

Algorithmen und Programmierung I

WS 2004 / 2005

Übung 3 (25 + 5 Punkte)

Ausgabe: Mo. 8.11.2004

Abgabe: Mi. 17.11.2004, 10.00

Verspätet abgegebene Lösungen werden nicht akzeptiert.

Die Abgabe erfolgt in Zweiergruppen. Die Gruppe darf nur mit ausdrücklicher Genehmigung des Tutors gewechselt werden.

Abgegeben werden müssen

- Alle Lösungen einmal ausgedruckt. **Handschriftliche Lösungen werden nicht akzeptiert.**
- Programme, Testdaten, Testergebnisse in elektronischer Form in Absprache mit den Tutorinnen und Tutoren.

Aufgabe 2.7 siehe 2. Übungszettel

Aufgabe 3.1 (5 Punkte)

Schreiben Sie den Ausdruck

- $a + b * 3 * c - (d + 3)$ in korrekter Funktionsnotation von Haskell. a,b,c,d seien schon definierte Konstanten.
- $(==>)$ praemisse $((\&\& x (|| y z))$, x, y und z sind Variablen. Gesucht ist ein syntaktisch korrekter Ausdruck mit den folgenden Variablenbindungen $x \leftarrow \text{True}$, $y \leftarrow \text{True}$, $z \leftarrow \text{False}$

Sind die folgenden Ausdrücke korrekt? Wert angeben, wenn korrekt, sonst kurze Begründung, maximal ein Satz)

- $3 + 2 * 5$
- $(+) 3 (2*5)$
- $(+) 3 (2^*) 5$
- $(+) 3 2 * 5$
- $(3+) (2^*) 5$
- $(3+2 *) 5$
- $(3+2) * 5$

j) Gegeben die folgenden Ausdrücke::

$2^7 + 2^3 + 1$, $2^9 + 2^2 + 1$, geben Sie mindestens 3 gemeinsame Abstraktionen an

$3^2 + 5$, $3^3 * 5$, gesucht sind alle möglichen gemeinsamen Abstraktionen

Unter "gemeinsamen Abstraktionen" werden Abstraktionen verstanden, unter denen die entstehenden anonymen Funktionen gleich sind.

Aufgabe 3.2 (5 Punkte)

Es soll eine Funktion zur Berechnung von 2^n geschrieben werden. Für gerade n, etwa $n=2*k$ gilt:

$2^n = 2^{2*k} = 2^{(k)^2}$, (lies: $2^{(2*k)} = (2^k)^2$ oder $2 \text{ hoch } (2*k) = (2 \text{ hoch } k) \text{ hoch } 2$).

für ungerade gilt: $2^n = 2^{2*k+1} = 2^{2*k} * 2$

Schreiben Sie eine rekursive Funktion, die diese Eigenschaften ausnutzt. Vergleichen Sie die subjektive Laufzeit für größere Argumente, z.B. 2^{100000} mit der "naiven" Implementierung

```
power2 n = if (n == 0) then 1 else 2*power2 (n-1)
```

Aufgabe 3.3 (5 Punkte)

Bekanntlich können sich Funktionen in ihrer Definition auf eine oder mehrere Funktionen abstützen.

a) Entwerfen Sie eine einfache Sprache, mit der diese Beziehung zwischen Programmen ausgedrückt werden kann (z.B. $P \rightarrow Q, R$; für P stützt sich auf Q und R ab.). Stützt sich P auf sich selbst ab, wird das gekennzeichnet, z.B durch einen Strich ('): $P \rightarrow Q, R, P'$; . Definieren Sie die Syntax dieser Sprache in BNF.

b) Gegeben eine Menge solcher "Stützen". Geben Sie einen Algorithmus in Umgangssprache an, der für ein Programm P alle Programme findet, auf die sich P direkt oder indirekt abstützt. Mehrfaches Vorkommen eines Q in der Liste der "Stützen" von P ist erlaubt. **Voraussetzung: alle Programme sind nichtrekursiv.**

c) Verändern Sie den Algorithmus so, dass kein Programm mehrfach auftreten kann.

Aufgabe 3.4 (5 Punkte)

a) Gegeben sind die folgenden Definitionen:

```
add1 (x,y) = x + y
```

```
add2 x y = x + y
```

Welche der folgenden Ausdrücke haben den Wert True oder False bzw. sind undefiniert oder fehlerhaft?

Begründen Sie Ihre Aussage jeweils knapp.

(1) `add1 == add2`

(2) `add1 (1,2) == add2 2 1`

(3) `add2 2 == add1 (2,0)`

(4) `add1 (3,4) == (add2 3) 4`

(5) `add1 (1) (2) == add2 2 1`

(6) `add2 (2) (1) == add1 (1,2)`

b) Definieren Sie eine Funktion `haskellCurry`, die eine Funktion $f :: (a,b) \rightarrow c$ in eine Funktion $f' :: a \rightarrow b \rightarrow c$ transformiert. Der Wert von f für (a,b) soll der gleiche sein wie für f' mit den Argumenten a und b. Definieren Sie außerdem eine Umkehrfunktion `haskellUnCurry`, die das Gegenteil bewirkt (Signatur angeben!)

Aufgabe 3.5 (5 Punkte).

Die Funktion zur Bestimmung der Quadratwurzel (A. 2.7) soll verallgemeinert werden. Abstrahieren Sie dazu von der speziellen Funktion "Quadratwurzel" und definieren Sie eine Funktion `newtonIteration`, die für eine beliebige Funktion eine Nullstelle bestimmt. Testen Sie Ihre Funktion, in dem sie die Quadrat- und die Kubikwurzel ziehen, ferner mit der Funktion, anhand derer Newton sein Verfahren demonstrierte: $y^3 - 2y - 5 = 0$.

Hinweis: Da wir die Ableitung einer Funktion f, z.B. des obigen Polynoms, jetzt noch nicht symbolisch differenzieren können (das ist eine etwas größere Übung ;) ist es nötig, den Wert der Ableitung an der Stelle x_i ebenfalls zu approximieren. Die Approximation ergibt sich direkt aus der Definition der Ableitung einer Funktion an der Stelle x_i :

$$f'(x_i) = \lim_{h \rightarrow 0} (f(x_{i+h}) - f(x_i)) / h$$

$f'(x_i)$ wird einfach also durch den Differenzenquotienten mit kleinem h approximiert. Die Berechnung geschieht am Besten mit einer separaten Funktion, auf die sich `newtonIteration` abstützt.