6. Aufgabenblatt zu Funktionale Programmierung vom Mi, 22.11.2017.
Fällig: Mi, 29.11.2017 (15:00 Uhr)

Themen: *Funktionen über algebraischen Datentypen*

Zur Frist der Zweitabgabe: Siehe „Hinweise zu Organisation und Ablauf der Übung“ auf der Homepage der LVA.

Aufgabe

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe6.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also wieder ein “gewöhnliches” Haskell-Skript schreiben. Versehen Sie wieder wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

1. Wir erweitern die Operationen auf der Menge aussagenlogischer Ausdrücke von Aufgabenblatt 4 um eine weitere Operation, und zwar die logische Implikation, sowie um existentiell und allquantifizierte Ausdrücke. Wir erweitern die Modellierung von Aufgabenblatt 4 dazu wie folgt:

```
type Wahrheitswert = Bool
data VName         = N1 | N2 | N3 | N4 | N5 deriving (Eq,Ord,Enum,Show)
newtype Variable   = Var VName deriving (Eq,Ord,Show)

instance Enum Variable where
 fromEnum (Var name) = fromEnum name
 toEnum n = Var (toEnum n :: VName)

data Ausdruck = K Wahrheitswert                -- Logische Konstante
              | V Variable                     -- Logische Variable
              | Nicht Ausdruck                 -- Logische Negation
              | Und Ausdruck Ausdruck          -- Logische Konjunktion
              | Oder Ausdruck Ausdruck         -- Logische Disjunktion
              | Impl Ausdruck Ausdruck         -- Logische Implikation
              | Esgibt Variable Ausdruck       -- Existentiell quantifizierter Ausdruck
              | Fueralle Variable Ausdruck     -- Allquantifizierter Ausdruck
              deriving (Eq,Show)

type Belegung = Variable -> Wahrheitswert      -- Total definierte Abbildung
```

   Dabei gilt: Ein existentiell quantifizierter Ausdruck ist genau dann *wahr*, wenn es eine Belegung der gebundenen Variable gibt, so dass der Ausdruck den Wahrheitswert *wahr* hat. Ein allquantifizierter Ausdruck ist genau dann *wahr*, wenn der Ausdruck für alle Belegungen der gebundenen Variable den Wahrheitswert *wahr* hat.

   - Schreiben Sie eine Haskell-Rechenvorschrift

     ```
     evaluiere :: Ausdruck -> Belegung -> Wahrheitswert
     ```

     die angewendet auf einen logischen Ausdruck a und eine Belegung b, den Wahrheitswert von a unter b liefert. Sie dürfen davon ausgehen, dass Belegungen stets total definiert sind.

   - Schreiben Sie eine Haskell-Rechenvorschrift

     ```
     ist_tautologie :: Ausdruck -> Ausdruck -> Wahrheitswert
     ```

     die überprüft, ob die beiden Argumentausdrücke logisch äquivalent sind und entsprechend `True` oder `False` als Resultat liefert.

   - Schreiben Sie eine Haskell-Rechenvorschrift

```
schreibe :: Ausdruck -> String
```

die einen Ausdruck in Form einer Zeichenreihe darstellt. Dabei soll gelten: Ist der Ausdruck von der Form

- `K b`, `b`∈{`True,False`}, so soll in Abhängigkeit des Werts von `b` die Zeichenreihe “wahr” bzw. “falsch” ausgegeben werden,
- `V (Var n)`, `n`∈{`N1,N2,N3,N4,N5`}, so soll in Abhängigkeit des Werts von `n` die Zeichenreihe “N1”, “N2”, “N3”, “N4” oder “N5” ausgegeben werden.
- `Nicht a`, so soll die Zeichenreihe “(”++“neg”++“ ”++`schreibe a`++“)” ausgegeben werden,
- `Und a1 a2`, so soll die Zeichenreihe “(”++`schreibe a1`++“ ”++“und”++“ ”++`schreibe a2`++“)” ausgegeben werden,
- `Oder a1 a2`, so soll die Zeichenreihe “(”++`schreibe a1`++“ ”++“oder”++“ ”++`schreibe a2`++“)” ausgegeben werden,
- `Impl a1 a2`, so soll die Zeichenreihe “(”++`schreibe a1`++“ ”++“=>”++“ ”++`schreibe a2`++“)” ausgegeben werden,
- `Esgibt v a`, so soll die Zeichenreihe “(”++“EG”++“ ”++`schreibe v`++“.”++“ ”++`schreibe a`“)” ausgegeben werden,
- `Fueralle v a`, so soll die Zeichenreihe “(”++“FA”++“ ”++`schreibe v`++“.”++“ ”++`schreibe a`“)” ausgegeben werden,

wobei “ ” die aus genau einem Leerzeichen bestehende Zeichenreihe ist.

2. Im Zuge einer Effizienzsteigerung der Meldedatenverwaltungssoftware soll die Speicherung von Personendaten von einer Listen- auf eine Suchbaumspeicherung umgestellt werden.

