

第五章：中间代码生成(2)



基于LL(1)方法

【1】 $E \rightarrow T$

【2】 $E \rightarrow E + T$

【3】 $E \rightarrow E - T$

【4】 $T \rightarrow F$

【5】 $T \rightarrow T * F$

【6】 $T \rightarrow T / F$

【7】 $F \rightarrow (E)$

【8】 $F \rightarrow i$

$F = P$

(1) $E \rightarrow TE_s$

(2) $E_s \rightarrow \varepsilon$

(3) $E_s \rightarrow +TE_s$

(4) $E_s \rightarrow -TE_s$

(5) $T \rightarrow PT_s$

(6) $T_s \rightarrow \varepsilon$

(7) $T_s \rightarrow *PT_s$

(8) $T_s \rightarrow /PT_s$

(9) $P \rightarrow C$

(10) $P \rightarrow id$

(11) $P \rightarrow (E)$

• 简单表达式的带有动作符的LL(1)文法

(1) $E \rightarrow TEs$

(2) $Es \rightarrow \varepsilon$

(3) $Es \rightarrow +T\#GenCode(+)\#Es$

(4) $Es \rightarrow -T\#GenCode(-)\#Es$

(5) $T \rightarrow PTs$

(6) $Ts \rightarrow \varepsilon$

(7) $Ts \rightarrow *P\#GenCode(*)\#Ts$

(8) $Ts \rightarrow /P\#GenCode(/)\#Ts$

(9) $P \rightarrow C\#Push(C)\#$

(10) $P \rightarrow id\#Push(id)\#$

(11) $P \rightarrow (E)$

- 当遇到常量C和简单变量id时，把它们的语义信息压入语义栈；
- 当处理完一个运算符（+，-，*，/）的右分量时，该运算符的左、右运算分量已经分别存放在语义栈 Sem 的次栈顶和栈顶的位置，因此可以生成相应的运算符的四元式，并把运算结果的语义信息压入语义栈。

简单表达式的LL(1)分析表

	C	id	()	+	-	*	/	#
E	1	1	1						
E s				2	3	4			2
T	5	5	5						
Ts				6	6	6	7	8	6
P	9	10	1 1						

产生式	Predict集
(1) $E \rightarrow T E_s$	C, id, (
(2) $E_s \rightarrow \varepsilon$	#,)
(3) $E_s \rightarrow + T E_s$	+
(4) $E_s \rightarrow - T E_s$	-
(5) $T \rightarrow P T_s$	C, id, (
(6) $T_s \rightarrow \varepsilon$	#,), +, -
(7) $T_s \rightarrow * P T_s$	*
(8) $T_s \rightarrow / P T_s$	/
(9) $P \rightarrow C$	C
(10) $P \rightarrow id$	Id
(11) $P \rightarrow (E)$	(

表达式 $x + y * z$ 的中间代码生成过程(1)

分析栈S

输入流T

动作

# E	$x + y * z \#$	$LL[E, id] = [1]$
# EsT	$x + y * z \#$	$LL[T, id] = [5]$
# EsTs P	$x + y * z \#$	$LL[P, id] = [10]$
# EsTs #Push (id)# id	$x + y * z \#$	Match
# EsTs #Push (id)#	$+ y * z \#$	#Push (id)#
# EsTs	$+ y * z \#$	

Top Top	intptr (x.level, x.off, dir)

语义栈Sem

- (1) $E \rightarrow T Es$
- (2) $Es \rightarrow \epsilon$
- (3) $Es \rightarrow + T \# \text{GenCode}(+) \# Es$
- (4) $Es \rightarrow - T \# \text{GenCode}(-) \# Es$
- (5) $T \rightarrow P Ts$
- (6) $Ts \rightarrow \epsilon$
- (7) $Ts \rightarrow * P \# \text{GenCode}(*) \# Ts$
- (8) $Ts \rightarrow / P \# \text{GenCode}(/) \# Ts$
- (9) $P \rightarrow C \# \text{Push}(C) \#$
- (10) $P \rightarrow id \# \text{Push}(id) \#$
- (11) $P \rightarrow (E)$

