Sven Eric Panitz

Lehrbriefe Funktionale Programmierung

Kreuzworträtsel

Kreuzworträtsel

Sven Eric Panitz

9. März 2025

Inhaltsverzeichnis

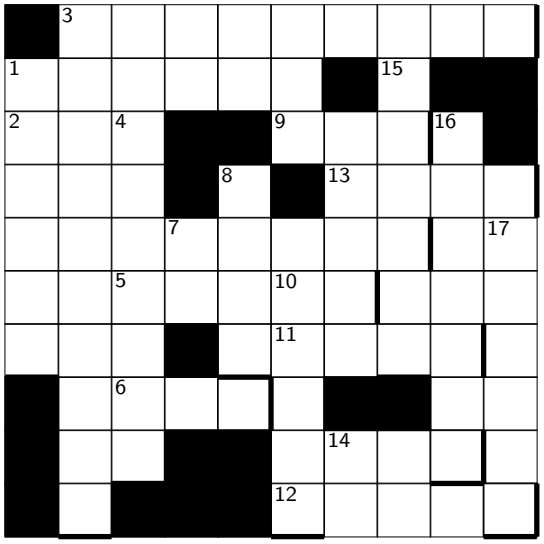
1 Kreuzworträtsel

In dieser Aufgabe soll ein kleines Programm entwickelt werden, dass für eine Liste von Fragen und Antwotren ein einfaches Kreuzworträtsel generieren kann. Bei einem Kreuzworträtsel sind Wörter in einem Gitter einzutragen, so dass in den Zellen des Gitters jeweils ein Buchstaben steht. Bei einem einfachen Kreuzworträtsel sind die Fragen nummeriert und das Startfeld für die Antwort ist mit der entsprechenden Nummer markiert. Diese einfachen oder gewöhnlichen Kreuzwotrrätsel werden auch als *Deutsche Kreuzworträtsel* bezeichnet im Gegensatz zu *Schwedenrätseln*, bei denen die Fragen direkt im Rätselgitter notiert sind.

Manche Felder n einfachen Kreuzworträtseln enthalten in der Lösung keine Buchstaben und werden deshalb schwarz ausgefüllt gedruckt.

In einfachen Kreuzworträtseln ist es mitunter notwendig Wortenden durch eine dickere Gitterlinie zu markieren.

Ein Beispiel für ein einfaches Kreuzworträtsel, dass das hier erstellte Programm generiert hat, findet sich in Abbildung 1.



Horizontal

1: Oder doch lieber tun?
3: Kann auch schwere Arbeit sein, so zu konstruieren.
5: Begrüßung
6: Persönliches Fürwort
7: Da tappen manche falsch herum im Nebel
9: Pronomen
11: Heizvorrichtug
12: Ganz Großartig
13: Nicht unbedingt der Preis
14: Weiblicher Artikel

Vertikal

2: Fährt unter dem Pflaster
3: In einem Land lebend
4: Zu beachten!
8: Wo wir alle leben
10: Örtchen
15: Limes
16: Nahestehende doch nicht verwandr
17: Antikes Volk und moderne Städter

Abbildung 1: Ein mit dem Zielprogramm generiertes Kreuzworträtsel

2 Kreuzworträtsel in Haskell

Wir definieren das Modul für Kreuzworträtsel:

Cross.lhs

```
> module Cross where
```

Wir verwenden für die Darstellung eines Kreuzworträtsels eine Liste von Listen. Somit werden wir auf einige Listenfunktionen zurückgreifen und importieren das entsprechende Modul.

Cross.lhs

```
> import Data.List
```

2.1 Der Datentyp für Kreuzworträtsel

Bei einem Kreuzworträtsel sind Wörter horizontal von links nach rechts und vertikal von oben nach unten einzutragen. Hierfür benötigen wir einen Aufzählungstypen:

Cross.lhs

```
> data Direction = Horizontal|Vertical deriving (Eq,Show)
```

Eine kleine Hilfsfunktion soll es ermöglichen zwischen den zwei Werten umzuswitchen.

