Probeklausur 1

Programmierung 1

Hinweise

- Schreiben Sie ihren Namen und ihre Matrikelnummer auf jedes Blatt, das in die Wertung eingehen soll.
- Schreiben Sie *lesbar* und *ordentlich*. Unleserliches wird nicht gewertet.
- Die Aufgaben haben in der Regel einfache Lösungen. Denken Sie zuerst eine Weile nach, bevor Sie anfangen zu schreiben! Hilfsprozeduren sind oft hilfreich.
- Die Punkteangaben sollten Ihnen als Richtlinie für den Zeitaufwand dienen.
- Auf den Aufgabenblättern sollte genug Platz sein, um die Lösungen zu notieren. Sollten Sie dennoch mehr Papier benötigen, melden Sie sich. Sie dürfen kein eigenes Papier verwenden.

Aufgabe 1 (Programmiersprachliches 10 Punkte)

a) Betrachten Sie das folgende Programm

```
val x = 3+2
fun f y = x+y
val x = f x
fun g x = if x<x then f x else 0
```

(1) **1 Punkt** Was berechnet `g 52`?

(2) **3 Punkte** Geben Sie die Tripeldarstellung von `g` an.

(3) **3 Punkte** Zeichnen Sie den Syntaxbaum zur Prozedur `g`.

(4) **1 Punkt** Geben Sie den Typen der Prozedur `f` an.

b) **2 Punkte** Betrachten Sie das folgende Programm

```
exception R
fun f x =
 let
   exception R
 in
   raise R
 end
val ergebnis = (f 5; true) handle R => false
```

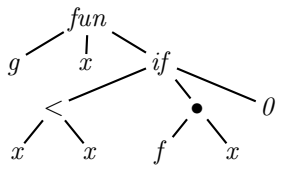
An welchen Wert wird der Bezeichner `ergebnis` gebunden, bzw. welche Fehlermeldung wird ausgegeben?

Lösung 1 (Programmiersprachliches 10 Punkte)

a) (1) 0

(2) `(fun g x = if x<x then f x else 0, int->int, [f:=(fun f y = x+y, int->int, [x:=5])])`

(3)



(4) $int \to int$

b) Es wird eine Ausnahme R geworfen, da die Ausnahme der zweiten Deklaration geworfen wird und *handle R* nur die Ausnahme der ersten Deklaration fängt.

Aufgabe 2 (Das Mysteriöse Programm 15 Punkte)

Geben Sie für alle Prozeduren, die Sie schreiben, den vollständigen Typen an.

```
val s = fn x => x+1
fun t x y = (x,y)

fun machBis p f t = if p t then t else machBis p f (f t)

fun f n =
 let
   val start = (t 0 1)
   fun schritt (i,m) = (s i,m * s i)
   val fertigWenn = fn (i,_) => i=n
   fun ergebnis (_,m) = m
 in ergebnis (machBis fertigWenn schritt start) end
```

a) **5 Punkte** Was ist der Typ der Prozedur `machBis`?

b) **10 Punkte** Geben Sie eine einfache Prozedur an, die semantisch äquivalent zur Prozedur `f` ist.

Lösung 2 (Das Mysteriöse Programm 15 Punkte)

a) $(\alpha \to bool) \to (\alpha \to \alpha) \to \alpha \to \alpha$

b)
```
fun fac n = if n<2 then 1 else n*fac (n-1)
```

Aufgabe 3 (Listen und Faltungen 15 Punkte)

Geben Sie für alle Prozeduren, die Sie schreiben, den vollständigen Typen an.

a) **6 Punkte** Ein Polynom vom Grad n ist eine Prozedur, die wie folgt definiert ist:

$$p(x) = \sum_{i=0}^{n} c_i x^i$$

Schreiben Sie eine Prozedur `poly`, die eine Liste mit den Koeffizienten `[c0,c1,...,cn]` sowie ein x als Argument bekommt und den Prozedurswert $p(x)$ ausrechnet.

b) **9 Punkte** Schreiben Sie mit Hilfe einer oder mehrerer Faltungen eine nichtrekursive Prozedur, die zu einem wie in 1 definierten Polynom die erste Ableitung an der Stelle x berechnet.

