

Aufgabe 1 (Ü) (*Typinferenz*)

Geben Sie jeweils einen Ableitungsbaum für folgende Aussagen an, und bestimmen Sie dabei den Typ τ :

- a) $\square \vdash \lambda xyz. x y (y z) : \tau$
- b) $[z : \tau_0] \vdash \text{let } x = \lambda yz. z y y \text{ in } x (x z) : \tau$

Aufgabe 2 (Ü) (*Konstruktive Logik*)

Geben Sie jeweils λ -Terme mit folgenden Funktionstypen an (mit Beweis):

- a) $(\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \gamma$
- b) $((((\alpha \rightarrow \beta) \rightarrow \alpha) \rightarrow \alpha) \rightarrow \beta) \rightarrow \beta$

Aufgabe 3 (Ü) (*Normalform*)

Zeigen Sie: Jeder typkorrekte λ^{\rightarrow} -Term besitzt eine Normalform.

Aufgabe 4 (Ü) (*Typinferenz in Gofer*)

Gegeben sei folgendes Term-Schema:

```
let  $x_0 = \lambda yz. z y y$   
  in let  $x_1 = \lambda y. x_0 (x_0 y)$   
    in let  $x_2 = \lambda y. x_1 (x_1 y)$   
      ...  
        in let  $x_n = \lambda y. x_{n-1} (x_{n-1} y)$   
          in  $x_n (\lambda z. z)$ 
```

Bis zu welchem n kann das Gofer-System den Typ des Terms berechnen?

Aufgabe 5 (Ü) (*Typinferenz in Gofer*)

Programmieren Sie den Algorithmus zur Typinferenz (ohne let-Polymorphismus) in Gofer. Gehen Sie dazu wie folgt vor:

- a) Definieren Sie einen Datentyp `Type`, der Typen von λ -Termen darstellt, und einen Datentyp `Term` für die λ -Terme selbst. Sie können sich dabei bei den Typen auf Typvariablen und den Funktionstyp, bei den λ -Termen auf Variablen, Applikation und Funktionsabstraktion beschränken.

b) Programmieren Sie den Unifikations-Algorithmus auf der Struktur der Typen. Das Rahmenprogramm von Blatt 7 enthält bereits eine Version der Unifikation für Terme.

c) Definieren Sie eine Funktion `constraints`, die zu einem Term t und einem Typ τ in einem Typkontext Γ mit Hilfe der Typinferenz-Regeln aus der Vorlesung eine Liste von Typ-Constraints berechnet. Bei dieser Berechnung ist es notwendig, neue Variablennamen für Typvariablen zu erzeugen. Es bietet sich deshalb an, für die Namen von Typvariablen die Sorten `Int` oder `Char` zu benutzen, und jeweils die "höchste" unbenutzte Variable als Ergebnis mitzuliefern. So steht diese Information für weitere rekursive Aufrufe zur Verfügung. Es könnte sich für `constraints` also folgende Signatur ergeben:

```
constraints :: Term -> Type -> [(VName,Type)] -> Char -> (Char, [(Type,Type)])
```

d) Definieren Sie eine Funktion, die zu einem beliebigen geschlossenem Term dessen allgemeinsten Typ berechnet.