Kapitel 8: Zwischencode

Aufgabe

Interne Programmdarstellung, um Maschinencode–Erzeugung zu vereinfachen (und Optimierungen zu ermöglichen)

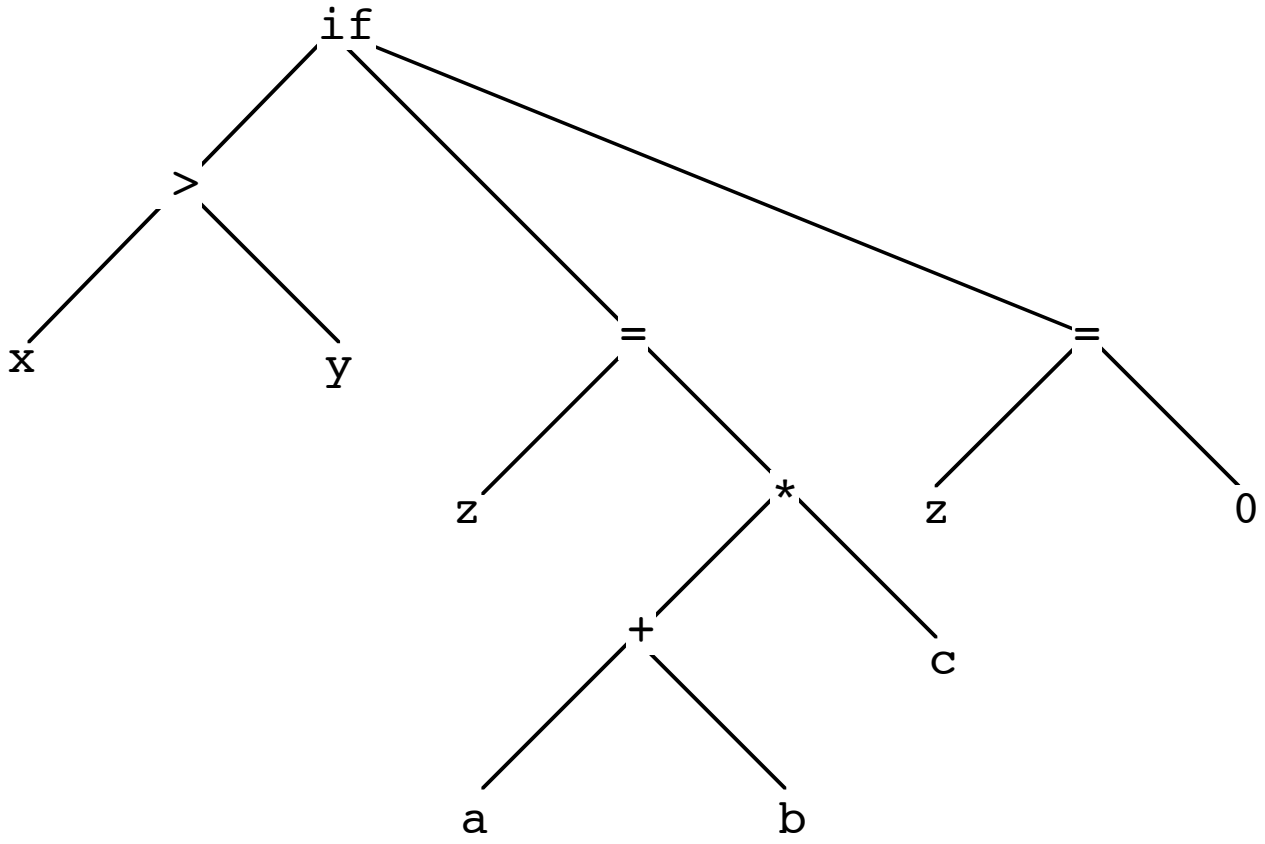
Themen

- **Sytaxbaum**
- **Linearisierungen**
- **Kontroll–Ausdrücke**
- **Kontroll–Anweisungen**

Syntaxbaum

Bei strukturierten Sprachen (ohne goto) kann die Programmstruktur vollständig durch den Syntaxbaum dargestellt werden, z.B.

