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

WS 2012/13
18.11.2012
Abgabe: 08.01.2013, 15:30

Einführung in die Informatik 2
10. Übung

Aufgabe G10.1 Den Beweis unter Beweis stellen

In der Vorlesung wurden zwei Implementierungen von Mengen als Listen eingeführt. Der Korrektheitsbeweis der ersten Implementierung (durch Listen mit Duplikaten, Folie 261) wurde Ihnen zur Übung überlassen. Führen Sie ihn zu Ende.

Hinweis: Die letzten zwei Simulationsbedingungen können mit Hilfe der Hilfssätze

$$\texttt{elem}\ x\ xs = x \in \alpha\ xs \qquad\qquad \texttt{length}\ (\texttt{nub}\ xs) = |\alpha\ xs|$$

bewiesen werden. Diese müssen Sie wiederum durch Induktion beweisen.

Aufgabe G10.2 Die Rückkehr der Paarliste

Wir wollen die abstrakten Mengen mittels der von H6.1 bekannten Paarlisten implementieren. Die Implementierungsdetails sollen mit Hilfe des Modulsystems von Haskell vor dem Benutzer versteckt werden.

Definieren Sie ein Modul `SetPairList`, welches die folgenden Typen/Funktionen exportiert.

```
newtype Set a = ...
empty :: Set a
insert :: Integral a => a -> Set a -> Set a
isin :: Integral a => a -> Set a -> Bool
size :: Integral a => Set a -> Integer
union :: Integral a => Set a -> Set a -> Set a
delete :: Integral a => a -> Set a -> Set a
```

Die Mengen sollen intern als Paarlisten dargestellt werden. Während wir in der Aufgabe H6.1 lediglich Paarlisten über `Int` betrachtet haben, sollen hier Paarlisten über beliebigen Typen aus der Typklasse `Integral` (Ganzzahlen) definiert werden.

Geben Sie die Abstraktionsfunktion α :: `Set a` $\to$ {`a`} im ~~Gangnam~~Haskell-Style (siehe Folie 259) an und formulieren Sie die Eigenschaften die gelten sollten, falls Ihre Implementierung korrekt ist. Sie brauchen die Beweise nicht zu führen, allerdings sollten Sie überlegen ob Sie eine Invariante für die Beweise brauchen werden. Definieren sie die Invariante als Haskell-Funktion `invar :: Integral a => Set a -> Bool` falls notwendig.

Aufgabe G10.3 Na warte, Schlange! (optional)

Eine *Warteschlange* ist eine klassische Datenstruktur mit FIFO-Semantik ("first in, first out"). Neue Elemente werden am Ende der Schlange hinzugefügt, alte Elemente werden am Kopf entfernt. Eine Haskell-Umsetzung dieses Konzepts könnte die Form eines Typs `Queue a` mit folgenden Operationen annehmen:

```
empty :: Queue a
isEmpty :: Queue a -> Bool
enqueue :: a -> Queue a -> Queue a   -- herein
dequeue :: Queue a -> (a, Queue a)   -- heraus
```

Die Operationen haben die folgenden charakterisierenden Eigenschaften (für alle `x` und `q`):

```
isEmpty empty
not (isEmpty (enqueue x q))
dequeue (enqueue xN (... (enqueue x2 (enqueue x1 empty)) ...))
    == (x1, (enqueue xN (... (enqueue x2 empty) ...)))
```

Die Funktion `dequeue` ist auf `empty` nicht definiert.

Ihre Aufgaben sind die Folgenden:

