Формални Езици и Езикови Процесори
ТУ, кат. КС, летен семестър 2012

Лекция 18

Тема:

Генерация на Код (Транслационни Схеми)

Съдържание:

- Генерация на код (Транслационни Схеми)
- Определение
- Видове
- Трансиращи граматики
- Разширени ТС с действия по време на компилация
- Примери

Въведение в ТС

Проблем: Превод на първична програма в обектен код. Две граматики описват този процес:

- Граматика G1 за първичния входен език
- Граматика G2 за целевия обектен език

Първичните програми са символни низове според синтаксиса на G1 = (Σ1, N1, P1, S1)

Обектните програми са символни низове според синтаксиса на G2 = (Σ2, N2, P2, S2)

Въведение в ТС

Граматиките G1 и G2 са свързани:

$\Sigma 1 \neq \Sigma 2$

$N1 = N2 = N$

$S1 = S2 = S$

Налице е релация м/у правилата на двете граматики. Всяко G1 правило има съответно правило от G2.

P1: $X \rightarrow \alpha$ P2: $X \rightarrow \beta$

Двете правила се обединяват в обща продукция

P: $X \rightarrow \alpha \mid \beta$

Как се чете “X→ α ▌β”

Записът X → α ▌ β се интерпретира така:

Нетермиалът X поражда низа α (source language), който се превежда в низа β (target language).

Още: Низът α се превежда в низа β.

Още: Низът β съответства на низа α.

Още: Низът β е преводът на низа α, т.е. β=TS(α).

Въвежда се транслационна схема като формална система: TS = (Σ1, Σ2, N, P, S)

Транслационни Схеми

Класификация:

ТС съхраняващи броя и реда на нетерминалните символи

ТС съхраняващи броя нетерминални символи и разменящи реда на нетерминалните символи

ТС разменящи броя и реда на нетерминаните символи

Транслационни Схеми

ТС, съхраняващи броя и реда на нетерминалните символи

$TS = (\Sigma_1, \Sigma_2, N, P, S)$

P: $X \rightarrow \alpha \mid \beta$

$\alpha = \alpha_1 X_1 \alpha_2 X_2 \alpha_3 \ldots \alpha_n X_n \alpha_{n+1}$

$\alpha_i \in \Sigma_1^*, \; i=1..n+1$

$\beta = \beta_1 X_1 \beta_2 X_2 \beta_3 \ldots \beta_n X_n \beta_{n+1}$

$\beta_i \in \Sigma_2^*, \; i=1..n+1$

ТС пример 1

ТС за превод от инфиксен в постфиксен запис

1. E	→	T	▌	T
2. E	→	E + T	▌	E T +
3. T	→	F	▌	F
4. T	→	T * F	▌	T F *
5. F	→	a	▌	a
6. F	→	(E)	▌	E

Демо: top-down ляв каноничен СА за:

a + a * a

(a + a) * a

a + a * a

E →	E + T	E T +
→	T + T	T T +
→	F + T	F T +
→	a + T	a T +
→	a + T * F	a T F * +
→	a + F * F	a F F * +
→	a + a * F	a a F * +
→	a + a * a	a a a * +

(a + a) * a

E	→	T	T
	→	T * F	T F *
	→	F * F	F F *
	→	(E) * F	E F *
	→	(E + T) * F	E T + F *
	→	(T + T) * F	T T + F *
	→	(F + T) * F	F T + F *
	→	(a + T) * F	a T + F *
	→	(a + F) * F	a F + F *
	→	(a + a) * F	a a + F *
	→	(a + a) * a	a a + a *

Транслационни Схеми

ТС, съхраняващи броя нетерминални символи и разменящи реда на нетерминалните символи