表达式 $x + y * z$ 的中间代码生成过程(2)

分析栈S

输入流T

动作

EsTs

+ y * z #

LL [Ts,+] = [6]

#Es

+ y * z #

LL [Es,+] = [3]

Es #GenCode (+)# T +

+ y * z #

Match

Es #GenCode (+)# T

y * z #

LL [T,id] = [5]

Es #GenCode (+)#TsP

y * z #

LL [P,id] = [10]

Es #GenCode (+)# Ts#Push (id)# id

y * z #

Top →	intptr	(x.level, x.off, dir)

语义栈Sem

- (1) $E \rightarrow T Es$
- (2) $Es \rightarrow \epsilon$
- (3) $Es \rightarrow + T \# \text{GenCode} (+) \# Es$
- (4) $Es \rightarrow - T \# \text{GenCode} (-) \# Es$
- (5) $T \rightarrow P Ts$
- (6) $Ts \rightarrow \epsilon$
- (7) $Ts \rightarrow * P \# \text{GenCode} (*) \# Ts$
- (8) $Ts \rightarrow / P \# \text{GenCode} (/) \# Ts$
- (9) $P \rightarrow C \# \text{Push} (C) \#$
- (10) $P \rightarrow id \# \text{Push} (id) \#$
- (11) $P \rightarrow (E)$

表达式 $x + y * z$ 的中间代码生成过程(3)

分析栈S

输入流T

动作

Es# GenCode (+) # Ts # Push (id) # id
Match

y * z #

Es# GenCode (+) # Ts # Push (id)

* z #

Push (id)

Es # GenCode (+) # Ts

* z #

Top →	intptr	(y.level, y.off, dir)
Top →	intptr	(x.level, x.off, dir)

语义栈Sem

- (1) $E \rightarrow T Es$
- (2) $Es \rightarrow \epsilon$
- (3) $Es \rightarrow + T \# \text{GenCode}(+) \# Es$
- (4) $Es \rightarrow - T \# \text{GenCode}(-) \# Es$
- (5) $T \rightarrow P Ts$
- (6) $Ts \rightarrow \epsilon$
- (7) $Ts \rightarrow * P \# \text{GenCode}(*) \# Ts$
- (8) $Ts \rightarrow / P \# \text{GenCode}(/) \# Ts$
- (9) $P \rightarrow C \# \text{Push}(C) \#$
- (10) $P \rightarrow id \# \text{Push}(id) \#$
- (11) $P \rightarrow (E)$

表达式 $x + y * z$ 的中间代码生成过程(4)

分析栈S

输入流T

动作

Es # GenCode (+) # Ts

* z #

LL [Ts, *] = [7]

Es # GenCode (+) # Ts # GenCode (*) # P*

* z #

Match

Es # GenCode (+) # Ts # GenCode (*) # P

z #

Top →		
	intptr	(y.level, y.off, dir)
	intptr	(x.level, x.off, dir)

语义栈Sem

- (1) $E \rightarrow T Es$
- (2) $Es \rightarrow \epsilon$
- (3) $Es \rightarrow + T \# \text{GenCode}(+) \# Es$
- (4) $Es \rightarrow - T \# \text{GenCode}(-) \# Es$
- (5) $T \rightarrow P Ts$
- (6) $Ts \rightarrow \epsilon$
- (7) $Ts \rightarrow * P \# \text{GenCode}(*) \# Ts$
- (8) $Ts \rightarrow / P \# \text{GenCode}(/) \# Ts$
- (9) $P \rightarrow C \# \text{Push}(C) \#$
- (10) $P \rightarrow id \# \text{Push}(id) \#$
- (11) $P \rightarrow (E)$

表达式 $x + y * z$ 的中间代码生成过程(5)

