

11. Übung „Übersetzerkonstruktion“

Abgabe am 4. Juli in der Vorlesung

Aufgabe 33

5 Punkte

Bei der Zwischencodeerzeugung wurden in der Vorlesung neue Marken mittels

```
newlabel/extendlabel und newtemp/extendtemp
```

erzeugt. Hierbei sind Marken Strings, die bei größeren Programmen natürlich recht groß werden können.

Als effizientere Alternative können Ints als Marken verwendet werden:

```
data Label = Label Int  
data Temp = Temp Int
```

Diese können dann in der Berechnung durchgereicht (entspr. L-Attributierung) und bei Bedarf verwendet werden. Hierbei sollte darauf geachtet werden, dass möglichst kleine Marken verwendet werden.

Geben Sie entsprechende Übersetzungen für Zuweisung, `if-then-else`, `repeat-until` und `for`-Schleife¹ an.

Aufgabe 34

7 Punkte

Wir erweitern die Bedingungen des Programms aus Aufgabe 31 wie folgt:

```
var x,n : Int;  
x := 7;  
n := 0;  
while x>4 && n>=0 do  
  n := n+1;  
  if x mod 2 = 0 && x>=0  
  then x := x div 2  
  else x := 3 * x + 1  
od;  
print(n)
```

¹mit Pascal-Semantik: `for i:=n to k step d do` wobei $i, n, k, k \in \text{Int}$ und i der Reihe nach die Werte $n, n+d, n+2d, \dots$ annimmt. Die Schleife terminiert, falls der Wert der Zählvariablen i größer als k ist.

Passen Sie zunächst den erzeugten Zwischencode an. Verwenden Sie das in der Vorlesung vorgestellte Verfahren, um aus dem entstehenden Zwischencode die Basisblockdarstellung zu gewinnen.

Aufgabe 35

8 Punkte

Wie wir bereits in der Übung diskutiert haben, ist die Übersetzung von Prozeduren, welche sich wechselseitig aufrufen, nicht mit Hilfe einer L-Attributierung möglich. Als vereinfachtes Modell von Prozeduren und Prozeduraufrufen betrachten wir folgende Sprache:

$$\begin{aligned} Prg &\longrightarrow Proc ; Prg \mid \varepsilon \\ Proc &\longrightarrow \mathbf{proc} \ Id \ Stm \\ Stm &\longrightarrow \mathbf{call} \ Id \ Stm \mid \varepsilon \\ Id &\longrightarrow \mathbf{a} \mid \dots \mid \mathbf{z} \end{aligned}$$

Wir betrachten als Vereinfachung nur parameterlose Prozeduren, deren Rumpf nur den Aufruf einer anderen Prozedur ermöglicht. Implementieren Sie einen Compiler, welcher Code für ein gegebenes Programm berechnet. Der erzeugte Code bestehe nur aus einer Folge von Programmadressen und `call/return`-Befehlen, wie in folgendem Beispiel:

proc a; proc c call a call b; proc b call c liefert den Code:

- 0: return
- 1: call 0
- 2: call 4
- 3: return
- 4: call 1
- 5: return

Verwenden Sie also keine symbolischen Adressen, sondern erzeugen Sie direkt absolute Programmadressen.