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

3. Aufgabenblatt zu Funktionale Programmierung von Do, 29.10.2020.

Erstabgabe: Fr, 06.11.2020 (12:00 Uhr)

(Beurteilt: Teil A; ohne Abgabe und Beurteilung: Teil B)

Themen: *Typsynonyme, algebraische Datentypen, Funktionen auf algebraischen Datentypen, Typklassen, Muster, literate Haskell-Skripte (Kap. 1 bis Kap. 6)*

- **Teil A, programmiertechnische Aufgaben:** Besprechung am ersten Übungsgruppentermin, der auf die *Zweitabgabe* der programmiertechnischen Aufgaben folgt.
- **Teil B, Papier- und Bleistiftaufgaben:** Besprechung am ersten Übungsgruppentermin, der auf die *Erstabgabe* der programmiertechnischen Aufgaben folgt.
- **Teil C, Termin- und organisatorische Hinweise:** Für Vorlesung, Klein- und Großübungsgruppen.

Wichtig

1. Befolgen Sie die Anweisungen aus den Begleitdateien zu Angabe 1 sorgfältig, um ein reibungsloses Zusammenspiel mit dem Testsystem sicherzustellen. Wenn Sie Fragen dazu haben, stellen Sie diese bitte im TISS-Forum zur Lehrveranstaltung.
2. Erweitern Sie für die für diese Angabe zu schreibenden Rechenvorschriften die zur Verfügung gestellte Rahmendatei

   `Angabe3.lhs`

   und legen Sie diese auf oberstem Niveau in Ihrem *home*-Verzeichnis ab. Achten Sie darauf, dass "Gruppe" Leserechte für diese Datei hat. Wenn nicht, setzen Sie diese Leserechte mittels `chmod g+r Angabe3.lhs`.

   Löschen Sie keinesfalls eine Deklaration aus der Rahmendatei! Auch dann nicht, wenn Sie einige dieser Deklarationen nicht oder nicht vollständig implementieren wollen. Löschen Sie auch nicht die Modul-Anweisung `module Angabe3 where` am Anfang der Rahmendatei.
3. Der Name der Abgabedatei ist für Erst- und Zweitabgabe ident!
4. Benutzen Sie keine selbstdefinierten Module! Wenn Sie (für spätere Angaben) einzelne Rechenvorschriften früherer Lösungen wiederverwenden möchten, kopieren Sie diese bitte in die neue Abgabedatei ein. Eine `import`-Anweisung für selbstdefinierte Module schlägt für die Auswertung durch das Abgabesystem fehl, weil Ihre Modul-Datei, aus der importiert werden soll, vom Testsystem nicht mit abgesammelt wird.
5. Ihre Programmierlösungen werden stets auf der Maschine `g0` mit der dort installierten Version von GHCi überprüft. Stellen Sie deshalb sicher, dass sich Ihre Programme (auch) auf der `g0` unter GHCi so verhalten, wie von Ihnen gewünscht, und überzeugen Sie sich bei jeder Abgabe davon! Das gilt besonders, wenn Sie für die Entwicklung Ihrer Haskell-Programme mit einer anderen Maschine, einer anderen GHCi-Version oder/und einem anderen Werkzeug wie etwa Hugs arbeiten!

A Programmiertechnische Aufgaben (beurteilt, max. 50 Punkte)

Erweitern Sie zur Lösung der programmiertechnischen Aufgaben die Rahmendatei `Angabe3.lhs`. Kommentieren Sie die Rechenvorschriften in Ihrem Programm 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 stets deren syntaktische Signatur (kurz: Signatur), explizit an.

Wir führen geschlossene lückenlose Intervalle ganzer Zahlen und darüber eine Intervallarithmetik ein (lies '$=_{df}$' als 'ist definitionsgemäß gleich', lies ' $\Longleftrightarrow_{df}$ ' als 'ist definitionsgemäß äquivalent zu/mit'):

