Funktionale Programmierung SS 2013 - Aufgabenblatt 3

Prof. Dr. U. Kastens
Institut für Informatik, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn
13.05.2013

Aufgabe 1 (Listenrepräsentation durch run-length-encoding)

Listen mit vielen aufeinanderfolgenden gleichen Elementen können durch run-length-encoding kompakter repräsentiert werden. Eine run-length kodierte Liste besteht aus Paaren, die jeweils den Wert eines Listenelements und die Anzahl seiner Wiederholungen enthält.

Beispiel: aus

```
["a","a","a","b","c","c","b","b"]
```

wird mit run-length-encoding

```
[("a",3), ("b",1), ("c", 2), ("b", 2)].
```

Schreiben Sie die Funktionen `rlencode` und `rlexpand`, die zu einer normalen Liste die run-length-kodierte Liste liefern und umgekehrt.
Hinweis: Zum Kodieren braucht man typischerweise entweder Hilfsfunktionen zum Zählen und Entfernen gleicher Elemente am Listenanfang oder einen Hilfsparameter, der das Zählen erlaubt. Wenn Sie Zeit haben, realisieren Sie beide Versionen und vergleichen Sie sie. Was sind vernünftige Vergleichskriterien?

Aufgabe 2 (Datentypen mit Konstruktoren: Verzeichnisbäume)

Betrachten Sie folgende Datentypdefinition:

```
(* Verzeichnisbäume mit einfachen Dateien und Verzeichnissen *)
datatype dirtree = file of string | dir of string * dirtree list;
```

und drei Werte von diesem Typ:

```
val d1 = dir("Urlaub-2011", [file("pic1.jpg"), file("pic2.jpg")]);
val d2 = dir("Urlaub-2012", [file("a.jpg"), file("b.jpg"), file("c.jpg")]);
val d3 = dir("MeineUrlaube", [file("sonne.jpg"), d1, d2, file("ich.jpg")]);
```

Schreiben Sie eine Funktion `filecount`, die die Anzahl der Dateien in einem Verzeichnisbaum liefert (ohne die Verzeichnisse).

Aufgabe 3 (Datentypen mit Konstruktoren: Kantorowitsch-Bäume)

a) Definieren Sie einen Datentyp für binäre Kantorowitsch-Bäume, mit denen wir arithmetische Ausdrücke über ganzen Zahlen modellieren wollen. An arithmetischen Operatoren sollen `plus`, `minus`, `times` und `divide` zulässig sein. Hier sind drei Beispiel-Werte:

```
val t1 = leaf(13);
val t2 = inner(t1, plus, leaf(12));
val t3 = inner(inner(leaf(2), times, t1), minus, t2);
```

b) Schreiben Sie eine Funktion `evaluate`, die den Wert des modellierten Ausdrucks liefert.
Hinweis: Der Operator für ganzzahlige Division ist `div`.

c) Formen Sie Ihren Datentyp in einen polymorphen Typen um, so dass an den Blättern beliebige Werte gespeichert werden können. Sind die Definitionen von `t1`, `t2`, `t3` und `evaluate` noch korrekt?