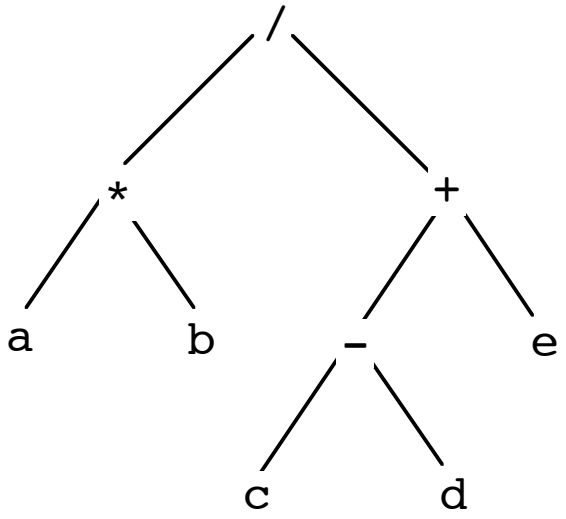
```
if (x>y) z=(a+b)*c;
else z=0;
```



Syntaxbaum und Linearisierungen

Infix

`a*b/(c-d+e)`

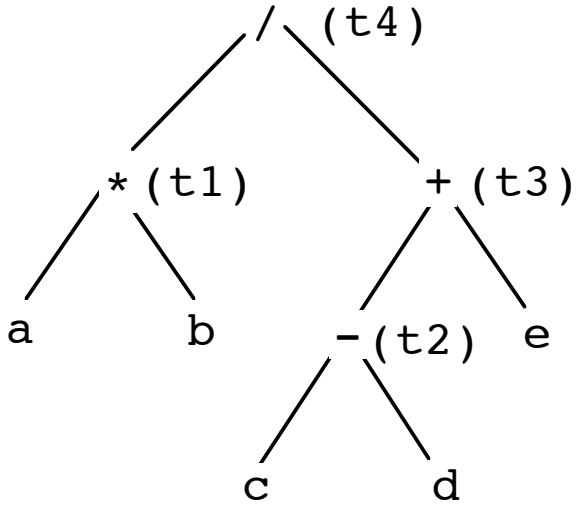


Postfix

```
a b * c d - e + /
```

Prefix

```
/ * a b + - c d e
```



Quadrupel

```
t1 = a  * b
t2 = c  - d
t3 = t2 + e
t4 = t1 / t3
```

Befehle des Quadrupel-Codes

```
z = x
z = op x
z = x op y

goto z
if x rop y goto z

z = @ x        (dereferenzieren)
@ z = x
```

AG zur Erzeugung von Quadrupel-Code

Zuweisung und Ausdrücke

Produktion	Regeln
`S -> V = E`	`S.c = E.c ‖ gen(V.a '=' E.a)`
`E -> E1 + E2`	`E.a = newtemp` `E.c = E1.c ‖ E2.c ‖ gen(E.a '=' E1.a '+' E2.a)`
`E -> V`	`E.a = V.a` `E.c = V.c`
`V -> id`	`V.a = lookadr(id.x)` `V.c = ''`

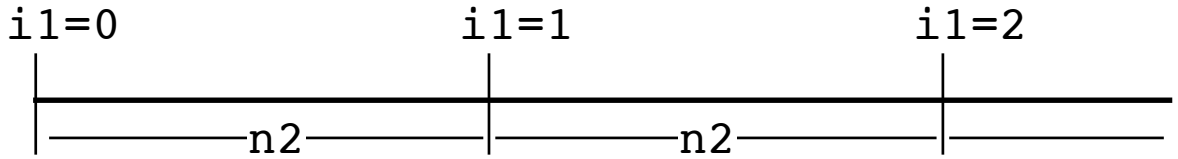
```
v = x+y+z
```

Quadrupel-Code

```
t1 =  x + y
t2 = t1 + z
 v = t2
```

Quadrupel-Code für indizierte Variable

Deklaration `a[n1,n2]`
Adresse des Elements `a[i1,i2]:  (i1*n2 + i2)*w + adr(a)`



Beispiel-Deklaration `a[10,5]`
Adresse des Elements `a[u,v]:  (u*5 + v)*4 + adr(a)`
Quadrupel für `x = a[u,v]`

```
t1 = 5  * u
t2 = t1 + v
t3 = 4  * t2
t4 = t3 + adr(a)
t5 = @ t4
 x = t5
```

Quadrupel-Code für Boolesche Ausdrücke

Numerische Methode

```
if B goto *+3
t1 = 0
goto *+2
t1 = 1
```

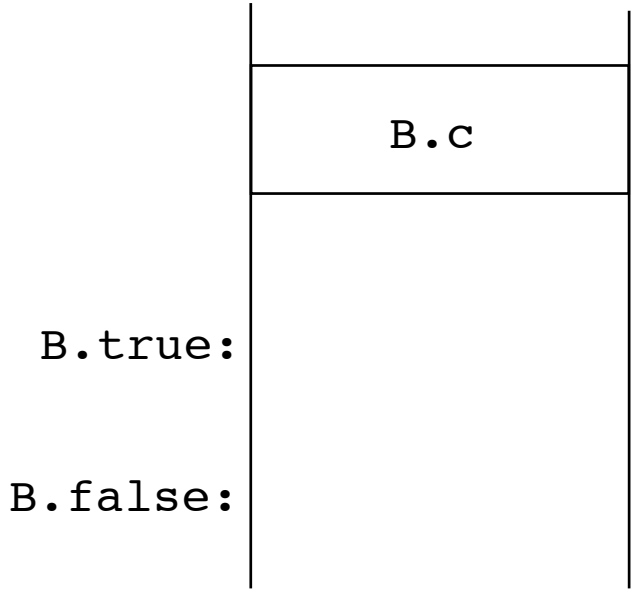
Kontrollfluß-Methode

```
if B goto Ltrue
goto Lfalse
```

