ALP I Funktionale Programmierung
WS 2012/2013
Prof. Dr. Margarita Esponda

Endklausur

Name: ..... **Vorname:**..... **Matrikel-Nummer:**.....

1. Aufgabe (12 Punkte)

a) Was verstehen Sie unter dem Begriff ***Referentielle Transparenz***?

Mögliche Lösungen:

- Eine Funktion liefert immer bei gleicher Eingabe das gleiche Ergebnis.
- Der Wert eines Ausdrucks hängt nur von den Werten der aktuellen Parameter ab.
- Keine Seiteneffekte.

b) Definieren Sie eine ***uncurrifizierte*** Version der **map**-Funktion.

Lösung:

```
map :: (a -> b) -> [a] -> [b]
map  (_, [])     =  []
map  (f, (x:xs)) =  f x : map (f, xs)
```

c) Was ist ein statisches Typsystem im Kontext von Programmiersprachen?

Lösung:

Der Datentyp der Funktionen wird statisch während der Übersetzungszeit des Programms abgeleitet.

Es findet eine statische Bindung der Datentyp der Funktionen auf Grund von explizite Deklarationen oder Typinferenz.

Nachdem der Datentyp einer Funktion oder Ausdruck festgelegt wird, kann keine Veränderung mehr statt finden.

d) Definieren Sie eine möglichst einfache Funktion, die nach mindestens einem ihrer Argumente ***strikt*** ist. Erläutern Sie kurz Ihre Antwort.

Lösung: add a b = a + b

Begründung: $add \perp b = \perp$ oder $add\ a \perp = \perp$

2. Aufgabe (32 Punkte)

a) Definieren Sie einen (\\)-Operator, der aus einer Liste **xs** alle Elemente, die sich in der Liste **ys** befinden, entfernt.

Lösung:

```
(\\) :: (Eq a) => [a] -> [a] -> [a]
(\\) [] ys = []
(\\) (a:xs) ys | elem a ys   = (\\) xs ys
               | otherwise   = a:(\\) xs ys
```

b) Programmieren Sie mit Hilfe Ihres (\\)-Operators eine Funktion, die aus einer beliebigen Liste natürlicher Zahlen die kleinste natürliche Zahl findet, die **nicht** in der Liste vorkommt.

Anwendungsbeispiel: **firstNatNotIn** [2, 0, 1, 8, 9, 12, 6] => 3

Lösung:

```
firstNatNotIn :: [Integer] -> Integer
firstNatNotIn xs = head ([0..] \\ xs)
```

c) Was ist die Komplexität der Funktionen in a) und b)? Erläutern Sie Ihre Antwort.

Lösung:

a) $O(n^2)$ + Begründung ...

b) $O(n^2)$ + Begründung ...

3. Aufgabe (6 Punkte)

Die **freq**-Funktion berechnet, wie oft ein angegebenes Element in einer Liste vorkommt.

Anwendungsbeispiele: **freq** 3 [2, 1, 4, 8, 9, 3, 3, 0] => 2
freq 'a' "abcdea abda" => 4

Schreiben Sie eine Definition der **freq**-Funktion, in der nur die **foldl**-Funktion und eine anonyme Funktion verwendet wird.

Lösung:

```
freq :: (Eq a) => a -> [a] -> Integer
freq a  = foldl (\x y -> if (y==a) then x+1 else x) 0
```

4. Aufgabe (20 Punkte)

Betrachten Sie folgende Funktionsdefinitionen:

```
(.)  ::  (b -> c) -> (a -> b) -> a -> c
(.)  f  g  a  =  f (g  a)                          ----------------  (.) .1

filter :: (a -> Bool) -> [a] -> [a]
filter p []        =  []                           ----------------  filter.1
filter p (x:xs)  | p x         =  x:filter p xs    ----------------  filter.2
                 | otherwise =  filter p xs        ----------------  filter.3

map :: (a -> b) -> [a] -> [b]
map  _  []       =  []                             ----------------  map.1
map  f  (x:xs)   =  f x : map f xs                 ----------------  map.2
```