Cross.lhs

```
> changeDirection Horizontal = Vertical
> changeDirection Vertical = Horizontal
```

Kommen wir zum eigentlichen Datentyp für das Kreuzworträtsel. In einem Gitter werden Wörter angeordnet. Die ersten beiden Zahlenwerten geben die Zeilen- und Spaltenanzahl des Rätsels an. Es folgt eine Liste der Fragen. Und schließlich kommen die einzelnen Felder der Rätsels. Diese sind ein einer Liste von Zeilen dargestellt. Die Elemente der Felder sind von einem Wert des Typs `Maybe`. Dabei signalisiert der Wert `Nothing`, dass in diesem Feld keine Eintrag vorzunehmen ist.

Cross.lhs

```
> data Grid a = Grid Int Int [Q a][[Maybe (Maybe Int,Border,a)]]
>   deriving (Eq,Show)
```

Auch wenn es fraglich ist, ob wir jemals Rätsel mit etwas anderem als Buchstaben als Einträge in den einzelnen Zellen programmieren wollen, ist die Defiintion noch polymorph über den Typ der Zelleneinträge. Im Rahmen dieser Aufgabe wird der Typ `a` aber immer `Char` sein.

Es wird in der Defiintion ein weiterer Aufzählungsdatentyp verwendet. `Border` soll angeben, ob und wo eine Zelle eine dickere Linie für die Markierung eines Wortendes benötigt.

Es gibt in dieser Aufzählung vier Werte. In der Zelle ender kein Wort, es endet ein horizontales Wort, es endet ein vertikales Wort und schließlich der Fall, in dem sowohl vertikal als auch horizontal ein Wort endet.

Cross.lhs

```
> data Border = None|Right|Bottom|RightBottom
>   deriving (Eq,Show)
```

Die Einzelnen Fragen des Rätsels haben eine Nummer, eine Position im Gitter, die Richtung, die eigentliche Frage und schließlich die einzutragende Antwort.

Cross.lhs

```
> data Q a = Q Int (Int,Int) Direction String [a]
>   deriving (Eq,Show)
```

Für eine schnelle kurze Testausgabe sei eine simple Funktion zur Anzeige des Gitters eines Kreuzworträtsels definiert:

Cross.lhs

```
> --instance Show a => Show (Grid a) where
> showMe (Grid _ _ ws xss) =
>  show ws ++"\n\n"
>    ++(concat
>      $intersperse "\n"
>      $map (\xs-> concat
>                  $map (\x->maybe "' '" (\(_,_,c)->show c) x) xs) xss)
```

Alternativ könnte man diese Funktion auch als Instanz der Typklasse `Show` definieren. Wir haben darauf verzichtet, um mit der generierten Instanz von `Show` direkt Haskell-Quelltext zu bekommen, den wir für das Schreiben der Testfälle verwendet haben.

Ziel soll es nun sein, mit einem einfachen Aufruf, ein Kreuzworträtsel zu generieren. Hierfür wird die Funktion `create` vrogesehen, die quadratische Rätsel für eine Liste von Frage-Antworten-Paare erzeugt.

Cross.lhs

```
> create :: Eq a => Int -> [(String,[a])] -> [Grid a]
> create w (x:xs)
>   = map addNumbering
>     $foldl (\rs y -> (selectBest$concat$map (insertWord y) rs))
>     [(firstWord x (newGrid w))] xs
```

Die einzelnen Funktionen sind dieser Aufgabe umzusetzen. Dieses sind im einzelnen:

- `newGrid` zum Erzeugen eines leeren quadratischen Rätselgitters.
- `firstWord` zum Einfügen eines ersten Wortes zentriert in einem leeren Rätsel.
- `insertWord` zum Einfügen eines weiteren Wortes in ein Rätsel.
- `selectBest` zum Selektieren der Rätsel, mit den meisten durch mehrere Wörter geteilten Buchstaben.
- `addNumbering` zum Durchnummerieren der Fragen im Rätsel und dem Hinzufügen der dickeren Trennbalken bei Wortenden.

