Prof. Dr. Manfred Schmidt-Schauß
Institut für Informatik

Fachbereich Informatik und Mathematik/ Institut für Informatik
Johann Wolfgang Goethe-Universität Frankfurt am Main

Einführung in die funktionale Programmierung

Wintersemester 2024/2025

Aufgabenblatt Nr. 3

Abgabe: Dienstag 26.11.24 in Moodle zu EFP

Bitte schicken Sie auch die Haskell-Quelltexte

Aufgabe 1 (15 Punkte)

Die folgenden KFPT-Ausdrücke seien gegeben:

a) $(\lambda y.((\lambda x.x)\ \texttt{True}))$

b) $((\lambda y.\texttt{case}_{\text{List}}\ y\ \texttt{of}\ \{(\texttt{Cons}\ a\ b) \to b;\ \texttt{Nil} \to \texttt{Nil}\})\ \texttt{True})$

c) $((\lambda z.\texttt{case}_{\text{Bool}}\ z\ \texttt{of}\ \{\texttt{True} \to \texttt{True};\ \texttt{False} \to (\texttt{True}\ \texttt{Nil})\})\ \texttt{True})$

d) $(\texttt{case}_{\text{Paar}}\ (\texttt{Paar}\ \texttt{True}\ (\lambda z.(z\ z)))\ \texttt{of}\ \{(\texttt{Paar}\ a\ b) \to (b\ b)\})$

e) $(\texttt{case}_{\text{Bool}}\ (\lambda x.\texttt{True})\ \texttt{of}\ \{\texttt{True} \to \texttt{False};\ \texttt{False} \to \texttt{True}\})$

f) $(\texttt{Cons}\ (\texttt{True}\ \texttt{True})\ (\texttt{Cons}\ \texttt{True}\ (\texttt{Cons}\ \texttt{True}\ \texttt{Nil})))$

Geben Sie für jeden der obigen Ausdrücke an, welche der folgenden Aussagen auf ihn zutreffen.

1. Der Ausdruck *ist eine WHNF.*
2. Der Ausdruck *ist eine FWHNF.*
3. Der Ausdruck *ist eine CWHNF.*
4. Der Ausdruck *hat eine WHNF.*
5. Der Ausdruck *hat eine FWHNF.*
6. Der Ausdruck *hat eine CWHNF.*
7. Der Ausdruck *ist dynamisch getypt.*
8. Der Ausdruck *ist direkt dynamisch ungetypt.*
9. Der Ausdruck *konvergiert.*

Für die Lösung der Aufgabe reicht eine Tabelle von folgender Form aus:

	1	2	3	4	5	6	7	8	9
a)									
b)									
c)									
d)									
e)									
f)									

Dabei sollte angekreuzt sein, welche Ausdrücke die entsprechende Eigenschaft erfüllen.

Aufgabe 2 (35 Punkte)

Der folgende Datentyp `Expr` stellt KFPT-Ausdrücke dar, die die Typen List und Bool verwenden. Er ist polymorph über dem Typ für die Variablen definiert:

```
data Expr a =
   Var a                                    -- x       = Var "x"
 | App (Expr a) (Expr a)                    -- (e1 e2) = App e1 e2
 | Lam a (Expr a)                           -- \x.e    = Lam "x" e
 | ListCons (Expr a) (Expr a)               -- a:as    = ListCons a as
 | ListNil                                  -- []      = ListNil
 | BoolTrue                                 -- True    = BoolTrue
 | BoolFalse                                -- False   = BoolFalse
 | CaseList (Expr a) (Expr a) (a,a,Expr a)-- case_List e of {[] -> e1; (x:xs) -> e2)
                                            -- = CaseList e (e1) ("x",xs",e2)
 | CaseBool (Expr a) (Expr a) (Expr a)    -- case_Bool e of {True -> e1; False -> e2)
                                            -- = CaseBool e e1 e2
  deriving(Eq,Show)
```

Auf der Webseite zur Vorlesung finden Sie den Haskell-Quellcode für den Datentyp, einige Beispiele und eine Funktion `substitute`, welche die Substitution durchführt (`substitute e1 e2 x` berechnet gerade `e1`[`e2`/`x`]).

