Eine Sache lernt man, indem man sie macht.
Cesare Pavese (1908-1950)
italien. Schriftsteller

Für das Können gibt es nur einen Beweis, das Tun.
Marie von Ebner-Eschenbach (1830-1916)
österr. Schriftstellerin

4. Aufgabenblatt zu Funktionale Programmierung von Mo, 28.10.2019.
Erstabgabe: Mo, 04.11.2019 (12:00 Uhr)
(Beurteilt: Teil A; ohne Abgabe und Beurteilung: Teil B)
Themen: *Funktionen über algebraischen Datentypen, Typklassen und Instanzbildungen, 'literate' Haskell-Skripte*

Besprechung von Aufgabenlösungen, Zweitabgabefrist

- **Teil A, programmiertechnische Aufgaben:** Am ersten Übungsgruppentermin, der auf die *Zweitabgabe* der programmiertechnischen Aufgaben folgt.
- **Teil B, Papier- und Bleistiftaufgaben:** Am ersten Übungsgruppentermin, der auf die *Erstabgabe* der programmiertechnischen Aufgaben folgt.
- **Teil C, Terminhinweise:** Für Übungsgruppen, Vorlesung, Tutorensprechstunde.
- **Zweitabgabefrist:** Siehe „Hinweise zu Organisation und Ablauf der Übung“ auf der Webseite der Lehrveranstaltung.

A Programmiertechnische Aufgaben (beurteilt, max. 50 Punkte)

Schreiben Sie zur Lösung der folgenden Aufgaben ein **'literates' Haskell-Skript** und legen Sie es in einer (Abgabe-) Datei namens `Angabe4.lhs` in Ihrem Home-Verzeichnis auf der Maschine `g0` ab. Der Name der Abgabedatei ist für Erst- und Zweitabgabe ident.

Kommentieren Sie Ihr Programm wieder zweckmäßig, aussagekräftig und problemangemessen. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten. Versehen Sie alle Funktionen, die Sie zur Lösung der Aufgaben benötigen, mit ihren Typdeklarationen, d.h. geben Sie deren syntaktische Signatur oder kurz, Signatur, explizit an.

Einführung, Motivation: In der Praxis ist es lästig, dass wir für das Rechnen mit den auf Aufgabenblatt 3 eingeführten `ZZ`-Werten (wie auch mit den dort eingeführten `IN_1`-Werten) (bislang) nicht auf die üblichen arithmetischen Operator- (`(+)`, `(*)`, `(-)`,...) und Relatorsymbole (`(==)`, `(/=)`,...) zurückgreifen können, sondern unsere auf Aufgabenblatt 3 eingeführten Nichtstandardoperationen und -relationen `plus`, `mal`, `minus`, `gleich`, `ungleich`,... verwenden müssen. Ein naiver Abhilfeversuch in Form von Deklarationen

```
(+) :: ZZ -> ZZ -> ZZ
(+) = plus
...
(/=) :: ZZ -> ZZ -> Bool
(/=) = ungleich
```

schlägt fehl (überlegen Sie sich, warum?), aber wir können die Typen `IN_1` und `ZZ` zu Instanzen der Typklassen `Eq, Ord, Num` und `Enum` machen, um die Überladung der Operator- und Relatorsymbole `(+)`, `(*)`, `(-)`, `(==)`, `(/=)`,... und weiterer in diesen Klassen

vorgesehener Operatoren und Relatoren wie für Werte der Typen `Int`, `Integer`, `Float`, `Double` vordefiniert auch für Werte der Typen `IN_1` und `ZZ` verfügbar zu machen.

