Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
L. Noschinski, L. Hupel, Dr. J. Blanchette, M. Eberl

WS 2014/15
14.11.2014
Abgabe: 21.11.2014

Einführung in die Informatik 2
6. Übung

Aufgabe G6.1 Der Unterschied

Was ist der Unterschied zwischen

1. `(div 5)` und `(`‘`div`‘ `5)`?
2. `(+ 7)` und `((+) 7)`?
3. `map (: [])` und `map ([] :)`?
4. `flip . flip` und `id`?
5. `[x, y, z]` und `x : y : z`?

Aufgabe G6.2 „Wer bin ich – und wenn ja, was ist mein Typ?“

Gegeben seien die folgenden Funktionen:

```
f $ z = f z

foldl f z [] = z
foldl f z (x : xs) = foldl f (f z x) xs

isCons = not . null

g = (.) map

oo = (.).(.)
```

1. Finden Sie den allgemeinsten Typ von `($)`, `foldl`, `isCons` und `g` und `oo` ohne Hilfe von GHCi heraus. Vergleichen Sie Ihre Hypothesen mit der Ausgabe von GHCi.
2. In welchem Kontext könnte `oo` als nützlich erweisen?

Aufgabe G6.3 Induktion

Beweisen Sie die Gleichung

```
filter p . filter p = filter p
```

per Extensionalität mit anschließender Induktion. Die notwendigen Definitionen finden Sie gewohnt in der `cprf`-Datei auf der Übungsseite.

Hinweis: Sie werden eine Fallunterscheidung durchführen müssen.

Aufgabe G6.4 Teile und Falte (optional)

Die Funktion `partition :: (a -> Bool) -> [a] -> ([a], [a])` aus der Bibliothek `Data.List` teilt eine Liste in zwei Teillisten. Die eine Teilliste enthält die Elemente der Eingabeliste, die das gegebene Prädikat erfüllen. Die zweite Teilliste enthält die übrigen Elemente. Beide Teillisten sollen die Elemente in der gleichen Reihenfolge wie in der Eingabeliste enthalten.

Die folgenden Beispiele sind alle `True` (`xs` ist eine beliebige Liste):

```
partition (\x -> True) xs == (xs, [])
partition (\x -> False) xs == ([], xs)
partition even [4, 4, 1, 2] == ([4, 4, 2], [1])
```

In Blatt 5 haben Sie `partition` bereits implementiert. Implementieren Sie nun `partition` mittels `foldr` ohne die Verwendung weiterer rekursiver Funktionen.

Aufgabe H6.1 Pfadfinder

Eines der grundlegenden Prinzipien von Unix-basierten Betriebssystemen ist „alles ist eine Datei“. Nun müssen Dateien auch irgendwie organisiert werden. In dieser Aufgabe befassen Sie sich mit Ordnerstrukturen und entsprechenden Operationen darauf.

Ein *Pfad* ist eine Sequenz von *Dateinamen*, getrennt durch den *Pfadseparator* (hier: '`/`'). Dateinamen dürfen nicht leer sein, können aber praktisch beliebige Zeichen enthalten, außer den Pfadseperator selbst.[1] Beginnt ein Pfad mit dem Pfadseperator, so wird dieser *absolut* genannt, ansonsten *relativ*. Für diese Aufgabe ist festgelegt, dass Pfade nie mit dem Pfadseperator enden dürfen, außer es handelt sich um den leeren absoluten Pfad, der die *Wurzel* des Dateisystems repräsentiert. Ein leerer relativer Pfad existiert nicht. Mehrere aufeinanderfolgende Pfadseperatoren werden als ein einziger interpretiert.

1. Schreiben Sie eine Funktion `valid :: String -> Bool`, die ermittelt, ob ein angegebener Pfad syntaktisch gültig ist.

   Beispiele:

   ```
   valid "/"              -- gültig: Wurzelpfad (absolut)
   valid "/home"          -- gültig: absoluter Pfad
   not (valid "/home/")   -- ungültig

   valid "home"           -- gültig: relativer Pfad
   valid "home/otto"      -- gültig: relativer Pfad
   not (valid "")         -- ungültig!
   ```

2. Einige Dateinamen sind besonders: Ein einzelner Punkt ('`.`') steht für den aktuellen Pfad; ein doppelter Punkt ('`..`') steht für den übergeordneten Pfad. Der übergeordnete Pfad des Wurzelpfads ist der Wurzelpfad selbst.

[1] In der Realität gibt es noch weitere Einschränkungen, die uns aber hier nicht interessieren.

Beispiel: Die Pfade '`/home/otto/..`', '`/home/.`', '`/home//.`' und '`/home`' sind äquivalent.

Definieren Sie eine Funktion `normalize :: String -> String`, die einen gegebenen gültigen Pfad *normalisiert*, d.h. die kürzeste zur Eingabe äquivalente Darstellung liefert.

Beispiele:

```
normalize "/home/otto/.." == "/home"
normalize "/home/.././usr/.." == "/"
normalize ".." == ".."
normalize "../.." == "../.."
normalize "home/.." == "."
```

Hinweis: Hierzu kann es nützlich sein, den übergebenen Pfad in die einzelnen Dateinamen aufzusplitten.