- Intervall: $\langle m, n\rangle =_{df} \{k \mid m \leq k \leq n\}$
- Leeres Intervall: $\langle\rangle =_{df} \emptyset$
- Identifikation:
  - $\langle\rangle$ und $\langle p, q\rangle \in \{\langle m, n\rangle \mid n < m\}$ sind verschiedene Darstellungen des leeren Intervalls und werden miteinander identifiziert; $\langle\rangle$ dient dabei als kanonische Darstellung des leeren Intervalls.
  - Einelementige Intervalle $\langle m, m\rangle$ werden mit der ganzen Zahl m identifiziert und umgekehrt.
- Element: $p \in \langle m, n\rangle \iff_{df} p \in \{k \mid m \leq k \leq n\}$
- Gleich: $\langle m, n\rangle = \langle p, q\rangle \iff_{df} k \in \langle m, n\rangle \Leftrightarrow k \in \langle p, q\rangle$
- Ungleich: $\langle m, n\rangle \neq \langle p, q\rangle \iff_{df} \neg(\langle m, n\rangle = \langle p, q\rangle)$
- Kleiner: $\langle m, n\rangle < \langle p, q\rangle \iff_{df} \langle m, n\rangle \subset \langle p, q\rangle$
- Kleinergleich: $\langle m, n\rangle \leq \langle p, q\rangle \iff_{df} \langle m, n\rangle \subseteq \langle p, q\rangle$
- Größer: $\langle m, n\rangle > \langle p, q\rangle \iff_{df} \langle m, n\rangle \supset \langle p, q\rangle$
- Größergleich: $\langle m, n\rangle \geq \langle p, q\rangle \iff_{df} \langle m, n\rangle \supseteq \langle p, q\rangle$
- Addition: $\langle m, n\rangle + \langle p, q\rangle =_{df} \{r + s \mid r \in \langle m, n\rangle \wedge s \in \langle p, q\rangle\}$
- Subtraktion: $\langle m, n\rangle - \langle p, q\rangle =_{df} \{r - s \mid r \in \langle m, n\rangle \wedge s \in \langle p, q\rangle\}$
- Multiplikation: $\langle m, n\rangle * \langle p, q\rangle =_{df} \alpha(\{r * s \mid r \in \langle m, n\rangle \wedge s \in \langle p, q\rangle\})$, wobei der Abschlussoperator α auf Teilmengen von $\mathbb{Z}$ folgendermaßen definiert ist:
  $\alpha(M)=N \iff_{df} M \subseteq N \wedge N \in K \wedge \forall N' \in K.\ M \subseteq N' \Rightarrow N \subseteq N'$ mit
  $K=_{df}\{P \subseteq \mathbb{Z} \mid \exists z_1, z_2 \in \mathbb{Z}.\ P = \{z \in \mathbb{Z} \mid z_1 \leq z \leq z'\}\}$
- Absolut(betrag): $\alpha(\langle m, n\rangle) =_{df} \begin{cases} \langle m, n\rangle & \text{falls } 0 \leq m \leq n \\ \langle -n, -m\rangle & \text{falls } m \leq n < 0 \\ \langle 0, n\rangle & \text{falls } m < 0 \leq n \\ \langle\rangle & \text{falls } m > n \end{cases}$

Zur Modellierung von Intervallen verwenden wir nachfolgende Datentypen. Dabei dient der Konstruktor `Leer` als kanonische Repräsentation des leeren Intervalls; der Konstruktor `Ungueltig` repräsentiert einen Sonderwert, der keinem Intervall entspricht.

```
> type UntereSchranke = Int
> type ObereSchranke  = Int
> data Intervall      = IV (UntereSchranke,ObereSchranke)
>                       | Leer
>                       | Ungueltig
```

A.1 Machen Sie den Datentyp `Intervall` zu einer Instanz der Typklasse `Show`. Die Instanzbildung soll dabei so erfolgen, dass `Intervall`-Werte wie in den folgenden Beispielen gezeigt, ausgegeben werden:

```
show Ungueltig ->> "Kein Intervall"
show Leer ->> "<>"
show (IV (2,5)) ->> "<2,5>"
show (IV (5,2)) ->> "<>"
```

Nutzen Sie bei der Instanzbildung aus, dass aufgrund der Protoimplementierungen in `Show` nicht die Bedeutung aller Funktionen in `Show` explizit implementiert werden muss.

A.2 Machen Sie den Datentyp `Intervall` zu einer Instanz der Typklasse `Eq`, so dass die Relatoren `(==)` und `(/=)` die für Intervalle eingeführte Bedeutung erhalten. Vergleiche mit `Ungueltig` sollen stets zum Aufruf `error "Vergleich nicht moeglich"` führen.

