Technische Universität München
Fakultät für Informatik *Peter Ziewer*
Prof. Dr. H. Seidl

Abstrakte Maschinen

Sommersemester 2006

6. Übungsblatt *Abgabetermin: Do, 22. Juni 2006 in der Vorlesung*

Aufgabe 1: *Code-Erzeugung* *10 Punkte*

Erzeugen Sie Call-by-value-Code für folgende Ausdrücke! Geben Sie zu jeder Instruktion den aktuellen Kellerpegel an. Bestimmen Sie außerdem die jeweilige Adressumgebung zur Übersetzung der Ausdrücke die auf `in` folgen (also `y*g` bzw `fib 4`).

a)
```
1 + let
        x = g + 10;
        y = x * 4
    in y * g
```
mit $\rho = \{g \mapsto (G, 1)\}$ und $kp = 1$.

b)
```
letrec
    fib = fn x => if x <= 1 then 1
                  else (fib (x-1)) + (fib (x-2))
in fib 4
```
mit $\rho = \emptyset$ und $kp = 0$.

Aufgabe 2: *Code-Optimierung für Call by Need* *10 Punkte*

Die Codeerzeugungsfunktion nach der CBN-Strategie generiert bei jedem Variablenzugriff eine `eval`-Instruktion. Diese überprüft, ob die Variable bereits ausgewertet ist oder ihr Wert noch berechnet werden muss. Falls die Variable bereits ausgewertet wurde, ist die `eval`-Instruktion überflüssig. So ist bei der Übersetzung des Ausdrucks $(a + a + a)$ eine `eval`-Instruktion nur nach dem ersten Vorkommen der Variablen a notwendig.
Durch Erweiterung der Codeerzeugungsfunktion um ein weiteres Argument A, welches die Menge der bereits ausgewerten Variablen enthält, lassen sich überflüssige `eval`-Instruktionen einsparen.
Die Codeerzeugungsfunktion für Variablenzugriffe lautet dann wie folgt:

$$\texttt{code}_V\ x\ \rho\ kp\ A\ = \begin{cases} \texttt{getvar}\ x\ \rho\ kp & \text{, falls } x \in A \\ & \\ \texttt{getvar}\ x\ \rho\ kp & \text{, sonst} \\ \texttt{eval} & \end{cases}$$

a) Geben Sie eine formale Definition von $A[e]$ für beliebige PuF-Ausdrücke e, so dass $A[e]$ die Menge der Variablen in e ist, die bereits ausgwertet werden mussten, um e auswerten zu können!

b) Modifizieren Sie die Codeerzeugungsfunktion für PuF-Ausdrücke, um überflüssige `eval`-Instruktionen einzusparen!