LVA 185.A03 Funktionale Programmierung (WS 20)

Leit- und Kontrollfragen II

Di, 13.10.2020

Stoff: Vorlesungsteil II – Kapitel 2, 3, 4, 5 und 6

Grundlagen: Vordefinierte Datentypen, Funktionen, Typsynonyme/Neue Typen/Typklassen, Algebraische Datentypdeklarationen, Muster

(Ohne Abgabe, ohne Beurteilung; zur Selbsteinschätzung)

Teil II, Kapitel 2 'Vordefinierte Datentypen'

1. Welche vordefinierten Datentypen bietet Haskell?
2. Welche davon sind strukurierte Datentypen, welche unstrukturierte?
3. Welche Datentypen gibt es in Haskell zur Darstellung von Zahlen? Wie unterscheiden sie sich?
4. Schließen Sie von den Namen der vordefinierten Datentypen auf die Namen jedes Datentyps in Haskell. Müssen die Namen von von Datentypen mit einem Groß- oder einem Kleinbuchstaben beginnen? Oder ist das egal?
5. Welche Werte enthält der Datentyp der Wahrheitswerte?
6. Wann sprechen wir von einem Operator? Wann von einer Operation?
7. Welche vordefinierten Operatoren gibt es auf Gleitkommazahlen? Nennen Sie einige Beispiele.
8. Wozu dienen die vordefinierten Funktionen `ord` und `char`? Welche Signatur haben diese Funktionen?
9. Wie sind Zeichenkonstanten in Haskell dargestellt? Geben Sie einige Beispiele, die Ihre Antwort illustrieren.
10. Welches ist der vordefinierte Operator für Paare in Haskell? Welches der für Tripel?
11. Welches ist der vordefinierte Operator für Listen in Haskell?
12. Wie ist die leere Liste in Haskell dargestellt? Wie das leere Tupel?
13. Wie kann man ein Element als neues Kopfelement an den Anfang einer Liste setzen?
14. Wie kann man ein Element als (neues) letztes Element an einer Liste anfügen?
15. Welchen Typ haben die folgenden Ausdrücke?
    (a) `["sin","cos","tan"]`
    (b) `[sin,cos,tan]`
    (c) `[]`
    (d) `("sin","cos","tan")`
    (e) `(sin,cos,tan)`
    (f) `()`
    (g) `[("sin","cos","tan")]`
    (h) `[(sin,cos,tan)]`
    (i) `[()]`
    (j) `(["sin","cos","tan"])`
    (k) `([sin,cos,tan])`
    (l) `([])`

16. Mit welchem Operator kann man auf ein bestimmtes Element in einer Liste zugreifen?
17. Welchen Index hat das erste Element einer Liste?
18. Was macht die Funktion `concat`?
19. Ergänzen Sie die Signatur der Funktion `anzElem` zur Berechnung der Zahl der Elemente einer Liste ganzer Zahlen um die fehlende Implementierung:

    ```
    anzElem :: [Int] -> Int
    ```

20. Gibt es in Haskell eine vordefinierte Funktion, die diese Aufgabe sogar in größerer Allgemeinheit leistet? Wenn ja, wie heißt sie und in welchem Sinn ist sie 'allgemeiner'?
21. Wie heißen die vordefinierten Funktionen in Haskell, um auf das Kopfelement und den Rest einer Liste ohne das Kopfelement zuzugreifen? Welchen Typ haben diese Funktionen?
22. Wie heißen die vordefinierten Funktionen in Haskell, um auf erste bzw. das zweite Element eines Paars zuzugreifen? Welchen Typ haben diese Funktionen?
23. Gibt es vordefinierte Funktionen, mit denen man auf die Elemente von Tripeln und Quadrupeln zugreifen kann?
24. Beschreiben Sie die Menge der geraden Zahlen von 500 bis 1000 (jeweils einschließlich) mithilfe eines
    (a) Listenaufzählungsausdrucks.
    (b) Listenkomprehensionsausdrucks.
25. Welcher Name wird für den Typ `[Char]` in Haskell auch verwendet?

