Prof. Dr. Manfred Schmidt-Schauß
Künstliche Intelligenz/Softwaretechnologie

Fachbereich Informatik (20)
Johann Wolfgang Goethe-Universität Frankfurt am Main

Funktionale Programmierung I

Sommersemester 2000

Übungsblatt Nr. 11

Abgabe: Dienstag 20. Juni 2000 vor der Vorlesung

Aufgabe 1 (10 Punkte)

Das erste Argument von `foldl` und `foldr` ist ja bekanntlich eine Funktion vom Typ (a `->` b `->` a) bzw. (a `->` b `->` b), die jeweils ein Listenelement und ein Element des Ergebnistyps verarbeitet.

Eine Verallgemeinerung ist, auch den Listenrest als zusätzliches Argument zu übergeben. Definieren Sie in diesem Sinne eine Funktion `fold`

$$\texttt{fold} :: ([a] \texttt{ -> } a \texttt{ -> } b \texttt{ -> } b) \texttt{ -> } b \texttt{ -> } [a] \texttt{ -> } b$$

und zeigen Sie, daß `foldr` als Spezialfall obiger Funktion definierbar ist, d.h.

$$\texttt{foldr}\ f\ z\ xs\ =\ \texttt{fold}\ \ldots$$

Aufgabe 2 (15 Punkte)

Geben Sie das Gesetz für die `foldl-concat` Verschmelzung an:

$$\texttt{foldl}\ f\ a\ \texttt{.}\ \texttt{concat}\ =\ \ldots$$

und beweisen Sie es.

Aufgabe 3 (15 Punkte)

Implementieren Sie wahlweise eine der Funktionen `map` oder `filter` mit Hilfe von `fold` aus Aufgabe 1.

Versuchen Sie, außer [] und (:) keine weiteren Fuktionen zu benutzen, und beweisen Sie die kontextuelle Äquivalenz.

Aufgabe 4 (10 Punkte)

Wie wir in Aufgabe 3 an `map` und `filter` gesehen haben, können die meisten listenverarbeitenden Funktionen durch die member-Funktionen der Klasse

```
class ListLike l where
  empty :: l a
  cons  :: a -> l a -> l a
  fold  :: (l b -> b -> a -> a) -> a -> l b -> a
```

realisiert werden. Definieren Sie für den folgenden Baumdatentypen

```
data T a = Leaf
         | Node a (T a) (T a)
```

eine *Sicht* als listenähnlichem Datentypen, indem Sie ihn zu einer Instanz der Klasse `ListLike` machen. Dadurch können `map` und `filter` aus Aufgabe 3 direkt umgesetzt werden und stehen sofort auf binären Bäumen zur Verfügung.