A.1 Machen Sie den Typ `ZZ` von Aufgabenblatt 3 ohne Verwendung von `deriving`-Klauseln zu Instanzen der Typklassen

- `Eq`
- `Ord`
- `Enum`
- `Num`

Die Bedeutung der auf Aufgabenblatt 3 für `ZZ`-Werte eingeführten Operationen (`plus`, `mal`,...) und Relationen (`gleich`, `kleiner`,...) soll dabei auf den jeweils passenden Operator (`(+)` für `plus`, `(*)` für `mal`,...) oder Relator (`(==)` für `gleich`, `(<)` für `kleiner`,..) in einer dieser Typklassen übertragen werden. Die Bedeutungr aller weiteren in den Klassen vorgesehenen Operatoren und Relatoren für `ZZ`-Werte soll mit derjenigen für `Integer`-Werte übereinstimmen.

A.2 Machen Sie den Typ `ZZ` von Aufgabenblatt 3 ohne Verwendung von `deriving`-Klauseln zu einer Instanz der Typklasse `Show`. Die Funktion `show` soll `ZZ`-Werte hexadezimal als Zeichenreihe über dem Alphabet (oder: Zeichenvorrat)

$$\{\text{'-'},\text{'0'},\text{'1'},\text{'2'},\ldots,\text{'9'},\text{'A'},\text{'B'},\text{'C'},\text{'D'},\text{'E'},\text{'F'}\}$$

ausgeben (mit Ausnahme der null ohne führende Nullen, ggf. mit einem führenden `'-'`-Zeichen für negative Werte).

Aufrufbeispiele

```
show Null ->> "0"
show (Plus (Nf (Nf (Nf Eins)))) ->> "4"
show (Plus (Nf (Nf (Nf (Nf Nf (Nf (Nf (Nf (Nf Eins)))))))))) ->> "A"
show (Minus (Nf (Nf (Nf (Nf Nf (Nf (Nf (Nf (Nf Eins)))))))))) ->> "-A"
show (Minus (Nf (Nf (Nf Eins)))) ->> "-4"
```

A.3 Wir führen folgende Typklasse ein:

```
class Binaer a where
 a_zu_binaer :: a -> String
 binaer_zu_a :: String -> a
```

Machen Sie den Typ `Integer` und die Typen `IN_1` und `ZZ` von Aufgabenblatt 3 zu Instanzen der Typklasse `Binaer`. Die Funktion `a_zu_binaer` soll bei Instanzbildungen so implementiert werden, dass `a_zu_binaer` angewendet auf einen `Integer`-, `IN_1`- oder `ZZ`-Wert die jeweilige Binärdarstellung des Arguments über dem Alphabet

$$\{\text{'-'},\text{'0'},\text{'1'}\}$$

liefert, mit Ausnahme der null ohne führende Nullen, ggf. mit einem führenden `'-'`-Zeichen für negative Argumentwerte.

Entsprechend soll die Funktion `binaer_zu_a` für alle Instanzen folgendermaßen implementiert werden: Wird `binaer_zu_a` auf eine Zeichenreihe angewendet, die eine

gültige Binärdarstellung einer ganzen Zahl ist, so liefert `binaer_zu_a` den entsprechenden `a`-Wert des Arguments, wenn es diesen gibt. Ist das nicht der Fall, liefert `binaer_zu_a` den kleinstmöglichen `a`-Wert als Resultat.

Dabei gilt: Eine Zeichenreihe z ist *gültige* Binärdarstellung einer ganzen Zahl, wenn z die Zeichenreihe `"0"` ist oder eine Zeichenreihe, die mit der Zeichenreihe `"-1"` oder der Zeichenreihe `"1"` beginnt, gefolgt von einer leeren oder nichtleeren Zeichenreihe über dem Alphabet `{'0','1'}`.

Nutzen Sie bei den Instanzbildungen für die Typklassen in den Aufgaben A.1 und A.2 die in der Klassen gegebenen Protoimplementierungen bestmöglich aus. Implementieren Sie deshalb bei jeder Instanzbildung nur eine minimal nötige Menge von Funktionen, um das Verhalten aller Funktionen der Typklasse 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).

Testen Sie alle Funktionen auch mit weiteren eigenen Testdaten (ohne Abgabe).

