编译原理

Compiler Construction Principles

朱 青

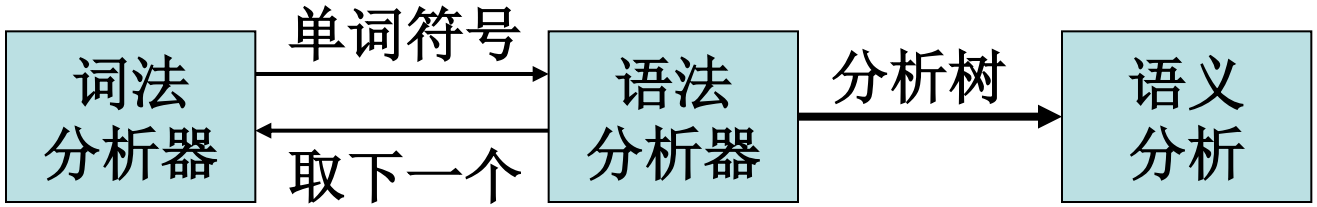
信息学院计算机系，

中国人民大学，

zqruc2012@aliyun.com

第4章:语法分析-自下而上分析

语法分析的重要性:



语法分析方法:

1)自顶向下分析法.

2)自底向上分析法.

第4章:语法分析-自下而上分析

- 4.1 自底向上分析法
- 4.2 算符优先分析法
- 4.3 LR分析器
- 4.4 LR(0)分析表的构造
- 4.5 SLR分析表的构造
- 4.6 LR(1) 和LALR(1)分析表简介
- 4.7 软件工具Yacc

4.1 自底向上分析法:

自底向上分析法是从输入串开始,逐步进行“归约”,直至归到文法的开始符号.或者说,从语法树的末端开始,步步向上归约,直到根结.

1 归约与分析树:

“移进-归约”法: 用栈把输入符号一个一个的移进栈中,当栈顶形成某个产生式的候选式时,即把栈顶的这一部分替换成(归约为)该产生式的左部符号.

假定文法G为: (5.1) p83

(1) S⟶aAcBe

(2) A⟶b

(3) A⟶Ab 将输入串abbcde归约到S.

(4) B⟶d

步骤:	1	2	3	4	5	6	7	8	9	10
动作:	进 a	进 b	归 (2)	进 b	归 (3)	进 c	进 d	归 (4)	进 e	归 (1)
									e	
							d	B	B	
				b		c	c	c	c	
		b	A	A	A	A	A	A	A	
	a	a	a	a	a	a	a	a	a	S

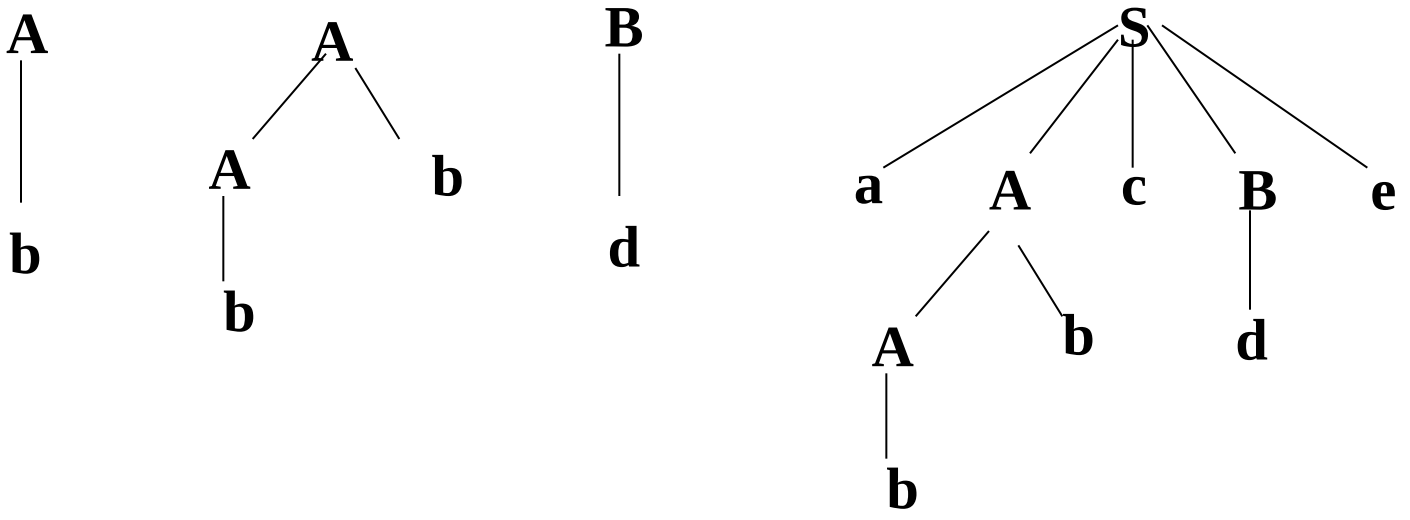
在整个“移进-归约”过程中,进行了四次归约.归约的关键问题是:精确定义“可归约串”.不同的可归约串形成了不同的自下而上分析法.