ТС пример 2

1. S	→	a	A	B	b	▮	B	A b
2. A	→	c	A			▮	c	A
3. B	→	d	B			▮	B	d
4. A	→		c			▮	c	
5. B	→		d			▮	d	

Демо: top-down ляв каноничен СА за:

a c c d d b

a c c d d b

$$
\begin{aligned}
S &\rightarrow a\ \underline{A}\ B\ b \quad | \quad B\ \underline{A}\ b \\
&\rightarrow a\ c\ \underline{A}\ B\ b \quad | \quad B\ c\ \underline{A}\ b \\
&\rightarrow a\ c\ c\ \underline{B}\ b \quad | \quad \underline{B}\ c\ c\ b \\
&\rightarrow a\ c\ c\ d\ \underline{B}\ b \quad | \quad \underline{B}\ d\ c\ c\ b \\
&\rightarrow a\ c\ c\ d\ d\ b \quad | \quad d\ d\ c\ c\ b
\end{aligned}
$$

$$d\ d\ c\ c\ b = TS(a\ c\ c\ d\ d\ b)$$

Транслиращи Граматики

ТГ са въведени от Rosenkrantz & Stirnz

Транслиращите граматики ТГ са модификация на ТС, съхраняващи броя и реда на нетерминалните символи

$TS=(\Sigma_1, \Sigma_2, N, P, S)$

Транслиращи Граматики

$TS = (\Sigma_1, \Sigma_2, N, P, S)$

P: $X \rightarrow \alpha \mid \beta$

$\alpha = \alpha_1 X_1 \alpha_2 X_2 \alpha_3 \ldots \alpha_n X_n \alpha_{n+1}$, $\alpha_i \in \Sigma_1^*$, $X_i \in N$

$\beta = \beta_1 X_1 \beta_2 X_2 \beta_3 \ldots \beta_n X_n \beta_{n+1}$, $\beta_i \in \Sigma_2^*$

низове α_i, β_i се слепват в единен низ.

Така, продукциите на ТС $X \rightarrow \alpha \mid \beta$ се променят в

$X \rightarrow \alpha_1 \{\beta_1\} X_1 \alpha_2 \{\beta_2\} X_2 \alpha_3 \{\beta_3\} \ldots \alpha_n \{\beta_n\} X_n \alpha_{n+1} \{\beta_{n+1}\}$

Транслиращи Граматики - пример

1. E → T
2. E → E + T {*add;*}
3. T → F
4. T → T * F {*mul;*}
5. F → **a** {*load a;*}
6. F → (E)

Транслиращи Граматики - пример

Ляв каноничен разбор за a + a * a

E 2→ E + T {*add;*}
1→ T + T {*add;*}
3→ F + T {*add;*}
5→ **a** {*load a;*} + T {*add;*}
4→ **a** {*load a;*} + T * F {*mul;*} {*add;*}
3→ **a** {*load a;*} + F * F {*mul;*} {*add;*}
5→ **a** {*load a;*} + a {*load a;*} * F {*mul;*} {*add;*}
5→ **a** {*load a;*} + a {*load a;*} * a {*load a;*} {*mul;*} {*add;*}

Транслиращи Граматики - пример

Десен каноничен разбор за a + a * a

E 2→ E + T {*add;*}
4→ E + T * F {*mul;*} {*add;*}
5→ E + T * a {*load a;*} {*mul;*} {*add;*}
3→ E + F * a {*load a;*} {*mul;*} {*add;*}
5→ E + a {*load a;*} * a {*load a;*} {*mul;*} {*add;*}
1→ T + a {*load a;*} * a {*load a;*} {*mul;*} {*add;*}
3→ F + a {*load a;*} * a {*load a;*} {*mul;*} {*add;*}
5→ a {*load a;*} + a {*load a;*} * a {*load a;*} {*mul;*} {*add;*}

Транслиращи Граматики - пример

Както левият разбор 2 1 3 5 4 3 5 5 , така и десният разбор 2 4 5 3 5 1 3 5 продуцират изход

a{*load a;*} **+ a**{*load a;*} *** a**{*load a;*} {*mul;*} {*add;*}

входният низ е показан bold, а обектният низ е показан курсив заграден във фигурни скоби

Така, низът **a + a * a** се превежда в:

load a
load a
load a
mul
add

Разширени ТС

ТС с действия в периода на компилация.

Въвеждат се:

- Променливи в периода на компилация: имена от тип цял, принадлежат на ТС, ползват се в изрази и оператори в периода на компилация.
- Действия в периода на компилация: изпълняват се над променливите от периода на компилация. Например, операции (increment/decrement), оператор за присвояване, извикване на функции и др.

