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

5. Aufgabenblatt zu Funktionale Programmierung von Mo, 11.11.2019.
Erstabgabe: Mo, 18.11.2019 (12:00 Uhr)

Themen: *Funktionen höherer Ordnung, Rechnen mit Funktionen, Funktionen als Argument, echte und unechte Polymorphie, Typklassen, Instanzbildung (automatisch, manuell) für Typklassen*

Besprechung von Aufgabenlösungen, Zweitabgabefrist

- **Teil A, programmiertechnische Aufgaben:** Am ersten Übungsgruppentermin, der auf die *Zweitabgabe* der programmiertechnischen Aufgaben folgt.
- Teil B, Papier- und Bleistiftaufgaben: Entfällt auf den Angaben 5 bis 7.
- **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. 100 Punkte)

Schreiben Sie zur Lösung der folgenden Aufgaben ein gewöhnliches Haskell-Skript und legen Sie es in einer (Abgabe-) Datei namens `Angabe5.hs` 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.

A.1 Im fernen Land *Streikistanien* zeigen die politischen Parteien ihre Muskeln, indem sie ihre Anhänger regelmäßig zu Streiks aufrufen. Die Beeinträchtigung des täglichen Lebens ist um so höher je mehr Parteien am gleichen Tag zu einem Streik aufrufen. Deshalb ist es hilfreich, die Anzahl von Streiktagen innerhalb eines Zeitraums im voraus berechnen zu können. Das wird dadurch möglich, dass jede Partei immer nur zu eintägigen Streiks aufruft, die sich in festen Abständen wiederholen. Darüberhinaus ruft keine Partei an Freitagen und Sonntagen zu einem Streik auf. Streikaufrufe, die auf einen Freitag oder Sonntag fallen würden, entfallen.

Beispiel:

Partei	Streikaufruf-abstand in Tagen	Zeitraum in Tagen														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo
0	2		×		×		×		×		×					
1	3			×			×			×						×
2	4				×				×							
3	5										×					×

Im Beispiel gibt es 4 Parteien, die jeden zweiten, dritten, vierten bzw. fünften Tag zu einem eintägigen Streik aufrufen. In einem montags beginnenden 15-tägigen Zeitraum gibt es damit 8 Tage, an denen mindestens eine Partei zu einem Streik aufruft; 5 Tage, an denen mindestens zwei Parteien zu einem Streik aufrufen und keinen Tag, an dem 3 oder mehr Parteien zu einem Streik aufrufen.

Beginnt der 15-tägige Zeitraum hingegen an einem Mittwoch, so gibt es 7 Tage, an denen mindestens eine Partei zu einem Streik aufruft und 4 Tage, an denen mindestens zwei Parteien zu einem Streik aufrufen und keinen Tag, an dem 3 oder mehr Parteien zu einem Streik aufrufen.

Partei	Streikaufruf-abstand in Tagen	Zeitraum in Tagen														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		Mi	Do	Fr	Sa	So	Mo	Di	Mi	Do	Fr	Sa	So	Mo	Di	Mi
0	2		×		×		×		×						×	
1	3						×			×						×
2	4				×				×							
3	5															×

Die Aufgabe ist nun für einen vorgegebenen Zeitraum zu berechnen, an wie vielen Tagen das tägliche Leben durch den Aufruf einer oder mehrerer Parteien zu einem Streik beeinträchtigt ist.

Eine Partei ist durch den Abstand ihrer Streikaufrufe in Tagen vollständig charakterisiert; die verschiedenen Parteien in *Streikistanien* durch ihre Indexposition in einer Liste von Parteien eindeutig benannt und identifizierbar. Damit können wir die Aufgabe wie folgt modellieren:

```
type Nat0       = Int
type Nat1       = Int
data Wochentag  = Mo | Di | Mi | Do | Fr | Sa | So deriving (Eq,Show)
type Starttag   = Wochentag
type Zeitraum_in_Tagen            = Nat1
type Streikaufrufabstand_in_Tagen = Nat1
newtype Partei  = P Streikaufrufabstand_in_Tagen deriving Show
type Parteien   = [Partei]
type Streiktage = Nat0
type Anzahl_Parteien = Nat1
type Modellszenario  = (Starttag,Zeitraum_in_Tagen,Parteien)
```

Schreiben Sie folgende Haskell-Rechenvorschriften:

(a) `streiktage :: Modellszenario -> Streiktage`

...liefert die Zahl der Tage im Modellszenrio, an denen mindestens eine Partei zum Streik aufruft.

