Aufgabe 3 Strukturelle Induktion und Folds (9 Punkte)

1. Wir erinnern an den Datentyp der Listen und einige hierauf rekursiv definierte Standardfunktionen:

data List a = Nil | $Cons$ a (List a)

1) $snoc\ Nil\ y = Cons\ y\ Nil$
2) $snoc\ (Cons\ x\ xs)\ y = Cons\ x\ (snoc\ xs\ y)$
3) $Nil \oplus ys = ys$
4) $(Cons\ x\ xs) \oplus ys = Cons\ x\ (xs \oplus ys)$

Beweisen Sie mittels struktureller Induktion, dass

$$\forall e,\ xs,\ ys.\ xs \oplus (Cons\ e\ ys) = (snoc\ xs\ e) \oplus ys.$$

Geben Sie im Induktionsschritt die Induktionsannahme explizit an und erläutern Sie alle Schritte des Beweises.

Zu zeigen: $\forall xs\ \forall e\ \forall ys :\ \underbrace{\underbrace{xs \oplus (Cons\ e\ ys)}_{LS} = \underbrace{(snoc\ xs\ e) \oplus ys}_{RS}}_{A(xs)}$

Induktionsanfang $xs = Nil$: Sei e und ys beliebig.

$$LS = Nil \oplus (Cons\ e\ ys) \overset{3)}{=} Cons\ e\ ys$$
$$RS = (snoc\ Nil\ e) \oplus ys \overset{1)}{=} (Cons\ e\ Nil) \oplus ys \overset{4)}{=} Cons\ e\ (Nil \oplus ys) \overset{3)}{=} Cons\ e\ ys$$
$\Rightarrow LS = RS$ ✓

Induktionsvorraussetung: Sei xs eine List und es gelte $A(xs)$.

Induktionsschritt: Zu zeigen: $\forall x$ gilt: $A(Cons\ x\ xs)$

Sei e vom Typ a und ys vom Typ List a beliebig.

Zu zeigen: $\underbrace{(Cons\ x\ xs) \oplus (Cons\ e\ ys)}_{LS} \overset{!}{=} \underbrace{(snoc\ (Cons\ x\ xs)\ e) \oplus ys}_{RS}$

$$LS \overset{4}{=} Cons\ x\ (xs \oplus (Cons\ e\ ys)) =$$
$$\overset{IV}{=} Cons\ x\ ((snoc\ xs\ e) \oplus ys) =$$
$$\overset{4)}{=} (Cons\ x\ (snoc\ xs\ e)) \oplus ys$$
$$= (snoc\ ((Cons\ x\ xs)\ e) \oplus ys$$

data List a = Nil | $Cons$ a (List a)

1 $snoc\ Nil\ y = Cons\ y\ Nil$
2 $snoc\ (Cons\ x\ xs)\ y = Cons\ x\ (snoc\ xs\ y)$
3 $Nil \oplus ys = ys$
4 $(Cons\ x\ xs) \oplus ys = Cons\ x\ (xs \oplus ys)$

Beweisen Sie mittels struktureller Induktion, dass

$\forall e,\ xs,\ ys.\ xs \oplus (Cons\ e\ ys) = (snoc\ xs\ e) \oplus ys.$

Geben Sie im Induktionsschritt die Induktionsannahme explizit an und erläutern Sie alle Schritte des Beweises.

miro

2. Gegeben sei der folgende Datentyp, den man sich als nichtleere Liste von Paaren vorstellen kann:

data Twins a = *End a a* | *More a* (Twins a) a

Geben Sie die *fold*-Funktion für Twins, *foldtw*, inklusive ihres Typs an. Nutzen Sie diese, um eine Funktion

identicalTwins : Twins a → List a

zu definieren, die Elemente nur dann in die Zielliste übernimmt, wenn die beiden Werte übereinstimmen. Beispielsweise soll gelten:

identicalTwins (*More* 1 (*More* 2 (*More* 1 (*End* 2 0) 1) 2) 3) = [2,1]

Hinweis: Sie dürfen die üblichen Konstrukte wie λ-Abstraktion, *if* ... *then* ... *else* verwenden und annehmen, dass Elemente mittels == auf Gleichheit überprüft werden können.

Startakkumulator

```
fold :: (a -> a -> b -> b) -> b -> Twins a -> b
```

aktueller Akkumulator

neuer Akkumulator

```
identicalTwins twins = fold (\x y akk -> if x==y then Cons x akk else akk) Nil twins
```

Liste

data List a = *Nil* | *Cons a* (List a)

kann man weglassen

```
fold :: (a -> a -> b -> b) -> b -> Twins a -> b
fold f akk (End x y) = f x y akk
fold f akk (More x twins y) = fold f (f x y akk) twins
```