分析栈S

输入流T

动作

Es # GenCode (+) # Ts # GenCode (*) # P
[10]

z # LL[P,id] =

Es # GenCode (+) # Ts # GenCode (*) # # Push (id) # id

z #

Match

Es # GenCode (+) # Ts # GenCode (*) # # Push (id)

#

Top →		
	intptr	(y.level, y.off, dir)
	intptr	(x.level, x.off, dir)

语义栈Sem

- (1) $E \rightarrow T Es$
- (2) $Es \rightarrow \epsilon$
- (3) $Es \rightarrow + T \# \text{GenCode}(+) \# Es$
- (4) $Es \rightarrow - T \# \text{GenCode}(-) \# Es$
- (5) $T \rightarrow P Ts$
- (6) $Ts \rightarrow \epsilon$
- (7) $Ts \rightarrow * P \# \text{GenCode}(*) \# Ts$
- (8) $Ts \rightarrow / P \# \text{GenCode}(/) \# Ts$
- (9) $P \rightarrow C \# \text{Push}(C) \#$
- (10) $P \rightarrow id \# \text{Push}(id) \#$
- (11) $P \rightarrow (E)$

表达式 $x + y * z$ 的中间代码生成过程(6)

分析栈S

输入流T

动作

Es # GenCode (+) # Ts # GenCode (*) # # Push (id) # # #Push (id)#

Es # GenCode (+) # Ts # GenCode (*) # # #GenCode (*)#

Es # GenCode (+) # Ts

(MULTI, (y.level, y.off, dir), (z.level, z.off, dir), (-1, t_1 , dir))

intptr	(z.level, z.off, dir)	← Top
intptr	(y.level, y.off, dir)	← Top
intptr	(x.level, x.off, dir)	

语义栈Sem

- (1) $E \rightarrow T Es$
- (2) $Es \rightarrow \epsilon$
- (3) $Es \rightarrow + T \# \text{GenCode}(+) \# Es$
- (4) $Es \rightarrow - T \# \text{GenCode}(-) \# Es$
- (5) $T \rightarrow P Ts$
- (6) $Ts \rightarrow \epsilon$
- (7) $Ts \rightarrow * P \# \text{GenCode}(*) \# Ts$
- (8) $Ts \rightarrow / P \# \text{GenCode}(/) \# Ts$
- (9) $P \rightarrow C \# \text{Push}(C) \#$
- (10) $P \rightarrow id \# \text{Push}(id) \#$
- (11) $P \rightarrow (E)$

表达式 $x + y * z$ 的中间代码生成过程(7)

分析栈S

输入流T

动作

Es # GenCode (+) # Ts

#

LL [Ts, #] = [6]

Es # GenCode (+)

#

#GenCode (+)#

Es

(MULTI, y, z, t₁)

#

LL [Es, #] = [2]

#

(ADDI, x, t₁, t₂)

#

Success

(MULTI, (y.level, y.off, dir), (z.level, z.off, dir), (-1, t₁, dir))

(ADDI, (x.level, x.off, dir), (-1, t₁, dir), (-1, t₂, dir))

		Top ↑ Top ↑
intptr	(-1, t ₁ , dir)	
intptr	(-1, t ₂ , dir)	

语义栈Sem

- (1) $E \rightarrow T Es$
- (2) $Es \rightarrow \epsilon$
- (3) $Es \rightarrow + T \# \text{GenCode}(+) \# Es$
- (4) $Es \rightarrow - T \# \text{GenCode}(-) \# Es$
- (5) $T \rightarrow P Ts$
- (6) $Ts \rightarrow \epsilon$
- (7) $Ts \rightarrow * P \# \text{GenCode}(*) \# Ts$
- (8) $Ts \rightarrow / P \# \text{GenCode}(/) \# Ts$
- (9) $P \rightarrow C \# \text{Push}(C) \#$
- (10) $P \rightarrow id \# \text{Push}(id) \#$
- (11) $P \rightarrow (E)$

