5. Aufgabenblatt zu Funktionale Programmierung vom 26.11.2003, fällig: 03.12.2003

Themen: *Polymorphe Funktionen auf Mengen, Mengen als Listen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der im folgenden angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe5.hs` in Ihrem Homeverzeichnis ablegen. Kommentieren Sie Ihre Programme wieder aussagekräftig und benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

- Schreiben Sie eine polymorphe Haskell-Rechenvorschrift `addComp`, die ein Element `a` und eine Liste `L` von Listen von Elementen des Typs von `a` als Argumente nimmt und `a` am Anfang jeder Komponentenliste von `L` einfügt. Beispiel: Angewandt auf `42` und die Liste `[[1,2,3],[],[4,5],[6]]` soll die Funktion `addComp` das Resultat `[[42,1,2,3],[42],[42,4,5],[42,6]]` liefern. Achten Sie darauf, dass der Typ von `addComp` so allgemein wie möglich ist (Hinweis: Typvariablen).

- Mengen können in Haskell durch Listen repräsentiert werden. Die Elemente einer Menge können dabei in beliebiger Reihenfolge in der Liste auftauchen, aber höchstens einmal. In dieser Aufgabe wollen wir homogene Mengen über den numerischen Datentypen von Haskell implementieren, d.h. eine Menge enthält entweder nur Elemente vom Typ `Int` oder nur vom Typ `Float`, etc. (Hinweis: `Num a =>...`). Schreiben Sie polymorphe Haskell-Rechenvorschriften, die die folgenden Operationen auf homogenen Mengen über numerischen Datentypen realisieren. Achten Sie darauf, dass Ihre Funktionen die Mengeneigenschaft ihrer Mengenargumente erhalten.

  - `isElem x S` ...liefert den Wahrheitswert "wahr", wenn `x` Element der Menge `S` ist, sonst "falsch".
  - `insert x S` ...fügt `x` zu `S` hinzu.
  - `delete x S` ...entfernt `x` aus `S`.

- Wie in der vorigen Aufgabe gesehen, lassen sich Mengen durch Listen darstellen. Mengen von Mengen lassen sich entsprechend durch Listen von Listen darstellen. Schreiben Sie eine polymorphe Haskell-Rechenvorschrift `isSetElem`, die eine im Sinn der vorigen Teilaufgabe homogene Menge über den numerischen Datentypen als erstes

Argument übernimmt und überprüft, ob diese als Element in der als zweites Argument übergebenen Menge von Mengen als Element vorkommt. In diesem Fall liefert die Funktion `isSetElem` den Wahrheitswert “wahr” zurück, sonst “falsch”. Sie können davon ausgehen, dass die Elemente der Mengen der Menge von Mengen vom selben Typ sind, wie die Elemente der Menge des ersten Arguments.

- In dieser Aufgabe betrachten wir wieder die schon eingeführten homogenen Mengen über den numerischen Datentypen von Haskell. Schreiben Sie eine polymorphe Haskell-Rechenvorschrift `powerSet`, die angewendet auf eine durch eine Liste gegebene Menge deren Potenzmenge berechnet. Die Potenzmenge wird dabei durch eine Liste von Listen repräsentiert. An die Reihenfolge dieser Listen in der Gesamtliste wie an die Reihenfolge der Elemente innerhalb einer Liste wird keine Anforderung gestellt. Stützen Sie die Funktion `powerSet` auf die Funktion `addComp` aus der ersten Teilaufgabe ab.