Prof. Dr. Manfred Schmidt-Schauß
M.Sc. Nils Dallmeyer

Goethe-Universität Frankfurt am Main/ Institut für Informatik
Künstliche Intelligenz/Softwaretechnologie

Grundlagen der Programmierung 2

Sommersemester 2018

Aufgabenblatt Nr. 4

Abgabe: Mittwoch 9. Mai 2018 **vor!** der Vorlesung

Aufgabe 1 (40 Punkte)

Faith Connors genießt die Aussicht und frische Luft auf einem Baukran des im Bau befindlichen Anansi Towers der Stadt Glass. In einer Stadt der völligen technischen Überwachung ist es ihre Aufgabe als sogenannte Runnerin, Pakete und Informationen unbemerkt zu überliefern. Dies geschieht meistens über Wege auf den Dächern, die nur teilweise von Sicherheitspersonal überwacht werden.

a) Selbst wenn Faith aktuell im Anansi Tower noch Einiges zu erledigen hat, plant sie schon die nächsten Laufwege durch Glass. Dabei plant sie nur grob nach Stadtvierteln und muss in zeitlich beliebiger Reihenfolge die folgenden nummerierten Stadtviertel besuchen: Zephyr Transithub (1), Charter Hill (2), Centurian Yards (3), Concord Plaza (4), Eden Village (5), Crystal Valley (6), Shimmering Heights (7). Dabei hat sie bereits einige Erfahrungen gesammelt:

   - Der Weg von *Charter Hill* über *Centurian Yards* zum *Zephyr Transithub* ist ein sehr sicherer Weg, weshalb er genau so benutzt werden muss. Da der *Zephyr Transithub* außerdem ein guter Rückzugsort ist, sucht sie den *Zephyr Transithub* immer erst zum Schluss auf.
   - Ab 15:00 Uhr trifft man in den *Shimmering Heights* auf viel Sicherheitspersonal, weshalb die *Shimmering Heights* zuerst besucht werden müssen. Da die *Shimmering Heights* recht abgelegen sind, beeinträchtigt dies die anderen Wege nicht (kein guter Weg führt hindurch).
   - Faith findet den Weg vom *Eden Village* ins *Concord Plaza* sehr einfach, weshalb sie diesen Weg immer benutzt.
   - Vor einer Woche hat sich Faith verrechnet und ist den folgenden Weg gelaufen: *Shimmering Heights, Eden Village, Concord Plaza, Crystal Valley, Charter Hill, Centurian Yards, Zephyr Transithub*. Allerdings ist es ein großer Umweg, von den *Shimmering Heights* nach *Eden Village* zu laufen, da das *Crystal Valley* bereits auf dem Weg liegt. Daher möchte sie den genannten Weg auf keinen Fall nutzen. Ansonsten interessiert sie sich aber nicht für Umwege, das heißt es sind keine weiteren Bedingungen bezüglich Umwegen erforderlich.

   Faith hat den `ghci` auf Ihrem Smartphone installiert und weiß, dass sie die Anzahl der möglichen Lösungen mit einer List Comprehension sehr gut einschränken kann, wenn sie die oben definierten Nummern als Kodierung für die Stadtviertel verwendet. Leider kann Sie sich nicht mehr genau erinnern, wie man so etwas programmiert und kam bisher nur auf den folgenden unvollständigen Ausdruck, bei dem jedes der oben genannten Stadtviertel genau einmal im 7-Tupel vorkommen muss und nur durchquerte Viertel nicht Teil des 7-Tupels sind (z.B. von `a` nach `b` könnte sie theoretisch durch die ganze Stadt laufen und dabei mehrere andere Viertel durchqueren):

```
[(a,b,c,d,e,f,g) | a <- [1..7], ]
```

   Helfen Sie Faith, indem Sie die List Comprehension vervollständigen. In welcher Reihenfolge sollte sie die Stadtviertel unter Berücksichtigung ihrer Erfahrungen besuchen? Wie viele mögliche Reihenfolgen gibt es? (15 Punkte)