Implementieren Sie in Haskell und zeigen Sie dass alle Ihre Funktionen mindestens einen Ausdruck sinnvoll verarbeiten, indem Sie das Ergebnis angeben.

a) eine der Funktionen `isFWHNF`, `isCWHNF` oder `isWHNF`, die einen KFPT-Ausdruck erwarten und prüfen, ob dieser eine FWHNF, eine CWHNF bzw. eine WHNF ist. (Jede der Funktionen hat den Typ `Expr a -> Bool`). (5 Punkte)
Als Beispiel: `isWHNF`:

```
isWHNF e =  case e of Lam x y -> True
              ListCons _ _ -> True
              ListNil   -> True
              BoolTrue -> True
              BoolFalse -> True
```

mit einem Testaufruf:

```
*Main> isWHNF (ListCons BoolFalse ListNil)
True
```

b) die Funktion `betaReduce :: Expr String -> Expr String`, die einen KFPT-Ausdruck erwartet und diesen – falls möglich – *unmittelbar* β-reduziert. Ist dies nicht möglich, so soll die Eingabe unverändert zurück gegeben werden. (5 Punkte)

c) die Funktion `caseReduce :: Expr String -> Expr String`, die einen KFPT-Ausdruck erwartet und diesen – falls möglich – *unmittelbar* case-reduziert. Ist dies nicht möglich, so soll die Eingabe unverändert zurück gegeben werden. (5 Punkte)

d) die Funktion `isUntyped :: Expr String -> Bool`, die einen KFPT-Ausdruck erhält und prüft, ob dieser *direkt dynamisch ungetypt* ist. (7 Punkte)

e) die Funktion `oneStepNormalOrder :: Expr String -> Expr String`, die einen KFPT-Ausdruck erhält und einen Normalordnungsreduktionsschritt für den Ausdruck ausführt. Ist kein solcher Schritt möglich, so soll die Eingabe unverändert zurück gegeben werden. (8 Punkte)

f) die Funktion `calcResult :: KFPTAusdruck -> Result`, die einen KFPT-Ausdruck erhält und diesen solange in Normalordnung auswertet, bis feststeht, ob der Ausdruck eine WHNF hat (in diesem Fall soll `TerminatesWith` e zurückgeliefert werden, wobei e die erhaltene WHNF ist) oder divergiert (in diesem Fall soll `Diverges` zurückgeliefert werden). Dabei sei `Result` der folgende Datentyp:

```
data Result = TerminatesWith (Expr String)| Diverges
 deriving(Eq,Show)
```

Die Funktion `calcResult` darf für getypte divergierende Ausdrücke nicht terminieren. (5 Punkte)

Hier die Kodierung von `substitute` als Hilfestellung.

```
-- ************************************************************************************
-- ************************************************************************************
-- ***** Substitution
-- *****
-- *****

-- substitute e1 e2 x  entspricht e1[e2/x]

substitute (Var y) e x
  | x==y = e
  | otherwise = Var y

substitute (App e1 e2) e x=
 App (substitute e1 e x) (substitute e2 e x)

substitute (Lam y e1) e2 x
  | y == x    = (Lam y e1)
  | otherwise = (Lam y (substitute e1 e2 x))

substitute (ListCons e1 e2) e x =
 ListCons (substitute e1 e x) (substitute e2 e x)

substitute (CaseBool e e1 e2) e3 x =
  CaseBool
   (substitute e e3 x)
   (substitute e1 e3 x)
   (substitute e2 e3 x)
```

```
substitute (CaseList e e1 (y1,y2,e2)) e3 x =
  CaseList
   (substitute e e3 x)
   (substitute e1 e3 x)
   (y1,y2, (if x `elem` [y1,y2]
             then e2
             else (substitute e2 e3 x)))
-- alle anderen Faelle
substitute e e3 x = e
-- *****
-- *****
-- ********************************************************************************
-- ********************************************************************************
```

Aufgabe 3 (15 Bonusunkte)

Zeigen Sie mit vollständiger Induktion, dass die Normalordnungsreduktion für folgenden Ausdruck nicht terminiert:
$((\lambda x.(x\ x))\ (\lambda x.(x\ x\ x)))$.