

Fordítóprogramok

12. előadás

Aszalós László

2015. november 25.

3 című kód (TAC, 3AC)

$x := y \text{ op } z$

- x, y, z címek
- változók, konstansok, vagy a fordító által generált segédváltozók címei
- a címek a szimbólumtábla bejegyzéseire mutató pointerek

3 című kód (TAC, 3AC)

- ❶ `x := y op z`
- ❷ `x := op y`
- ❸ `x := y`
- ❹ `if x relop y goto L`
- ❺ `param x1 ... param xn call p,n`
- ❻
 - ▶ `x := y[i],`
 - ▶ `x[i] := y`
- ❼
 - ▶ `x := &y` (`x` értéke `y` címe)
 - ▶ `x := *y` (`x` értéke az `y` pointer által mutatott cím tartalmával lesz egyenlő)
 - ▶ `*x := y` (az `x` pointer által mutatott objektum tartalma lesz egyenlő `y` tartalmával)

Példa a 3 című kód használatára

$$x = (-b + \sqrt{b^2 - 4ac}) / (2a)$$

t1 := b * b

t2 := 4 * a

t3 := t2 * c

t4 := t1 - t3

t5 := sqrt(t4)

t6 := 0 - b

t7 := t5 + t6

t8 := 2 * a

t9 := t7 / t8

x := t9

Példa a 3 című kód használatára

```
for (i = 0; i < 10; ++i) {  
    b[i] = i*i;  
}
```

	t1 := 0	; initialize i
L1:	if t1 >= 10 goto L2	; conditional jump
	t2 := t1 * t1	; square of i
	t3 := t1 * 4	; word-align address
	t4 := b + t3	; address to store i*i
	*t4 := t2	; store through pointer
	t1 := t1 + 1	; increase i
	goto L1	; repeat loop
L2:		

Attribútumnyelvtanból 3 című utasítások

- Szintetizált attribútum nevek:
 - ▶ temp_hely :int;
 - ▶ azon_hely :int;
- Nemterminálisok és attributumok:
 - ▶ utasitas;;
 - ▶ kif: temp_hely;
 - ▶ tag: temp_hely;
 - ▶ tenyezo: temp_hely;
- Tokenek:
 - ▶ azonosito = betu (betu | szam | ' ')* : azonhely;
- Terminálisok: ':=' , '+' , '-' , '*' , '/' , '(' , ')'

Attribútumnyelvtanból 3 című utasítások

Szabályok és szemantikus függvények:

① `utasitas -> azonosito := kif;`

`do`

`print (azonosito.azon_hely :=kif.temp_hely);`

`end`

② `kif0 -> kif1 + tag;`

`do`

`kif0.temp_hely := newtemp;`

`print (kif0.temp_hely ':'= kif1.temp_hely '+'
tag.temp_hely);`

`end`

Attribútumnyelvtanból 3 című utasítások

③ kif 0 -> kif 1 - tag;

do

kif0.temp_hely := newtemp;

print (kif0.temp_hely ':= ' kif1.temp_hely '-'
tag.temp_hely);

end

④ kif0 -> tag;

do

kif.temp_hely := tag.temp_hely;

end

⑤ tag0 -> tag1 * tenyezo;

do

tag0.temp_hely := newtemp;

print (tag 0 .temp_hely ':= ' tag 1 .temp_hely '*'
tenyezo.temp_hely);

end

Attribútumnyelvtanból 3 című utasítások

❖ tag0 -> tag1 / tenyezo;

do

tag 0 .temp_hely := newtemp;

print (tag0.temp_hely ':= ' tag1.temp_hely '/'
tenyezo.temp_hely);

end

❖ tag -> tenyezo;

do

tag.temp_hely := tenyezo.temp_hely;

end

Attribútumnyelvtanból 3 című utasítások

8 `tenyezo -> '(' kif ')';`

`do`

`tenyezo.temp_hely := kif.temp_hely;`

`end`

9 `tenyezo -> azonosito;`

`do`

`tenyezo.temp_hely := azonosito.azon_hely;`

`end`

3 című kód átalakítása gépi utasításokká

- minden egyes 3 című utasítás 1 vagy több gépi utasítássá fordul
- minden egyes változónak szükséges egy cím, melyen az értéke tárolódik
 - ▶ regiszter, verem, heap, statikus tárolás
 - ▶ hagyomány
 - ★ atomi értékek, címek: regiszter
 - ★ tömbök: heap
 - ★ globális változók: statikus

Függvényhívás

függvényhívás kódja

- Új bejegyzés készítése a verembe,
 - ▶ utasításszámláló értékének mentése
- paraméterek beállítása/tárolása
- utasításszámláló értékének mentése
- regiszterek, flag-ek mentése

visszatérés kódja

- visszatérési érték mentése
- utasításszámláló és regiszterek visszaállítása
- veremből utolsó bejegyzés törlése

