Exercise sheet for lecture 10— Traversable Functors

1 map via traverse

Implement `map` via `traverse` for arbitrary traversable functors. This proves, that `Traverse` is an extension of `Functor` and that `traverse` is a generalization of `map`.

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

Hints

- Use the `Traverse` type class from cats.
- Note, that `map` doesn't take an `Applicative` as a `using` parameter. But to use `traverse` inside of `map` an `Applicative` is required. Choose a fitting `Applicative` instance yourself. You can use any of the ones shown in the lecture and exercise sheets as well as those in Cats.
- Note that every monad is also an applicative functor.

2 Traverse instance for binary trees

The binary trees from the earlier exercise sheets are, surprise surprise, traversable functors.

a) Implement a `Traverse` instance for binary trees. In Cats, a `Traverse` instance needs implementations of `foldLeft`, `foldRight` and `traverse`. You can copy the first two from the `Foldable` instance (watch out for calling them correctly, or you'll get an endless recursion!). For `traverse`, a *non* tail-recursive solution is sufficient.

b) Think about, how `sequence` behaves on trees, like seen for other types in the lecture.

3 Accumulating with State

a) Using the function `mapAccum` introduced in the lecture, we can finally write a `reverse` function, which can reverse every traversable functor. Implementieren this function for arbitrary `Traverse`.

Hints: For this function, a stack is required. Luckily, a `List` is a stack and we've seen in the lecture, how to turn any `Traversable` into a list with `toList`.

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

The function should fulfill the following law:

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

b) Use `mapAccum` to implement a general version of `foldLeft` for the `Traverse` trait. This implementation is pretty similar to `toList`. But instead of using a list as accumulator, a `B` is used for accumulation with help from the function `f`.

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