算符优先分析法中,用“最左素短语”

规范归约分析法中,用“句柄”.

语法分析过程中,可用一棵分析树来表示.在自下而上分析过程中,每一步归约都可以画出一棵子树来.归约完成,形成一棵分析树.

以上一题为例,用语法树表示归约的过程.



2 规范归约简述

若G是一个文法,S是开始符号,

假定αβχ是文法G的一个句型,如果有

$S \overset{*}{\Longrightarrow} \alpha A \chi$ 且 $A \overset{+}{\Longrightarrow} \beta$

则称β是一个关于非终结符号A的,句型αβχ的短语. 如果有

$S \overset{*}{\Longrightarrow} \alpha A \chi$ 且 $A \Longrightarrow \beta$

则称β是直接短语.

一个句型的最左直接短语称为该句型的句柄.

例如: 以下述文法为例 ,(a*a+a)是文法的句子,找出此文法的直接短语和句柄.

$$E \longrightarrow T|E+T$$
$$T \longrightarrow F|T*F \qquad (3.2)$$
$$F \longrightarrow (E)|a$$

考虑文法(3.2)的一个句型a1*a2+a3:

- **a1,a2,a3,a1*a2 都是句型a1*a2+a3的 短语,**
- **a1,a2和a3是直接短语,**
- **a1是最左直接短语,**
- **a2+a3不是句型a1*a2+a3短语,因为有E=>a2+a3但不存在从文法的开始符号E到a1*E的推导.**

假定文法G为: (5.1) p83

(1) S⟶aAcBe

(2) A⟶b

(3) A⟶Ab 将输入串abbcde归约到S.

(4) B⟶d

对文法(5.1)的句子abbcde逐步寻找句柄,并用相应产生式的左部符号去替换,得到如下归约过程:(画底线的部分是句柄).

句型	归约过程
a<u>b</u>bcde	(2) A⟶b
a<u>Ab</u>cde	(3) A⟶Ab.
aAc<u>d</u>e	(4) B⟶d
<u>aAcBe</u>	(1) S⟶aAcBe
S	

规范归约定义:

假定a是文法G的一个句子,称序列 an,an-1,…,a0 是a的一个规范归约,此序列应满足:

1) an=a

2)a0为文法的开始符,即a0=S

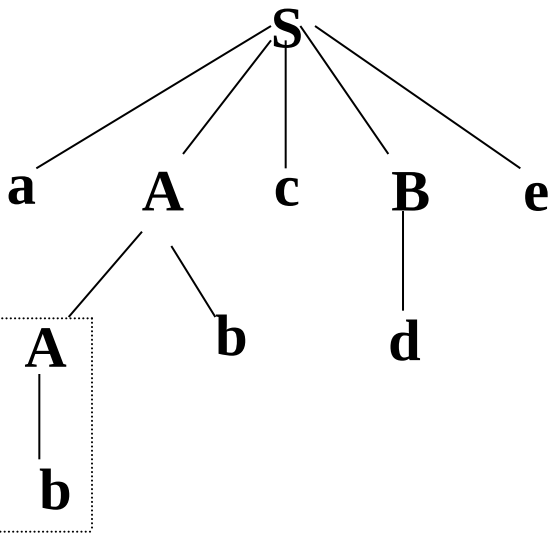
3)对任何i,0<i<=n,ai-1是从ai经把句柄替换为相应产生式的左部符号而得到的.

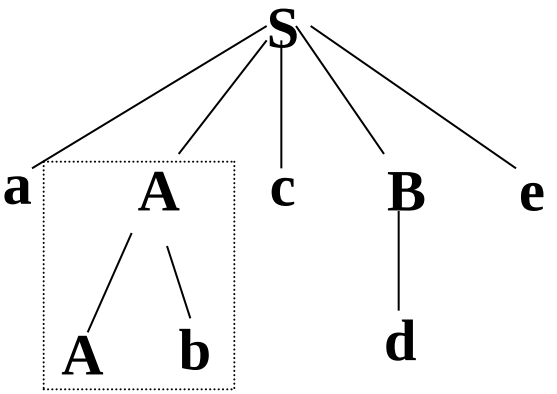
规范归约是关于a的一个最右推导的逆过程.规范归约也称最左归约.

