Friedrich-Alexander-Universität
Technische Fakultät

FAU

Theorie der Programmierung

Übung 09 - Strukturelle Induktion

Leon Vatthauer

26. Juni 2023

Aufgabe 1

Beweise mittels struktureller Induktion

Wir betrachten erneut den Datentyp der Listen sowie die folgenden Funktionen:

```
data List a where
    Nil : () → List a
    Cons : a → List a → List a
```

```
length : List a → Nat
length Nil = 0
length (Cons x xs) = 1 + length xs
```

```
reverse : List a → List a
reverse Nil = Nil
reverse (Cons x xs) = snoc (reverse xs) x
```

```
snoc : List a → List a
snoc Nil y = Cons y Nil
snoc (Cons x xs) y = Cons x (snoc xs y)
```

Wir zeigen einige Eigenschaften dieser Funktionen. Beweisen Sie diese jeweils durch Induktion über der Struktur der Argumentliste. Rechtfertigen Sie hierbei Ihre Schritte und geben Sie jeweils ihre Induktionshypothese an.
Hinweis: Wir erinnern daran, dass $s = t$ als $s \leftrightarrow_{\beta\delta} t$ zu lesen ist. Außerdem können Sie jederzeit zuvor bewiesene Eigenschaften verwenden.

Aufgabe 1

Beweise mittels struktureller Induktion

Wir betrachten erneut den Datentyp der Listen sowie die folgenden Funktionen:

```
data List a where
    Nil : () → List a
    Cons : a → List a → List a
```

```
length : List a → Nat
length Nil = 0
length (Cons x xs) = 1 + length xs
```

```
reverse : List a → List a
reverse Nil = Nil
reverse (Cons x xs) = snoc (reverse xs) x
```

```
snoc : List a → List a
snoc Nil y = Cons y Nil
snoc (Cons x xs) y = Cons x (snoc xs y)
```

Beweisen Sie mittels Induktion:

1. $\forall$`x xs. length (snoc xs x) = 1 + length xs`

Aufgabe 1

Beweise mittels struktureller Induktion

Wir betrachten erneut den Datentyp der Listen sowie die folgenden Funktionen:

```
data List a where
    Nil : () → List a
    Cons : a → List a → List a
```

```
length : List a → Nat
length Nil = 0
length (Cons x xs) = 1 + length xs
```

```
reverse : List a → List a
reverse Nil = Nil
reverse (Cons x xs) = snoc (reverse xs) x
```

```
snoc : List a → List a
snoc Nil y = Cons y Nil
snoc (Cons x xs) y = Cons x (snoc xs y)
```

Beweisen Sie mittels Induktion:

1. $\forall$`x xs. length (snoc xs x) = 1 + length xs`
2. $\forall$`xs. length (reverse xs) = length xs`

Aufgabe 2

Eine binäre Funktion: Listenkonkatenation

Wir betrachten die folgende Definition einer Funktion zur Listenkonkatenation:

```
(⊕) : List a → List a → List a
Nil ⊕ ys = ys
(Cons x xs) ⊕ ys = Cons x (xs ⊕ ys)
```

Wir möchten die folgende Eigenschaft mittels struktureller Induktion beweisen:

```
∀xs ys. length (xs ⊕ ys) = length xs + length ys
```

1. Über welche Liste(n) sollten wir induzieren, über das erste Argument von (_ ⊕ _), über das zweite, oder über beide? Warum?

Aufgabe 2

Eine binäre Funktion: Listenkonkatenation

Wir betrachten die folgende Definition einer Funktion zur Listenkonkatenation:

```
(⊕) : List a → List a → List a
Nil ⊕ ys = ys
(Cons x xs) ⊕ ys = Cons x (xs ⊕ ys)
```

Wir möchten die folgende Eigenschaft mittels struktureller Induktion beweisen:

```
∀xs ys. length (xs ⊕ ys) = length xs + length ys
```

1. Über welche Liste(n) sollten wir induzieren, über das erste Argument von (_ ⊕ _), über das zweite, oder über beide? Warum?
2. Beweisen Sie die oben angegebene Eigenschaft; begründen Sie Ihre Schritte und geben Sie explizit an, an welcher Stelle die Induktionshypothese verwendet wird.

Aufgabe 3.1

Induktion über Bäume

Wir erinnern uns an den induktiven Datentyp der binären Bäume von Blatt 8 und die Funktion `size`, die die Knoten eines Baums zählt:

```
data Tree a where
    Leaf : a → Tree a
    Inner : a → Tree a → Tree a → Tree a
```

```
size : Tree a → Nat
size (Leaf x) = 1
size (Inner x l r) = 1 + size l + size r
```

Definieren Sie induktiv eine Funktion `inorder : Tree a → List a`, die die Elemente eines Baumes gemäß einer In-Order-Traversierung von links nach rechts ausgibt. Zeigen Sie dann per struktureller Induktion über Bäume, dass

$$\forall t.\ \texttt{length (inorder t) = size t}$$

Aufgabe 3.2

Induktion über Bäume

Wir betrachten im folgenden einen parametrischen induktiven Datentyp für Bäume, deren Blätter Elemente vom Typ *a* enthalten, und deren innere Knoten jeweils bis zu drei Kinder haben, selbst aber keine Werte enthalten:

```
data VarTree a where
    VLeaf : a → VarTree a
    Node1 : VarTree a → VarTree a
    Node2 : VarTree → VarTree a → VarTree a
    Node3 : VarTree → VarTree a → VarTree a → VarTree a
```

Hierbe ist also `Node1` ein Knoten mit einem Nachfolger `m`, `Node2 l r` ein Knoten mit einem linken Nachfolger `l` und rechtem Nachfolger `r`, und `Node3 l m r` ein Knoten mit linkem Nachfolger `l`, mittlerem Nachfolger `m` und rechtem Nachfolger `r`.

Definieren Sie induktiv eine Funktion `mirror : VarTree a → VarTree a`, die einen solchen Baum spiegelt und zeigen Sie per struktureller Induktion, dass `mirror` eine **Involution** darstellt, das heißt:

$$\forall t.\ \texttt{mirror (mirror t) = t}$$