Praktische Informatik 1 WS 97/98

Prof. Bernd Krieg-Brückner
Heino Gärtner
Shi Hui
Kolyang
Christoph Lüth

2. Übungsblatt

Ausgabe: 6. November 1997

Bearbeitungszeit: zwei Praktikumstermine

$\boxed{1}$ *Wurzelbehandlung* *Wichtung: 30%*

Die positive ganzzahlige Wurzel einer natürlichen Zahl n ist eine Zahl k, so daß gilt:

$$k^2 \leq n \wedge n < (k+1)^2$$

Definiert eine Funktion `wurzel`, die sie berechnet, also: `wurzel n = k`.

$\boxed{2}$ *It's Logic, Captain, but not as we know it.* *Wichtung: 30%*

Der Haskell-Typ `Bool` hat zwei Werte, nämlich `True` und `False`. Nun ist es aber nicht gesagt, daß immer alles wahr oder falsch sein muß; die Welt ist meistens etwas komplizierter. Deshalb gibt es die sogenannte *mehrwertige Logik*. Wir werden in dieser Aufgabe die *dreiwertige Logik* betrachten, die außer `True` und `False` noch einen dritten Wert, `Maybe`, kennt. Die logischen Operatoren heißen jetzt `not3`, `and3` und `or3`. Aber wie sollte man sie definieren — welchen Wert sollte z.B. `False and3 Maybe` haben? Oder `not3 Maybe`?

1. Überlegt Euch plausible Werte für die logischen Operatoren für die dreiwertige Logik, und definiert diese anhand dreier Wertetabellen. Begründet Eure Definitionen.

2. Definiert den Aufzählungstypen `Boolean3` mit den drei Werten `True`, `False` und `Maybe`.

3. Definiert die Funktionen

```
not3 :: Bool3-> Bool3
and3 :: Bool3-> Bool3-> Bool3
or3  :: Bool3-> Bool3-> Bool3
```

welche die Definition aus der ersten Teilaufgabe oben implementieren.

$\boxed{3}$ *Organische Primzahlgewinnung* *Wichtung: 40%*

Primzahlen spielen eine wichtige Rolle in der Zahlentheorie, und auch in der Informatik (z.B. für Verschlüsselungsverfahren). Das folgende Verfahren, das sogenannte *Sieb des Eratosthenes* stammt von dem griechischen Computerpionier Eratosthenes, der vor zweitausend Jahren die Firma $\iota\beta\mu$, den Vorläufer eines heute weltweit tätigen Computerkonzerns, gründete, und bestimmt alle Primzahlen in einem Abschnitt von 1 bis n der natürlichen Zahlen.

- man fängt an mit der kleinesten Primzahl 2, und streiche alle Vielfachen von 2 aus dem Abschnitt;
- dann setze man das Verfahren mit der nächsten verbliebenen Zahl in dem Abschnitt fort, solange bis man zum Ende des Abschnitts gelangt.

Die übrigbleibenden Zahlen sind alles Primzahlen. Wir wollen jetzt das Sieb des Eratosthenes als Haskell-Funktion implementieren. Dazu wird der Abschnitt der natürlichen Zahlen als eine Liste modelliert, was sich in Haskell z.B. als `[1..100]` schreiben läßt. Der Algorithmus wird dann schrittweise durch folgende Funktionen definiert:

1. Definiere eine Funktion

```
divides :: Int-> Int-> Bool
```

die `True` liefert genau dann, wenn das erste Argument das zweite ohne Rest teilt, also z.B.

```
divides 5 25 = True
divides 3 7  = False
```

2. Unter Benutzung von `divides` definiere eine Funktion

```
excludeMultiples :: Int-> [Int]-> [Int]
```

die alle ganzzahligen Vielfachen des ersten Arguments aus der Argumentliste entfernt, z.B.

```
excludeMultiples 4 [3, 4, 7, 8, 8, 10, 12, 17] = [3, 7, 10, 17]
```

3. Unter Benutzung von `excludeMultiples` definiere eine Funktion

```
sieve :: [Int]-> [Int]
```

welche die Argumentliste, anfangend mit der ersten Zahl in der Liste, wie oben beschrieben "durchsiebt". Darauf aufbauend, definiere eine Funktion

```
primes :: Int-> [Int]
```

welche für n alle Primzahlen zwischen 1 und n zurückgibt.