Lehrstuhl für Softwaretechnik und Programmiersprachen
Professor Dr. Michael Leuschel

Compilerbau - Wintersemester 2021/22

Praktisches Übungsblatt 9

Besprechung der Aufgaben am 07.01.22 um 16:30 Uhr in 25.12.02.55 und gleichzeitig online per BBB
(evtl. auch früher)
Fragen an lukas.lang@hhu.de
Die Bearbeitung und Abgabe ist freiwillig.
Bringen Sie Ihre Lösung mit zum Übungstermin (z.B. USB-Stick)

Aufgabe 9.1

Schreiben Sie einen 3-Address Code Generator für eine kleine Sprache von arithmetischen Ausdrücken:

1. Die Programme bestehen aus einer Abfolge von Zuweisungen.
2. Links einer Zuweisung steht ein Bezeichner. Dieser besteht aus einer Abfolge von Klein- und Grossbuchstaben.
3. Rechts eines Zuweisungsoperators (:=) steht ein Ausdruck.
4. Ein Ausdruck kann aus Bezeichern, Zahlen sowie den Verknüpfungen '+', '-', '*', '/' bestehen.

Die (abstrakte) Grammatik hierzu ist:

1. S → id = E ;
2. E → E + E
3. E → E - E
4. E → E * E
5. E → E / E
6. E → -E
7. E → (E)
8. E → num
9. E → id

Das Format des Zwischencodes ist:

1. x = y op z
2. x = op y

3. x = y

Ein Beispielprogramm:

```
a := b*-c+b*-c;
b := a+-(b+c);
a := 42+a;
```

Erzeugt folgende Ausgabe:

```
t1= b
t2= c
t3=minus t2
t4=t1*t3
t5= b
t6= c
t7=minus t6
t8=t5*t7
t9=t4+t8
a= t9
t10= a
t11= b
t12= c
t13=t11+t12
t14=minus t13
t15=t10+t14
a= t15
t16= 42
t17= a
t18=t16+t17
a= t18
```

a. Erzeugen Sie mit Hilfe von SableCC einen Parser und einen AST. Finden Sie hierzu zuerst eine konkrete Grammatik.

b. Schreiben Sie einen Visitor der 3-Address Code erzeugt. Die naive Version von oben ist ausreichend.

c. Wieso ist der hier erzeugte Code alles andere als optimal?

d. Wie könnte man den Algorithmus aus b) verbessern um die Anzahl der temporären Variabelen veringern?

Tipp zur Lösung: Attributieren Sie den AST. Es sollte zwei Attribute geben: Code und Adresse. Hierbei enthält Adresse die Position des Ausdrucks (d.h. die letzte temporäre Variable t_i). Lassen Sie eine Zählervariable beim generieren der temporären Register mitlaufen um die nächte Nummer zu vergeben.

Aufgabe 9.2
Die Sprache wird nun um if-Anweisungen und while-Schleifen erweitert. Diese werden bei einem Wert ungleich Null durchlaufen.

Die (abstrakte) Grammatik aus der vorherigen Aufgabe wird um folgende Regeln erweitert:

1. S → if (E) S else S
2. S → while (E) S
3. S → S S |ϵ

Der Zwischencode wird erweitert um:

1. goto L
2. if x goto L
3. ifFalse x goto L
4. if x relop y goto L

Hierbei ist L ein Label und relop ein Vergleichsoperator (=,<,>)

Ein Beispielprogramm:

```
i := 4;
sum := 0;
while(i)
    sum := sum + i;
    i := i-1;
```

Erzeugt folgende Ausgabe:

```
t1=4
i=t1
t2=0
sum=t2
L0:
      t3= i
      if t3 goto L1
      t4=sum
      t5=i
      t6=t4+t6
      sum=t6
      t7=i
      i=t7-1
goto L0
L1:
```

a. Erzeugen Sie mit Hilfe von SableCC einen Parser und einen AST.

b. Schreiben Sie einen Visitor der 3-Address Code erzeugt .

c. Was müsste am Visitor aus Aufgabenteil b verändert werden um SSA Code zu erzeugen. Welches Problem tritt hier bei if-Anweisungen auf?

Zu diesem Zettel wird es keine Musterlösung geben!