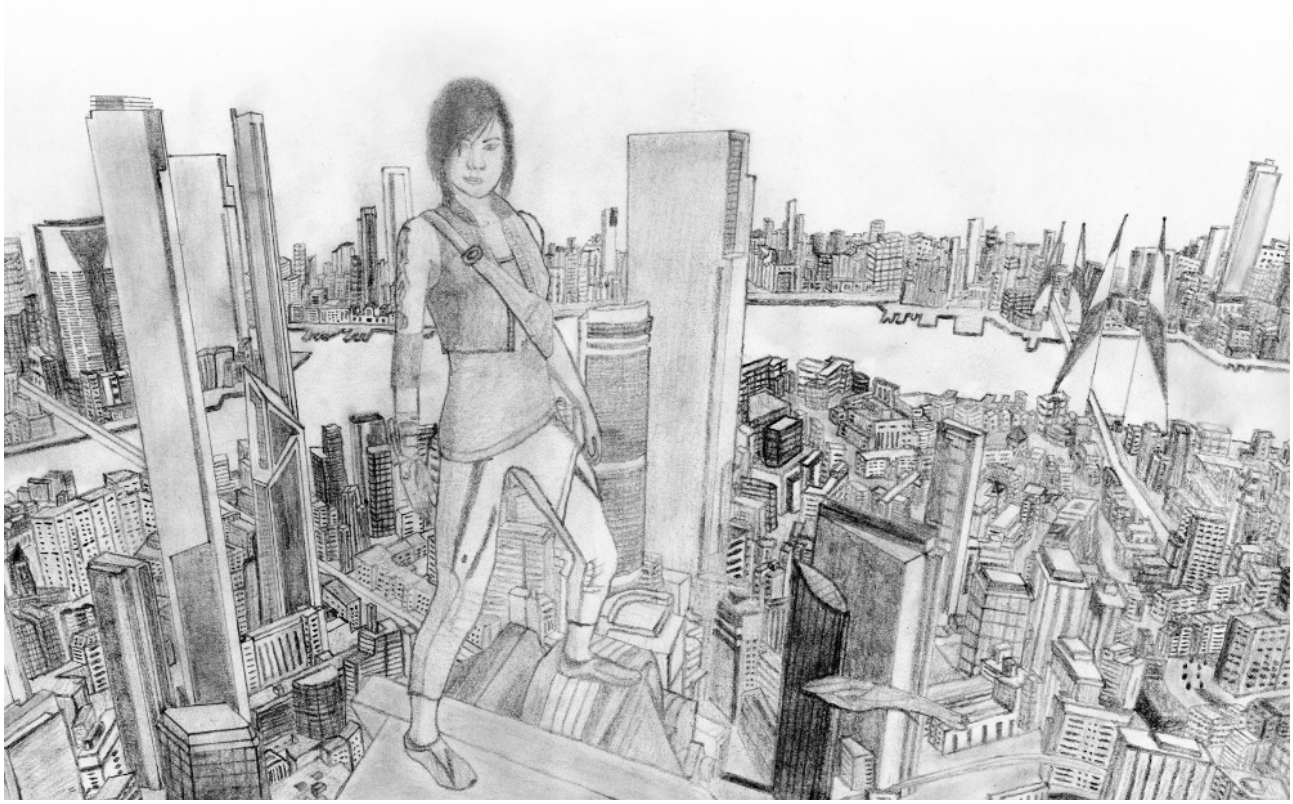
b) Für jede Lieferung merkt sich Faith, wie viel Geld in Euro sie für die Lieferung bekommt und in welches Stadtviertel die Lieferung gebracht werden muss. Die zu liefernden Objekte können zerbrechlich sein und haben ein Gewicht in Gramm und Abmessungen in Zentimeter. Also lässt sich die Tragetasche von Faith wie folgt unter Verwendung der Datentypen `Tragetasche`[1], `Lieferung`, `Objekt` und der Typsynonyme `Bezahlung`, `Zielort`, `Zerbrechlich`, `Gewicht`, `Abmessungen` in Haskell darstellen:

```
data Tragetasche = Tragetasche [Lieferung]
  deriving(Eq,Show)
data Lieferung = Lieferung Objekt Bezahlung Zielort
  deriving(Eq,Show)
data Objekt = Objekt Zerbrechlich Gewicht Abmessungen
  deriving(Eq,Show)
type Bezahlung = Float
type Zielort = String
type Zerbrechlich = Bool
type Gewicht = Float
type Abmessungen = (Float,Float,Float)
```

b1) Faith muss zwei Objekte für je 100 Euro nach *Eden Village* liefern. Das erste Objekt ist zerbrechlich, 50x50x50cm groß und wiegt 30g, das zweite Objekt ist nicht zerbrechlich und hat die Abmessungen 229x324x5mm und ein Gewicht von 15g. Geben Sie Faith' Tragetasche unter Verwendung der obigen Datentypen an. (5 Punkte)

b2) Implementieren Sie eine Funktion `gesamtGewicht :: Tragetasche -> Float`, die das Gesamtgewicht von Faith' Lieferungen berechnet. (10 Punkte)

b3) Faith hat ein zerbrechliches Objekt nicht markiert, weiß aber noch die Abmessungen. Geben Sie eine Funktion `aendereZerbrechlich :: Tragetasche -> Abmessungen -> Tragetasche` an, welche alle Objekte mit den übergebenen Abmessungen als zerbrechlich markiert. (10 Punkte)



Nils Dallmeyer, 2018

[1] `deriving(Eq,Show)` ermöglicht, dass Objekte des entsprechenden Datentyps angezeigt und verglichen werden können.

