

0. Übungsblatt

Ausgabe: 06.04.17

Abgabe: 13.04.17

Dieses Übungsblatt wird nicht bewertet und soll dazu dienen, sich mit der funktional-objektorientierten Sprache Scala bekannt zu machen. Wir raten dazu, dieses Übungsblatt dennoch zu bearbeiten, da wir im nächsten Übungsblatt direkt anfangen werden, mit Scala zu arbeiten.

0.1 *Arithmetische Ausdrücke*

Wir definieren einfache arithmetische Ausdrücke durch folgende BNF:

$$E ::= E + E \mid E * E \mid E - E \mid E / E \mid \text{Zahl} \mid \text{Variable}$$

1. Definieren Sie einen algebraischen Datentyp in Scala, der arithmetische Ausdrücke dieser Art repräsentiert.¹
2. Definieren Sie eine Methode `toString`, welche diese Ausdrücke als Zeichenkette formatiert.
3. Definieren Sie eine Methode `eval`, die einen arithmetischen Ausdruck zu einer ganzen Zahl auswertet. Variablen können dabei erstmal als 0 ausgewertet werden.
4. Erweitern Sie die Methode `eval`, so dass sie als zusätzliches Argument eine `Map` nimmt, welche Variablen Werte zuordnet.
5. Erweitern Sie die Methode `eval` weiter, so dass sie eine Option zurückgibt, die undefiniert ist (`None`), wenn durch 0 dividiert wird oder eine undefinierte Variable ausgewertet wird.
6. Definieren Sie eine Methode `subst`, welche zu einem Ausdruck eine Variable und einen zweiten Ausdruck nimmt, und alle Vorkommen der Variablen durch den zweiten Ausdruck ersetzt.

0.2 *Primzahlen*

Schreiben Sie die zwei Funktionen `isPrime` und `primeFactors`, die testen, ob eine gegebene ganze Zahl eine Primzahl ist, und die für eine gegebene Zahl die Primfaktoren berechnet:

```
def isPrime(int: Int): Boolean = ???  
def primeFactors(int: Int): List[Int] = ???
```

Nutzen sie nun ihre Implementation von `isPrime` um die Funktion `primeFactors` zu überprüfen. Folgendes sollte für alle $x \geq 1$ gelten:

```
(primeFactors(x) == Nil) == (x == 1)  
primeFactors(x).forall(isPrime)  
primeFactors(x).product == x
```

¹Sie brauchen keinen Parser für die BNF zu schreiben.