Hier der Beispielaufruf der Funktion `create` zum Erzeugen des Kreuzworträtsels aus Abbildung 1.

Cross.lhs

```
> ex = create  10
>   [("Begrüßung","HALLO")
>   ,("Wo wir alle leben","WELT")
>   ,("Örtchen","LOKUS")
>   ,("Heizvorrichtug","OFEN")
>   ,("Zu beachten!","WICHTIG")
>   ,("Ganz Großartig","SUPER")
>   ,("Nahestehende doch nicht verwandr","FREUNDE")
>   ,("Antikes Volk und moderne Städter","ROEMER")
>   ,("Nicht unbedingt der Preis","WERT")
>   ,("In einem Land lebend","LANDSLEUTE")
>   ,("Kann auch schwere Arbeit sein, so zu konstruieren.","LEICHTBAU")
>   ,("Limes","GRENZE")
>   ,("Persönliches Fürwort","ICH")
>   ,("Pronomen","IHR")
>   ,("Da tappen manche falsch herum im Nebel","LEBEN")
>   ,("Oder doch lieber tun?","LASSEN")
>   ,("Fährt unter dem Pflaster","UBAHN")
>   ,("Weiblicher Artikel","DIE")]
```

2.2 Instanz von ToLaTeX

Die Kreuzworträtsel sollen schließlich in einem Druckdokument angezeigt werden. Hierfür bietet sich LaTeXan. Der Vorteil an LaTeXist, dass es nur eine Textgrundlage benötigt, um sauber gesetzte Dokumente zu erzeugen. Somit lassen sich ohne Zusatzbibliotheken auf einfache Weise Dokumente in jeder Programmiersprache druckbare Dokumente generieren. Für LaTeXgibt es ein Paket, das den Satz von unterschiedlichen Kreuzworträtseln ermöglicht [Neu20].

Dadurch, dass wir die Rätsel als LaTeXausgeben können, lässt sich mit einen einfachen Aufruf, ein generiertes Rästel im LaTeX-Format in eine Datei schreiben:

Cross

```
writeFile "ex.tex"$toLaTeX $head ex
```

Da mehrere Typdefinitionen in die Definition eines Rätsels definiert sind, und diese Teile alle eine Darstellung in LaTeXbnötigen, bietet sich an, eine Typklasse `ToLaTeX` zu definieren. Dann ist es möglich, eine Funktion `toLaTeX` für diese unterschiedlichen Teiltypen zu überladen.

Hier die notwendige Typklasse:

Cross.lhs

```
> class ToLaTeX a where
>   toLaTeX :: a -> String
```

Entsprechend des Pakets [Neu20] zum Setzen von Kreuzowrträtseln, werden die Trennbalkeninformationen in LaTeXgesetzt.

Cross.lhs

```
> instance ToLaTeX Border where
>   toLaTeX None = ""
>   toLaTeX Cross.Right = "[r]"
>   toLaTeX Bottom = "[b]"
>   toLaTeX RightBottom = "[rb]"
```

Das Erzeugen des LaTeX-Codes für ein Rätsel ist wenig inspirierender Haskell-Code. Er folgt hier weitgehend unkommentiert.

Cross.lhs

```
> instance Show a => ToLaTeX (Grid a) where
>  toLaTeX (Grid w h ws gss) =
```

```
>    "\\begin{Puzzle}{"++show w++"}{"++show h++"}% \n"
>    ++latexGrid gss++"\\end{Puzzle}\n\n"
>    ++"\\paragraph*{Horizontal}~\\\\\n"
>    ++(concat $map showQuestion hors)
>    ++"\\paragraph*{Vertikal}~\\\\\n"
>    ++(concat $map showQuestion vers)
>     where
>        latexGrid  = concat.(map latexRow)
>        latexRow rs = "|"++
>          (intercalate "|"
>          $map (maybe "*"
>                 (\(a,b,c) -> ((maybe "[]" (\x->"["++show x++"]") a)
>                        ++toLaTeX b++[(show c!!1)]))) rs)
>           ++"|.\\\\\n"
>
>        (ver,hor) = partition (\(Q _ _ d _ _)->d==Vertical) ws
>        hors = sortOn (\(Q nr _ _ _ _)->nr) hor
>        vers = sortOn (\(Q nr _ _ _ _)->nr) ver
>        showQuestion (Q nr _ _  q _)
>          = "{\\bfseries "++show nr++": }"++q++"\\\\\n"
```