Zeigen Sie mittels struktureller Induktion über Listen, dass für jede endliche Liste **xs** gilt:

(filter p) **.** (map f) = (map f) **.** (filter (p **.** f))

Lösung:

Zuerst zeigen wir die Eigenschaft für die leere Liste [].

$$\begin{aligned}
(filter\ p)\cdot(map\ f)\ [] &=_{(.).1} \quad filter\ p\ (map\ f\ []) \\
&=_{map.1} \quad filter\ p\ [] \\
&=_{filter.1} \quad [] \\
&=_{map.1} \quad map\ f\ [] \\
&=_{filter.1} \quad map\ f\ (filter\ (p.f)\ []) \\
&=_{(.).1} \quad (map\ f)\cdot(filter\ (p.f))\ []
\end{aligned}$$

I.V: Die Eigenschaft gilt für eine beliebige Liste der Länge n.

$$(filter\ p)\ .\ (map\ f)\ xs\ =\ (map\ f)\ .\ (filter\ (p\ .f))\ xs$$

Zur zeigen ist:

$$(filter\ p)\ .\ (map\ f)\ (\mathrm{x}{:}xs)\ =\ (map\ f)\ .\ (filter\ (p\ .f))\ (\mathrm{x}{:}xs)$$

mit x irgendein Element mit geeigneten Datentyp:

1. Fall p (f x) = (p.f) x ist falsch (False)

$$\begin{aligned}
(filter\ p)\cdot(map\ f)\ (x:xs) &=_{(.).1,\ map.2} \quad filter\ p\ ((f\ x):\ map\ xs) \\
&=_{filter.3} \quad filter\ p\ (map\ f\ xs) \\
&=_{(.).1} \quad (filter\ p)\cdot(map\ f)\ xs \\
&=_{I.V.} \quad (map\ f)\cdot(filter\ (p\cdot f)\ xs) \\
&=_{(.).1} \quad map\ f\ (filter\ (p\cdot f)\ xs) \\
&=_{filter.3} \quad map\ f\ (filter\ (p\cdot f)\ (x:xs)) \\
&=_{(.).1} \quad (map\ f)\cdot(filter\ (p\cdot f))\ (x:xs)
\end{aligned}$$

2. Fall p (f x) = (p.f) x ist wahr (True)

$$\begin{aligned}
(filter\ p)\ .\ (map\ f)\ (x:xs) &=_{(.).1} \quad filter\ p\ (f\ x:\ map\ f\ xs) \\
&=_{filter.2} \quad f\ x\ :\ filter\ p\ (map\ f\ xs) \\
&=_{(.).1} \quad f\ x\ :\ (filter\ p)\cdot(map\ f)\ xs \\
&=_{I.V.} \quad f\ x\ :\ (map\ f)\cdot(filter\ (p.f)\ xs) \\
&=_{(.).1} \quad f\ x\ :\ map\ f\ (filter\ (p\cdot f)\ xs) \\
&=_{map.2} \quad map\ f\ (x\ :\ filter\ (p\cdot f)\ (xs)) \\
&=_{filter.2} \quad (map\ f)\cdot(filter\ (p\cdot f))\ (x:xs) \\
&=_{(.).1} \quad (map\ f)\cdot(filter\ (p\cdot f))\ (x:xs)
\end{aligned}$$

5. Aufgabe (18 Punkte)

a) Reduzieren Sie den folgenden Lambda-Ausdruck zur Normalform:

$$(\lambda xy.\, xy(\lambda xy.\, \mathrm{y}))(\lambda xy.\, x)(\lambda xy.\, \mathrm{y})$$

Lösung:

$$\begin{aligned}(\lambda xy.\, xy(\lambda xy.\, \mathrm{y}))(\lambda xy.\, x)(\lambda xy.\, \mathrm{y}) &\Rightarrow_\beta (\lambda y.\, (\lambda xy.\, \mathrm{x})\ y\ (\lambda xy.\, y))(\lambda xy.\, \mathrm{y}) \\ &\Rightarrow_\beta (\lambda xy.\, \mathrm{x})(\lambda xy.\, \mathrm{y})(\lambda xy.\, y) \\ &\Rightarrow_\beta (\lambda y.\, (\lambda xy.\, \mathrm{y}))(\lambda xy.\, y) \\ &\Rightarrow_\beta (\lambda xy.\, \mathrm{y}) \\ &\equiv F\end{aligned}$$

b) Schreiben Sie einen Lambda-Ausdruck, der die Länge einer Liste berechnet. Sie können dabei die Funktionen **S**(Nachfolger), **NIL**(Test ob leere Liste), **TAIL** und **Y**(Fixpunkt-Operator) als gegeben verwenden.