例：将下列表达式翻译成四元式序列。

~~$$X*2+A*(i+1)/(j+1)$$~~

其中i和j为整型变量，其它为实型变量。

1. (FLOAT, 2, _, t_1)
2. (MULTF, X, t_1 , t_2)
3. (ADDI, i, 1, t_3)
4. (FLOAT, t_3 , _, t_4)
5. (MULTF, A, t_4 , t_5)
6. (ADDI, j, 1, t_6)
7. (FLOAT, t_6 , _, t_7)
8. (DIVF, t_5 , t_7 , t_8)
9. (ADDF, t_2 , t_8 , t_9)

实型 整

变量的中间代码生成

$V \rightarrow id \text{ \#push}$ 将 id 压入语义栈 【标识符变量】

$V \rightarrow *V \text{ \#pushA}$ 取 $sem(s-1)$, 取其址压入 【指针变量】

$V \rightarrow V.id \text{ \#dom}$ 【域选择变量】

$\#dom$ 中要执行的动作:

1. 对 $sem(s-1)$ 和 $sem(s-2)$ 进行类型检查主要是检查域名是否和标识符类型匹配
2. $new(t)$; t 为间接取址
3. 找到 $sem(s-1)$ 域名在结构类型中的偏移 x
4. $Gen([], sem(s-2), x, t)$
5. 退栈: $pop(2)$
6. 进栈: $push(t)$

变量的中间代码生成

□ 下标变量: $V \rightarrow V[E] \#Addnext$

#Addnext中要执行的动作的思想:

1. 取出E的值
2. -数组下界
3. *size
4. +初始地址

例：C语言的数组
int A[5][6];

A	VarKin	TypePtr	dir	Level	Off	
---	--------	---------	-----	-------	-----	--

Size	Kind	Low	Up	ElemType
5*6*size(int)	ArrayTy	0	4	

Size	Kind	Low	Up	ElemType
6*size(int)	ArrayTy	0	5	IntPtr

A[0]	A[0][0]	A[0][1]	A[0][2]	A[0][3]	A[0][4]	A[0][5]
A[1]	A[1][0]	A[1][1]	A[1][2]	A[1][3]	A[1][4]	A[1][5]
A[2]	A[2][0]	A[2][1]	A[2][2]	A[2][3]	A[2][4]	A[2][5]
A[3]	A[3][0]	A[3][1]	A[3][2]	A[3][3]	A[3][4]	A[3][5]
A[4]	A[4][0]	A[4][1]	A[4][2]	A[4][3]	A[4][4]	A[4][5]




int A[5][6];

设起始地址为0

int A[D₁][D₂];

$\text{Addr}(A[E_1][E_2])$
 $= E_1 \times S_1 + E_2 \times S_2$

$S_1 = D_2, S_2 = 1$

A[0]		A[0][0]	0
		A[0][1]	1
		
		A[0][5]	5
		A[1][0]	6
		A[1][1]	7
A[1]		
		A[1][5]	11
		
.....	
A[4]		A[4][0]	24
		A[4][1]	25
		
		A[4][5]	29

假设有数组类型的变量声明如下:

$A : \text{array } [L_1..U_1] [L_2..U_2] \dots [L_n..U_n] \text{ of } T ; (n \geq 1)$

数组类型的内部表示 其中 $D_i = U_i - L_i$



假设有数组类型的变量声明如下:

$A : \text{array } [L_1..U_1] [L_2 .. U_2] \dots [L_n .. U_n] \text{ of } T ; (n \geq 1)$

将size转化为S则有:



变量的中间代码生成

假设有数组类型的变量声明如下:

A :array $[L_1..U_1][L_2..U_2]...$

$[L_n..U_n]$ of T ; ($n \geq 1$)

w 数组A占单元大小: $\text{size}(A) = D_1 * D_2 * ... * D_n * \alpha$

其中 $D_i = U_i - L_i + 1$, α 为类型T所占单元数。

w 下标变量取址: $a[i_1][i_2][i_3]...[i_n]$

$= \text{addr}(a) + (i_1 - L_1) * S_1 + (i_2 - L_2) * S_2 + ... + (i_n - L_n) * S_n$