1. Die einfachste Implementierung eines Schlangentyps `Queue a` basiert auf einer Liste `[a]`. Elemente werden am Ende der Liste hinzugefügt und vom Kopf entfernt. Definieren Sie diesen Typ in einem eigenen Modul `Queue` und implementieren Sie die obigen Operationen sowie `==`. Stellen Sie sicher, dass keine Implementierungsdetails sichtbar sind.
2. Eine effizientere Implementierung von `Queue a` benötigt zwei Listen (*front*, *kcab*), wobei *front* den Kopf und *kcab* das umgedrehte Hinterteil (die Schwanzspitze) der Schlange darstellt. Die abstrakte Schlange $[\![x_1, \ldots, x_n]\!]$ ist also für ein beliebiges $k \in \{1, \ldots, n\}$ als $\big([x_1, \ldots, x_k], [x_n, \ldots, x_{k+1}]\big)$ konkret dargestellt. Beispiele:
   - `([], [])` entspricht der leeren Schlange.
   - `([1, 2], [])`, `([1], [2])` und `([], [2, 1])` entsprechen der Schlange $[\![1, 2]\!]$.
   - `("lange", "egnalhcS")` entspricht der Schlange $[\![\texttt{l}, \texttt{a}, \texttt{n}, \texttt{g}, \texttt{e}, \texttt{S}, \texttt{c}, \texttt{h}, \texttt{l}, \texttt{a}, \texttt{n}, \texttt{g}, \texttt{e}]\!]$.

   Definieren Sie diesen Typ in einem eigenen Modul `SuperQueue`. Beachten Sie, dass `==` eine sinnvolle Semantik bezüglich verschiedener Darstellungen einer einzigen Schlange hat. Überlegen Sie, warum diese Implementierung angeblich effizienter ist als die mit einer einzigen Liste.
3. Beweisen Sie, dass `SuperQueue` eine Warteschlange simuliert.

   Hinweis: Wählen Sie zuerst einen angemessenen abstrakten Schlangentyp und eine Abstraktionsfunktion α. (Es ist keine Invariante nötig.) Leiten Sie dann die Simulationsbedingungen her (siehe Folie 265) und führen Sie die Beweise.

Wichtig: Dieses Mal besteht Ihre Übungsabgabe aus mehreren Dateien. Laden Sie alle diese Dateien (und nicht nur `Exercise_10.hs`) in das übliche Verzeichnis (`exercise_10`) hoch.

Aufgabe H10.1 BKA, BSI und SMS (8 Punkte)

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) war vom Bundeskanzleramt beauftragt worden, die SMS-App der Kanzlerin auf seine Sicherheit hin zu untersuchen. Dabei musste das BSI feststellen, dass die in Java geschriebene App nicht nur völlig unverständlich war, sondern auch gravierende Sicherheitslücken aufwies. Daraufhin beschloss das BSI eine Ausschreibung für eine SafeKanzlerSMS-App: Der Kern muss in Haskell programmiert werden und einen Korrektheitsbeweis haben.

Ein Modul des Kerns soll einen abstrakten Datentyp `SafeMap k v` für eine Assoziationstabelle realisieren, die Schlüssel vom Typ `k` auf Werte vom Typ `v` abbildet. Dieser soll die folgenden Operationen unterstützen:

```
-- builds a new, empty SafeMap
empty :: Eq k => SafeMap k v

-- updates the value for key k to value v
-- (inserts the key if necessary)
update :: Eq k => k -> v -> SafeMap k v -> SafeMap k v

-- if a key k is present in the map, return its value v
-- as Just v, else Nothing
lookup :: Eq k => k -> SafeMap k v -> Maybe v
```

1. Schreiben Sie ein Module `SafeMap` (in der Datei `SafeMap.hs`), das den abstrakten Datentyp `SafeMap` implementiert und nur die oben beschriebene Schnittstelle exportiert. Im Hinblick auf die nächste Aufgabe ist eine einfache Implementierung sinnvoll.
2. Geben Sie eine Abstraktionsfunktion α, die eine `SafeMap` zu einer geeigneten mathematischen Struktur abstrahiert, und wenn notwendig auch eine Invariante.
3. Geben Sie die Simulationsbedingungen für die Funktionen `empty` und `update` an (siehe Folie 265).

Aufgabe H10.2 Join (7 Punkte + 5 Bonuspunkte)

Für die Ablage der Übungspunkte verwendet die Übungsleitung einfache Textdateien der folgenden Form:

```
roi.soleil@etat.moi,Louis XIV,5,3,4
veni.vidi.bibi@haskell.tum.de,Julius Caesar,1,2,3,4
```

Das heißt, in jeder Zeile steht zunächst die E-Mail-Adresse, dann der Name und dann die Punktzahlen für die Hausaufgaben einer Woche. Die Einträge sind jeweils durch ein `','` getrennt und die Anzahl der Punktzahlen kann in jeder Zeile variieren.