Wichtig:

1. **'Literates' Haskell-Skript:** Denken Sie bitte daran, dass Sie für die Lösung dieses Aufgabenblatts wieder ein **'literates'** Haskell-Skript schreiben sollen, kein gewöhnliches.
2. **Wiederverwendung früherer Lösungsteile:** Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen wieder verwenden möchten, kopieren Sie diese bitte in die neue Abgabedatei ein. Ein `import` (eigener Module) 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!
3. **Abgaben auf `g0` auf gewünschtes Verhalten prüfen:** Aufgabenlösungen werden stets auf der Maschine `g0` unter Hugs überprüft. Überzeugen Sie sich deshalb bitte, dass Ihre Lösungen sich auf der `g0` unter Hugs wie von Ihnen erwartet und gewünscht verhalten. Das gilt besonders, wenn Sie für die Entwicklung Ihrer Haskell-Programme mit einem anderen Werkzeug oder einer anderen Maschine als der `g0` arbeiten!

B Papier- und Bleistiftaufgaben (ohne Abgabe/Beurteilung)

B.1 Einige Funktionen der Typklassen `Eq`, `Ord`, `Enum`, `Num` sind für ganze Zahlen total definiert, für natürliche Zahlen nur partiell.

i. Welche sind das? Aus welchen Klassen sind sie?

ii. Wie sollte man diese Funktionen implementieren, wenn man den Typ `IN_1` von Aufgabenblatt 3 in möglichst 'natürlicher' Weise zu Instanzen der Typklassen `Eq`, `Ord`, `Enum`, `Num` machen will? Begründen Sie Ihre Antwort jeweils kurz.

B.2 Gibt es einen Unterschied zur Instanzbildung für den Typ `Integer`, wenn man in Aufgabe A.3 auch den Typ `Int` zu einer Instanz der Typklasse `Binaer` machen möchte? (Hinweis: Betrachten Sie besonders die Implementierung von `binarer_zu_a` und die Nebenbedingung "Ist das nicht der Fall, liefert `binaer_zu_a` den kleinstmöglichen `a`-Wert als Resultat".)

B.3 Gegeben ist die Funktion `hoch`:

```
type Nat0 = Int
type Nat1 = Int

hoch :: Nat0 -> Nat0 -> Nat1
hoch m n = if n == 0 then 1 else m * hoch m (n-1)
```

i. Welchen Wert hat der Ausdruck `hoch (2*3) (1+1)`?

ii. Werten Sie den Ausdruck `hoch (2*3) (1+1)` Schritt für Schritt in applikativer Ordnung aus.

iii. Werten Sie den Ausdruck `hoch (2*3) (1+1)` Schritt für Schritt in nomaler Ordnung aus.

C Terminhinweise

1. **Übungsgruppen**: Alle Übungsgruppen finden in dieser Woche von heute, Montag, 28.10.2019, bis Donnerstag, 31.10.2019, statt.

   Alle weiteren geplanten Übungsgruppentermine finden Sie auf der Webseite zur Lehrveranstaltung. Mögliche Änderungen werden dort oder/und unmittelbar in den (von möglichen Änderungen betroffenen) Übungsgruppen bekanntgegeben.

2. **Vorlesung:** Der nächste Vorlesungstermin findet morgen, Dienstag, den 29.10.2019, von 08:15 Uhr bis 09:45 Uhr im Informatik-Hörsaal in der Treitlstraße statt.

   Alle weiteren geplanten Vorlesungstermine finden Sie auf der Webseite zur Lehrveranstaltung. Mögliche Änderungen werden ebenfalls dort oder/und in der Vorlesung bekanntgegeben.

3. **Tutorensprechstunde:** Die Tutorensprechstunde findet regelmäßig freitags von 12 bis 13 Uhr im `complang`-Labor statt; wegen des Feiertags in dieser Woche das nächste Mal am Freitag, den 08.11.2019. Sie finden das Labor im Innenhof des Institutsgebäudes in der Argentinierstraße 8.