Разширени ТС – формална с-ма

ТС като формална система се задава като

$TS = (\Sigma 1, \Sigma 2, N, P, S)$

Разширена ТС се задава като

$TS = (\Sigma 1, \Sigma 2, Vc, Lc, N, P, S)$

Vc – множество променливи активни в периода на компилация

Lc – език (операции/оператори) за работа с променливите от множеството Vc

Разширени ТС - пример

Разширена ТС за превод на оператор за присвояване <var> = <expression>

V = E

В инструкции на Асемблер за процесор с класическа регистрова архитектура, чийто регистри с общо предназначение са означени reg0, reg1, reg2, … reg15.

Ръчно проигран пример

a = b

Нека reg0 да съхрани/съдържа адреса на a

mov reg0, #a

Нека reg1 да съхрани/съдържа стойността на b

mov reg1, b

Необходимо е съдържанието на reg1 да се копира в паметта, индиректно адресируема по съдържанието на регистър reg0

mov (reg0), reg1

Разширени ТС - пример

Инструкции:

move dest,src	
op dst,src	op ={mul ,div, mod, add, sub}

Видове адресация:

move regx,a	register – direct
move regx,#21	register – immediate operand
move (regx),#a	register indirect – immediate
move (regx),a	register indirect – direct
move regx,regy	register - register

Разширени ТС - пример

- Vc – променливи в период на компилация:
  - Само една променива: int p = 0;
- Lc – действия в период на компилация:
  - Операция increment {+} смисъл: p=p+1
  - Операция decrement {-} смисъл: p=p-1
  - Фунция STR(p) връща “reg”+itoa(p)

Разширени ТС - пример

1. S → V = E	V E move (STR(p-2)),STR(p-1) {- -} ‘\n’
2. V → a	move STR(p),#a {+} ‘\n’
3. E → T	T
4. E → E + T	E T add STR(p-2),STR(p-1) {-} ‘\n’
5. T → F	F
6. T → T * F	T F mul STR(p-2),STR(p-1) {-} ‘\n’
7. F → a	move STR(p),a {+} ‘\n’
8. F → (E)	E

a = a + a * a

1. S → V = E (rule 1)
2. → a = E (rule 2)
4. → a = E + T (rule 4)
3. → a = T + T (rule 3)
5. → a = F + T (rule 5)
7. → a = a + T (rule 7)
6. → a = a + T * F (rule 6)
5. → a = a + F * F (rule 5)
7. → a = a + a * F (rule 7)
7. → a = a + a * a (rule 7)

SA(a = a + a * a) = 1 2 4 3 5 7 6 5 7 7

SA(a = a + a * a) = 1243576577

1. S → V E move (STR(p-2)),STR(p-1) {- -} ‘\n’
2. → move STR(p),#a {+} ‘\n’ E move (STR(p-2)),STR(p-1) {- -} ‘\n’
4. → move STR(p),#a {+} ‘\n’ E T add STR(p-2),STR(p-1) {-} ‘\n’ move (STR(p-2)),STR(p-1) {- -} ‘\n’
3. → move STR(p),#a {+} ‘\n’ **T** T add STR(p-2),STR(p-1) {-} ‘\n’ move (STR(p-2)),STR(p-1) {- -} ‘\n’
5. → move STR(p),#a {+} ‘\n’ **F** T add STR(p-2),STR(p-1) {-} ‘\n’ move (STR(p-2)),STR(p-1) {- -} ‘\n’
7. → move STR(p),#a {+} ‘\n’ move STR(p),a {+} ‘\n’ T
   add STR(p-2),STR(p-1) {-} ‘\n’ move (STR(p-2)),STR(p-1) {- -} ‘\n’
6. → move STR(p),#a {+} ‘\n’ move STR(p),a {+} ‘\n’ T F mul STR(p-2),STR(p-1) {-} ‘\n’
   add STR(p-2),STR(p-1) {-} ‘\n’ move (STR(p-2)),STR(p-1) {- -} ‘\n’
5. → move STR(p),#a {+} ‘\n’ move STR(p),a {+} ‘\n’ F F mul STR(p-2),STR(p-1) {-} ‘\n’
   add STR(p-2),STR(p-1) {-} ‘\n’ move (STR(p-2)),STR(p-1) {- -} ‘\n’
7. → move STR(p),#a {+} ‘\n’ move STR(p),a {+} ‘\n’ move STR(p),a {+} ‘\n’ F mul STR(p-2),STR(p-1) {-} ‘\n’
   add STR(p-2),STR(p-1) {-} ‘\n’ move (STR(p-2)),STR(p-1) {- -} ‘\n’
7. → move STR(p),#a {+} ‘\n’ move STR(p),a {+} ‘\n’ move STR(p),a {+} ‘\n’ move STR(p),a {+} ‘\n’
   mul STR(p-2),STR(p-1) {-} ‘\n’ add STR(p-2),STR(p-1) {-} ‘\n’ move (STR(p-2)),STR(p-1) {- -} ‘\n’

