Allgemeine Fragen:

1. Was versteht man unter dem Begriff Referenzielle Transparenz?
2. Was ist ein statisches Typsystem im Kontext einer Programmiersprache?
3. Geben Sie eine Funktion an, die nach mindestens einem Ihrer Argumente strikt ist. Erklären Sie in diesem Zusammenhang den Begriff „bottom".
4. Definieren Sie eine uncurryfizierte Version der Funktion foldr.
5. Definieren Sie eine uncurryfizierte Version der Funktion filter.
6. Geben Sie den allgemeinsten Typ, den die Funktion f haben kann an.
   ```
   f a b c
     | a b = a b
     | otherwise = a c
   ```
   Begründen Sie Schritt für Schritt, wie Sie zu Ihrer Schlussfolgerung kommen.
7. Geben Sie den allgemeinsten Typ, den die Funktion f haben kann an.
   ```
   f a b c
     | b c = [c+2]
     | otherwise = a c
   ```
   Begründen Sie Schritt für Schritt, wie Sie zu Ihrer Schlussfolgerung kommen.
8.

Einfache Funktionen (Einstiegsaufgaben):

1. Definieren Sie eine rekursive Funktion getUntil, die als Argument ein Prädikat p und eine Liste bekommt und so lange Elemente der Liste, die das Prädikat nicht erfüllen, in die Ergebnisliste einfügt.
2. Die Funktion freq berechnet, wie oft ein angegebenes Element in einer Liste vorkommt. Schreiben Sie eine Definition der freq-Funktion, in der nur die foldl und eine anonyme Funktion verwendet wird.
3. Variationen: Schreiben Sie die folgenden Funktionen als nicht-endrekursive Funktion, als endrekursive Funktion und als Funktion höherer Ordnung (mit foldr/foldl):
   a. Fibonacci
   b. Fakultät
   c. sumQuad
4. Schreiben Sie eine Filterfunktion unter Verwendung eines Listengenerators.
5. Definieren Sie eine Funktion sumGerade, die eine Liste aus ganzen Zahlen bekommt und die Summe aller geraden Zahlen der Liste berechnet. Verwenden Sie dafür die foldl filter-Funktion und eine anonyme Funktion.

Funktionen (Fortgeschrittene Aufgaben):

1. Schreiben Sie ein Haskell-Programm, das in einer Liste vom Typ [Integer] ein Tripel (a,b,c) mit a+b=c findet, falls es eines gibt. Dabei müssen a,b,c von verschiedenen Stellen der Eingabeliste kommen. (Sie dürfen gleich sein, falls in der Eingabeliste gleiche ZTahlen mehrfach vorkommen.) Sie dürfen die Funktion sort verwenden, die eine Folge der Länge n in O(n*log(n)) sortiert.
   a. Schreiben Sie das Programm.
   b. Analysieren Sie die Laufzeit Ihrer Funktion.
2. Schreiben Sie eine Funktion tiefeKonst::Ausdruck->Int, die die Tiefe der am tiefsten geschachtelten Konstanten in einem arithmetischen Ausdruck bestimmt. Wenn der Ausdruck

keine Konstanten enthält, soll das Ergebnis -1 sein. Der Datentyp für arithmetische Ausdrücke ist folgendermaßen korrigiert:
data Ausdruck = Konst Int | Var String | Plus Ausdruck Ausdruck | Mal Ausdruck Ausdruck deriving (Eq,Show)

3. Schreiben Sie ein Haskell-Programm vielfachheit::[Int]-> Int, das für eine Eingabeliste x von nicht-negativen Zahlen zählt, wie oft jede Zahl in ihr enthalten ist. Das Ergebnis ist eine Lsite b, wobei b!!i angibt, wie oft i in x vorkommt. Sie dürfen die Funktion sort verwenden, die eine Folge der Länge n in O(n*log(n)) sortiert.
   Beispiel: vielfachheit [1,3,5,3,1] = [0,2,0,2,0,1]
   a. Schreiben Sie das Programm.
   b. Analysieren Sie die Laufzeit Ihrer Funktion.
4. In Atlantis wurde mit Gulden und Dukaten bezahlt. Die Münzen gab es in folgenden Größen:
   Dukaten: 1,2,7,13,23,53
   Gulden: 1,3
   Schreiben Sie eine Haskell-Funktion, die einen Betrag entgegen nimmt und diesen in Münzen zerlegt. Es sollen so wenig Münzen wie möglich zurückgegeben werden.
5. Schreiben Sie einen abstrakten Datentypen Stapel, der einen Stapel von Werten vom Typ a darstellt. Der Stapel soll folgende Operationen erlauben:
   a. leer::ST a --erzeugt einen leeren Stapel
   b. push:: (ST a, a) -> ST a --legt ein Element auf einen Stapel
   c. top:: ST a -> a --gibt das oberste Element vom Stapel zurück
   d. pop::ST a -> ST a --löscht das oberste Element vom Stapel
   e. istLeer::ST a -> Bool --prüft ob Stapel leer ist
6. Definieren Sie für folgenden algebraischen DT für binäre Suchbäume aus der Vorlesung data BSTree a = Nil | Node a (BSTree a) (BSTree a) die search Funktion, die eine Element x::a in einem Buam t::BSTree a sucht und die insert Funktion, die ein neues Element x::a in einen Baum t::BStTree einsortiert.