3 Aufgaben

Implementieren Sie nun die fehlenden Funktionen für die Generierung von Kreuzworträtseln. Bis einschließlich der Funktion `insertWordInLine` sind Unit-Tests hinterlegt. Für die weiteren Funktionen wurde darauf verzichtet, weil sie unnötig alternative Lösungen einschränken würden. Auch kann man sich in den Folgeaufgaben durch die Ausgabe des LaTeX-Druckbildes von der Güte der Lösung überzeugen.

Aufgabe 1

a) Die erste Funktion, die zur Lösung benötigt wird, beschäftigt sich nur mit Listen. Es soll ein Element in eine Liste an eine bestimmte Position ersetzt werden durch ein anderes Element. Das erste Element der Liste habe dabei den Index 0. Ist der Index kein gültiger Index der Liste, so findet keine Änderung für die Ergebnisliste statt.

Cross.lhs

```
> replaceAt :: (Eq t1, Num t1) => t1 -> t2 -> [t2] -> [t2]
> replaceAt i y xs = []
```

b) Auch diese Aufgabe beschäftigt sich noch nicht mit dem Rätsel, sondern allgemein mit Listen. Es sollen Elemente, die mit einer übergebenen Vergleichsfunktion als gleich ausgewertet werden, zusammen gefasst werden. Das Ergebnis ist eine Liste von Listen. Die inneren Listen haben Elemente, die nach der übergebenen Funktion gleich sind.

Cross.lhs

```
> clusterBy :: (t -> t -> Bool) -> [t] -> [[t]]
> clusterBy eq xs = []
```

Ein Beispielaufruf für Strings gruppiert diese in die Buchstaben des Wortes:

Cross

```
*Cross> clusterBy (==) "mississippi"
["m","iiii","ssss","pp"]
```

Beachten Sie, dass diese Funktion nicht diesselbe ist wie die Standardfunktion `groupBy` des Listenmoduls. Dort werden nur direkt hintereinander in der Liste auftauchende Element bei Gleichheit gruppiert:

Cross

```
Cross> groupBy (==) "mississippi"
["m","i","ss","i","ss","i","pp","i"]
```

c) Schreiben Sie jetzt die erste Funktion für Kreuzworträtsel. Es soll ein initiales leeres Rätselgitter in quadratischer Form erstellt werden. Die Anzahl der Zeilen und Spalten wird als Argument übergeben. Die Fragenliste des Rätsels ist noch leer. Alle Einträge in der verschachtelten Liste sind noch mit dem Wert `Nothing` belegt.

Cross.lhs

```
> newGrid :: Int -> Grid a
> newGrid _ = Grid 0 0 [][]
```

d) In dieser Aufgabe sollen Sie in ein leeres Rätsel ein erstes Wort möglichst zentriert in horizontaler Weise einfügen. Da angenommen wird, dass noch keinerlei Wörter in das Rätsel eingetragen sind, sind keine beschränkungen auf bereits eingetragene Wörter zu befürchten und das Wort kann frei platziert werden.