Lösung:

$$Y(\lambda rl.\{NIL\}l0(S(r(\{TAIL\}l))))$$

b) Zeigen Sie, dass folgende Lambda- und Kombinatoren-Ausdrücke äquivalent sind.

$$\lambda x.\lambda y.(xx) \quad \equiv \quad S(KK)(SII)$$

1. Lösung:

$$\begin{aligned}T\left[\lambda x.\lambda y.(xx)\right] &\Rightarrow_5 T\left[\lambda x.\, T\left[\lambda y.(xx)\right]\right] \\ &\Rightarrow_3 T\left[\lambda x.\, K\ T\left[xx\right]\right] \\ &\Rightarrow_{6,1} T\left[\lambda x.\, K\ (xx)\right] \\ &\Rightarrow_4 S\ T\left[(\lambda x.\, K)\right]\ T\left[(\lambda x.\, xx)\right] \\ &\Rightarrow_3 S\ (KK)\ T\left[(\lambda x.\, xx)\right] \\ &\Rightarrow_4 S\ (KK)\ (S\ T\left[(\lambda x.x)\right]\ T\left[(\lambda x.x)\right]) \\ &\Rightarrow_{2,2} S\ (K\ K)\ (S\ I\ I)\end{aligned}$$

2. Lösung:

$$\lambda x.\lambda y.(xx)\ a\ b \Rightarrow \lambda y.(aa)\ b \Rightarrow a\ a$$

$$\begin{aligned} S(KK)(SII)\ a\ b &\Rightarrow (K\ K)\ a\ ((SI\ I)\ a)\ b \\ &\Rightarrow K\ ((SI\ I)\ a)\ b \\ &\Rightarrow (S\ I\ I)\ a \\ &\Rightarrow I\ a\ (I\ a) \\ &\Rightarrow a\ a \end{aligned}$$

6. Aufgabe (12 Punkte)

Zeigen Sie, dass die Funktion **gerade**, die entscheiden kann, ob eine natürliche Zahl gerade ist oder nicht, primitiv rekursiv ist, indem Sie diese nur unter Verwendung von primitiv rekursiven Funktionen definieren. Das bedeutet, wenn Sie für die Definition andere Hilfsfunktionen verwenden, müssen Sie auch zeigen, dass diese primitiv rekursiv definierbar sind.

Lösung:

$$\begin{aligned} equalZero\ (\ 0\ ) &= (S \circ Z)\ () \qquad oder = C_1^0 \\ equalZero\ (\ S(n)\ ) &= Z\ (equalZero(n),\ n\ ) \end{aligned}$$

$$equalZero = not$$

$$\begin{aligned} gerade\ (\ 0\ ) &= (S \circ Z)\ () \qquad oder = C_1^0 \\ gerade\ (\ S(n)\ ) &= not\ (\ \pi_1^2(\ gerade\ (\ n\ ),\ n\ )\ ) \end{aligned}$$

Transformationsregeln, um Lambda-Terme in SKI-Terme zu verwandeln.

1)	T[x]	=> **x**	
2)	T[λx.x]	=> **I**	
3)	T[λx.*E*]	=> (**K** T[*E*])	wenn x nicht frei in *E* ist
4)	T[λx.(E_1 E_2)]	=> (**S** T[λx.E_1] T[λx.E_2])	
5)	T[λx.λy.*E*]	=> T[λx.T[λy.*E*]]	falls x frei in *E* ist
6)	T[(E_1 E_2)]	=> (T[E_1] T[E_2])	