Bedauerlicherweise ist einigen Tutoren die Namensliste abhanden gekommen, so dass sie in ihren Bewertungen das Namensfeld ganz ausgelassen haben:

```
roi.soleil@etat.moi,5,3,4
veni.vidi.bibi@haskell.tum.de,1,2,3,4
```

Zwar hat ihnen die Übungsleitung mittlerweile eine Namensliste mit Zeilen der Form

```
roi.soleil@etat.moi,Louis XIV
veni.vidi.bibi@haskell.tum.de,Julius Caesar
```

zukommen lassen, aber bei über 500 Studenten wäre es sehr viel Arbeit, diese von Hand zusammenzuführen. Daher hat sich die Übungsleitung entschlossen, die Studenten zu bitten, ein Programm zu schreiben, das als Eingabe die beiden unteren Dateien nimmt und daraus die obere Datei macht.

Formaler: Es ein Programm gesucht, das zwei Dateien als Eingabe nimmt. Jede Zeile dieser Dateien ist durch `','` in Felder getrennt. Das erste Feld einer Zeile ist das Indexfeld. Sie dürfen annehmen, dass die Zeilen einer Datei aufsteigend nach dem Indexfeld sortiert sind und dass der Wert des Indexfeldes in einer Datei immer eindeutig ist. Die Ausgabe des Programmes ist der *Join* der beiden Dateien: Für jedes Indexfeld I, dass in beiden Dateien vorkommt, soll die Zeile

$$I\,,A_1\,,\ldots,A_m\,,B_1\,,\ldots,B_n$$

ausgegeben werden, wobei $I\,,A_1\,,\ldots,A_m$ die Zeile aus der ersten Datei und $I\,,B_1\,,\ldots,B_n$ die Zeile aus der zweiten Datei mit dem passenden Indexfeld ist.

1. Der Aufruf von `Join names.txt missing.txt` soll die zusammengeführte Liste ausgeben. Da Sie bisher noch nicht gelernt haben, wie man in Haskell Dateien lesen und Text ausgeben kann, haben wir für Sie ein geeignetes Gerüst bereitgestellt (`Join.hs`), in dem Sie nur noch die Funktion `doJoin :: [String] -> [String] -> [String]` geeignet implementieren müssen. Die beiden Listen enthalten jeweils die Zeilen der beiden Dateien.

   Zum Kompilieren öffnen Sie eine Kommandozeile und wechseln in das Verzeichnis mit Ihrer Lösung. Mit `ghc -main-is Join Join.hs` kompilieren Sie Ihr Programm in eine ausführbare Datei `Join` (`Join.exe` unter Windows). Testen Sie Ihr Programm auf den beigelegten Beispieldateien (`grades.txt`, `missing.txt` und `names.txt`).

2. **Bonusaufgabe.** Wenn das Programm als `Join -f` *`Format File1 File2`* aufgerufen wird, werden nur die Felder ausgegeben, die in *`Format`* angegeben sind.

   So soll z.B. `Join -f A1,B4,B1 names.txt missing.txt` unter anderem die folgenden Zeilen ausgeben:

   ```
   Louis XIV,,5
   Julius Caesar,4,1
   ```

Beachten Sie das leere zweite Feld in der zweiten Zeile – das fünfte Feld in der entsprechenden Zeile von `missing.txt` ist leer.

Dazu müssen Sie zwei Funktionen implementieren:

```
parseFormat :: String -> Maybe [Field]
doJoinFormat :: [Field] -> [String] -> [String] -> [String]
```

mit

```
data Field = A Int | B Int
```

Dabei soll sich `doJoinFormat` ähnlich wie `doJoin` verhalten, aber nur die Felder im ersten Parameter (in der angegebenen Reihenfolge) ausgeben. Hat eine Zeile weniger Felder als im Parameter angegeben, sollen die überzähligen Felder leergelassen werden.

