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

Aufgabenblatt 3 zu Funktionale Programmierung von Do, 31.10.2024

Abgabe: Fr, 08.11.2024, 23:30 Uhr

(*Verbesserung in Anwesenheit* in Kalenderwoche 47 vom 20.11.2023, ohne Beurteilung)

Themen: *Typklassen, Instanzbildungen für Typklassen, algebraische Datentypen, Typsynome, neue Typen, Überladung, rekursives Rechnen auf algebraischen und neuen (Daten-) Typen*

Stoffumfang: *Kapitel 1 bis Kapitel 9.*

Schreiben Sie zur Lösung der programmiertechnischen Aufgaben in Abschnitt A ein gewöhnliches Haskell-Skript `<dateiname.hs>` (s. Kap. 1.3.2). Kommentieren Sie die Rechenvorschriften in Ihrem Programm aussagekräftig. Verwenden Sie, wo sinnvoll, Hilfsfunktionen. Vereinbaren Sie sprechende Namen für häufig verwendete Werte (z.B. `pi = 3.14 :: Float`) und Typen (z.B. `type Schuhgroesse = Int`). Versehen Sie Funktionen, die Sie zur Lösung benötigen, immer mit ihren Typdeklarationen, d.h. geben Sie stets deren syntaktische bzw. Funktionssignatur (kurz: Signatur) an.

A Programmiertechnische Aufgaben

A.1 Wir führen folgende Datentypen zur Darstellung natürlicher Zahlen in Binärdarstellung ein:

```
data Ziffer  = Null | Eins
data NatZahl = L Ziffer -- L wie ‘Letzte Ziffer‘
             | E Ziffer NatZahl -- E wie ‘Eine weitere Ziffer’
```

Machen Sie den Typ `NatZahl` mithilfe von `instance`-Deklarationen (keine `deriving`-Klauseln verwenden!) zu Instanzen der Typklassen:

(a) `Eq`
(b) `Ord`
(c) `Enum`
(d) `Num`
(e) `Bound`
(f) `Show`
(g) `Read`

Die Elementfunktionen der verschiedenen Typklassen (Gleichheits- und Ungleichheitstest aus Typklasse `Eq`, Vergleichstests aus Typklasse `Ord`, algebraische Operatoren aus `Num`, etc.) sollen dabei ihre offensichtliche ‘natürliche’ Bedeutung für natürliche Zahlen erhalten.

Operationen wie minus, die aus dem Zahlbereich natürlicher Zahlen herausführen können, können Sie nach Wahl mit einem Fehler (Aufruf von `error "<diesen Text durch eine freigewählte treffende Fehlermeldung ersetzen>"`: sog. Panikmodusbehandlung (siehe Kapitel 16.2)) oder mit dem Resultat `Null` implementieren.

Überlegen Sie sich für andere Funktionen wie z.B. `maxBound` in Typklasse `Bound`, die keine offensichtliche Bedeutung für natürliche Zahlen und damit `NatZahl`-Werte haben, eine passende Implementierung und kommentieren Sie Ihre Entwurfsentscheidung knapp, aber gut nachvollziehbar.

Für die Typklasse `Show` soll die Instanzbildung so erfolgen, dass `NatZahl`-Werte als Zeichenreihen über dem Zeichenvorrat (= Alphabet) {'0','1'} ausgegeben werden:

```
show (L Null) ->> "0"
show (L Eins) ->> "1"
show (E Eins (E Eins (L Null))) ->> "110"
show (E Null (E Null (E Eins (E Eins (L Null))))) ->> "110"
```

Wie im Beispiel gezeigt, sollen führende Nullen für von null verschiedene Zahlen in der Ausgabe unterdrückt werden.

Für die Typklasse `Read` soll die Instanzbildung entsprechend umgekehrt erfolgen: Zeichenreihen über dem Alphabet {'0','1'} werden in den entsprechenden `NatZahl`-Wert überführt, wobei möglicherweise führende Nullen in der Eingabezeichenreihe unterdrückt werden. Überlegen Sie sich wieder eine passende Fehlerbehandlung, wenn die Eingabezeichenreiche andere Zeichen als '0' und '1' enthält (Überlesen, Panikmodus,...) und kommentieren Sie Ihre Entwurfsentscheidungn kapp, aber gut nachvollziehbar.

Machen Sie bei den Instanzbildungen für alle Typklassen bestmöglichen Gebrauch der Protoimplementierungen in den Typklassen und implementieren Sie deshalb immer nur die jeweils minimal nötige (Teil-) Menge von Elementfunktionen der Typklassen bei den Instanzbildungen.

A.2 Um `NatZahl`-Werte auf dem Bildschirm ausgeben zu können, haben wir in Aufgabe A.1 den Typ `NatZahl` zu einer Instanz der Typklasse `Show` gemacht. Eine zweite Instanzbildung für `NatZahl` und `Show` im selben Programm für eine zweite Darstellungsart von `NatZahl`-Werten ist nicht möglich. Deshalb geben wir `NatZahl`-Werten mithilfe einer `newtype`-Deklaration eine neue '(Typ-) Identität', womit auch gleich ein wichtiger Anwendungsfall von `newtype`-Deklarationen demonstriert wird:

```
newtype NatZahl' = NI NatZahl -- NI wie 'neue Identität'
```

Machen Sie den Typ `NatZahl'` mithilfe von `instance`-Deklarationen (keine `deriving`-Klauseln verwenden!) zu Instanzen der Typklassen:

(a) `Eq`

(b) `Show`

Der Gleichheits- und Ungleichheitstest der Typklasse `Eq` sollen dabei wieder ihre offensichtliche 'natürliche' Bedeutung für natürliche Zahlen erhalten. Die Ausgabe von `NatZahl'`-Werten auf dem Bildschirm soll in Dezimaldarstellung erfolgen, wobei führende Nullen wie in Aufgabe A.1 behandelt werden sollen.

