Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Norbert Schirmer

Vorlesung Gleichungslogik
Wintersemester 2006/07
Übungsblatt 13
Abgabe 31. 01, Besprechung am 1. Februar 2007

Aufgabe 1 (H) (*Let-Polymorphie*)

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

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

Aufgabe 2 (H) (*Konstruktive Logik*)

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

a) $(\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$

b) $((((\alpha \to \beta) \to \alpha) \to \alpha) \to \beta) \to \beta$

Aufgabe 3 (Ü) (*Normalform*)

Zeigen Sie: Jeder typkorrekte $\lambda^{\to}$-Term besitzt eine Normalform.

Aufgabe 4 (P) (*Typinferenz von Gofer*)

Gegeben sei folgendes Term-Schema:

$$
\begin{array}{l}
\text{let } x_0 = \lambda yz.\ z\ y\ y \\
\quad \text{in let } x_1 = \lambda y.\ x_0\ (x_0\ y) \\
\qquad \text{in let } x_2 = \lambda y.\ x_1\ (x_1\ y) \\
\qquad\quad \ldots \\
\qquad\qquad \text{in let } x_n = \lambda y.\ x_{n-1}\ (x_{n-1}\ y) \\
\qquad\qquad\quad \text{in } x_n\ (\lambda z.\ z)
\end{array}
$$

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

Aufgabe 5 (P) (*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" unbenutze 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.