Teil II, Kapitel 3 'Funktionen'

1. Was ist ein
    (a) bedingter
    (b) bewachter
    (c) Wächter-

    Ausdruck? Geben Sie je ein Beispiel an.
2. Was ist der *stets erfüllte Wächter*(*ausdruck*)?
3. Die Funktion `sumQuadrat` berechnet die Summe der ersten n Quadratzahlen ab 1:

    ```
    sumQuadrat :: Int -> Int
    sumQuadrat n = if n == 1 then 1 else n*n + sumQuadrat (n-1)
    ```

    Schreiben Sie `sumQuadrat` mithilfe
    (a) bewachter Ausdrücke
    (b) von Mustern
    (c) eines `case`-Ausdrucks
    (d) einer anonymen λ-Abstraktion
4. Die Nutzung von Mustern in Funktionsdefinitionen macht Selektorfunktionen unnötig. Was ist damit gemeint? Geben Sie ein Beispiel an, das Ihre Antwort illustriert.
5. Welche Vorteile versprechen musterbasierte Funktionsdefinitionen?
6. Auf welche Weise(n) lassen sich in Funktionsdefinitionen lokale Deklarationen einführen und nutzen? Geben Sie (je) ein illustrierendes Beispiel an.

7. Bewachte Ausdrücke sind syntaktischer Zucker für (i.a.) geschachtelte Fallunterscheidungen. Geben Sie ein illustrierendes Beispiel an.

8. Wozu dient das `';'` in Haskell?

9. Welche Bezeichnungen werden üblicherweise, konventionsgemß für

   (a) Zeichenwerte
   (b) Wahrheitswerte
   (c) Ganze Zahlen
   (d) Werte beliebigen Typs

   und für Listen von

   (a) Zeichenwerten
   (b) Wahrheitswerten
   (c) ganzen Zahlen
   (d) Werten beliebigen Typs

   gewählt?

10. Welche Bezeichnungskonvention gilt dann vermutlich für Listen von Listen von

    (a) Zeichenwerten
    (b) Wahrheitswerten
    (c) ganzen Zahlen
    (d) Werten beliebigen Typs?

11. Gegeben ist (ohne Implementierung) die Signatur der (Editor-) Funktion:

```
type Text = String
type Suchwort = String
type Ersatzwort = String
suche_und_ersetze :: Text -> Suchwort -> Ersatzwort -> Text
```

    und der Funktionsterm:

```
text = "Sehr langer Text, sehr langer Text, sehr langer Text"
suchwort = "sehr"
ersatzwort = "noch viel laengerer"
suche_und_ersetze text suchwort ersatzwort
```

    Klammern Sie die Funktionssignatur und den Funktionsterm vollständig, aber nicht überflüssig.

12. Welche Stelligkeit haben Haskell-Funktionen? Illustrieren Sie Ihre Antwort anhand der Funktion `suche_und_ersetze`.

13. Geben ist die vollständig, aber nicht überflüssig geklammerte Funktionsdefinition:

```
(binom :: (Int -> (Int -> Int)))
((binom n) k) = (div (fac n) ((fac k) * (fac (n-k))))
```

    Nutzen Sie Haskells Klammereinsparungsregeln für Funktionssignaturen und Funktionsterme aus, um so viele Klammern wie möglich wegzulassen, ohne dass sich die Bedeutung ändert.

14. Ist die Funktion `binom` aus der vorigen Aufgabe curryfiziert oder uncurryfiziert deklariert?

15. Geben Sie die jeweils fehlende Deklarationsvariante für `binom` an.

16. Was spricht für eine curryfizierte Deklaration von `binom`? Was ggf für eine uncurryfizierte?

17. Wozu dienen die Funktionen `curry` und `uncurry`?
18. Was bedeutet Infix-, was Prä- und Postfixschreibweise für Operatoren?
19. Sind stets alle drei Schreibweisen fúr einen Operator anwendbar?
20. Nennen Sie eins oder mehrere Beispiele für Operatoren, die üblicherweise als Präfix-, Postfix- oder Infixoperator gebraucht werden.
21. Gibt es in Haskell für jede dieser drei Operatorschreibweisen mindestens je ein Beispiel?
22. Was versteht man in Haskell unter einem Operatorabschnitt?
23. Funktionen sind oft nur partiell definiert, z.B. die Fakultäts- und die Fibonacci-Funktion. Was sollte man bei der Definition partiell definierter Funktionen beachten?
24. Geben Sie je ein Beispiel für eine Funktionsdefinition an, bei der Ihre Hinweise aus der vorigen Aufgabe beachtet worden sind bzw. nicht.
25. Wozu dient die Abseitsregel in Haskell? Was besagt sie?