Die Funktion `parseFormat` soll den *`Format`*-String aus dem Aufruf in eine Liste von `Field`s umwandeln. Dieser String ist eine durch Kommas getrennte Liste von `A`i und `B`i, wobei i eine nicht-negative Ganzzahl ist. Ein solcher String wäre z.B. `"A1,B1,B2,A2"`. Kann der String nicht geparst werden, wird `Nothing` zurückgegeben, sonst `Just xs`, wobei `xs` die berechnete Liste ist. Sie können die Funktion `readInt :: String -> Maybe Int` benutzen, um die Zahlen zu parsen.

Aufgabe W10.1 Mursi-Codes (weihnachtlich-winterliche Wettbewerbsaufgabe, **0 Punkte**)

In der Vorlesung wurde der Huffman-Algorithmus, der einen optimalen Präfixcode erzeugt, vorgestellt. In dieser arabisch-winterlichen Wettbewerbsaufgabe geht es um eine Variante dieses Prinzips. Wie Sie sicherlich schon wissen wurde infolge der ägyptischen Revolution ein neuer Computertypus eingeführt, die sogenannte *Mursi-Maschine*. Diese erhielt ihren Namen zu Ehren des Professor Mohammed Mursi, Ph.D., Direktor a. D. der Abteilung für Materialwissenschaft an der Universität Zagazig.

Die Mursi-Maschinen sind binär. Dennoch werden in der wissenschaftlichen Literatur die Namen *Isa* (0) und *Ayyaat* (1) für die Bits verwendet. Die Besonderheit dieser Maschinen besteht darin, dass ein Ayyaat zweimal so viel Speicherplatz wie ein Isa beansprucht. Dementsprechend braucht das Wort IsaAyyaatAyyaatIsaAyyaat mehr Platz als IsaIsaIsaIsaIsaIsaIsa.

Der Master of Competition interessiert sich sehr für diese neue Technologie und sucht deshalb eine Funktion

```
morsiCodeFromFreqs :: [(a, Integer)] -> [(a, String)]
```

die einen effizienten Präfixcode für die gegebenen Zeichen-Frequenz-Paare erzeugt.[1] Die Eingabe soll keine Duplikate enthalten und die Frequenzen sollen strikt positiv sein. Isa wird in der Ausgabe als `i` und Ayyaat als `A` dargestellt.

Beispiel anhand der Lösung des MC:

[1] Morsi ist die übliche englische Schreibweise von Mursi – daher dieser Funktionsname.

```
morsiCodeFromFreqs [('z', 2), ('d', 3), ('f', 5), ('s', 7),
                    ('e', 11), ('h', 100)] ==
  [('e', "AA"), ('s', "AiA"), ('z', "AiiAA"), ('d', "AiiAi"),
   ('f', "Aiii"), ('h', "i")]
```

In diesem Beispiel hat der erzeugte Code die Gesamtkosten 241 „Isas“:

$$2 \times \underbrace{\texttt{AiiAA}}_{\texttt{8i}} + 3 \times \underbrace{\texttt{AiiAi}}_{\texttt{7i}} + 5 \times \underbrace{\texttt{Aiii}}_{\texttt{5i}} + 7 \times \underbrace{\texttt{AiA}}_{\texttt{5i}} + 11 \times \underbrace{\texttt{AA}}_{\texttt{4i}} + 100 \times \texttt{i} \;=\; 241 \times \texttt{i}$$

Das Wort `fezzed` wird als `Aiii AA AiiAA AiiAA AA AiiAi` kodiert.

Bei der Punktevergabe ist die Effizienz des erzeugten Präfixcodes entscheidend. Sollte ein(e) einzige(r) Teilnehmer(in) eine Lösung einreichen, die für jeden unserer Tests einen genauso effizienten oder effizienteren Code als alle anderen eingereichten Lösungen erzeugt, wird diese(r) mit 30 Wettbewerbspunkten (anstatt der üblichen 20) geehrt.

Bei der Jagd nach Effizienz sollten Sie die Korrektheit Ihrer Funktion nicht aus den Augen verlieren. Korrektheit heißt, dass das Ergebnis einen gültigen Präfixcode bildet. Lösungen, die die Korrektheitstests nicht bestehen, bekommen keine Punkte. Um teilzunehmen müssen Sie Ihre 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.