Komplexität

1. Analysieren Sie die Komplexität bezüglich der Reduktionen folgender Funktionen. Nehmen Sie an, dass die inSet-Funktion einen linearen Aufwand O(n) hat.

```
data Menge a = Menge [a]
insertSet x (Menge ys) | inSetx (Menge ys) = Menge ys
                       | otherwise = Menge (x:ys)
union (Menge []) m = m
union (Menge (x:xs)) m = insertSet x (union ( Menge xs) m)
```

Lambda-Kalkül:

1. Welche Variablen kommen im Ausdruck
   (\x.((\x.\a.(ax(xx)yx))(\a.xa))
   gebunden vor? Welche Variabeln kommen frei vor (bezogen auf den gesamten Ausdruck)? Reduzieren Sie den Ausdruck so weit wie möglich.
2. Reduzieren Sie den folgenden Lambda-Ausdruck zur Normalform:
   (\xy.xy(\xy.y))(\xy.x)(\xy.y)

3. Schreiben Sie einen Lambda-Ausdruck, der die Länge einer Liste berechnet. Sie können dabei die Funktion S (Nachfolger) , TNIL (Test, ob Liste leer), HEAD (Kopf einer Liste), TAIL (Rest einer Liste) und Y (Fixpunktoperator) als gegeben verwenden.
4. Schreiben Sie einen Lambda-Ausdruck, der die ersten n natürlichen Zahlen addiert. Sie können dabei die Funktionen P (Vorgänger) und Z (Vergleich auf Null) als gegeben verwenden.
5. Schreiben Sie die Fibonacci-Funktion in einer endrekursiven Version als Lambda-Ausdruck. Sie dürfen alle in der Vorlesung erwähnten Makros als gegeben verwenden.
6. Zeigen Sie, dass folgende Lambda- und Kombinatoren-Ausdrücke äquivalent sind.
   (\x.\y.xx) = S (S (KS) (S(KK)I)) (S(KK)I)
7. Zeigen Sie, dass folgende Lambda- und Kombinatoren-Ausdrücke äquivalent sind.
   (\x.y(yx)) = S(Ky)(S(Ky)I)

Induktion:

1. Welche Formel muss man für x einsetzen, damit die Gleichung
   drop m . drop n = drop x
   für alle m und n (auch für negative Werte) gilt?
   Beweisen Sie die Gleichung dann durch strukturelle Induktion.

```
drop n xs
    | n <= 0 = xs
drop _ [] = []
drop n (_:xs) = drop (n-1) xs
```

   Lsg für x = (max(0,m)+max(0,n))

2. Zeigen Sie mittels struktureller Induktion über Listen, dass für jede endliche Liste xs gilt:
   (filter p). (map f) = (map f) . (filter (p.f))

```
(.) f g a = f (g a)                    c.1
filter p [] = []                       f.1
filter p (x:xs)
    | p x = x:filter p xs              f.2
    | otherwise = filter p xs          f.3
map _ [] = []                          m.1
map f (x:xs) = f x : map f xs          m.2
```

3. Zeigen Sie mit struktureller Induktion über die Liste xs die Gültigkeit folgender Gleichung:
   take n xs ++ drop n xs = xs

   Definitionen:

```
drop 0 xs = xs
drop (n+1) [] = []
drop (n+1) (x:xs) = drop n xs

take 0 xs = []
take (n+1) [] = []
take (n+1) (x:xs) = x:(take n xs)
```

4. map f xs ++ map f ys = map f (xs++ys)

Primitive Rekursion

1. Zeigen Sie, dass die Funktion exp(x,y) = y^x mit x,y € N primitiv rekursiv ist.
2. Zeigen Sie, dass die Fakultätsfunktion primitiv rekursiv ist.
3. Zeigen sie dass die Funktion half (floor von n/2) primitiv rekursiv ist.

Sie dürfen keine Funktion als gegeben voraussetzen.