HTWK Leipzig, Fakultät IM

Prof. Dr. Sibylle Schwarz sibylle.schwarz@htwk-leipzig.de

Arbeitpaket zu KW 19 zum Modul „Fortgeschrittene Programmierung“
Sommersemester 2020

In dieser Woche geht es um Rekursion, die in funktionalen Sprachen eine zentrale Rolle spielt. Ergänzend zum Abschnitt im Buch lernen wir gleichzeitig die Deklaration rekursiver Datentypen kennen. Deklarationen von Aufzählungstypen mit `data` haben wir schon gesehen.
In dieser Woche kommt die Deklaration eines rekursiven Datentyps für Peano-Zahlen hinzu:

```
data Nat = Z | S Nat
```

In dieser Definition kommt der Typ `Nat` auf beiden Seiten vor, der Datentyp `Nat` ist also rekursiv. In rekursiven Datentypen gibt es jeweils Konstruktoren für

- Basiselemente (hier `Z` für das Basiselement 0) und
- die rekursiven Operationen (hier `S` für die Operation Nachfolger).

Darüber funktioniert das Pattern Matching in Funktionen auf solchen Datentypen, für jeden Konstruktor muss also ein Fall definiert werden, z.B.

```
double :: Nat -> Nat
double x = case x of
   Z    -> Z
   S x' -> S ( S ( double x' ) )
```

Auch hier erkennen wir das typische Muster der Rekursion, in der Definition von `double ( S x' )` kommt `double x'` vor.

Lesen und Verstehen

Kapitel 4 Rekursion als Entwurfstechnik
im Buch „Haskell-Intensivkurs - Ein kompakter Einstieg in die funktionale Programmierung“
Überprüfen Sie dabei wieder alle Beispiele mit `ghci`.
Ergänzend konsultieren Sie bitte die Folien auf der Homepage zum Modul. Diese werden auch in den Übungsaufgaben verwendet.

Begriffe und Testfragen

1) Was ist lineare Rekursion?
2) Welche Arten von Rekursion gibt es außerdem? Wodurch unterschieden sich diese?

Übungsaufgaben

Für Serie 4 gibt es insgesamt 10 Aufgaben-Punkte, je einen für Musterlösungen zu

- Teilaufgaben 4.1.b und 4.1.c (aber nicht für 4.1.a, Lösung dazu steht im Buch)
- je einen für jede Teilaufgabe von 4.3.a und 4.3.b
  Dabei können die „späteren“ Aufgaben die Aussagen aus den „früheren“ voraussetzen, also z.B. beim Beweis der Kommutativität der Addition deren Assoziativität voraussetzen (nicht jedoch beim Beweis der Assoziativität die Kommutativität).

Wieder sind die Aufgaben im Buch (Abschnitt 4.7) gut zum Selbsttest geeignet.

Autotool

Haskell-Programme mit Peano-Zahlen