A.3 Schreiben Sie zwei Konvertierungsfunktionen `k` und `k'`, die `NatZahl`- in `NatZahl'`-Werte überführen und umgekehrt. Führende Nullen sollen dabei jeweils erhalten bleiben:

```
k  :: NatZahl -> NatZahl'
k' :: NatZahl' -> NatZahl
```

A.4 Schreiben Sie weitere Konvertierungsfunktionen, die `Nat0`-Werte in `NatZahl`- bzw. `NatZahl'`-Werte überführen und umgekehrt:

```
type Nat0 = Int
k1 :: Nat0 -> NatZahl
k2 :: NatZahl -> Nat0
k3 :: Nat0 -> NatZahl'
k4 :: NatZahl' -> Nat0
```

A.5 Beschreiben Sie für jede Rechenvorschrift in einem Kommentar knapp, aber gut nachvollziehbar, wie die Rechenvorschrift vorgeht.

A.6 Testen Sie alle Funktionen umfassend mit aussagekräftigen eigenen Testdaten.

B Papier- & Bleistiftaufgaben (ohne Abgabe)

B.1 Wie ändert sich die Ausgabe von `NatZahl`-Werten, wenn Sie die Instanzdeklaration von `Show` durch eine entsprechende `deriving`-Klausel ersetzen? Warum ist das so?

B.2 (a) Für welche Typklassen in Aufgabe A.1 wäre die Instanzbildung überhaupt mit einer `deriving`-Klausel möglich?

(b) Hätten sich für diese Typklassen dieselben Bedeutungen der Elementfunktionen der Klassen ergeben wie durch Ihre `instance`-Deklarationen?

Begründen Sie Ihre Antwort jeweils.

B.3 Warum ist es nicht nötig, in Aufgabe A.4 eine Vorgabe über die Behandlung führender Nullen zu machen?

B.4 Welche Typdeklarationen auf diesem Aufgabenblatt sind Deklarationen von

(a) Typsynonymen

(b) Neuen Typen

(c) Algebraischen Typen?

Welche der algebraischen Typdeklarationen führen

i. Aufzählungstypen

ii. Produkttypen

iii. Summentypen

ein?

Woran erkennt man das jeweils?

B.5 In welchen Aufgaben auf diesem Aufgabenblatt geht es um Überladung?

Iucundi acti labores.
Getane Arbeiten sind angenehm.

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

C Das Sprachduell (ohne Abgabe)

...ist Haskell oder eine andere Sprache besser zur Lösung folgender Aufgabe geeignet?

City-Maut

Wie in anderen Städten wird auch in Wien überlegt, eine City-Maut einzuführen, um die Verkehrsströme besser kontrollieren und lenken zu können. Für die Einführung einer City-Maut sind verschiedene Modelle denkbar: in unserem Modell liegt der Fokus auf innerstädtischen Nadelöhren.

Unter einem *Nadelöhr* verstehen wir eine Verkehrsstelle, die auf dem Weg von einem Bezirk A zu einem Bezirk B in der Stadt passiert werden muss, für das es also keine Möglichkeit zur Umfahrung gibt.

Um den Verkehr an Nadelöhren zu beeinflussen, sollen genau dort Mautstationen eingerichtet und Verkehrsverursachungsgebühren eingehoben werden.

In der Musterstadt MS mit den sechs Bezirken A, B, C, D, E, F und den sieben in beiden Richtungen befahrbaren Routen $B \leftrightarrow C$, $A \leftrightarrow B$, $C \leftrightarrow A$, $D \leftrightarrow C$, $D \leftrightarrow E$, $E \leftrightarrow F$ und $F \leftrightarrow C$ führt jede Fahrt von Bezirk A in Bezirk E unvermeidbar durch Bezirk C: Bezirk C ist also ein Nadelöhr und muss deshalb gemäß unseres Modells mit einer Mautstation versehen werden.

Ihre Aufgabe ist es nun, ein Programm zu schreiben, das für eine durch seine Bezirke und Routen wie oben beschriebene Stadt die Anzahl und Namen aller Nadelöhre bestimmt.

Der Einfachheit halber gehen wir davon aus, dass anstelle von Bezirksnamen fortlaufende Bezirksnummern beginnend mit 1 verwendet werden und der Bezirks- und Routenplan in Form eines Tupels zur Verfügung gestellt wird, das die Anzahl der Bezirke und die möglichen in jeweils beiden Richtungen befahrbaren direkten Routen von Bezirk zu Bezirk innerhalb der Stadt angibt. In Haskell könnte dies durch einen Wert des Datentyps `CityPlan` modelliert werden:

```
type Bezirk      = Int
type AnzBezirke  = Int
type Route       = (Bezirk,Bezirk)
newtype CityPlan = CP (AnzBezirke,[Route])
```

Gültige Bezirks- und Routenpläne müssen offenbar einigen Wohlgeformtheitsanforderungen genügen. Überlegen Sie sich, welche das sind und wie Sie sie überprüfen können, so dass Ihr Nadelöhrsuchprogramm schließlich nur auf wohlgeformte Bezirks- und Routenpläne angewendet wird.

C.1 Treffen Sie Ihre Wahl für das Sprachduell! (ist es nicht Haskell, kommen Sie am Ende des Semesters noch einmal auf das Duell zurück!)

C.2 Schreiben Sie in der von Ihnen gewählten Sprache ein Programm, das die Nadelöhre zu einem vorgelegten Bezirks- und Routenplan aufdeckt.

(*Lösungsvorschläge für das Sprachduell werden, soweit es die Zeit erlaubt, in einer der kommenden Übungen besprochen.*)