Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Lars Noschinski, Dr. Jasmin Blanchette, Dmitriy Traytel

WS 2012/13
27.11.2012
Abgabe: 04.12.2012, 15:30

Einführung in die Informatik 2
7. Übung

Aufgabe G7.1 Wildcard-Muster

Gegeben sei ein endliches Alphabet Σ. Ein *Wildcard-Zeichen* ist entweder

(a) ein normales Zeichen $a \in \Sigma$;
(b) eine Teilmenge $A \subseteq \Sigma$;
(c) das Sonderzeichen ? (Fragezeichen); oder
(d) das Sonderzeichen $*$ (Asterisk).

Ein *Wildcard-Muster* ist eine Liste von Wildcard-Zeichen.

1. Ergänzen Sie die Definition von `WildChar`, damit sie der obigen Beschreibung entspricht.

```
data WildChar = ...
data WildPat = WildPat [WildChar]
```

   Für Σ soll Ihre Implementierung `Char` verwenden. Teilmengen können als Listen dargestellt werden.

2. Implementieren Sie die folgenden Funktionen, die Wildcard-Zeichen bzw. -Muster in Strings umwandeln. Die Menge $\{a_1, \ldots, a_n\}$ wird als `[`$a_1 \ldots a_n$`]` dargestellt (z.B. `[aeiouy]`).

```
stringFromWildChar :: WildChar -> String
stringFromWildPat :: WildPat -> String
```

3. Registrieren Sie die Typen `WildChar` und `WildPat` als Instanzen der Typklasse `Show`, damit GHCi Werte dieser Typen anzeigen kann.

4. Implementieren Sie die Parsingfunktion

```
wildPatFromString :: String -> WildPat
```

   Unerwartete Vorkommen von `[` und `]` sollen als normale Zeichen behandelt werden.

5. Schreiben Sie QuickCheck-Tests für Ihre Konvertierungsfunktionen, wobei Sie sich überlegen sollten, unter welchen Bedingungen die folgenden Eigenschaften gelten:

```
stringFromWildPat (wildPatFromString s) == s
wildPatFromString (stringFromWildPat p) == p
```

6. Schreiben Sie eine Funktion

```
matchWildPat :: WildPat -> String -> Bool
```

   die einen Zielstring mit einem Wildcard-Muster vergleicht. Das Sonderzeichen ? (Fragezeichen) stellt ein beliebiges Zeichen, $*$ (Asterisk) eine beliebige Sequenz von (null oder

mehr) Zeichen und die Teilmenge $\{a_1, \ldots, a_n\}$ ein beliebiges Zeichen a_j $(1 \leq j \leq n)$ dar. Das Muster muss den vollständigen Zielstring erkennen.

Für `match = matchWildPat . wildPatFromString` sollen die folgenden Beispiele alle `True` sein (vgl. Aufgabe G4.4):

```
match "?bc" "abc"            not (match "a" "")
match "a*f" "abcdef"         not (match "a*f" "abcde")
match "a***b?" "abc"         not (match "a***b" "abc")
match "[abc]" "b"            not (match "[abc]" "d")
match "[abc" "[abc"          not (match "[abc" "a")
match "][" "]["              not (match "[]" "[]")
```

Aufgabe G7.2 Sortieren mit Bäumen

In der Vorlesung (Folien 209, 210) wurde ein Datentyp `Tree` eingeführt. Die dazu gehörige Funktion `insert :: Ord a => a -> Tree a -> Tree a` fügt Elemente geordnet in einen Baum ein, sofern sie noch nicht im Baum vorhanden sind.

Benutzen Sie `insert`, um eine Funktion `treeSort :: Ord a => [a] -> [a]` zu implementieren, die Listen sortiert – und dabei Duplikate entfernt. Bauen Sie dazu zunächst einen geordneten Baum auf und wandeln Sie diesen dann in eine sortierte Liste um.