(b) `superstreiktage :: Modellszenario -> Streiktage`

...liefert die Zahl der Tage im Modellszenario, an denen alle Parteien zum Streik aufrufen.

(c) `grossstreiktage :: Modellszenario -> Anzahl_Parteien -> Streiktage`

...liefert die Zahl der Tage im Modellszenario, an denen mindestens so viele Parteien zum Streik aufrufen, wie durch den Wert des `Anzahl_Parteien`-Arguments angegeben ist.

(d) `streiktage_am :: Modellszenario -> Wochentag -> Anzahl_Parteien -> Streiktage`

...liefert die Zahl der Tage im Modellszenario, an denen mindestens so viele Parteien zum Streik aufrufen, wie durch den Wert des `Anzahl_Parteien`-Arguments angegeben ist und die auf den durch den Wert des `Wochentag`-Arguments bestimmten Wochentag fallen.

(e) `wird_gestreikt :: Modellszenario -> Nat1 -> Bool`

...liefert `True`, wenn am Tag n mindestens eine Partei zum Streik aufruft, sonst `False`.

Nehmen Sie bei Bedarf weitere Instanzbildungen für Typklassen vor.

A.2 Arithmetische und logische Ausdrücke über einer (jeweils endlichen) Menge arithmetischer und logischer Variablen, arithmetischen und logischen Konstanten und arithmetischen und logischen Operatoren und Relatoren lassen sich in Haskell folgendermaßen modellieren:

```
data Arith_Variable = A1 | A2 | A3 | A4 | A5 | A6 deriving (Eq,Show)
data Log_Variable   = L1 | L2 | L3 | L4 | L5 | L6 deriving (Eq,Show)

data Arith_Ausdruck = AK Int                -- Arithmetische Konstante
                    | AV Arith_Variable  -- Arithmetische Variable
                    | Plus Arith_Ausdruck Arith_Ausdruck  -- Addition
                    | Minus Arith_Ausdruck Arith_Ausdruck -- Subtraktion
                    | Mal Arith_Ausdruck Arith_Ausdruck   -- Multiplikation
                    deriving (Eq,Show)

data Log_Ausdruck = LK Bool                           -- Logische Konstante
                  | LV Log_Variable                   -- Logische Variable
                  | Nicht Log_Ausdruck                -- Logische Negation
                  | Und Log_Ausdruck Log_Ausdruck     -- Logische Konjunktion
                  | Oder Log_Ausdruck Log_Ausdruck    -- Logische Disjunktion
                  | Gleich Arith_Ausdruck Arith_Ausdruck  -- Wertgleichheit
                                                          -- arith. Ausdruecke
                  | Kleiner Arith_Ausdruck Arith_Ausdruck -- Linker Ausdruck
                                                          -- echt wertkleiner
                                                          -- als rechter
                                                          -- Ausdruck
                  deriving (Eq,Show)

type Arith_Variablenbelegung = Arith_Variable -> Int -- Total definierte Abb.
type Log_Variablenbelegung   = Log_Variable -> Bool   -- Total definierte Abb.
type Variablenbelegung       = (Arith_Variablenbelegung,Log_Variablenbelegung)

links :: (Either a b) -> a
links (Left x) = x

rechts :: (Either a b) -> b
rechts (Right y) = y
```

```
class Evaluierbar a where
 evaluiere :: a -> Variablenbelegung -> Either Int Bool
```

Schreiben Sie eine Evaluierungsfunktion für arithmetische und logische Ausdrücke. Machen Sie dazu die Typen

(a) `Arith_Ausdruck`

(b) `Log_Ausdruck`

zu Instanzen der Typklasse `Evaluierbar`.

