9. Aufgabenblatt zu Funktionale Programmierung vom 07.01.2004, fällig: 14.01.2004

Themen: *Funktionale, Listen und Mengen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe9.hs` auf oberstem Niveau in Ihrem Homeverzeichnis ablegen. Kommentieren Sie Ihre Programme wieder aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Ihre Lösungen sind besonders gut, wenn sich die Rümpfe der zu implementierenden Funktionen wo möglich auf andere Funktionale, Funktionen und Funktionskomposition abstützen, nicht aber auf Fallunterscheidungen, elementare Operatoren wie $+$, $*$, etc.

- In ähnlicher Weise wie in der ersten Aufgabe von Aufgabenblatt 8 soll in dieser Aufgabe ein curryfiziertes polymorphes Haskell-Funktional `listFilt` zur Filterung von Listen von Listen geschrieben werden. Neben einer Liste von Listen als zweites Argument erwartet `listFilt` als erstes Argument eine Wahrheitswertfunktion, die festlegt, welche Elemente der Listenelemente des zweiten Arguments verworfen bzw. ins Ergebnis übernommen werden. Beispielsweise soll der Aufruf `listFilt (>3) [[2],[2,4,3,5],[],[34,5]]` das Resultat `[[],[4,5],[],[34,5]]` liefern.

  Geben Sie eine entsprechende Implementierung von `listFilt` an.

  Um dieses Funktional zu testen, sollen zusätzlich (Filter-) Funktionen `fOne`, `fTwo` und `fThree` implementiert werden, mit denen Elementlisten über Zeichenreihen und ganzen Zahlen gefiltert werden können. Dabei wird `fOne` alle Zeichenreihen verwerfen, die aus einer ungeraden Zeichenanzahl bestehen oder mit einem Großbuchstaben beginnen, `fTwo` alle Zahlen verwerfen, die keine Fibonacci-Zahl sind (dabei gelte: $f_0 = 0$, $f_1 = 1$ und $f_n = f_{n-1} + f_{n-2}$ für $n \geq 2$) und `fThree` alle Zeichenreihen verwerfen, die keine Palindrome sind (zur Definition von Palindromen siehe Aufgabenblatt 2). Die folgenden Beispiele illustrieren die gewünschten Funktionalitäten.

  1. `listFilt fOne [["abc","de"],["XYz"],[],["rTcE","ZBN"]]`
     `= [["de"],[],[],["rTcE"]]`

2. `listFilt fTwo [[1,2,55,17,3],[24,35],[],[56,13,15,55]] = [[1,2,55,3],[],[],[13,55]]`
3. `listFilt fThree [["otto","Otto","P"],[],["lotto"]] = [["otto","P"],[],[]]`

- Mengen (mit Elementen über einem gemeinsamen Grundtyp) lassen sich in Haskell als Listen repräsentieren (vgl. Aufgabenblatt 5). Allerdings ist nicht jede solche Liste eine Menge, da Listen anders als Mengen Duplikate enthalten können. Auch liefert der standardmäße Vergleich von Listen auf Gleichheit keinen gültigen Test für Gleichheit von Mengen, da anders als bei Listen bei Mengen die Reihenfolge der Elemente keine Rolle spielt. In dieser Aufgabe sollen daher Haskell-Rechenvorschriften zur Behandlung und Manipulation von Mengen darstellenden Listen geschrieben werden. Dabei soll davon ausgegangen werden, dass nur solche Typen als Grundtypen für Mengen betrachtet werden, deren Elemente mithilfe des Standardoperators (==) auf Gleichheit überprüft werden können.

  1. Schreiben Sie eine Haskell-Rechenvorschrift `extractSet`, die aus einer als Argument übergebenen Liste durch Entfernen etwaiger Duplikate eine Menge erzeugt und diese als Resultat zurückgibt. Für das Resultat von `extractSet` soll gelten, dass es "elementmaximal" ist, d.h. jede andere Menge, die durch Streichen von Elementen in der Argumentliste herleitbar wäre, hat höchstens so viele Elemente wie die von `extractSet` gelieferte Menge, wäre also eine Teilmenge davon.
  2. Schreiben Sie curryfizierte Haskell-Rechenvorschriften
     (a) `union`, die als Ergebnis die Vereinigungsmenge zweier Argumentmengen berechnet,
     (b) `inters`, die als Ergebnis den Durchschnitt zweier Argumentmengen berechnet,
     (c) `diff`, die als Ergebnis die Differenz zweier Mengen berechnet.
  3. Schreiben Sie curryfizierte Haskell-Rechenvorschriften, die folgende Prädikate auf Listen bzw. Mengen realisieren:
     (d) `isSet` überprüft, ob eine als Argument vorgelegte Liste eine Menge ist, und gibt einen entsprechenden Wahrheitswert aus.
     (e) `isEqual` überprüft, ob zwei als Argument vorgelegte Listen (genauer Mengen) gleich sind, und gibt einen entsprechenden Wahrheitswert aus.

(f) `isElem` überprüft, ob ein als erstes Argument übergebenes Element in der als zweites Argument übergebenen Liste (genauer Menge) als Element vorkommt, und gibt einen entsprechenden Wahrheitswert aus.

`isElem` und `isEqual` werden nur auf Listen angewendet, die eine Menge darstellen. Für `isSet` gilt diese Einschränkung (aus naheliegenden Gründen) nicht.

Achten Sie darauf, dass die Funktionen `extractSet`, `union`, `inters` und `diff` stets Mengen als Ergebnis zurückliefern.