Aufrufbeispiele:

```
IV (2,5) == IV (2,5) ->> True
IV (2,5) == IV (5,2) ->> False
IV (2,5) /= IV (2,5) ->> False
IV (5,2) == Leer ->> True
IV (2,5) /= Ungueltig ->> error "Vergleich nicht moeglich"
```

Nutzen Sie wieder aus, wenn möglich, dass aufgrund der Protoimplementierungen in `Eq` möglicherweise nicht die Bedeutung beider Relatoren in `Eq` explizit implementiert werden muss.

A.3 Machen Sie den Datentyp `Intervall` zu einer Instanz der Typklasse `Ord`, so dass die Relatoren `(>)`, `(>=)`, `(<)` und `(<=)` die für Intervalle eingeführte Bedeutung erhalten. Vergleiche mit `Ungueltig` sollen stets zum Aufruf `error "Vergleich nicht moeglich"` führen, wobei für die übrigen Operatoren in `Ord` ansonsten kein besonderes Verhalten gefordert wird. Nutzen Sie wieder aus, wenn möglich, dass aufgrund der Protoimplementierungen in `Ord` möglicherweise nicht die Bedeutung aller Relatoren in `Ord` explizit implementiert werden muss.

A.4 Machen Sie den Datentyp `Intervall` zu einer Instanz der Typklasse `Num`, so dass die Operatoren `(+)`, `(-)`, `(*)` und `abs` die für Intervalle eingeführte Bedeutung erhalten. Operationen mit `Ungueltig` sollen stets zum Aufruf `error "Vergleich nicht moeglich"` führen. Nutzen Sie wieder aus, wenn möglich, dass aufgrund der Protoimplementierungen in `Num` möglicherweise nicht die Bedeutung aller Operatoren in `Num` explizit implementiert werden muss.

A.5 Intervallwerte der Form `IV (m,m)` identifizieren wir mit der ganzen Zahl m und umgekehrt. Machen Sie unter dieser Identifikation den Datentyp `Intervall` zu einer Instanz der Typklasse `Enum`. Dabei sollen alle Aufrufe von Funktionen mit `Ungueltig` oder mehrelementigen Intervallwerten (als eines von möglicherweise mehreren Argumenten) zum Aufruf von `error "Operation nicht moeglich"` führen. Nutzen Sie wieder aus, wenn möglich, die Protoimplementierungen in `Enum` bestmöglich aus.

A.6 Wir führen die Typklasse `Kanonisch` ein:

```
> class Kanonisch a where
>  kanonisch :: a -> a
```

Machen Sie den Datentyp `Intervall` zu einer Instanz von `Kanonisch`. Intervalle der Form `IV (m,n)` werden auf `Leer` abgebildet für `m>n`; alle anderen Intervallwerte werden ident abgebildet.

A.7 Wir führen die Typklasse `Element` ein:

```
> class Element a where
>  is_elem :: Int -> a -> Maybe Bool
```

Machen Sie den Datentyp `Intervall` zu einer Instanz von `Element`, so dass die wahrheitswertsfunktionsähnliche Rechenvorschrift `is_elem` die Bedeutung der für Intervalle eingeführten Elementfunktion bekommt. Angewendet auf `Ungueltig` als Intervallwert liefert `is_elem` den Wert `Nothing`.

A.8 Wir führen die Typklasse `Code` ein:

```
> class Element a => Code a where
>  codiere   :: [Int] -> a
>  decodiere :: a -> Maybe [Int]
```

Machen Sie den Datentyp `Intervall` zu einer Instanz von `Code`, so dass `codiere` und `decodiere` ganzzahlige Listen in die entsprechenden Intervallwerte überführen und umgekehrt. Im Detail: Angewendet auf eine echt aufsteigende lückenlose ganzzahlige Liste liefert `codiere` den entsprechenden Intervallwert, der diese Liste repräsentiert, ansonsten den Sonderwert `Ungueltig`. Umgekehrt liefert `decodiere` angewendet auf einen Intervallwert die davon repräsentierte aufsteigende lückenlose ganzzahlige Liste, ansonsten den Wert `Nothing`.