Angewendet auf einen arithmetischen oder logischen Ausdruck a und eine Variablenbelegung b, liefert die überladene Funktion `evaluiere` den Wert von Ausdruck a unter der Variablenbelegung b als Wert vom Typ `Either Int Bool`. Das ist ein Wert vom Muster `Left z` mit `z` ganze Zahl, wenn a ein arithmetischer Ausdruck ist, und ein Wert vom Muster `Right b` mit `b` Wahrheitswert, wenn a ein logischer Ausdruck ist. Die Funktionen `links` und `rechts` erlauben diese Werte vollständig auszupacken, indem sie die Konstruktoren `Left` und `Right` abstreifen. Sie dürfen bei der Implementierung davon ausgehen, dass Belegungen für arithmetische und logische Variablen stets total definiert sind.

Nehmen Sie bei Bedarf weitere Instanzbildungen für Typklassen vor.

Aufrufbeispiele:

```
avb = \ av -> 3 :: Arith_Variablenbelegung
lvb = \ lv -> True :: Log_Variablenbelegung
vb1  = (avb,lvb) :: Variablenbelegung
vb2 = (\ av -> if av == A1 then 42 else avb av,
       \ lv -> if lv /= L6 then False else (mod (avb A3) 3 == 0)
      :: Variablenbelegung

aa1 = Mal (Plus (AV A1) (AV A2)) (Mal (AV A1) (AV A2))
aa2 = AK 42
la1 = Nicht (Kleiner aa1 aa1)
la2 = Oder (Nicht (Gleich aa1 aa1)) (Oder (Nicht (LV L3)) (Nicht la1))

links (evaluiere aa1 vb1)  ->> links (Left 54) ->> 54
links (evaluiere aa1 vb2)  ->> links (Left 5670) ->> 5670
links (evaluiere aa2 vb1)  ->> links (Left 42) ->> 42
links (evaluiere aa2 vb2)  ->> links (Left 42) ->> 42
rechts (evaluiere la1 vb1) ->> rechts (Right True) ->> True
rechts (evaluiere la1 vb2) ->> rechts (Right True) ->> True
rechts (evaluiere la2 vb1) ->> rechts (Right False) ->> False
rechts (evaluiere la2 vb2) ->> rechts (Right True) ->> True
```

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

Wichtig:

1. **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!
2. **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 für dieses und auch alle weiteren Aufgabenblätter sich auf der `g0` unter Hugs wie von Ihnen erwartet und gewünscht verhalten; überzeugen Sie sich bei jeder Abgabe davon. 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

Entfällt auf den Angaben 5 bis 7.

C Terminhinweise

1. **Übungsgruppen**: Die nächsten Übungsgruppentreffen finden diese Woche statt, von heute, Montag, 11.11.2019, bis Donnerstag, 14.11.2019.
2. **Vorlesung:** Der nächste Vorlesungstermin ist morgen, Dienstag, den 12.11.2019, von 08:15 Uhr bis 09:45 Uhr im Informatik-Hörsaal in der Treitlstraße.
3. **Tutorensprechstunde:** Die nächste Tutorensprechstunde findet kommende Woche statt, am Freitag, den 22.11.2019, im `complang`-Labor (am Freitag dieser Woche, 15.11.2019, ist aus Anlass des "Tag des Landespatron" lehrveranstaltungsfrei). Sie finden das `complang`-Labor im Innenhof des Institutsgebäudes in der Argentinierstraße 8.

Alle weiteren geplanten Vorlesungs- und Übungsgruppentermine finden Sie auf der Webseite zur Lehrveranstaltung. Mögliche Änderungen werden dort oder/und in der Vorlesung bzw. in den von möglichen Änderungen betroffenen Übungsgruppen bekanntgegeben.