

Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion

Institut für Informatik
Christian-Albrechts-Universität zu Kiel
Prof. Dr. M. Hanus, S. Fischer



8. Übung zur Vorlesung „Prinzipien von Programmiersprachen“ Wintersemester 2006/2007

Abgabe: Dienstag, 19.12.2006 in der Vorlesung

Aufgabe 27

In dieser Aufgabe geht es um unterschiedliche Auswertungs-Strategien.

- (a) Wir betrachten folgendes funktionales Programm:

```
double x = x+x
```

Geben Sie **alle** möglichen Auswertungen des Ausdrucks `double (double 3)` an. Welches sind die leftmost-innermost (LI) und die leftmost-outermost (LO) Auswertungen?
Skizzieren Sie die lazy Auswertung von `double (double 3)` (nicht die Launchbury-Semantik!).

- (b) In der Vorlesung wurde die Semantik des `if-then-else` diskutiert. Zeigen Sie, dass es auch in einer strikten Sprache, wie ML wichtig ist, dass dieses Konstrukt nicht strikt in seinem zweiten und dritten Argument ist. Geben Sie hierzu die Semantik der in der Vorlesung definierten `fac`-Funktion für den Ausdruck `fac 1` an.

Aufgabe 28

Implementieren Sie in Haskell folgende Funktionen:

- (a) `rev :: [Int] -> [Int]` zum Umdrehen einer Liste von Zahlen.
`rev [1,2,3]` ergibt z.B. `[3,2,1]`.
Welche Komplexität hat ihre Implementierung (ggf. experimentelle Komplexitätsermittlung)? Können Sie eine lineare Implementierung angeben?
- (b) `insert :: Int -> [Int] -> [[Int]]`
zum Einfügen einer Zahl an allen möglichen Positionen einer Liste von Zahlen.
`insert 3 [1,2]` ergibt z.B. `[[3,1,2], [1,3,2], [1,2,3]]`
Geben Sie eine zweite Version dieser Funktion an, welche kein Pattern-Matching (auch nicht im `case`) verwendet.
- (c) `perm :: [Int] -> [[Int]]`
zur Berechnung aller möglichen Permutationen einer Liste von Zahlen.
`perm [1,2,3]` ergibt z.B. `[[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]`.
Ihre Funktion kann die Permutationen in der Ergebnisliste auch in einer anderen Reihenfolge ausgeben. Wichtig ist nur, dass alle Permutationen enthalten sind.

Aufgabe 29

Definieren Sie einen Datentyp `Exp` zur Repräsentation arithmetischer Ausdrücke in Haskell. Neben den zweistelligen Konstruktoren `Add`, `Sub`, `Mult` und `Div` soll `Exp` einen einstelligen Konstruktor `Num` zur Repräsentation von `Int`-Zahlen enthalten.

Implementieren Sie eine Funktion `eval :: Exp -> Int` zur kanonischen Auswertung eines Ausdrucks.

Machen Sie Ihr Programm robust gegen den “Division-durch-0-Fehler”. Definieren Sie hierzu einen Datentyp `OptInt`, welcher neben einem Ergebnis (vom Typ `Int`) auch den Fehlerwert repräsentiert. Unter Verwendung dieses Fehlertyps ändert sich der Typ der Auswertungsfunktion wie folgt: `eval :: Exp -> OptInt`. Passen Sie die Implementierung entsprechend an.