Cross.lhs

```
> firstWord :: (String,[a]) -> Grid a -> Grid a
> firstWord _ g = g
```

Folgender Testaufruf zeigt, wie das erste Wort in das Rätselgitter eingefügt werden soll:

Cross

```
*Cross> firstWord ("Frage","ANTWORT") $newGrid 10
[Q 0 (1,5) Horizontal "Frage" "ANTWORT"]

' '' '' '' '' '' '' '' '' '' '' '
' '' '' '' '' '' '' '' '' '' '' '
' '' '' '' '' '' '' '' '' '' '' '
' '' '' '' '' '' '' '' '' '' '' '
' '' '' '' '' '' '' '' '' '' '' '
' ''A''N''T''W''O''R''T'' '' '
' '' '' '' '' '' '' '' '' '' '' '
' '' '' '' '' '' '' '' '' '' '' '
' '' '' '' '' '' '' '' '' '' '' '
' '' '' '' '' '' '' '' '' '' '' '
```

e) Für manche Arbeitsschritte kann es sinnvoll sein, das komplette Rätsel zu transponieren, so dass die Zeilen zu Spalten werden und die Spalten zu Zeilen. Damit werden vertikale Wörter zu horizontalen Wörtern und umgekehrt. Bei nicht quadratischen Rätseln ändert sivch auch die Dimensionsangabe.

Schreiben Sie eine Funktion, die ein Rätselgitter transponiert.

Cross.lhs

```
> transposeGrid :: Grid a -> Grid a
> transposeGrid g = g
```

f) Nun soll in eine Zeile des Rätsels ein Wort eingefügt werden. Dieses kann auf unterschiedliche Weise, aber auch gar nicht möglich sein. Das Ergebnis ist deshalb eine Liste von allen Möglichkeiten, wie das Wort eingetragen werden kann.

Cross.lhs

```
> insertWordInLine
>   :: (Eq a1, Eq a2, Num a3) =>
>      [a2]
>      -> [Maybe (Maybe a1, Border, a2)]
>      -> [(a3, [Maybe (Maybe a1, Border, a2)])]
> insertWordInLine _ _ = []
```

Hier ein Beispielaufruf dieser Funktion. In eine Zeile, die mit Wort `hallo` beginnt und anschließend noch vier freie Felder hat, soll das Wort `lok` eingefügt werden.

Cross

```
*Cross> insertWordInLine "lok" [Just (Nothing,None,'h'),Just (Nothing,None,'a'
),Just (Nothing,None,'l'),Just (Nothing,None,'l'),Just (Nothing,None,'o'),Noth
ing,Nothing,Nothing,Nothing]
[(3,[Just (Nothing,None,'h'),Just (Nothing,None,'a'),Just (Nothing,None,'l'),J
ust (Nothing,None,'l'),Just (Nothing,None,'o'),Just (Nothing,None,'k'),Nothing
,Nothing,Nothing]),(5,[Just (Nothing,None,'h'),Just (Nothing,None,'a'),Just (N
othing,None,'l'),Just (Nothing,None,'l'),Just (Nothing,None,'o'),Just (Nothing
,None,'l'),Just (Nothing,None,'o'),Just (Nothing,None,'k'),Nothing]),(6,[Just
(Nothing,None,'h'),Just (Nothing,None,'a'),Just (Nothing,None,'l'),Just (Nothi
ng,None,'l'),Just (Nothing,None,'o'),Nothing,Just (Nothing,None,'l'),Just (Not
hing,None,'o'),Just (Nothing,None,'k')])]
```

Um das Ergebnis besser lesen zu können, bietet sich an, es kompakt als Stringliste darzustellen:

Cross

```
*Cross> map (map (maybe ' ' (\(_,_,c)->c)))$map snd line
["hallok   ","hallolok ","hallo lok"]
```

Die Funktion findet drei Möglichkeiten, die Buchstaben `lok` in der Zeile zu platzieren. Einmal unter Verwendung der Buchstaben `lo` des Wortes `hallo` und zweimal unter Werverndung der vier freien Felder.

Solche Überlappungen, wie im ersten dieser drei Felder sind bei Kreuzworträtseln allerings nicht erlaubt. Wir müssen sie aber für einzelne Zeilen zunächst untersuchen, weil nicht gesagt ist, dass die verwendeten schon eingetragenen Buchstaben `lo` nicht durch verikale Wörter in diese Zeile gekommen sind. Dann wäre die erste der drei Lösungen sogar ünschenswert.

g) Mit der zuletzt umgesetzten Funktion sind wir in der Lage, zu berechnen, auf welche Art ein Wort in eine Zeile eingefügt werden kann. Das lässt sich sich nutzen. Zum einen, indem wir in allen Zeilen des Rätsels schauen, wie sich dort das Wort einfügen lässt.