SA(a = a + a * a) = 1243576577

1. S → V
   E
   move (STR(p-2)),STR(p-1) {- -} ‘\n’

2. → move STR(p),#a {+} ‘\n’
   E
   move (STR(p-2)),STR(p-1) {- -} ‘\n’

4. → move STR(p),#a {+} ‘\n’
   E
   T
   add STR(p-2),STR(p-1) {-} ‘\n’
   move (STR(p-2)),STR(p-1) {- -} ‘\n’

SA(a = a + a * a) = 1243576577

3. → move STR(p),#a {+} ‘\n’
T
T
add STR(p-2),STR(p-1) {-} ‘\n’
move (STR(p-2)),STR(p-1) {- -} ‘\n’

5. → move STR(p),#a {+} ‘\n’
F
T
add STR(p-2),STR(p-1) {-} ‘\n’
move (STR(p-2)),STR(p-1) {- -} ‘\n’

7. → move STR(p),#a {+} ‘\n’
move STR(p),a {+} ‘\n’
T
add STR(p-2),STR(p-1) {-} ‘\n’
move (STR(p-2)),STR(p-1) {- -} ‘\n’

SA(a = a + a * a) = 1243576577

6. → move STR(p),#a {+} ‘\n’
move STR(p),a {+} ‘\n’
T
F
mul STR(p-2),STR(p-1) {-} ‘\n’
add STR(p-2),STR(p-1) {-} ‘\n’
move (STR(p-2)),STR(p-1) {- -} ‘\n’

5. → move STR(p),#a {+} ‘\n’
move STR(p),a {+} ‘\n’
F
F
mul STR(p-2),STR(p-1) {-} ‘\n’
add STR(p-2),STR(p-1) {-} ‘\n’
move (STR(p-2)),STR(p-1) {- -} ‘\n’

SA(a = a + a * a) = 1243576577

7. → move STR(p),#a {+} ‘\n’
move STR(p),a {+} ‘\n’
move STR(p),a {+} ‘\n’
F
mul STR(p-2),STR(p-1) {-} ‘\n’
add STR(p-2),STR(p-1) {-} ‘\n’
move (STR(p-2)),STR(p-1) {- -} ‘\n’

7. → move STR(p),#a {+} ‘\n’
move STR(p),a {+} ‘\n’
move STR(p),a {+} ‘\n’
move STR(p),a {+} ‘\n’
mul STR(p-2),STR(p-1) {-} ‘\n’
add STR(p-2),STR(p-1) {-} ‘\n’
move (STR(p-2)),STR(p-1) {- -} ‘\n’

Изпълнение на действията в периода на компилация

Междинна форма	Окончателна форма	
		// p=0
move STR(p),#a {+}	move reg0,#a	// p=1
move STR(p),a {+}	move reg1,a	// p=2
move STR(p),a {+}	move reg2,a	// p=3
move STR(p),a {+}	move reg3,a	// p=4
mul STR(p-2),STR(p-1) {-}	mul reg2,reg3	// p=3
add STR(p-2),STR(p-1) {-}	add reg1,reg2	// p=2
move (STR(p-2)),STR(p-1) {- -}	move (reg0),reg1	// p=0

Благодаря
За
Вниманието