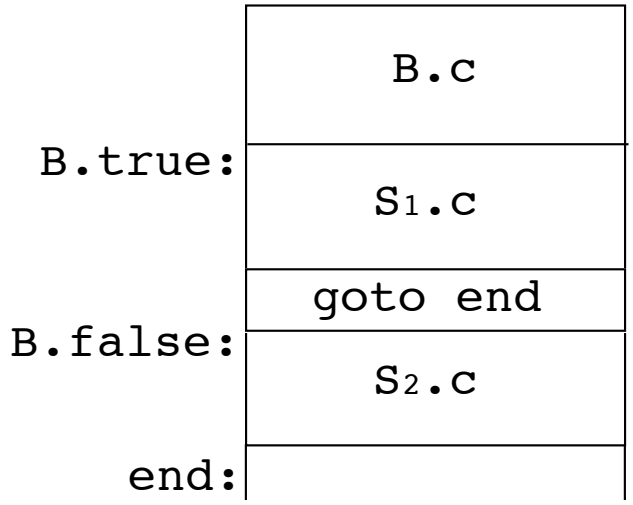
AG für Booleschen Ausdruck (Kontrollfluß–Methode)

Produktion	Regeln
`B -> E1 rop E2`	`B.c = gen('if'E1.a rop.x E2.a'goto'B.true) ‖ gen('goto'B.false)`

Die Sprungziele `B.true` und `B.false` sind ererbte Attribute
Ihre Werte müssen vom Kontext (z.B. if–Anweisung) übergeben werden



AG für if–Anweisung



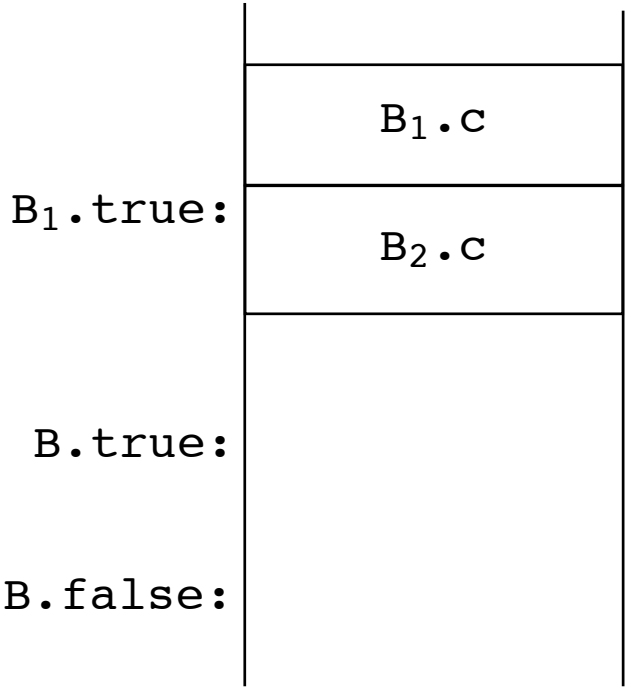
Produktion	Regeln

`S -> if B then` S_1 `else` S_2	`B.true = newlabel`
	`B.false = newlabel`
	`end = newlabel`
	`S.c = B.c ‖`
	`gen(B.true':') ‖` S_1`.c ‖`
	`gen('goto'end) ‖`
	`gen(B.false':') ‖` S_2`.c ‖`
	`gen(end':')`

AG für Booleschen Operator AND

Produktion	Regeln

`B -> B1 AND B2`	`B1.true = newlabel`
	`B1.false = B.false`
	`B2.true = B.true`
	`B2.false = B.false`
	`B.c = B1.c ‖ gen('B1.true':') ‖ B2.c`



Beispiel: Quadrupel–Code für if–Anweisung

Quellprogramm

```
if (a>b && c>d) {
   y = x+1;
else
   y = x-1;
}
```

Quadrupel

```
   if a>b goto L4
   goto L2
L4:if c>d goto L1
   goto L2
L1:t1 = x+1
    y = t1
   goto L3
L2:t2 = x-1
    y = t2
L3:
```

Marken der if–Anweisung

```
B.true=L1
B.false=L2
end=L3
```