Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
L. Noschinski, L. Hupel, Dr. J. Blanchette, M. Eberl

WS 2014/15
09.10.2014
Abgabe: 17.10.2014, 08:00

Einführung in die Informatik 2
1. Übung

Aufgabe G1.1 Auswertung

1. Definieren Sie eine Funktion

   ```
   threeAscending :: Integer -> Integer -> Integer -> Bool
   ```

   die genau dann `True` zurückgibt, wenn die Parameter in der angegebenen Reihenfolge streng monoton steigend sind.

2. Definieren Sie eine Funktion

   ```
   fourEqual :: Integer -> Integer -> Integer -> Integer -> Bool
   ```

   die genau dann `True` zurückgibt, wenn alle Parameter gleich sind.

3. Geben Sie zu den beiden folgenden Ausdrücken je eine zeilenweise Auswertung an:

   ```
   threeAscending (2+3) 5 (11 `div` 2)
   fourEqual (2+3) 5 (11 `div` 2) (21 `mod` 11)
   ```

Aufgabe G1.2 This one goes to 11

1. Schreiben Sie eine rekursive Funktion `fac :: Integer -> Integer`, die die Fakultätsfunktion berechnet.
2. Schreiben Sie eine Funktion `sum_eleven :: Integer -> Integer`, die für den Parameter n die Summe $\sum_{i=n}^{n+10} i$ berechnet.

   Hinweis: Gegebenenfalls ist eine Hilfsfunktion nützlich.

Aufgabe G1.3 Maximum

Gegeben sei eine Funktion `g` mit

```
g :: Integer -> Integer
g n = if n < 10 then n*n else n
```

1. Definieren Sie eine rekursive Funktion `max_g :: Integer -> Integer`, so dass `max_g n` die Zahl $0 \leq$ `i` $\leq$ `n` ausgibt, für die `g i` am größten ist. Schreiben Sie diese Funktion so, als ob Sie nichts über `g` wüssten.
2. Schauen Sie sich die Funktion `g` an und überlegen Sie, wann `max_g n` $\neq$ `n` gilt. Verwenden Sie dies, um einen QuickCheck-Test für `max_g` zu schreiben.

Aufgabe H1.1 Ohnegleichen (Wettbewerbsaufgabe)

Diese Aufgabe ist vom Master of Competition Senior gestellt. Sie wird als Hausaufgabe korrigiert, zählt aber zusätzlich als Teil des informalen Wettbewerbs.[1]

Implementieren Sie eine Funktion `allDistinct :: Integer -> Integer -> Integer -> Integer -> Integer -> Bool`, die fünf Argumente bekommt und entscheidet, ob alle Argumente ungleich sind.

Beispiele:

```
allDistinct 1 2 3 4 5   -- liefert True zurück
allDistinct 1 2 3 2 5   -- liefert False zurück
allDistinct 1 2 3 2 1   -- liefert False zurück
```

Im Wettbewerb gewinnt die kürzeste Lösung im Hinblick auf die Anzahl der Token. Token sind Identifier, Operatoren, Klammern, Zahlen usw.[2] Die Länge der Identifier ist unerheblich. Der offizielle Tokenzähler ist auf der Vorlesungswebseite erhältlich.[3] Alle Funktionen aus der Haskell-Standardbibliothek sowie eigene Hilfsfunktionen sind erlaubt. Die vollständige Lösung (außer Standardfunktionen) muss innerhalb der Kommentare {-WETT-} und {-TTEW-} abgegeben werden, um als Wettbewerbsbeitrag zu zählen. Zum Beispiel:

```
{-WETT-}
allDistinct :: Integer -> Integer -> Integer -> Integer -> Integer -> Bool
allDistinct a b c d e = ...
{-TTEW-}
```

Der MC Sr. hat für seine Lösung 24 Token (exklusive Typsignaturen) gebraucht. Unterbieten Sie ihn!

> **Wichtig:** Wenn Sie diese Aufgabe als Wettbewerbsaufgabe abgeben, stimmen Sie zu, dass Ihr Name ggf. auf der Ergebnisliste auf unserer Internetseite veröffentlicht wird. Sie können diese Einwilligung jederzeit widerrufen, indem Sie eine Email an `fp@fp.in.tum.de` schicken. Wenn Sie nicht am Wettbewerb teilnehmen, sondern die Aufgabe allein im Rahmen der Hausaufgabe abgeben möchten, lassen Sie bitte die {-WETT-} ... {-TTEW-} Kommentare weg. Bei der Bewertung Ihrer Hausaufgabe entsteht Ihnen hierdurch kein Nachteil.

Aufgabe H1.2 Rekursion

Die Funktion `f :: Integer -> Integer -> Integer` sei folgendermaßen auf den natürlichen Zahlen definiert:

[1] `http://www21.in.tum.de/teaching/info2/WS1415/wettbewerb.html`
[2] `http://www21.in.tum.de/teaching/info2/WS1415/wettbewerb.html#token`
[3] `http://www21.in.tum.de/teaching/info2/WS1415/Tokenize.hs`

$$f\ m\ n = \begin{cases} 0 & \text{für } m = 0 \text{ oder } n = 0 \\ 1 + f\ (f\ (n-1)\ m)\ (f\ (m-1)\ n) & \text{sonst} \end{cases}$$

1. Setzen Sie die obige rekursive Funktionsdefinition in Haskell um. Für negative Parameter darf Ihre Funktion sich beliebig verhalten.
2. Evaluieren Sie Ihre in 1 definierte Funktion für verschiedene nichtnegative Parameter. Stellen Sie anhand der Ergebnisse eine Vermutung auf, was diese Funktion berechnet.
3. Prüfen Sie Ihre in 2 aufgestellte Vermutung mithilfe eines geeigneten QuickCheck-Tests. Beachten Sie hierbei, dass der QuickCheck-Test auf den Definitionsbereich (d.h. nichtnegative Zahlen) eingeschränkt werden muss.

   Hinweis: QuickCheck-Eigenschaften lassen sich mithilfe des „`==>`“-Operators einschränken, z.B. `x >= 0 ==> x^3 >= 0`.

Aufgabe H1.3 Potenzen

Eine Potenz a^b können Sie in Haskell mit dem Operator `^` berechnen, z.B. ergibt `2 ^ 3` den Wert `8`. Definieren Sie eine Funktion `isPower :: Integer -> Integer -> Bool`, wobei der Aufruf `isPower` x a genau dann `True` zurückliefern soll, wenn es eine positive Ganzzahl b gibt, so dass $x = a^b$ gilt. Sie dürfen annehmen, dass alle Eingaben positiv sind.