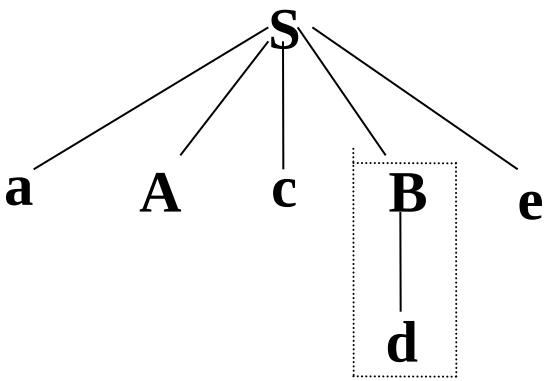
在形式语言中,最右推导常被称为规范推导.由规范推导所得的句型成为规范句型.规范归约的实质是,在移进过程中,当发现栈顶呈现句柄时,就用相应产生式的左部符号进行替换.

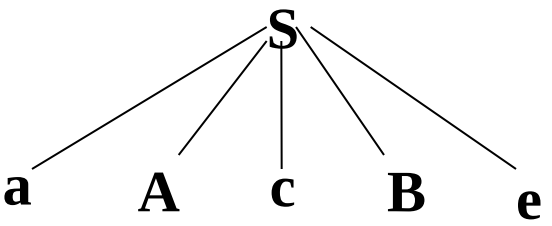
上述概念可以从语法树中形象地给出.并用修剪语法树的方法,阐明分析过程.

例如: 句子**abbcde**的语法树为


S
a
A
c
B
e
A
b
d
b


S
a
A
c
B
e
A
b
d


S
a
A
c
B
e
d


S
a
A
c
B
e

3 “移进-归约”分析法的栈实现:

移进-归约分析器使用符号栈和输入缓冲区.用“#”作为栈底.分析器的工作过程中,对符号栈的使用有四类操作:“移进”,“归约”,“接受”,和“出错处理”.

值得注意:任何可归约串的出现必须在栈顶.

例如:对于文法(5.2)P88,输入串i1*i2+i3的分析(规范归约)步骤如下:

例:只含*, +和( )的算术表达式的文法

$$E \longrightarrow E+E|E*E|(E)|i \quad (3.1)$$

例如:对于文法(3.1) ,输入串i1*i2+i3的分析(规范归约)步骤如下:

步骤	符号栈	输入串	动作
0	#	i1*i2+i3#	预备
1	# i1	*i2+i3#	进
2	#E	*i2+i3#	归,E→i
3	#E*	i2+i3#	进
4	#E*i2	+i3#	进
5	#E*E	+i3#	归,E→i
6	#E	+i3#	归,E→E*E
7	#E+	i3#	进
8	#E+i3	#	进
9	#E+E	#	归,E→i
10	#E	#	归,E→E+E
11	#E	#	接受

例如: 以下述文法为例说明 $i_1*i_2+i_3$ 是文法的句子,及术语举例.

$$E \longrightarrow T|E+T$$
$$T \longrightarrow F|T*F \qquad (5.2)$$
$$F \longrightarrow (E)|i$$

对于文法(5.2),输入串 $i_1*i_2+i_3$ 的分析(规范归约)步骤如 P88 例5.3 所示。

步骤	符号栈	输入串	动作
0	#	$i_1 * i_2 + i_3$ #	预备
1	# i_1	$* i_2 + i_3$ #	进
2	# F	$* i_2 + i_3$ #	归,用 F→i
3	# T	$* i_2 + i_3$ #	归,用 T→F
4	# T *	$i_2 + i_3$ #	进
5	# T * i_2	$+ i_3$ #	进
6	# T * F	$+ i_3$ #	归,用 F→i
7	# T	$+ i_3$ #	归,用 T→T * F
8	# E	$+ i_3$ #	归,用 E→T
9	# E +	i_3 #	进
10	# E + i_3	#	进
11	# E + F	#	归,用 F→i
12	# E + T	#	归,用 T→F
13	# E	#	归,用 E→E + T
14	# E	#	接受

第4章:语法分析-自下而上分析

- 4.1 自底向上分析法
- 4.2 算符优先分析法
- 4.3 LR分析器
- 4.4 LR(0)分析表的构造
- 4.5 SLR分析表的构造
- 4.6 LR(1) 和LALR(1)分析表简介
- 4.7 软件工具Yacc

4.2 算符优先分析法

易于手工编译,特别适用于表达式,不是规范归约.此归约过程是按终结符的优先级进行.

1 直观算符优先分析法

算符优先分析法的关键是定义两个可能相继出现的终结符之间的优先级.