Teil II, Kapitel 4 'Typsyonyme, Neue Typen, Typklassen'

1. Wozu dienen Typsynonyme? Wofür sind sie hilfreich?
2. Was leisten Typsynonyme nicht?
3. Geben Sie ein je Beispiel an, die Ihre Antworten auf die vorigen beiden Fragen illustrieren.
4. Wozu dienen Neue Typen in Haskell? Wie werden sie deklariert?
5. Neue Typen führen wie Typsynonyme zu höherer Typsicherheit. Richtig oder falsch? Begründen Sie Ihre Antwort.
6. Typ- und Datenkonstruktorname Neuer Typen dürfen übereinstimmen. Richtig oder falsch?
7. Wozu dienen Typklassen in Haskell?
8. Illustrieren Sie am Beispiel der Typklasse `Eq` den grundsätzlichen Aufbau von Typklassen in Haskell.
9. `Eq` ist eine vordefinierte Typklasse in Haskell. Nennen Sie weitere Beispiele vordefinierter Typklassen. Wozu dienen sie?
10. Typklassen in Haskell bilden eine Hierarchie. Was ist damit gemeint?
11. Wie ist die hierarchische Beziehung zwischen den Typklassen, die Sie in der vorvorherigen Aufgabe angegeben haben
12. Typklassen sehen für ihre Elemente oft sehr viele Funktionen vor. Müssen diese Funktionen immer alle vom Programmierer ausgeführt werden, wenn ein Typ zu einem Elemet der Typklasse gemacht werden soll?
13. Machen Sie folgende Typen zu einer Instanz der Typklasse `Eq`:
    (a) `newtype MyBool = MB Bool`
    (b) `newtype MyInt = MI Int`

    Dabei soll Gleichheit von `MyBool`-Werten analog zu der von `Bool`-Werten definiert sind; Gleichheit von `myInt`-Werten jedoch schon gegeben sein, wenn sich zwei Zahlen um höchsten 5 oder weniger unterscheiden; Ungleichheit, wenn sie sich um 6 oder mehr unterscheiden.
14. Wozu dienen `deriving`-Klauseln in Haskell?
15. Welche der Instanzbildungen aus der vorvorigen Aufgabe hätte man mit gleicher Bedeutug auch mit einer `deriving`-Klausel vornehmen könnnen? Begründen Sie Ihre Antwort.

16. Wann lohnt es sich, die in Haskell vordefinierten Typklassen um neue, eigene Typklassen zu erweitern?
17. Geben Sie ein Beipiel für eine selbst definierte Typklasse an, das Ihre Antwort auf die vorige Frage illustriert.
18. Was versteht man unter
    (a) Vererbung
    (b) Überschreiben

    im Zusammenhang mit Typklassen?
19. Illustrieren Sie Ihre Antwort auf die vorige Frage anhand eines oder mehrerer geeigneter Beispiele.
20. Vergleichen Sie Haskells Typklassen mit Klassen in Java oder anderen objektorientierten Sprachen. Welche Unterschiede gibt es?
21. Was verstehen wir unter überladenen Funktionen?
22. Wie lässt sich der Begriff Überladung weiter verfeinern?
23. Geben Sie Beispiele an, an denen diese Verfeinerung des Überladungsbegriffs gezeigt und erläutert werden kann.
24. Woran erkennt man überladene Funktionen?
25. Woran erkennt man die verfeinerten Überladungsbegriffe für Funktionen?

