigt.

```
def reverse[F[_],A](fa: F[A])(using Traverse[F]): F[A] = ???
```

Die Funktion sollte das folgende Gesetz erfüllen:

```
reverse(x).toList ::: reverse(y).toList == reverse(y.toList ::: x.toList)
```

b) Benutzen Sie `mapAccum`, um eine allgemeine Version von `foldLeft` für den `Traverse`-Trait zu implementieren. Diese Implementierung ist sehr ähnlich zu `toList`. Statt eine Liste als Akkumulator zu verwenden wird hier allerdings in einem `B` akkumuliert mithilfe der Funktion `f`.

```
def foldLeftViaMapAccum[F[_],A,B](fa: F[A], z: B)(f: (B, A) => B)(using
↪  Traverse[F]): B = ???
```