//注: 此处的 S_i 不同于书中P130给出的 S_i , 为第i层数组变量元素空间大小

变量的中间代码生成

□ 下标变量: $V \rightarrow V[E] \#Addnext$

#Addnext中要执行的动作:

1. $(-, sem(s-1), sem(s-2).low, t1)$
2. $(*, t1, si, t2)$
3. $([], sem(s-2), t2, t3)$
4. $pop(2); push(t3)$

例子

```
int i,m; float z;  
struct rt {int y; float x};  
rt a[10][10];
```

求: $a[5+i][6].x + m * z$ 的中间代码

□ $(+, 5, i, t1)$

□ $(*, t1, 30, t2)$

□ $([], a, t2, t3)$

□ $(*, 6, 3, t4)$

□ $([], t3, t4, t5)$

6. $([], t5, 1, t6)$

7. $(\text{FLOAT}, m, -, t7)$

8. $(*, t7, z, t8)$

9. $(+, t6, t8, t9)$

3x10.



1. 数组结构.

2. - li

3. size.

4. []

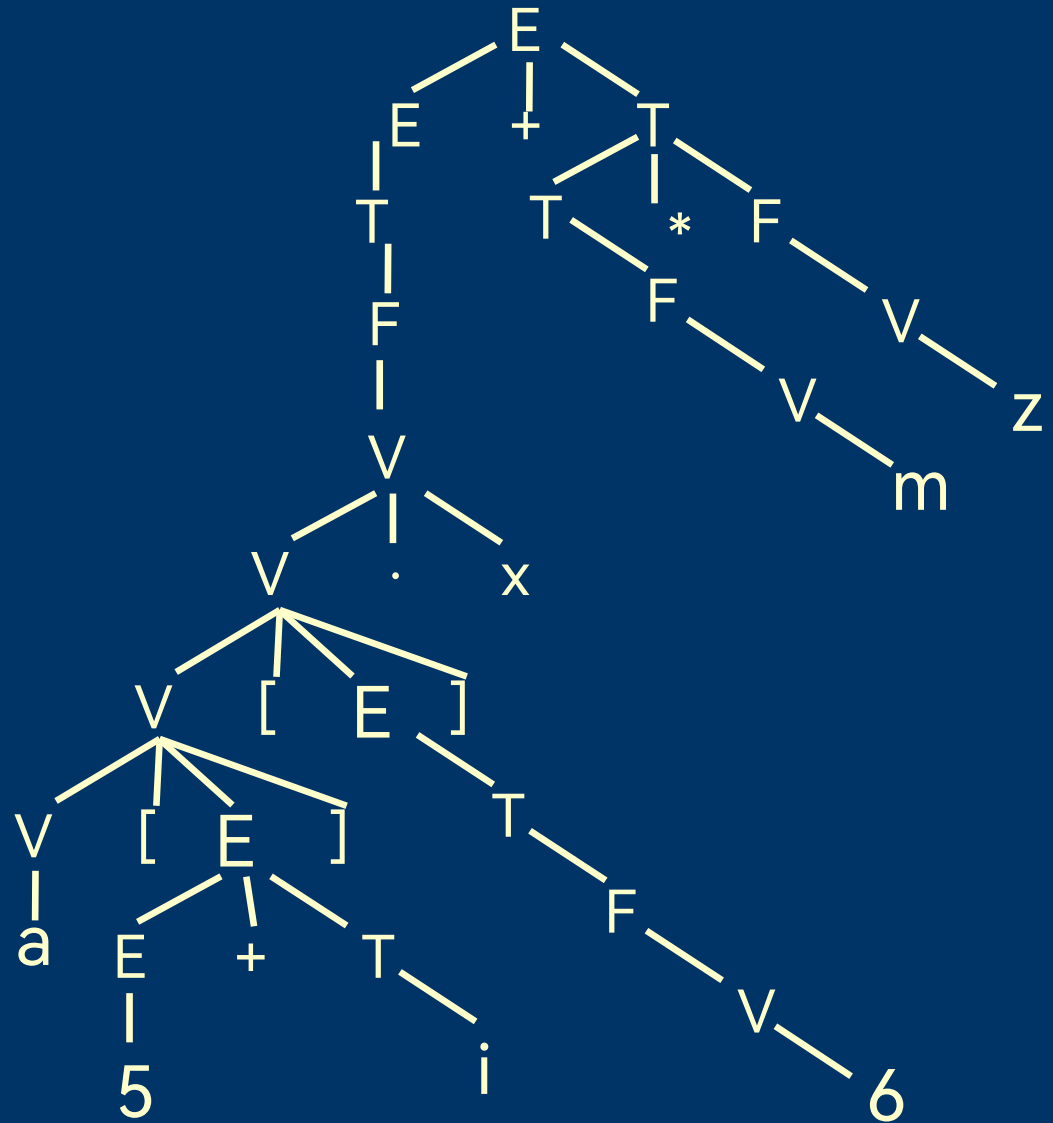
$<(-, t1, 0, t2)$

0+1

-x

a[5+i][6].x + m * z

- 【1】 $E \rightarrow T$
- 【2】 $E \rightarrow E + T$ #inc
- 【3】 $E \rightarrow E - T$ #dec
- 【4】 $T \rightarrow F$
- 【5】 $T \rightarrow T * F$ #MUL
- 【6】 $T \rightarrow T / F$ #DIV
- 【7】 $F \rightarrow (E)$
- 【8】 $F \rightarrow V$
- 【9】 $V \rightarrow \text{id}$ #Push
- 【9】 $V \rightarrow V.\text{id}$ #Dom
- 【9】
 $V \rightarrow V[E]\text{\#Addnext}$



练习题:

i,j:integer;

a : array[1..10][1..5] of integer ;

下标变量a [i +1] [j*i-2]的 中间代码 :

