Prof. Dr. Manfred Schmidt-Schauß
Institut für Informatik

Fachbereich Informatik und Mathematik/ Institut für Informatik
Johann Wolfgang Goethe-Universität Frankfurt am Main

Einführung in die funktionale Programmierung

Wintersemester 2022/2023

Aufgabenblatt Nr. 4

Abgabe: bis 12.12.2022 online, oder am 13.12.2022 in der Vorlesung.

Aufgabe 1 (20 Punkte)

Folgendes Wahrscheinlichkeits-Experiment soll in KFPTSprob programmiert werden:
Werfe einen fairen 6-er Würfel.
Wenn eine der Zahlen 2,3,4,5 gewürfelt wird, dann ist das das Ergebnis.
Wenn eine 1 gewürfelt wird, dann soll das Ergebnis Nichtterminierung sein.
Wenn die 6 gewürfelt wird, dann würfle erneut.

a) Programmiere die Funktion `wuerfel x = let wert = ...`

b) Warum benötigt `wuerfel` ein Argument, das nicht benutzt wird?

c) Wie ist die Wahrscheinlichkeit der Terminierung bzw. Nichtterminierung?

d) Was ist die Wahrscheinlichkeit, dass das Experiment mit einer gewürfelten 3 endet?

Aufgabe 2 (30 Punkte)

Ziel dieser Aufgabe ist es eine einfache Version von „Conways Spiel des Lebens“[1] [2] in Haskell zu implementieren. **Ein Teil des Quellcodes ist bereits auf der Webseite der Veranstaltung zu finden.**

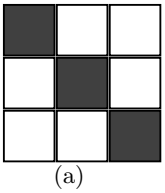
Das Spielfeld wird dargestellt durch *quadratische Matrizen*, deren Einträge entweder belegt (bevölkert), oder nicht belegt sind (ein Beispiel ist in Abbildung (a) zu sehen). Ein Eintrag an Position (i, j) in einer Matrix wird durch seine Zeile i und sein Spalte j bezeichnet, wobei die Zählung der Zeilen und Spalten bei 0 beginnt (wie für ein Beispiel in Abbildung (b) zu sehen ist). Solche Matrizen lassen sich in Haskell durch die folgende Datenstruktur darstellen:

```
data Matrix = Matrix [[Bool]] deriving(Eq)
```

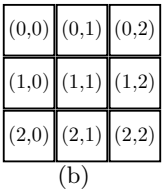
Ist in der i-ten Liste der `Matrix` der j-te Eintrag `True`, so gilt der Eintrag (i, j) der Matrix als belegt. Beispielsweise entspricht

```
Matrix [[True,False,False],[False,True,False],[False,False,True]]
```

der Matrix aus Abbildung (a).



(a)



(b)

Um solche Matrizen gut lesbar anzuzeigen, kann man folgende `Show`-Instanz verwenden:

[1] `http://de.wikipedia.org/wiki/Conways_Spiel_des_Lebens`

[2] John H. Conway is vor kurzem, am 11. April 2020, an Komplikationen durch Covid-19 gestorben

```
instance Show Matrix where
    show (Matrix m) =  unlines $ map showRow m
        where showRow = map (\x -> if x then '#' else ' ')
```

Machen Sie sich mit dem Programmcode auf der Webseite vertraut:

a) Eine Haskell-Funktion des Typs: `Matrix -> (Int, Int) -> Bool`, die eine Matrix und eine Matrix-Position als Eingabe erwartet und die Belegung des durch die Koordinate gegebenen Matrix-Eintrags berechnet.

b) Eine Haskell- Funktion `nachbarn :: (Int, Int) -> Int -> [(Int, Int)]` bestehend aus einer *List-Comprehension*, die eine Matrix-Position und die Dimension einer Matrix erwartet und die Positionen aller benachbarten Felder berechnet.
Beispielsweise liefert `nachbarn (1,1) 3` als Resultat die Liste `[(0,0),(0,1),(0,2),(1,0),(1,2),(2,0),(2,1),(2,2)]`.

c) Eine Haskell-Funktion `anzahlBelegterNachbarn :: (Int, Int) -> Matrix -> Int`, die in einer gegebenen Matrix zu einem durch seine Position gegebenen Feld die Anzahl seiner belegten Nachbarn berechnet. Beispielsweise liefert der Aufruf von `anzahlBelegterNachbarn` mit der Koordinate `(1,1)` und der Matrix aus Abbildung (a), als Resultat `2`.

Implementieren und testen Sie mit sinnvollen Beispielen:

a) (10 Punkte) Implementieren Sie in Haskell eine Funktion `zugGOL :: Matrix -> Matrix`, die den Nachfolger einer Matrix gemäß der Spielregeln des Spiel des Lebens berechnet:

   – Für eine Matrix-Zelle, die belegt ist:
     * Jede Zelle mit einem oder keinem Nachbarn stirbt.
     * Jede Zelle mit vier oder mehr Nachbarn stirbt.
     * Jede Zelle mit zwei oder drei Nachbarn überlebt.
   – Für eine Matrix-Zelle, die leer ist:
     * Jede Zelle mit drei Nachbarn wird belegt.

   Hier bedeutet „stirbt“: nicht belegt in der Matrix (`False`); und „überlebt“: belegt in der Matrix (`True`).

   Ruft man `zugGOL` auf dem Spielfeld von Abbildung (a) auf, soll das Resultat `Matrix [[False,False,False],[False,True,False],[False,False,False]]` sein.

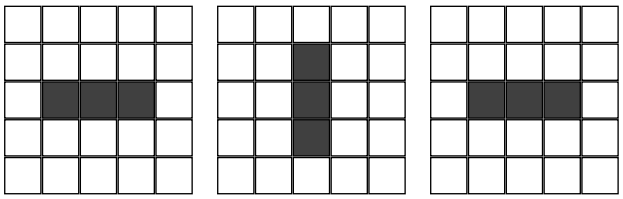
b) (15 Punkte) Implementieren Sie eine Funktion `zuege :: Matrix -> Int -> [Matrix]`, die zu einer Anfangsmatrix M eine Liste der Folgematrizen berechnet, die durch i aufeinanderfolgende Spielzüge im Spiel des Lebens aus M entstehen.

   Wenn `M` das Spielfeld aus Abbildung (a) ist, sollte `zuege M 2` als Resultat ergeben:

```
[Matrix [[False,False,False],[False,True,False],[False,False,False]],
 Matrix [[False,False,False],[False,False,False],[False,False,False]]]
```

c) (10 Punkte) Implementieren Sie eine Funktion `oszilliert :: Matrix -> Bool`, die zu einer gegebenen Matrix testet, ob der durch sie beschriebene Spielzustand oszilliert. Ein Spielzustand M *oszilliert*, wenn M nach i-Spielzügen ($i \geq 2$) wieder erreicht ist.

   Im Beispiel unten ist ein oszillierender Zustand zu sehen: Beginnend bei Matrix (c) ist nach 2 Zügen im Spiel des Lebens wieder der Startzustand erreicht.



(c) (d) (e)