Welche Tests sind sinnvoll, um sicherzustellen, dass treeSort wirklich sortiert?

Aufgabe G7.3 HTML Teil 1

Gegeben sei ein Datentyp zur Darstellung von einfachen HTML-Dokumenten:

```
data Html =
  Text String |
  Block String [Html]
```

Ein HTML-Dokument ist also

- entweder ein `Text`-Element mit einem `String`-Inhalt
- oder ein `Block`-Element mit einem `String`-Tag und einer Liste von geschachtelten HTML-Dokumenten.

Beispiele:

```
Text "Every string should learn to swim"
Block "head" []
Block "body" [Block "p" [Text "My cat"], Text "is not a float"]
```

Gesucht wird eine Funktion `plainHtml :: Html -> String`, die solche Dokumente in Text umwandelt.

Der Inhalt von `Text`-Elementen wird bis auf zwei Ausnahmen unverändert ausgegeben:

- Nicht-ASCII-Zeichen (d.h. Zeichen c, für die `Data.Char.ord` $c \geq 2^7$) sollen durch numerische HTML-Entitäten ersetzt werden. Beispiel: ÿ hat den Code 255 und wird deshalb als `&#255;` kodiert.
- Die schweizerdeutschen Buchstaben Ä, Ö, Ü, ä, ö und ü sollen aber durch symbolische HTML-Entitäten ersetzt werden. Beispiel: Ü wird als `&Uuml;` kodiert. Die folgende Assoziierungsliste (association list) enthält die notwendigen Informationen.

```
swissLetters =
  [(196, "Auml"), (214, "Ouml"), (220, "Uuml"),
   (228, "auml"), (246, "ouml"), (252, "uuml")]
```

  Ihr Code sollte diese Liste verwenden.

Elemente der Form `Block` s $[h_1, \ldots, h_n]$ werden als `<`s`>`$h_1^\star \cdots h_n^\star$`</`$s$`>` ausgegeben, wobei $h_j^\star$ die Übersetzung von h_j darstellt.

Beispiel:

```
grueezi =
  Block "html"
    [Block "head"
       [Block "author" [Text "der MC"],
        Block "date" [Text "27.11.2012"],
        Block "topsecret" []],
     Block "body"
       [Block "h1" [Text "Gr\252ezi!"],
        Block "p" [Text "\196b\228, genau. Sal\252. \
                        \Bis sp\246ter!"]]]
putStr (plainHtml grueezi)
```

gibt

```
<html><head><author>der MC</author><date>27.11.2012</date><top
secret></topsecret></head><body><h1>Gr&uuml;ezi!</h1><p>&Auml;
b&auml;, genau. Sal&uuml;. Bis sp&ouml;ter!</p></body></html>
```

aus.

Hinweis: Die Bibliotheksfunktionen `Data.Char.ord`, `Data.List.lookup` und `concatMap` können hilfreich sein.

Wichtig: Bei bisherigen Hausaufgaben gab es immer wieder eingereichte Lösungen die zwar inkorrekt waren aber dennoch alle unsere Tests bestanden haben. Das ist nichts Ungewöhnliches, da die Tests die Korrektheit nicht garantieren. Allerdings sind wir daran interessiert unsere Tests zu verbessern, indem eventuelle Lücken geschlossen werden.

Dabei zählen wir auf Ihre Unterstützung, die wir gerne mit zusätzlichen Hausaufgabenpunkten entlohnen werden. Falls Sie also eine inkorrekte Implementierung haben, die unsere Tests

besteht, können Sie zusätzliche Tests in Ihren hochgeladenen Lösungsvorschlag einschließen. Benutzen Sie bitte die Tags {-QC-}...{-CQ-} um die neuen Tests zu kennzeichnen.