$$\begin{aligned} & \text{Addr} (A [E_1] [E_2] \dots [E_n]) \\ &= \text{Addr} (A) + (E_1 - L_1) \times S_1 \times \text{Size} (T) \\ &\quad + (E_2 - L_2) \times S_2 \times \text{Size} (T) \\ &\quad + (E_3 - L_3) \times S_3 \times \text{Size} (T) \\ &\quad \dots \dots \dots \\ &\quad + (E_n - L_n) \times S_n \times \text{Size} (T) \end{aligned}$$

$$\begin{aligned} S_i &= D_{i+1} \times D_{i+2} \times D_{i+3} \times \dots \times D_n \\ &= (U_{i+1} - L_{i+1} + 1) \times \dots \times (U_n - L_n + 1) \end{aligned}$$

$$S_n = 1$$

1. (+, i, 1, t₁)
2. (-, t₁, 1, t₂)
3. (*, t₂, 5, t₃)
4. ([], a, t₃, t₄)
5. (*, j, i, t₅)
6. (-, t₅, 2, t₆)
7. (-, t₆, 1, t₇)
8. (*, t₇, 1, t₈)
9. ([], t₄, t₈, t₉)

赋值语句的中间代码

- 赋值语句的形式为：Left := Right,

- 赋值语句的四元式结构为：

Left 的中间代码

Right 的中间代码

(FLOAT, Right, —, t)

(ASSIG ,Right(t),- ,Left)

赋值语句的中间代码

- 赋值语句中间代码生成动作文法如下：

$S \rightarrow V := E \text{ \#Assig\#}$

- Assig需要做如下处理：

- 1.从语义栈中取出赋值号左右分量的语义信息；
- 2.比较类型是否相同，如果不同，则生成类型转换中间代码；
- 3.生成赋值四元式：
(ASSIG , Right (t) , - , Left)

赋值语句的中间代码的例子

□ i, j, x, y : integer;

□ a : array[1..10][1..5] of integer ;

赋值语句 $a[i][j] := x + y * 3$ 的中间代码(其中变量均为整型) :