Teil II, Kapitel 5 'Algebraische Datentypdeklarationen'

1. Wie erklärt sich die Bezeichnung *algebraische Datentypdeklarationen*?
2. Warum wird neben dem Konzept Neuer Typen in Haskell mit dem Konzept algebraischer Typen noch eine weitere Möglichkeit geschaffen, selbstdefinierte Datentypen in Haskell-Programmen einzuführen?
3. Nennen Sie einige Beispiele vordefinierter Aufzählungstypen in Haskell.
4. Wann spricht man von einem Aufzählungstyp, wann von einem Produkt-, wann von einem Summentyp?
5. Warum sind die Wahrheitswert `True` und `False` in Haskell großgeschrieben?
6. Definieren Sie einen Aufzählungstyp `Sternzeichen`, dessen Werte die 12 Sternzeichen des Jahreskreises.
7. Geben Sie ein Beispiel für einen Produkttyp an.
8. Geben Sie ein Beispiel für einen Summentyp an.
9. Wie ist eine algebraische Datentypdeklaration aufgebaut? Erklären Sie den Aufbau anhand Ihres Beispiels für einen Summentyp aus der vorigen Aufgabe und benennen Sie die vorkommenden Teile.
10. Datenwertkonstruktoren sind Abbildungen. Was sind ihre Argumente, was ihr Resultat? Benutzen Sie ein Beispiel, um Ihre Antwort zu illustrieren.
11. Definieren Sie einen Typ `Baum`, dessen Werte folgende Eigenschaften haben sollen. Blätter sind mit einer Abbildung auf den ganzen Zahlen benannt; innere Knoten sind mit einer ganzen Zahl und einer Zeichenreihe benannt und haben genau drei Teilbäume.
12. Geben Sie beispielhaft einige konkrete `Baum`-Werte an.
13. Algebraische Datentypen sind strukturierte Datentypen Warum bietet sich für Funktionsdefinitionen auf algebraischen Datentypen besonders die musterbasierte Funktionsschreibweise an? Illustrieren Sie Ihre Antwort auch anhand eines aussagekräftigen Beispieis.

14. Definieren Sie eine Funktion `infix`, die einen `Baum`-Wert in Infixordnung durchläuft (d.h. linker Teilbaum, Wurzel, rechter Teilbaum) und als Resultat die Liste der innere Knoten benennenden ganzen Zahlen in der Reihenfolge des Besuchs dieser Knoten liefert, d.h. eine Zahl kommt um so später in der Liste vor, je später ein Knoten von `infix` besucht wird.

15. Ändern Sie Ihre Funktion aus der vorigen Aufgabe so ab, dass eine Zahl um so früher in der Resultatliste vorkommt, je später der mit ihr benannte innere Knoten besucht wird.

16. Welche Vorteile bietet Haskells Feldsyntax im Zusammenhang mit algebraischen Datentypen efür einen Programmierer?

17. Zeigen Sie diese Vorteile anhand eines geeigneten Beispiels auf.

18. Welche Möglichkeiten bietet Haskell grundsätzlich, um Typdeklarationen für Leser eines Programms möglichst 'sprechend' zu machen?

19. Illustrieren Sie Ihre Antwort auf die vorige Frage anhand eines aussagekräftigen Beispiels, falls erforderlich, mehrerer.

20. Zur Wahl stehen folgende zwei Typvereinbarungen:

```
type Kilogramm = Float
newtype Kilogramm = Kg Float
```

    Was spricht für, was gegen die Vereinbarung von `Kilogramm` als

    (a) Typsynonym?

    (b) Neuer Typ?

21. Wann immer eine

    (a) `type`-Vereinbarung möglich ist, ist auch eine `newtype`-Vereinbarung möglich

        i. und umgekehrt.

        ii. aber nicht umgekehrt.

    (b) `newtype`-Vereinbarung möglich ist, ist auch eine `data`-Vereinbarung möglich

        i. und umgekehrt.

        ii. aber nicht umgekehrt.

    Richtig oder falsch? Begründen Sie jeweils Ihre Antwort.

