Technische Universität München *Thomas Gawlitza*
Fakultät für Informatik *Alex Berlea*
Prof. Dr. H. Seidl *Andrea Flexeder*

Compilerbau

Sommersemester 2006

11. Übungsblatt *Abgabetermin: Po. 17. Juli 2006, in der Vorlesung*

Aufgabe 1: *Typ-Inferenz* *14 Punkte*

In dieser Aufgabe erweitern wir die funktionale Kernsprache aus der Vorlesung, insbesondere das Typ-System, um benutzerdefinierte Daten-Typen. Es soll z.B. ermöglicht werden, einen Daten-Typen zur Repräsentation binärer Bäume wie folgt zu definieren:

```
datatype tree t = Leaf t | Node (tree t,tree t)
```

Gemäß dieser Definition ist `tree int` ein Typ dem z.B. der Wert `Node(Leaf 1,Leaf 2)` angehört. In diesem Kontext bezeichnet man `tree` als Typ-Konstruktor, sowie `Leaf` und `Node` als Werte-Konstruktoren. Im Allgemeinen sieht eine Typ-Definition wie folgt aus:

```
datatype <Typ-Bezeichner> (t1,...,tn) =
    <Konstruktor-Bezeichner_1> [<Typ-Ausdruck_1> ]
  | ....
  | <Konstruktor-Bezeichner_n> [<Typ-Ausdruck_n> ]
```

wobei in den Typ-Ausdrücken die Typ-Variablen `t1,...,tn` sowie der Typ-Bezeichner des eingeführten Typs verwendet werden kann. Dementsprechend wird der `case`-Ausdruck erweitert, so dass Pattern-Matching über Werte benutzer-definierter Typen erlaubt ist. Z.B. soll die Auswertung von

```
letrec
  count = fn tree =>
              case tree of
                 Leaf -> 1
               | Node(x,y) -> (count x)+(count y)
in count (Node(Leaf 1,Leaf 2))
```

die Anzahl der Blätter der Baumes `Node(Leaf 1,Leaf 2)`, also 2, liefern. Um das Typ-System entsprechend anzureichern ist folgendes zu tun:

1. Erweitere die Regeln zur Aufstellung der Term-Gleichungen zur Typ-Inferenz.
2. Erweitere dementsprechend den Algortihmus $\mathcal{W}$.
3. Berechne den Typ der obigen Funktion `count`.

Aufgabe 2: *Typ-Inferenz* *6 Punkte*

Definiere einen Ausdruck der Größe $O(n)$, dessen Typ die Größe $\Omega(2^{2^n})$ hat.