Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. M. Wirsing, M. Hammer

SoSe 2009
Übungsblatt 9

Programmierung und Modellierung

Aufgabe 9-1 Bäume und Ausnahmen

In Aufgabe 8-4 sollte in linearer Zeit geprüft werden, ob ein Baum ausgeglichen ist. Eine mögliche Implementierung ist diese:

```
local
  fun   balanced_aux Empty = (0, true)
      | balanced_aux (Build(_,l,r)) =
                        let
                          val (ld, lf) = balanced_aux l
                          val (rd, rf) = balanced_aux r
                        in
                          ((max ld rd) + 1,
                          ((abs(ld - rd) <= 1) andalso lf andalso rf))
                        end
in
  fun balanced t = #2(balanced_aux t)
end;
```

(**Hinweis**: `local` funktioniert ganz analog zu `let` – nur dass hier im `in ... end` Block Funktionen deklariert werden können, die auch ausserhalb sichtbar sind. Effektiv versteckt die obrige `local`-Konstruktion die Funktion `balanced_aux`. Sie können die Zeilen `local`, `in` und `end;` entfernen, wenn Sie möchten.)

Die Idee dieser Implementierung ist, für einen Baum ein Paar `(t,f)` zurückzugeben, wobei `t` die Tiefe des Baumes angibt und `f` ein boolescher Wert ist, der dann `true` ist, wenn der Baum ausgeglichen ist.

Schreiben Sie diese Funktion so um, dass intern eine Ausnahme verwendet wird um einen nicht ausgeglichenen Teilbaum zu melden. Effektiv soll dafür das Element `f` des Rückgabepaares ersetzt werden. Der Rückgabewert der Funktion `balanced` für unausgeglichene Bäume soll jedoch weiterhin `false` sein. Geben Sie Ihre Implementierung in einer Datei `9-1.sml` ab. Entspricht diese Verwendung von Ausnahmen Ihrer Idee von funktionaler Programmierung? Schreiben Sie dazu einen kurzen Kommentar in die Abgabedatei.

Aufgabe 9-2 FIFO-Buffer

In dieser Aufgabe soll ein FIFO-Buffer (auch als „Warteschlange“ bekannt) in Form einer SML-Struktur implementiert werden. Ein FIFO-Buffer ist eine Datenstruktur ähnlich einer Liste, die durch zwei Operationen modifiziert wird:

- `enqueue(b,e)` erzeugt einen FIFO-Buffer, der durch das Einstellen des Elements `e` in den Buffer `b` entsteht,
- `dequeue(b)` erzeugt einen FIFO-Buffer, der dadurch entsteht, dass das älteste Element aus `b` entfernt wird.

Zusätzlich benötigt man noch eine Operation `head(b)`, die für einen FIFO-Buffer `b` das älteste enthaltene Element zurückgibt.

a) Definieren Sie eine Signatur `FIFO`, die diese drei Methoden sowie einen geeigneten Typ deklariert. Ein FIFO-Buffer soll dabei selbstverständlich einen beliebigen Typ speichern können. Zudem soll eine Ausnahme deklariert werden, die bei einem Aufruf von `dequeue` oder `head` auf einem leeren FIFO-Buffer ausgelöst werden soll. Und zuletzt wird noch ein Wert für den leeren Buffer benötigt.

   Implementieren Sie die Signatur in einer Datei `9-2.sml`.

b) Implementieren Sie jetzt eine Struktur, die die in a) definierte Signatur umsetzt. Verwenden Sie dabei eine Liste als interne Repräsentation des FIFO-Buffers. Ergänzen Sie die Datei `9-2.sml` um ihre Implementierung.

c) Die Verwendung einer Liste bringt mit sich, dass `dequeue` und `head` (oder, alternativ, `enqueue`) eine lineare Laufzeitkomplexität haben. Das können wir zwar – ohne zusätzliche Hilfsmittel – nicht besser machen, aber es gibt eine einfache Datenstruktur, die garantiert, dass n Bearbeitungsschritte in einer Zeit linear in n durchführbar sind:

   Ein FIFO-Buffer wird durch zwei Listen `(a,b)` repräsentiert. Neue Elemente werden `a` vornangestellt; das älteste Element steht entweder am Anfang von `b` oder, falls `b` leer ist, am Ende von `a`. Sollte im zweiteren Fall ein lesender Aufruf stattfinden (`dequeue` oder `head`) soll die Liste `b` durch die umgedrehte Liste `a` ersetzt werden, und `a` auf die leere Liste gesetzt werden.

   Soll also `dequeue` auf dem Listenpaar `([1,2,3],nil)` aufgerufen werden, soll zuerst der FIFO-Buffer `(nil, [3,2,1])` erzeugt werden und dann das erste Element der zweiten Liste entfernt werden: `(nil, [2,1])`.

   Ergänzen Sie die Datei `9-2.sml` um eine weitere Struktur, die wie beschrieben funktioniert. Geben Sie (ausserhalb der Struktur) einige Tests an. Für das Umdrehen von Listen können Sie die Funktion `List.rev` verwenden.

Abgabe: Montag, den 6.7.2009, 12 Uhr, per UniWorx.