22. Definieren Sie einen Datentyp `Haustier`, wobei jeder `Haustier`-Wert den Namen, den Lieblingsplatz (Sofa, Fensterbank, Bett, Futternapf, Feld-Wald-und-Wiese) und Hoert-auf (Zuruf, manchmal, gar nicht) Information angibt mittels

    (a) `type`

    (b) `newtype`

    (c) `data`

    und stützen Sie diese Vereinbarungen auf 'sprechende' Typsynonyme.

23. Welche Vor- und Nachteile haben die drei Vereinbarungsvarianten aus der vorigen Aufgabe?

24. Haben Sie für die `data`-Variante der `Haustier`-Deklaration von der Feldsyntax Gebrauch gemacht? Wenn ja, geben Sie eine entsprechende `data`-Deklaration ohne Feldsyntax an; wenn nein, dann mit.

25. Vergleichen Sie die feldsyntaxnutzende und -nichtnutzende `data`-Deklarationen für `Haustier` miteinander. Welche Vor- und Nachteile haben Sie im Vergleich zueinander?

Teil II, Kapitel 6 'Muster und mehr'

1. Wie sieht der Musterausdruck (oder kürzer das Muster)

   (a) der leeren Liste

   (b) einer nichtleeren Liste

   aus?

2. Welche Ausdrücke passen auf folgende Muster(ausdrücke)? Geben Sie einige Beispiele an:

   (a) `[]`

   (b) `(x:[])`

   (c) `(x:xs)`

   (d) `(x:y:[])`

   (e) `(x:y:xs)`

3. Gibt es Muster(ausdrücke), auf die genau Listen folgender Werte passen, die Listen genau dieser Werte beschreiben? Listen mit mindestens einem Element, deren

   (a) erstes

   (b) letztes

   Element eine 1 ist? Geben Sie den entsprechenden Musterausdruck an oder erläutern Sie, warum es keinen entsprechenden Musterausdruck gibt.

4. Schreiben Sie eine musterdefinierte Haskell-Rechenvorschrift `laenge` zur Berechnung der Länge von Listen des Typs:

   ```
   data Liste = Leer | Kopf (String,Int) Liste
   ```

   Geben Sie dabei auch die Signatur der Funktion `laenge` an.

5. Welche Muster(ausdrücke) gibt es für die Beschreibung von

   (a) Wahrheitswerten

   (b) Gleitkommazahlen

   (c) Zeichen

   (d) Tupelwerten

   Welche Werte sind von diesen Mustern jeweils beschrieben?

6. Welche Namens-/Bezeichnungskonventionen gelten für die Muster(ausdrücke) aus der vorigen Aufgabe?

7. Was sind Konstruktormuster? Illustrieren Sie Ihre Antwort auch anhand eines Beispiels.

8. Was sind als-Muster? Wozu sind sie nützlich?

9. Welches Symbol dient zur Bezeichnung von als-Mustern?

10. Geben Sie einige konkrete Beispiele für als-Muster an.

11. Welche Definitionen der ersten 10 Haskell-Beispiele aus Kapitel 1.1.1 machen von Mustern Gebrauch?

    (a) Listen Sie diese Beispiele und die verwendeten Muster auf.

    (b) Beschreiben Sie informell mit Worten, welche Werte von diesen Mustern beschrieben werden, welche Werte auf sie passen?

12. Was haben Muster und Selektorfunktionen miteinander zu tun?

13. Illustrieren Sie den Zusammenhang von Mustern und Selektorfunktionen aussagekräftig anhand eines Beispiels.

14. Was sind i.a. Vorteile musterbasierte Definitionen?

15. Was sind mögliche Nachteile musterbasierter Definitionen, wenn Programme weiterentwickelt werden, wenn sich Parameter von Funktionen ändern, neu hinzukommen oder bisherige wegfallen?

16. Welcher Schreibweise und welchem Bildungsprinzip aus der Mathematik ist Listenkomprehension nachgebildet?

17. Beschreiben Sie als Listenkomprehensionen: Die Liste der

    (a) geraden ganzen Zahlen.

    (b) geraden ganzen Zahlen, die eine 2er-Potenz sind.

    (c) Nachfolger der geraden ganzen Zahlen, die eine 2er-Potenz sind.