Aufrufbeispiele:

```
codiere [] ->> Leer
codiere [2,3,4,5] ->> IV (2,5)
codiere [2,4,5] ->> Ungueltig
codiere [2,5,4] ->> Ungueltig
codiere [3,2,5,4] ->> Ungueltig
codiere [2,2,3,4,5] ->> Ungueltig
codiere [5,4..2] ->> Ungueltig
codiere [2..5] ->> IV (2,5)

decodiere Leer ->> Just []
decodiere Ungueltig ->> Nothing
decodiere (IV (2,5)) ->> Just [2,3,4,5]
decodiere (IV (5,2)) ->> Just []
```

A.9 Wir führen die Typklasse `ExTest` ein:

```
> class (Ord a,Enum a) => ExTest a where
>  extrahiere :: Maybe [a] -> [a]
>  ist_aufsteigend :: [a] -> Bool
>  ist_lueckenlos :: [a] -> Bool
>  ist_laL_Element :: a -> Maybe [a] -> Bool
```

Machen Sie den Datentyp `Int` zu einer Instanz von `ExTest`. Die Funktion `extrahiere` soll dabei von `Just`-Werten den Konstruktor abstreifen, ansonsten zum Aufruf von `error "Extraktion nicht moeglich"` führen. Die Wahrheitswertfunktionen `ist_aufsteigend` und `ist_lueckenlos` überprüfen, ob die Argumentliste aufsteigend geordnet ist bzw. keine Lücken enthält, (mit Ausnahme des kleinsten und größten) jedes Listenelement also Vorgänger bzw. Nachfolger eines anderen Listenelements ist. Die Wahrheitswertfunktion `ist_laL_Element` überprüft, ob das zweite Argument eine lückenlos aufsteigende Liste ist, die das erste Argument als Element enthält und liefert entsprechend `True` oder `False`. Ist das zweite Argument kein `Just`-Wert, ist das Resultat stets `False`.

Aufrufbeispiele:

```
extrahiere (Just [2..5]) ->> [2,3,4,5]
extrahiere (Just [5..2]) ->> []
extrahiere (Just []) ->> []
extrahiere Nothing ->> error "Extraktion nicht moeglich"

ist_aufsteigend [2..5] ->> True
ist_aufsteigend [2,5..21] ->> True
ist_aufsteigend [5..2] ->> True
ist_aufsteigend [5,4..2] ->> False

ist_lueckenlos [2..5] ->> True
ist_lueckenlos [2,5..21] ->> False
ist_lueckenlos [5..2] ->> True
ist_lueckenlos [5,4..2] ->> True
ist_lueckenlos [5,3,4,1,2] ->> True
ist_lueckenlos [5,3,4,5,1,3,5,2] ->> True

ist_laL_Element 3 (Just [2..5]) ->> True
ist_laL_Element 3 (Just [5..2]) ->> False
ist_laL_Element 3 (Just []) ->> False
ist_laL_Element 3 (Just [5,4..2]) ->> False
ist_laL_Element 8 (Just [2,5..21]) ->> False
ist_laL_Element 5 (Just [1,2,3,3,4,5,5,5]) ->> True
ist_laL_Element 3 Nothing ->> False
```

A.10 **Ohne Abgabe, ohne Beurteilung:** Hätten Sie das Ziel einiger Instanzbildungen auch mithilfe von `deriving`-Klauseln erreichen können? Begründen Sie Ihre Antwort.

A.11 **Ohne Beurteilung:** Beschreiben Sie für jede Rechenvorschrift in einem Kommentar knapp, aber gut nachvollziehbar, wie die Rechenvorschrift vorgeht.

A.12 **Ohne Abgabe, ohne Beurteilung:** Testen Sie alle Funktionen umfassend, auch mit eigenen Testdaten.

B Papier- und Bleistiftaufgaben (ohne Abgabe/Beurteilung)

B.1 Wie ändert sich der (Programmierer-) Blick auf ein Haskell-Programm beim Übergang von Programmen in *gewöhnlicher* zu *literater* Darstellung und umgekehrt?

B.2 Wer hat *literates Programmieren* geprägt? Wofür ist diese Person darüber hinaus bekannt (hier ist vieles aufzählbar)?

B.3 Einige Funktionen der Typklassen `Eq`, `Ord`, `Enum`, `Num` sind auf ganzen Zahlen total definiert, auf natürlichen Zahlen nur partiell. Welche aus welchen Klassen genau?

B.4 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.

Iucundi acti labores.
Getane Arbeiten sind angenehm.

Cicero (106 - 43 v.Chr.)
röm. Staatsmann und Schriftsteller

C Termin- und organisatorische Hinweise

1. **Zusammenlegung Kleinübungsgruppen**: Aufgrund geringer Anmeldezahlen für einige Freitagsgruppen werden die Gruppen
   - KÜG 5 und KÜG 6
   - KÜG 11 und KÜG 12

   zusammengelegt. Neuer Veanstaltungstermin für die so neu entstehenden Gruppen
   - KÜG 5&6
   - KÜG 11&12

   ist freitags von 11-13 Uhr für Präsenztermine und freitags von 11-12 Uhr für Online-Termine.