Aufgabe 2 (40 Punkte)

Geben Sie jeweils *List Comprehensions* in Haskell an, welche die folgenden Listen erzeugen.

a) Die unendliche Liste aller Zahlen $n \in \mathbb{N}_{>0}$, die ungerade sind, durch 13 mit Rest 3 teilbar sind und geteilt durch 20 den Rest 5 ergeben, d.h. `[185,445,705,965,1225,1485,1745,2005,...`

Die Funktion `take` kann beim Testen nützlich sein. (4 Punkte)

b) Aus einer gegebenen Liste `xs` von Paaren, wobei beide Paarkomponenten vom gleichen Typ sind, soll eine Liste von Elementen berechnet werden, in der die Paarkomponenten nacheinander eingefügt sind. D.h. für `[(1,2),(3,4),(5,6)]` soll die Liste `[1,2,3,4,5,6]` berechnet werden.

Tipp: Gesucht ist eine Funktion mit Parameter `xs`, deren Rumpf eine List Comprehension ist. (6 Punkte)

c) Die endliche Liste aller Tripel (a, b, c) für alle $a \in \{0, 3, 20, 25, 37, 97\}$, $b \in \{\texttt{True}, \texttt{False}\}$ und $c \in \{\texttt{'a'},\texttt{'b'},\texttt{'c'},\texttt{'d'}\}$ und zwar in der Reihenfolge, sodass zunächst die Buchstaben wechseln, danach die Zahlen und zuletzt die Wahrheitswerte, d.h. die Ausgabe beginnt mit:

```
[(0,True,'a'),(0,True,'b'),(0,True,'c'),(0,True,'d'),(3,True,'a'),(3,True,'b'),
 (3,True,'c'),(3,True,'d'),(20,True,'a'),(20,True,'b'),(20,True,'c'),(20,True,'d')...
```

(8 Punkte)

d) Die Liste aller Tripel (a, b, c), $a, b \in \{1..4\}$, wobei $c = a + b$ oder $c = a - b$ gilt und außerdem c gerade ist. Dabei wechseln die Zahlen bei b zuerst und danach die Zahlen bei a. Außerdem sollen zuerst alle Tripel mit $c = a+b$ und danach alle Tripel mit $c = a-b$ in der Liste vorkommen. Das heißt die Ausgabe beginnt mit `[(1,1,2),(1,3,4),(2,2,4),(2,4,6),(3,1,4),(3,3,6)...`

Tipp: Im Prädikat einer List Comprehension können anonyme Funktionen verwendet werden. (12 Punkte)

e) Die unendliche Liste `xs` aller 4-Tupel (a, b, c, d) mit $a, b, c, d \in \mathbb{N}$, so dass die Tupel in einer *fairen* Reihenfolge erzeugt werden. *Fair* bedeutet hierbei, dass `(a,b,c,d) ‘elem‘ xs` für jedes Tupel von Zahlen `(a,b,c,d)` nach *endlicher Zeit* terminiert. (10 Punkte)

Aufgabe 3 (20 Punkte)

Berechnen (Rechenweg erforderlich!) Sie die Menge der *freien Variablen* $FV(s_i)$ und der *gebundenen Variablen* $GV(s_i)$ für die folgenden Ausdrücke s_1 und s_2: (jeweils 10 Punkte)

a) $s_1 := \texttt{\textbackslash}y \texttt{ -> if } (x\ y) \texttt{ then } ((\texttt{\textbackslash}w \texttt{ -> } w)\ y) \texttt{ else } ((\texttt{\textbackslash}z \texttt{ -> } z)\ z)$

b) $s_2 := \texttt{let } g\ x = (\texttt{\textbackslash}x \texttt{ -> } \texttt{\textbackslash}y \texttt{ -> } (g\ y)) \texttt{ in let } w = g\ u\ (g\ y), z = w \texttt{ in } g$