Philipps Universität Marburg

Übungen zur „Praktischen Informatik III“, WS 2008/09

Prof. Dr. R. Loogen · Fachbereich Mathematik und Informatik · Hans-Meerwein-Straße, D-35032 Marburg

Nr. 9, Abgabe: 17. Dezember 2008 vor der Vorlesung

20. Typen zu Ausdrücken — 3 Punkte

Bestimmen Sie (falls möglich) den allgemeinsten Typ der folgenden Ausdrücke:

(a) `map foldr` (b) `foldr map` (c) `foldr foldr`

21. Typklassen — 6 Punkte

Gegeben seien die folgenden Deklarationen:

```
-- verschiedene Baumdefinitionen
data BinTree a   = Leaf a  | Node (BinTree a) (BinTree a)
data LabTree l a = LLeaf a | LNode l (LabTree l a) (LabTree l a)
data STree a     = Empty   | Split a (STree a) (STree a)
data RoseTree a  = RNode a [RoseTree a]
data AExp a      = Val a | Plus (AExp a) (AExp a) | Mult (AExp a) (AExp a)
-- Typklasse Tree
class Tree t where
  subtrees :: t -> [t] -- liefert direkte Teilbaeume
```

(a) Geben Sie für jede Baumart eine Instanzendeklaration für die Klasse `Tree` an. / 1

(b) Definieren Sie die folgenden Funktionen: / 5

`depth :: Tree t => t -> Int` bestimmt die Tiefe eines Baumes.

`size :: Tree t => t -> Int` bestimmt die Knotenanzahl eines Baumes.

`dfs :: Tree t => t -> [t]` liefert die Liste *aller* Teilbäume in Präfixordnung.

`bfs :: Tree t => t -> [t]` liefert die Liste *aller* Teilbäume ebenenweise.

22. Fehlersuche in Robinson-Programm — 3 Punkte

Der Weihnachtsmann hat den in der Vorlesung vorgestellten Robinsonschen Unifikationsalgorithmus in Haskell implementiert (siehe Rückseite und Datei `robinson.hs` auf der Internetseite zur Vorlesung).
Leider sind ihm einige Fehler unterlaufen, denn der Beispielaufruf mit den Typgleichungen der Typinferenz für `map` liefert `Nothing`.

Nun ist er betrübt, weil er dem Christkind ein korrektes Programm für die Lösung von Typgleichungen noch vor Weihnachten versprochen hat. Können Sie ihm helfen und das Programm von den Fehlern befreien?

```
module Robinson where

type TCon  = (String,Int) -- Name, Stelligkeit
data Types = TVar String | SType TCon [Types] | FType Types Types
             deriving Eq
type Subst = [(String,Types)]

-- zentrale Unifikationsfunktion, Ausgabe Nothing bedeutet fail
unify :: Types -> Types -> Maybe Subst
unify (TVar cs) t
    | occursNot cs t = Just [(cs,t)]
    | otherwise      = Nothing
unify t (TVar cs)
    | occursNot cs t = Just [(cs,t)]
    | otherwise      = Nothing
unify (SType tcon1 ts1) (SType tcon2 ts2)
    | tcon1 == tcon2 = unifylist ts1 ts2 []
    | otherwise      = Nothing
unify (FType t11 t12) (FType t21 t22)
                     = unifylist [t11,t12] [t21,t22] []
unify _ _            = Nothing

-- Unifikation von mehreren Gleichungen
unifylist :: [Types] -> [Types] -> Subst -> Maybe Subst
unifylist [] [] sigma  = Just sigma
unifylist (t1:ts1) (t2:ts2) sigma
    =  case unify (applySubst sigma t1) (applySubst sigma t2) of
         Nothing       -> Nothing
         (Just sigma1) -> unifylist ts1 ts2 sigma1

applySubst :: Subst -> Types -> Types
applySubst sigma (TVar cs)
   = head ([ t | (tv,t) <- sigma, tv == cs] ++ [TVar cs])
applySubst sigma (SType tcon ts)
   = SType tcon (map (applySubst sigma) ts)
applySubst sigma (FType t1 t2)
   = FType (applySubst sigma t1) (applySubst sigma t2)

occursNot :: String -> Types -> Bool
occursNot cs (TVar cs1)      = cs /= cs1
occursNot cs (SType tcon ts) = and (map (occursNot cs) ts)
occursNot cs (FType t1 t2)   = occursNot cs t1 && occursNot cs t2

-- Beispielaufruf
mapBsp = printResult $ unifylist
           [ TVar "a3"
           , TVar "a1"
           , FType (TVar "a1") (FType (SType ("List",1) [TVar "a2"])
                                      (TVar "a3"))
           , FType (TVar "a7") (FType (SType ("List",1) [TVar "a7"])
                                      (SType ("List",1) [TVar "a7"]))
           ]
           [ SType ("List",1)  [TVar "a4"]
           , FType (TVar "a2") (TVar "a5")
           , FType (TVar "a1") (FType (SType ("List",1) [TVar "a2"])
                                      (TVar "a6"))
           , FType (TVar "a5") (FType (TVar "a6") (TVar "a3"))
           ] []

-- Ausgabefunktionen (siehe Datei)
printResult              :: Maybe Subst -> IO ()
```