Wir werden die sinnvollen Tests in unsere Testsuite einpflegen. Wenn Ihre Tests Fehler unter den eingereichten Lösungen offenbaren, die wir nicht gefunden haben, bekommen Sie einen zusätzlichen Punkt gutgeschrieben (es gilt First-Come-First-Served).

Aufgabe H7.1 HTML Teil 2 (5 Punkte)

Diese Aufgabe ist eine Fortsetzung von G7.3 mit anderen Mitteln. Die Ausgabe von `plainHtml` ist nicht schön – es fehlen Zeilenumbrüche (`'\n'` in Haskell) und Einrückungen (mit Leerzeichen).

Schreiben Sie eine Funktion `prettyHtml :: Int -> Html -> String`, die zusätzlich die Blöcke einrückt. Das `Int`-Argument gibt die Größe der Einrückung an (als Anzahl von Leerzeichen). Zudem sollen leere Blöcke anstelle von `<s></s>` durch `<s />` dargestellt werden.

Beispiel: `putStr (prettyHtml 4 grueezi)` soll

```
<html>
    <head>
        <author>
            der MC
        </author>
        <date>
            27.11.2012
        </date>
        <topsecret />
    </head>
    <body>
        <h1>
            Gr&uuml;ezi!
        </h1>
        <p>
            &Auml;b&auml;, genau. Sal&uuml;. Bis sp&ouml;ter!
        </p>
    </body>
</html>
```

ausgeben. Achtung: Der äußerste Block ist *nicht* eingerückt.

Aufgabe H7.2 Baumkosmetik (10 Punkte)

Gegeben sei ein Datentyp zur Darstellung von Verzeichnisstrukturen:

```
data DirTree a =
  File a |
  Dir a [DirTree a]
```

Ihre Aufgabe ist es, solche Bäume gut lesbar auszugeben. Als Beispiel verwenden wir den Baum

```
exDir = Dir "" [Dir "usr" [Dir "lib" [File "vim"],
                           Dir "include" [File "string.h"]],
                Dir "bin" [File "ls", File "cat"]]
```

Für die beiden folgenden Teilaufgaben gilt: Zeilen werden durch `\n` getrennt; auch die letzte Zeile endet in `\n`. Dies erreichen Sie, indem Sie die Ausgabe als Liste von Zeilen aufbauen und mit der Funktion `unlines` zusammensetzen.

Überflüssige Leerzeichen am Ende einer Zeile sind nicht erlaubt. Die Reihenfolge der Verzeichniseinträge soll erhalten bleiben. Zur Darstellung der Datei- und Verzeichnisnamen benutzen Sie die Funktion `show`.

1. Schreiben Sie eine Funktion `plainDirTree :: Show a => DirTree a -> String`, die `DirTree`s durch Einrückung formatiert. Beispiel: `putStr (plainDirTree exDir)` soll ausgeben:

   ```
   /""
     /"usr"
       /"lib"
         "vim"
       /"include"
         "string.h"
     /"bin"
       "ls"
       "cat"
   ```

   Für jede Verzeichnisebene wird um zwei Leerzeichen eingerückt, vor Verzeichnissen wird zusätzlich `/` (Schrägstrich) gesetzt.

2. Schöner wird die Ausgabe, wenn die zusammengehörenden Dateien bzw. Verzeichnisse zusätzlich durch Linien verbunden werden. Schreiben Sie dazu die Funktion:

   ```
   prettyDirTree :: Show a => DirTree a -> String
   ```

Beispiel: `putStr (prettyDirTree exDir)` soll ausgeben:

```
+-\ ""
  +-\ "usr"
  | +-\ "lib"
  | | +-- "vim"
  | +-\ "include"
  |   +-- "string.h"
  +-\ "bin"
    +-- "ls"
    +-- "cat"
```

Vor einem Verzeichnis soll "`+-\␣`" und vor einer Datei "`+--␣`" stehen. Wie zuvor beträgt die Einrückung pro Verzeichnisebene zwei Zeichen, jedoch ist es nun entweder "`␣␣`" oder "`|␣`", abhängig davon, ob es noch einen weiteren Eintrag auf dieser Verzeichnisebene gibt.