- a változó/kifejezés értéke szerepelhet regiszterben, s veremben, háttértáron
- a változó értékének helye a szimbólumtáblában adott
- új tároló megnyitása az eredmény tárolására
 - ▶ segéd (temporal) változók,
 - ▶ elsőre végtelen sok regiszter adott
 - ▶ később regiszterallokációval pontosíthatunk

Tömbök kezelése

- $a := b[i]$ regiszterből

```
mult ri, elsize      (r1)
ld b(r1)              (ra)
```

- memóriából

```
ld mi                (ri)
mult ri, elsize      (r1)
ld b(r1)              (ra)
```

- $a[i] := b$ regiszterbe

```
mult ri, elsize      (r1)
st rb, a(r1)
```

- memóriából

```
ld mi, ri            (ri)
mult ri, elsize      (r1)
ld rb, a(r1)
```

Statikus memóriakezelés

- fordítási időben ismerni kell minden programelem méretét
 - ▶ dinamikus tömb, egymásba skatulyázott eljárások kizárva
- a program végrehajtási idejében egy időpillanatban minden elem csak egyszer fordulhat elő
 - ▶ rekurzív eljárások kizárva

Memória felosztása

- utasításrész
- adatrész
 - ▶ implicit paraméterek: szubrutinok visszatérési értékei, függvényhívások eredményei
 - ▶ aktuális paraméterek: címek, értékek
 - ▶ változók, tömbök értékei

Dinamikus memóriakezelés

- tetszőleges élettartamú objektumok (globális, static)
- skatulyázott élettartamú objektumok - aktivációs rekord
 - ▶ lokális változók területe
 - ▶ a blokkból elérhető változók (aktivációs rekordok listája/lánca)
 - ▶ explicit és implicit paraméterek

Kódoptimalizálás

- célunk az, hogy a program végrehajtási ideje és a program mérete csökkenjen
- optimalizálást a kódgenerálás során, vagy utána alkalmazhatunk

Lokális optimalizálás

Minden olyan utasítás, melyre ugró utasítással eljuthatunk, megjelöljük, s két jelölt utasítás közti rész lesz egy *alapblokk*. Egy alapblokknak egy belépési pontja van, s minden utasítása egyszer kerül végrehajtásra.

Tömörítés (lokális)

- konstansok összevonása:
 - ▶ ha egy kifejezés valamely része ismert konstans, akkor végrehajtja akkor a rá vonatkozó műveletet
 - ▶ $x+20-4/2$ helyett $x+18$
- konstans továbbterjesztése:
 - ▶ konstans értékű azonosítóknek az értékükkel való helyettesítése
- azonos kifejezések többszöri kiszámításának elkerülése:
- irányított körmentes gráf készítése az értékadások alapján, a ennek megfelelő kódgenerálás
- ablakoptimalizálás:
 - ▶ pár sornyi kódban minta keresése és cseréje: pl. regiszterbe mentés és visszatöltés helyettesíthető egy másolással, felesleges regisztermentés törölhető, stb.
- ciklusoptimalizálás

Ciklusoptimalizálás

- ciklus kifejtése
- parciális ciklus kifejtés
- ciklusok összevonása
- frekvenciaredukálás (invariáns változók értékadásait a cikluson kívülre)
- erős redukció
 - ▶ gyorsabban végrehajtható művelettel kiváltás: pl. szorzás helyett összeadás

Globális optimalizálás

alablokkok alkotta gráf vizsgálata

rendelkezésre álló kifejezések az adott alablokkba vezető bármely úton szerepel, komponenseinek értéke nem változott meg

nagyon foglalt kifejezések az alablokkból kivezető minden úton operandusai megváltoztatása előtt hivatkozunk rá (érdemes regiszterben tartani)

értékadások elérhetősége egy másik adatblokkban szereplő értékadásból idevezető úton az adott változónak nem adunk más értéket

élő változó egy változó élő, ha legalább egy követő blokkban felhasználjuk

- azonos kifejezések ismételt kiszámításának megszüntetése
- változók továbbterjesztése
- ciklusinvariánsok meghatározása, ciklus elé helyezése
- ciklusok indukciós változóinak megszüntetése (ugyanazzal a konstans értékkel változik minden esetben)

Gépfüggő optimalizálás

- eggyel növelés, csökkentés beépített inkrementáló, dekrementáló utasítással
- kettő hatványaival szorzás léptetéssel
- tömb elemeinek elérése indexelt címezéssel
- regiszterallokáció (mely változók, értékek kerülnek regiszterekbe)
- regiszterkijelölés (adott változó melyik regiszterbe kerül)