Dann können wir das Rätsel transponieren und auf den transponierten Rätsel wieder versuchen, das Wort einzufügen. So lässt sich das Wort auch vertikal einfügen.

Cross.lhs

```
> insertWord :: Eq a => (String, [a]) -> Grid a -> [Grid a]
> insertWord ws g = filter (not.hasOverlappings)
>   (insertWordHorizontal ws g
>   ++(map transposeGrid$insertWordHorizontal ws$transposeGrid g))
```

Man braucht sich also nur um den horizontalen Fall zu kümmern. Implementieren Sie diesen unter Verwendung der Funktion `insertWordInLine` der letzten Aufgabe.

Cross.lhs

```
> insertWordHorizontal :: Eq a =>
>     (String, [a]) -> Grid a -> [Grid a]
> insertWordHorizontal  _ g = [g]
```

Die durch `insertWordInLine` eventuell entstandenen Lösungen mit überlappenden Wörtern sind aus der Lösungsliste zu filtern. Da man die eingetragenen Frage-Antwortenpaare auch in den `Grid`-Daten notiert mit Position im Gitter, Orientierung und natürlich der Länge des Antwortwortes, so lässt sich diese Information aus dem `Grid` generieren.

Cross.lhs

```
> hasOverlappings :: Grid a -> Bool
> hasOverlappings _ = False
```

h) In den vorherigen Aufgaben wurden alle Möglichkeiten generiert, wie ein Wort in ein Rätsel eingefügt werden kann. Uns interessieren Lösungen, in denen möglichst viel Buchstaben von zwei Wörtern geteilt werdn. Dieses sind die Rätsel, in denen noch möglichst viele Felder mit `Nothing` belegt sind. Schreiben Sie eine Funktion, die nur die besten, also mit maximaler Anzahl von `Nothing`-Feldern belegte Felder haben.

Cross.lhs

```
> selectBest :: Eq a => [Grid a] -> [Grid a]
> selectBest gs = gs
```

i) Zwei Dinge haben wir bisher ausgeklammert. Die Nummerierung der Fragen im Rätsel und die Wortendemarkierungen. Schreiben Sie jetzt eine Funktion, die diese Information noch für ein Rätsel generiert.

Cross.lhs

```
> addNumbering :: Grid a -> Grid a
> addNumbering g = g
```

Wir sind nun in der Lage erste Kreuzworträtsel zu generieren. Wir sind dabei recht naiv vorgegangen, untersuchen nicht, ob es durch die unterschiedliche Reihenfolge, in der die Wörter nacheinander eingefügt werden, nicht bessere Lösungen gibt. Wir haben trotzdem schon durch

das Hauruckverfahren Probleme in der Performanz und bei größeren Kreuzworträtseln werden schnell die Grenzen des Beherrschbaren gesprengt.

Projektidee 1

Nehmen Sie diese Aufgabe als Grundlage, um bessere und größere Kreuzworträtsel zu generieren. Überlegen Sie, ob Sie dann mit den Kreuzworträtseln eine Webapplikation erstellen können, bei dem Serverseitig die Rätsel generiert werden und dann auf einer Webseite mit einem ELM-Programm[Fel16] zum Lösen als Rätselpause bereit gestellt werden können.

4 Lernzuwachs

- Arbeiten mit Standardlisten.
- Arbeiten mit dem Typ `Maybe`.
- Beispiel für die Verwendung einer eigenen Typklasse.

Literatur

[Fel16] Richard Feldman. *Elm in Action*. Manning, meap edition, 2016.

[Neu20] Gerd Neugebauer. A LaTeX Package for Typesetting Crossword Puzzles and More. `http://www.gerd-neugebauer.de/software/TeX/cwpuzzle/cwpuzzle.pdf`, 2020.