3. Die *Interpretation* eines Paars von Pfaden entspricht der Auflösung eines Pfades relativ zu einem absoluten Pfad. Definieren Sie eine Funktion `interpretPath`, die aus einem absoluten Pfad p und einen Pfad q denjenige (absoluten und normalisierten) Pfad p' ermittelt, in dem man sich befindet, wenn man von p aus beginnt und nach q wechselt. Falls q ein absoluter Pfad ist, dann „gewinnt" dieser. Sie dürfen davon ausgehen, dass p ein gültiger absoluter Pfad ist und q ein beliebiger gültiger Pfad.

   Beispiele:

   ```
   interpretPath "/" ".." == "/"
   interpretPath "/home" "/usr" == "/usr"
   interpretPath "/home/./otto" "../." == "/home"
   ```

Stellen Sie sicher, dass Ihre Funktionen stets vollständig normalisierte und gültige Pfade zurückgeben. Mit Ausnahme von `valid` erhalten Sie auch stets gültige Pfade als Eingabe.

Aufgabe H6.2 Falten, finden

Geben Sie eine Funktion `f` an, so dass

```
findAll :: Eq a => a -> [(a,b)] -> [b]
findAll k xs = foldr (f k) [] xs
```

alle zweiten Elemente der Tupel aus `xs` zurückliefert, deren erstes Element `k` ist. Die Reihenfolge der zurückgegebenen Elemente soll der in `xs` entsprechen.

Aufgabe H6.3 (Wettbewerbsaufgabe)

Jedes Jahr steht die Informatik 2-Übungsleitung vor dem gleichen Problem: Es gibt n Tutoren und k Übungsgruppenslots (mit $k \geq 2n$) von denen manche miteinander überlappen. Jeder Tutor soll nun genau zwei Übungsgruppen übernehmen, die sich natürlich zeitlich nicht überschneiden dürfen. Um diese Zuteilung zu erleichtern, sollen Sie nun ein Programm schreiben, das diese automatisch vornimmt.

Zur besseren Übersichtlichkeit definieren wir folgende Typsynonyme:

```
type Tutor = String
type Time = Integer
type TimeSlot = (Time, Time)
```

Zeiten werden der Einfachkeit halber als `Integer` kodiert.

Schreiben Sie nun eine Funktion

```
findMapping :: [Tutor] -> [TimeSlot] ->
                    (Bool, [(Tutor, TimeSlot, TimeSlot)])
```

Die Funktion erhält als Parameter die Liste der Tutoren `ts` sowie eine Liste der verfügbaren Zeitslots `ss`. Es kann mehrere gleiche Zeitslots geben, die Liste der Tutoren enthält jedoch keine Duplikate. Sie dürfen annehmen, dass es immer mindestens doppelt so viele Zeitslots wie Tutoren gibt und dass für jeden Zeitslot `(s,e)` die Eigenschaft `s < e` gilt.

Die Ausgabe ist ein Boolean sowie eine Zuordnung der Tutoren zu je zwei Zeitslots. Die Ausgabeliste muss natürlich jeden Tutor genau einmal enthalten; die Reihenfolge ist hierbei egal. Ein Zeitslot darf höchstens so häufig zugeordnet werden, wie er in `ts` vorkommt.

Der Boolean in der Ausgabe gibt an, ob es überhaupt eine Lösung gibt: Falls möglich ist, jedem Tutor zwei Gruppen zuzuordnen, soll `True` und so eine Zuordnung ausgegeben werden; ansonsten soll `False` und eine beliebige Liste (z.B. `[]`) ausgegeben werden.

Hinweis: Die Zeitslots `(1,3)` und `(2,3)` überlappen, die Zeitslots `(1,2)` und `(2,3)` jedoch *nicht*.

Beispiel:

```
findMapping ["A","B"] [(1,2),(1,2),(2,3),(2,3)] ==
    (True, [("A", (1,2), (2,3)), ("B", (1,2), (2,3))])

findMapping ["A","B"] [(1,2),(1,2),(1,3),(2,3)] ==
    (False, [])
```

Hauptkriterium für den Wettbewerb ist dieses Mal die Effizienz. Als Nebenkriterium wird der MC unter Umständen auch die Eleganz, Verständlichkeit und Formatierung des Codes heranziehen.

Hinweis: Für den Wettbewerb können Pakete aus der Haskell-Library – unter Umständen auch *unsafe* Pakete – hilfreich sein. Ihr Code wird dann jedoch nicht mehr auf dem Hausaufgabenserver kompilieren. Falls Sie etwas derartiges verwenden wollen, müssen Sie deshalb diese Lösung auskommentiert mit in die {-WETT-}-Tags schreiben und *zusätzlich* eine nicht auskommentierte Lösung, die auf dem Hausaufgabenserver kompiliert, ebenfalls in die {-WETT-}-Tags schreiben. Falls Sie sich nicht sicher sind, ob Sie eine bestimmte Library für den Wettbewerb verwenden können, fragen Sie den MC Jr. Manuel Eberl per E-Mail.

Wichtig: Wenn Sie diese Aufgabe als Wettbewerbsaufgabe abgeben, stimmen Sie zu, dass Ihr Name ggf. auf der Ergebnisliste auf unserer Internetseite veröffentlicht wird. Sie können diese Einwilligung jederzeit widerrufen, indem Sie eine Email an `fp@fp.in.tum.de` schicken. Wenn Sie nicht am Wettbewerb teilnehmen, sondern die Aufgabe allein im Rahmen der Hausaufgabe abgeben möchten, lassen Sie bitte die {`-WETT-`} . . . {`-TTEW-`} Kommentare weg. Bei der Bewertung Ihrer Hausaufgabe entsteht Ihnen hierdurch kein Nachteil.