    (d) Quadrate der Nachfolger der geraden ganzen Zahlen, die eine 2er-Potenz sind.

18. Was versteht man im Zusammenhang mit Listenkomprehension unter einem

    (a) Generator

    (b) Transformator

    (c) Filter

    (d) Test

    Illustrieren Sie Ihre Antworten anhand aussagekräftiger Beispiele.

19. Sind

    (a) `(:)`

    (b) `(++)`

    Listenkonstruktoren oder Listenoperatoren?

20. Was unterscheidet Listenkonstruktoren und Listenoperatoren?

21. Listenkonstruktoren und Listenoperatoren dürfen in gleicher Weise in Mustern verwendet werden. Richtig oder falsch? Begründen Sie Ihre Antwort.

22. Wie werden das

    (a) Variablenmuster

    (b) Jokermuster

    in Haskell geschrieben?

23. Welche Werte eines Typs beschreiben das

    (a) Variablenmuster?

    (b) Jokermuster?

24. Wie unterscheiden sich Variablen- und Jokermuster in der Verwendbarkeit?

25. Warum soll man sich für das Joker- und nicht das Variablenmuster entscheiden, wenn beide anwendbar sind?

Präludium, Teil I

1. Gegeben ist die Funktion:

   ```
   f :: Int -> Int
   f n = if n == 1 then 1 else (f (n-1)) + n^2
   ```

   (a) Welchen Wert hat der Aufruf `f 3`?

   (b) Wie verhält sich `f` beim Aufruf mit dem Wert (-3)?

   (c) Was berechnet `f` für Argumentwerte größer oder gleich null?

   (d) Wie verhält sich `f` für Aufrufe mit echt negativen Argumentwerten?

   (e) Werten Sie den Aufruf `f 2` Schritt für Schritt aus. Ein Schritt ist eine Expansion, eine Operatorauswertung oder die Wahl des Ausdrucks im dann- bzw. sonst-Fall der Fallunterscheidung nach Auswertung deren Bedingung zu wahr bzw. falsch. Die Reihenfolge der Rechenschritte ist nicht vorgegeben.

   (f) Wenden Sie eine andere Reihenfolge der Rechenschritte als in der vorigen Aufgabe an, um den Wert des Ausdrucks `f 2` zu berechnen.

2. Was passiert bei der Ausführung eines

   (a) imperativen

   (b) funktionalen

   Programms?

3. Was sind wichtige Kennzeichen imperativer/objektorientierter und funktionaler Programmierung?

4. Wie alt ungefähr ist der *Algorithmus von Euklid*?

5. Wofür sind die Werkzeuge `Hoogle` und `Hayoo` nützlich?

6. Wonach, nach wem ist die Programmiersprache Haskell benannt?

7. In präskriptiven Sprachen steht das 'wie' im Vordergrund, in deklarativen Sprachen das 'was'. Was ist damit gemeint?

8. Welche große Gruppen gibt es innerhalb der Familien

   (a) präskriptiver

   (b) deklarativer

   Sprachen? Was sind typische Beispiele von Sprachen dieser Gruppen?

9. Das Haskell-Vokabular, die Zahl seiner Schlüsselwörter ist sehr gering. Richtig oder falsch? Wieviele ungefähr?

10. Wie sind Identifikatoren in Haskell aufgebaut?

11. Was ist der sog. *Standard Prelude* von Haskell? Was beinhaltet er?

12. In seinem Artikel *Haskell: A Language for Modern Times* stellt Mihai Maruseac fest: *"Learning Haskell opens one's mind [...], which might produce clearer and shorter implementations."* Wie gelangt er zu dieser Einschätzung?

13. Welche Sprachkonstrukture sind für imperative, objektorientierte Sprachen zentral, unverzichtbar, fehlen aber in funktionalen Sprachen?

14. Welche Herausforderung muss man beim Übergang von objektorientierter zu funktionaler Programmierung meistern?

15. Warum (vielleicht) empfinden anfänglich viele den Übergang von imperativer/objektorientierter zu funktionaler Programmierung als 'Kulturschock'?