Zum Beispiel sind "`ls`" und "`cat`" mit "`␣␣`" eingerückt, denn "`bin`" ist der letzte Eintrag auf der zweiten Verzeichnisebene. Da "`bin`" auf "`usr`" folgt, sind alle Einträge zwischen den beiden Verzeichnissen mit "`|␣`" eingerückt.

Aufgabe H7.3 Korekrtur (5 Punkte, Wettbewerbsaufgabe)

Gesucht wird eine Funktion

```
unscrambleWords :: [String] -> [String] -> [String]
```

die eine Vokabelliste *vocabs* und eine Liste von teilweise falsch geschriebenen Wörtern *text* nimmt. Der Aufruf `unscrambleWords` *vocabs text* gibt eine Liste von korrigierten Wörtern (den *korrigierten Text*) zurück.

Ein *Wort* ist eine Liste von `Char`s aus den 26 kleinen englischen Buchstaben `a` bis `z`. Ein Wort ist *falsch geschrieben*, wenn es nicht in der Vokabelliste *vocabs* steht. Die Korrektur eines solchen Wortes besteht darin, es durch eine seiner in *vocabs* vorhandenen Permutationen zu ersetzen. Falls mehrere Permutationen vorhanden sind, kann eine beliebige gewählt werden. Falls keine vorhanden ist, soll das falsche Wort unverändert stehen bleiben.

Beispiele:

```
vocabs = ["i", "kyoto", "more", "now", "tokyo", "want", "won"]

unscrambleWords vocabs ["i", "want", "mroe"] ==
  ["i", "want", "more"]
unscrambleWords vocabs ["and", "i", "tnaw", "it", "now"] ==
  ["and", "i", "want", "it", "now"]
unscrambleWords vocabs ["tokyo", "", "kyoto"] ==
  ["tokyo", "", "kyoto"]
```

Bibliotheksfunktionen wie `Data.List.lookup` und `Data.List.permutations` sind erlaubt.

Schreiben Sie auch mindestens zwei sinnvolle QuickCheck-Tests.

Für den Wettbewerb zählt die Effizienz, vor allem die Skalierbarkeit. Ihre Implementierung sollte mit langen (bzw. kurzen) Vokabellisten, großen (bzw. kleinen) Texten, langen (bzw. kurzen) Wörtern und absurd vielen (bzw. keinen) Duplikaten möglichst effizient vorgehen. Der wichtigste Fall ist erfahrungsgemäß der, in dem die Vokabelliste im 10.000-Wort-Bereich und der Text im 100.000-Wort-Bereich liegt – denn der Master of Competition hat vor, seine reichhaltigen literarischen Ergüsse korrigieren zu lassen.

Die Lösung des MC benötigt ca. 2,5 Sekunden für das Beispiel

```
Exercise_7> import Data.List
Exercise_7> import Data.Hashable
Exercise_7> let vocabs = take 10000 (permutations "abcdefghi")
Exercise_7> let text = take 100000 (permutations "dichbagef")
Exercise_7> :set +s
Exercise_7> hash (unscrambleWords vocabs text)
-207789595
(2.54 secs, 848755064 bytes)
```

in GHCi. (Die Funktion `hash` wird hier verwendet, um die Ausgabe drastisch zu reduzieren. Installierung: `cabal install hashable`.) Versuchen Sie, ihn zu schlagen!

Lösungen, die sich effizienztechnisch ähnlich verhalten, werden bzgl. ihrer Lesbarkeit verglichen, wobei Formatierung und Namensgebung besonders gewichtet werden. Um teilzunehmen müssen Sie Ihre vollständige Lösung innerhalb der Kommentare {`-WETT-`} und {`-TTEW-`} eingeben.

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.