4. Aufgabenblatt zu Funktionale Programmierung vom Mi, 08.11.2017.
Fällig: Mi, 15.11.2017 (15:00 Uhr)

Themen: *Funktionen über algebraischen Datentypen; Typklassen und Instanzbildungen*

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

Aufgabe

Für dieses Aufgabenblatt sollen Sie die zur Lösung der unten angegebenen Aufgabenstellungen zu entwickelnden Haskell-Rechenvorschriften in einer Datei namens `Aufgabe4.lhs` im home-Verzeichnis Ihres Accounts auf der Maschine g0 ablegen. Wie für Aufgabenblatt 3 sollen Sie also wieder ein “**literate Script**” schreiben. Versehen Sie wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung benötigen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

1. In Haskell können wir natürliche Zahlen durch folgenden algebraischen Datentyp realisieren:

   ```
   data Nat = Null | N Nat
   ```

   Der Konstruktor `Null` steht dabei für die Zahl 0, der Konstruktorausdruck `N Null` (kurz für “Nachfolger von Null”) für die Zahl 1, der Konstruktorausdruck `N (N Null)` für die Zahl 2, derKonstruktorausdruck `N (N (N Null))` für die Zahl 3, usw. Auf diese Weise besitzt jede natürliche Zahl eine eindeutige Darstellung als Wert des Datentyps `Nat`.

   Machen Sie den Typ `Nat` ohne Verwendung von `deriving`-Klauseln zu Instanzen der Typklassen

   (a) `Show`
   (b) `Eq`
   (c) `Ord`
   (d) `Num`
   (e) `Enum`

   Nutzen Sie bei der Instanzbildung für jede dieser Typklassen die in der Klasse gegebenen Protoimplementierungen bestmöglich aus.

   Implemementieren Sie also jeweils bei jeder Instanzbildung nur eine minimal nötige Menge von Funktionen, um das Verhalten aller Funktionen der Typklasse auf Werten vom Typ `Nat` vollständig festzulegen. Für einige Typklassen haben Sie dabei einen Freiheitsgrad für die Wahl der minimalen Menge, den Sie ausnützen sollen (siehe Kapitel 4.3 der Vorlesung).

   Insgesamt sollen die Instanzbildungen so vorgenommen werden, dass die Werte vom Typ `Nat` zusammen mit den darauf definierten Funktionen und Relationen in natürlicher Weise als Implementierung eines endlichen Ausschnitts der natürlichen Zahlen aufgefasst werden können.

   Speziell gilt: Die Instanzbildung von `Nat` für die Typklasse

   - `Show` soll leisten, dass die Werte vom Typ `Nat` als Zeichenreihen über dem Zeichenvorrat {`'0'`,`'1'`} dargestellt werden, wobei die Zeichenreihe die Binärdarstellung des Wertes (mit Ausnahme der Null ohne führende Nullen) eines `Nat`-Werts ist.

     Der `Nat`-Wert `Null` hat also die Zeichenreihendarstellung `"0"`, die `Nat`-Werte `N Null` und `N (N Null)` die Zeichenreihendarstellungen `"1"` und `"10"` usw.
   - `Enum` soll leisten, dass `Int`-Werte in denjenigen `Nat`-Wert konvertiert werden, der dem Maximum des `Int`-Werts und Null entspricht.
   - `Num` soll leisten, dass `Integer`-Werte in denjenigen `Nat`-Wert konvertiert werden, der dem Maximum des `Integer`-Werts und Null entspricht. Weiters soll die Differenz zweier `Nat`-Werte durch das Maximum aus Null und ihrer Differenz in $\mathbb{Z}$ gegeben sein, dargestellt durch den entsprechenden `Nat`-Wert (Die Bedeutung von `negate` ist dann durch seine Protoimplementierung in der Klasse festgelegt, da nur eine Minimalmenge von Funktionen bei der Instanzbildung implementiert werden soll).

2. Aussagenlogische Ausdrücke über einer Menge von Wahrheitswertvariablen, den logischen Konstanten *wahr* und *falsch* und den logischen Operationen Negation, Konjunktion und Disjunktion lassen sich in Haskell folgendermaßen modellieren:

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

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

data Ausdruck = K Wahrheitswert              -- Logische Konstante
                | V Variable                 -- Logische Variable
                | Nicht Ausdruck             -- Logische Negation
                | Und Ausdruck Ausdruck      -- Logische Konjunktion
                | Oder Ausdruck Ausdruck     -- Logische Disjunktion
                deriving (Eq,Show)

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

Schreiben Sie eine Haskell-Rechenvorschrift

```
auswerten :: Ausdruck -> Belegung -> Wahrheitswert
```

Angewendet auf einen aussagenlogischen Ausdruck a und eine Belegung b, liefert die Funktion `auswerten` den Wahrheitswert von a unter b. Sie dürfen davon ausgehen, dass Belegungen stets total definiert sind.

Wichtig: Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen für dieses oder spätere Aufgabenblätter 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!

Denken Sie bitte daran, dass Sie für die Lösung dieses Aufgabenblatts ein "literate" Haskell-Skript schreiben sollen!

Haskell Live

Am Freitag, den 10.11.2017, werden wir uns in *Haskell Live* u.a. mit gerne auch von Ihnen eingebrachten Lösungsvorschlägen bereits abgeschlossener Aufgabenblätter beschäftigen, sowie mit einigen der schon speziell für *Haskell Live* gestellten Aufgaben.

Haskell Private

Sie können sich ab sofort zu *Haskell Private* anmelden. Nähere Hinweise und die URL zur Anmeldungsseite finden Sie auf der Homepage der Lehrveranstaltung.