Zur Erinnerung

$$p(x)' = \sum_{i=1}^{n} i c_i x^{i-1}$$

Lösung 3 (Listen und Faltungen 15 Punkte)

a)
```
fun poly' _     nil a = 0
  | poly' x (y::yr) a = x*a*y + poly' x yr (a*x)

fun poly x ys = poly' x (rev ys) 0
```

Besonders schön fanden wir:

```
fun poly' cs x  = foldr (fn (y,a) => a*x+y) 0 cs
```

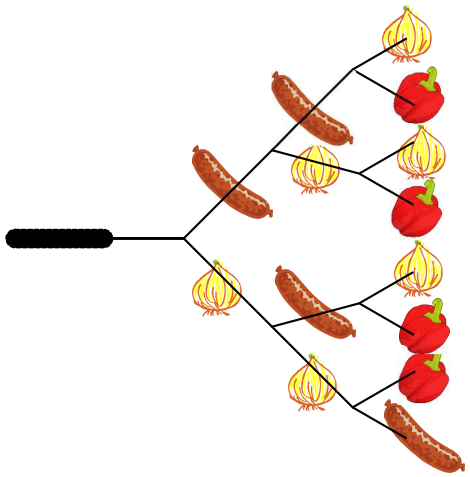
b)
```
fun ableitung cs = #2 (foldr (fn(c,(n,xs)) => (n-1,(c*n)::xs)) (
   length cs-1,nil) (tl cs))
```

```
fun abl cs x = foldr (fn (y,a) => a*x+y) 0 (ableitung cs)
```

Aufgabe 4 (Konstruktoren 25 Punkte)

Geben Sie für alle Prozeduren, die Sie schreiben, den vollständigen Typen an.

Sie haben eine grandiose Erfindung gemacht: Ein Grillspieß, auf den man nicht nur linear Dinge aufspießen kann. Er ist ein binärer Baum und sieht so aus



Und kann auf eine beliebige Größe erweitert werden. Toll!
Auf jeden Abschnitt Ihres Grillspießes kann man genau eine Zutat aufspießen. Die verfügbaren Zutaten sind:

Zutat	Köstlichkeit
Zwiebel	2
Paprika	1
Fleisch	3

Die Köstlichkeit einer Zutat gibt an, wie lecker sie ist.

a) **3 Punkte** Definieren Sie einen geeigneten Datentypen `Zutat`, der die verfügbaren Zutaten abbildet und eine regelbasierte Prozedur `mmh:Zutat->int`, die die Köstlichkeit einer Zutat berechnet!

b) **3 Punkte** Definieren Sie einen geeigneten Datentypen `GrSp`, der einen Grillspieß abbildet, auf den man wie oben beschrieben Zutaten aufspießen kann!

c) **10 Punkte** Man kann eine Zutat auf einem Grillspieß erst essen, wenn man alle Zutaten gegessen hat, die danach aufgespießt wurden. Geben Sie eine Prozedur `mampf` an, die die Zutaten eines Grillspießes in einer eßbaren Reihenfolge in einer Liste zurückgibt.

`mampf` soll dabei eine Ausnahme werfen, wenn die Reihenfolge nicht gesund ist. In einem gesunden Grillspieß isst man nicht zweimal hintereinander die gleiche Zutat.

Hinweis: Die Reihenfolge ist nicht eindeutig, mehrere Lösungen sind denkbar. Ob ein bestimmter Grillspieß gesund ist oder nicht, hängt also von ihrer Definition von `mampf` ab.

d) **9 Punkte** Die Köstlichkeit eines gesunden Grillspießes sei definiert als die durchschnittliche Köstlichkeit seiner Zutaten. Ein ungesunder Grillspieß hat die Köstlichkeit 10000.

Schreiben Sie eine Prozedur, die testet ob ein Grillspieß weniger köstlich als ein anderer ist.

Hinweis: Sie dürfen runden, wenn Sie möchten. Sie dürfen die Prozedur `mampf`. benutzen.

Lösung 4 (Konstruktoren 25 Punkte)

a)
```
datatype Zutat = Fleisch | Zwiebel | Paprika

fun mmh Fleisch = 3
  | mmh Zwiebel = 2
  | mmh Paprika = 1
```

b)
```
datatype GrSp = Blatt | Knoten of Zutat*Zutat*GrSp*GrSp
```

c)
```
exception Ungesund
fun mampf gs = let
   fun toList (Blatt) = nil
     | toList (Knoten (z1,z2,l,r)) = toList l @ toList r @ [z1,z2]
   val li = toList gs
   val testDouble xs = foldl (fn (x,s) => if x=s then raise
      Ungesund else x) (hd xs) (tl xs)
 in
   (testDouble li; li)
 end
```

d)
```
fun leckerer gr1 gr2 = let
   fun koestl xs = (foldl op+ 0 (map mmh xs)) div List.length xs
   val l1 = koestl (mampf gr1) handle Ungesund => 10000
   val l2 = koestl (mampf gr2) handle Ungesund => 10000
 in l1 <= l2 end
```