   Dazu werden die Typen der Meldedatenverwaltungssoftware um einen algebraischen Summentyp `Registerbaum` erweitert; die Typsynonyme `Von_Anschrift`, `Nach_Anschrift` werden nicht mehr benötigt:

```
type Nat1          = Int
type Wahrheitswert = Bool
type Name          = String
type Alter         = Nat1
data Geschlecht    = M | W | X deriving Show
type Gemeinde      = String
type Strasse       = String
type Hausnr        = Nat1
data Person        = P Name Alter Geschlecht Wohnsitze deriving Show
data Anschrift     = A Gemeinde Strasse Hausnr deriving Show
type Wohnsitze     = [Anschrift]
type Melderegister = [Person]
data Registerbaum  = Leer
                   | Verzweigung Registerbaum Person Registerbaum deriving (Eq,Show)
```

   - Schreiben Sie eine Haskell-Rechenvorschrift

```
migration :: Melderegister -> Registerbaum
```

     die die Migration des Melderegisters aus der bisherigen Listen- in die neue Suchbaumdarstellung überführt. Der Namenseintrag eines Personendatums wird dabei als Schlüssel verwendet. Dabei sollen Schlüsselwerte im linken Teilbaum eines Registerbaums lexikographisch stets kleiner als in der Wurzel sein, im rechten Teilbaum lexikographisch stets größer als in der Wurzel.

     Für die Migration der Daten arbeitet die Funktion `migration` die Einträge des Melderegisters von links nach rechts ab und trägt die Personendaten sukzessive in einen ursprünglich leeren Registerbaum ein.

     Die Migration der Daten soll zugleich dazu verwendet werden, fehlerhafte Daten zu löschen.

     Als fehlerhaft gelten Melderegistereinträge, für die bereits ein Personeneintrag gleichen Namens, jedoch mit anderer Alters- oder Geschlechtsangabe in den Registerbaum übernommen worden ist. Diese fehlerhaften Einträge werden bei der Migration der Daten verworfen und bleiben unberücksichtigt.

Nicht als fehlerhaft gelten Melderegistereinträge, die in Name, Alter und Geschlecht mit einem Personeneintrag übereinstimmen, der bereits in den Registerbaum übernommen worden ist. Hier werden die Wohnsitzlisten der beiden Einträge aneinandergehängt, wobei die Wohnsitzliste des bereits im Baum befindlichen Eintrags am Anfang zu stehen kommt.

- Kurz nach erfolgreicher Migration der Melderegisterdaten in den Registerbaum fällt auf, dass die Anschriftendaten vieler Personeneinträge noch Duplikate enthalten und nicht sortiert sind.

  In einem weiteren Schritt sollen deshalb die Anschriftenlisten aller Personeneinträge im Registerbaum von Duplikaten befreit und die jeweils verbleibenden Anschriften aufsteigend sortiert angeordnet werden. Eine Anschrift a steht nach der Sortierung weiter links in der Anschriftenliste als eine Anschrift b, wenn der Gemeindeeintrag von a lexikographisch kleiner ist als der von b. Stimmen die Gemeindeeinträge von a und b überein, steht a weiter links als b, wenn der Straßeneintrag von a lexikographisch kleiner ist als der von b. Stimmen auch die Straßeneinträge von a und b überein, so steht a weiter links als b, wenn der Hausnummereintrag von a kleiner ist als der von b.

  Schreiben Sie eine Haskell-Rechenvorschrift

  ```
  bereinige_Anschriften :: Registerbaum -> Registerbaum
  ```

  die die Anschriftenlisten eines Registerbaums in dieser Weise bereinigt.

- Nach einem Wechsel in der Ratsmehrheit wird beschlossen zur Erhöhung der Arbeitsplatzqualität und Steigerung der Arbeitszufriedenheit der Gemeindebediensteten durch Entschleunigung der Antwortzeiten der Melderegistersoftware wieder zur früheren und altbewährten Listenspeicherung von Meldedaten zurückzukehren. Die Opposition ist für diesen Beschluss allerdings nur ins Boot zu holen, wenn die frühere unsortierte Listenspeicherung der Meldedaten durch eine sortierte Speicherung nach Namen ersetzt wird. Zähneknirschen stimmt die Ratsmehrheit einer lexikographisch absteigenden Sortierung nach Namenseinträgen zu.

  Schreiben Sie eine Haskell-Rechenvorschrift

  ```
  rolle_rueckwaerts :: Registerbaum -> Melderegister
  ```

  die einen Registerbaum in der angegebenen Weise in ein nach Namen lexikographisch absteigend sortiertes Melderegister überführt.

  Überlegen Sie sich, ob und wie Sie den Registerbaum durchlaufen können, so dass sie das Melderegister ohne nachträgliche Sortierung und unter ausschließlicher Verwendung des Listenkonstruktors `(:)` und ohne Verwendung der Listenkonkatenation `(++)` korrekt aufbauen können.

Nehmen Sie weitere Typklasseninstanzbildungen vor, falls nötig oder hilfreich, um die Berechnung oder die Ausgabe von Ergebnissen im Zusammenspiel mit den vorgegebenen automatischen Instanzbildungen zu erleichtern oder ermöglichen.

Hinweis: Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen wieder verwenden möchten, so kopieren Sie diese in die neue Abgabedatei ein. Ein `import` schlägt für die Auswertung durch das Abgabeskript fehl, weil Ihre alte Lösung, aus der importiert wird, nicht mit abgesammelt wird. Deshalb: Kopieren statt importieren zur Wiederwendung!

Haskell Live

Der nächste *Haskell Live*-Termin findet am Freitag, den 24.11.2017, statt.

Haskell Private

Anmeldungen zu *Haskell Private* sind weiterhin möglich. Nutzen Sie die die Möglichkeit. *Code reviews* in *Haskell Private* geben Ihnen nicht nur individuelles Feedback, sie sind auch zentral, fremden Code zu verstehen, daran zu lernen und eigenen Code zu verbessern. Nähere Hinweise und die URL zur Anmeldungsseite finden Sie auf der Homepage der Lehrveranstaltung.