2. **Nächster Vorlesungstermin: Mittwoch, 04.11.2020**, 08:15-09:45 Uhr, Echtzeitvideokonferenz unter der bekannten Zoom-URL (siehe TISS):
   - 08:15-09:00 Uhr: Vorlesungsteil IV
   - 09:15-09:45 Uhr: Umgekehrtes Klassenzimmer zu Vorlesungsteil III

3. **Nächste Kleinübungsgruppentermine:**
   - **Kleinübungsgruppen in Präsenz:**
     - KÜG 1: Di, 03.11.2020, 16:00-18:00 Uhr, FAV HS 1, Favoritenstr.
     - KÜG 2: Do, 05.11.2020, 08:00-10:00 Uhr, FAV HS 1, Favoritenstr.
     - KÜG 3: Do, 05.11.2020, 11:00-13:00 Uhr, FAV HS 1, Favoritenstr.
     - KÜG 4: Do, 05.11.2020, 14:00-16:00 Uhr, FAV HS 1, Favoritenstr.
     - *Fr, 06.11.2020, 08:00-10:00 Uhr: Termin entfällt!*
     - KÜG 5&6: Fr, 06.11.2020, 11:00-13:00 Uhr, FAV HS 1, Favoritenstr.
   - **Kleinübungsgruppen als Echtzeitvideokonferenz:**
     - **Zoom-URLs für alle KÜGs: Siehe Tuwel!**
     - KÜG 7: Di, 03.11.2020, 16:00-17:00 Uhr, *online* via Zoom
     - KÜG 8: Do, 05.11.2020, 08:00-9:00 Uhr, *online* via Zoom
     - KÜG 9: Do, 05.11.2020, 11:00-12:00 Uhr, *online* via Zoom
     - KÜG 10: Do, 05.11.2020, 14:00-15:00 Uhr, *online* via Zoom
     - *Fr, 06.11.2020, 08:00-9:00 Uhr: Termin entfällt!*
     - KÜG 11&12: Fr, 06.11.2020, 11:00-12:00 Uhr, *online* via Zoom
     - KÜG 13: Mi, 04.11.2020, 15:00-16:00 Uhr, *online* via Zoom
     - KÜG 14: Fr, 06.11.2020, 15:00-16:00 Uhr, *online* via Zoom
     - KÜG 15: Fr, 06.11.2020, 17:00-18:00 Uhr, *online* via Zoom

   Mögliche Änderungen werden via TISS (-Aussendung) bzw. in den betroffenen Kleinübungsgruppen bekanntgegeben.

4. **Nächste Plenumsübungsgruppentermine:**
   - PÜG I: Di, 03.11.2020, 09:00-10:00 Uhr, Informatik-Hörsaal, Treitlstr.
   - PÜG II: Mi, 04.11.2020, 16:00-17:00 Uhr, Informatik-Hörsaal, Treitlstr.

5. **TISS-Gebäudezutrittsvalidierung:** Alle Teilnehmer von KÜG 1 bis 6 werden in der kommenden Woche zusätzlich den mit Räumen verbundenen SPK-Übungsgruppen zugeordnet. Dadurch wird die positive Validierung ihres Zutritts zum Gebäude in der Favoritenstr. durch das TUW-Zutrittsmanagementsystem möglich. In der Woche darauf wird das entsprechend für die Teilnehmer von KÜG 7 bis 12 so gehandhabt. Danach wiederholt sich dieser Rhythmus.

6. **Sitzplanerstellung:** Für alle in Präsenz stattfindenden KÜGs und PÜGs müssen entsprechend einer universitären Richtlinie zusätzlich zur Zutrittsvalidierung **zwingend Sitzpläne erstellt werden**. Dazu **muss jeder**, der bei einer in Präsenz stattfindenden KÜG oder PÜG anwesend ist, während der Veranstaltungszeit in einer in Tuwel verfügbaren **freien Textfeldaufgabe formlos seinen Sitzplan eintragen!**

COVID-19-Hinweis: Ein Tausch von Klein- und Plenumsübungsgruppen oder die Präsenzteilnahme an einer anderen als zur angemeldeten Klein- und Plenumsübungsgruppe sind **nicht möglich**, um nicht durch Vermischung der Gruppen eine Verbreitung des Corona-Virus zu begünstigen.