

# Übersetzerbau: Übung 14

von

Naja v. Schmude (4127652), Lisa Dohrmann (4130066)

## Aufgabe 1

Man entwickelt die Infos zu Lebendigkeit und nächster Verwendung von unten nach oben.

13: $x = 10 + y$	x nicht lebendig und keine nächste Verwendung, y lebendig, nächste Verwendung in 14
14: $y = y + 1$	y lebendig, nächste Verwendung in 15
15: $z = 10 + y$	z, y lebendig, nächste Verwendung in 16
16: $x = z + y$	x, y, z lebendig, keine nächste Verwendung

Die Symboltabelle entwickelt sich dabei wie folgt (von links nach rechts):

	Lebendigkeit	Nächste Verwendung
x	ja, nein	nein
y	ja, ja, ja, ja	nein, 16, 15, 14, 13
z	ja, ja, nein	nein, 16, nein

## Aufgabe 2

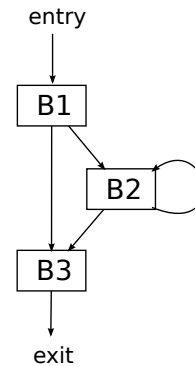
a) Das Programm

```
p = 0;
for (i=0; i < n; i++)
    p = p + a[i] * b[i];
```

soll in Drei-Adress-Code übersetzt werden.

```
1  p = 0
2  i = 0
3  if i >= n goto (10)
4  t1 = a[i]
5  t2 = b[i]
6  t3 = t1 * t2
7  p = p + t3
8  i = i + 1
9  if i < n goto (4)
10 ...
```

- b) Um den Flussgraphen zu konstruieren, muss man sich zunächst die Aufteilung in Grundblöcke klar machen. Laut Regel bilden die Zeilen 1-3 den Block  $B_1$ , die Zeilen 4-9 den Block  $B_2$  und Zeile 10 den Block  $B_3$ . Dann überlegt man sich, nach welchen Blöcken man das Programm verlässt, solche Blöcke verbindet man dann mit dem Knoten *exit*. Ansonsten gibt es immer dann eine Kante von Knoten  $i$  nach Knoten  $j$ , wenn man von der letzten Anweisung in Block  $i$  zum Anfang von Block  $j$  kommt.



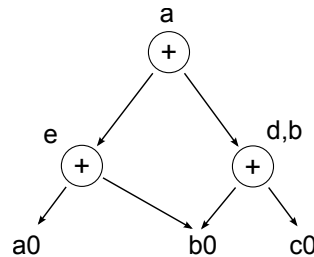
- c) Die Schleife taucht bei Block 2 auf.

### Aufgabe 3

- a) Der GAG zum Block

$d = b * c$   
 $e = a + b$   
 $b = b * c$   
 $a = e - d$

sieht wie folgt aus:



- b) Bedingung:  $a$  soll im Blockausgang lebendig sein.

Da  $b$  und  $d$  der gleiche Wert zugewiesen wird, das  $b$  aber danach nicht verwendet wird und im Blockausgang auch nicht mehr lebendig ist, muss es gar nicht erst berechnet werden. Es reicht folglich aus:

$d = b * c$   
 $e = a + b$   
 $a = e - d$

- c) Bedingung:  $a, b, c$  sollen im Blockausgang lebendig sein.

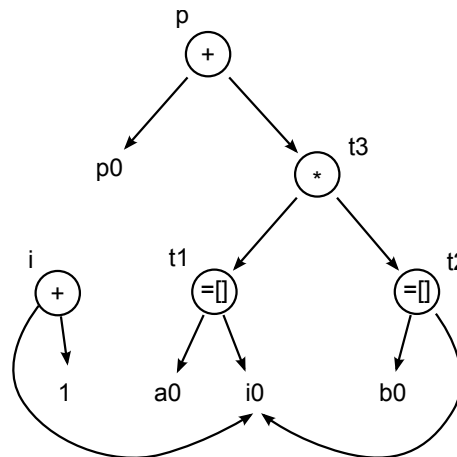
Hier kann nur insoweit optimiert werden, dass  $b$  nicht berechnet werden muss, sondern ihm einfach der Wert von  $d$  zugewiesen werden kann, was zumindest etwas Rechenaufwand einspart.

$d = b * c$   
 $e = a + b$   
 $b = d$   
 $a = e - d$

## Aufgabe 4

Nun erstellen wir für den Codeteil mit der Schleife aus 2a) den GAG. Die Zuweisung des Array-Inhalts an Position  $i$  behandeln wir so, wie im Drachenbuch beschrieben.

```
t1 = a[i]
t2 = b[i]
t3 = t1 * t2
p = p + t3
i = i + 1
```



## Aufgabe 5

## Aufgabe 6

- a) Eine Übersetzung, die ohne weitere Annahmen einfach sequentiell die gegebenen Anweisungen in Drei-Adress-Code umwandelt, sieht wie folgt aus

```
t1 = a + b
t2 = t1 + c
t3 = t2 + d
t4 = t3 + e
x = t4 + f
t5 = a + c
y = t5 + e
```

- b) Eine optimierte Variante kann durch Umstrukturierung der Anweisungen einige Zeilen und Register einsparen. Wir gehen davon aus, dass auch die Änderung der Befehlsreihenfolge erlaubt ist, wie wir es in der Vorlesung besprochen hatten.

```
t1 = a + c
y = t1 + e
t1 = y + b
t1 = t1 + d
x = t1 + f
```