两个终结符之间的优先关系:

a $\lessdot$ b: a的优先级低于b的优先级.

a $\doteq$ b: a的优先级等于b的优先级

a $\gtrdot$ b: a的优先级高于b的优先级

文法(3.1) E ⟶ E+E|E*E|E^E|(E)|i

的终结符之间的优先关系可用矩阵表示:

(p90-表5.1)

	+	*	^	i	(	)	#
+	`>	<`	<`	<`	<`	`>	`>
*	`>	>`	<`	<`	<`	`>	`>
^	`>	>`	<`	<`	<`	`>	`>
i	`>	`>	`>			`>	`>
(	<`	<`	<`	<`	<`	`=	
)	`>	`>	`>			`>	`>
#	<`	<`	<`	<`	<`		`=

从表中可以看出:

1) 优先级 ^ * + 由高到底
2) * + 左结合, ^ 右结合.
3) 运算对象i要先处理.
4) 先括号内后括号外.
5) 对于#, 任何终结符Q都高于#.

- 分析算法:

  是用两个栈:

  OPTR---运算符栈

  OPND---存放操作数和运算结果.

算法的基本步骤: (P93)

```
FUNC  exp_reduced:opendtype;
{  op----   + * ^  }
  initstack(optr);   push(optr,'#');
  initstack(opnd);
  read(w);
  while not  ((w='#') and (gettop(optr)='#')) do
    if w not in op  then [push(opnd,w);
                                read(w)  ]
       else case preced(gettop(optr),w) of
          '<':[push(optr,w),read(w);]
          '=':[x:=pop(optr);read(w);]
```

```
        '>':[theta:=pop(optr);
             b:=pop(opnd);
             a:=pop(opnd);
             push(opnd,operate(a,theta,b))   ]
           endc;
    return(gettop(opnd))
endf;
```

- 优先函数:

  在实际实现算法分析时;用两个优先函数f,g,把两个终结符Q与两个自然数f(Q)和g(Q)相对应.

使得:

若Q1<Q2 则 f(Q1) < g(Q2)

若Q1=Q2 则 f(Q1) = g(Q2)

若Q1>Q2 则 f(Q1) > g(Q2)

函数f成为入栈优先函数.

函数g成为比较优先函数.

	+	*	^	(	)	i	#
f	2	4	5	0	6	6	0
g	1	3	6	7	0	7	0

优先函数的优缺点:

优点: 便于比较,使用方便.

缺点:

1)每个终结符都对应一对优先函数, 而数是可以比较的,所以容易掩盖错误.

例如: i * + ( ) 认为是正确的句子.

2)不便区分一目运算“-”“+”,和二目运算“-”“+”.

2 算符优先文法和优先表的构造:

算符优先文法:由它生成的句型不出现两个非终结符并列的情况.即不含下列形式的句型: ……RS…… 其中R,S∈V_N.

若算符文法不含形如 P—→ε的产生式,对任何一对终结符(a,b),称:

1) a=' b,当且仅当有形如P—→…ab…或P—→…aEb…的产生式,

2) a<' b,当且仅当有形如P—→…aR…且R=±=>b…或R=±=>Qb...的产生式,

3) a>' b,当且仅当有形如P→ …Rb…且

$R \overset{+}{\Longrightarrow} \ldots a$或$R \overset{+}{\Longrightarrow} \ldots aQ$的产生式,

若这个文法的任何一对终结符(a,b)只满足下列关系之一

a=' b,　a<' b,　a>' b 则称该文法是算符先文法.

例如: 考虑下述文法,看其是否为算符优先文法:

(1)　E ⟶ T|E+T

(2)　T ⟶ F|T*F

(3)　F⟶ P|P^F

(4)　P ⟶ (E) |　i

由(4) 可得: (=‘)

由E⟶E+T及T==>T*F 可得: + <' *

由T⟶F|T*F及F=>P|P^F 可得: * <' ^

由E⟶E+T及 E=>E+T 可得: + >' +

由F⟶P^F 可得: ^ <' ^

由P⟶(E) 及

E=>E+T=>T+T=>T*F+T=>F*F+T

=>P^F*F+T=>i^F*F+T

可得: (<' + , (<' * , (<' ^ , (<' i.

同理可得:+ >'), * >') , ^ >') , i >').

所以这个文法是算符优先文法.

优先关系表的构造:

1) 构造文法的每个非终结符的首(尾)符号集合:

$FIRSTVT(P)=\{a \mid P \overset{+}{\Rightarrow} a\ldots$

或$P \overset{+}{\Rightarrow} Qa\ldots$, $a \in V_T$，P，$Q \in V_N\}$

$LASTVT(P)=\{a \mid P \overset{+}{\Rightarrow} \ldots a$

或$P \overset{+}{\Rightarrow} \ldots aQ$, $a \in V_T$，P，$Q \in V_N\}$

有了这两个集合就可以定义：<‘ >’而=‘可从文法直接找到.

如果形如$P \longrightarrow \ldots aQ \ldots$的产生式,且任何$b \in FIRSTVT(Q)$,满足$a <‘ b$.

同理, 如果形如$P \longrightarrow \ldots Qb \ldots$的产生式,且任何$a \in LASTVT(Q)$,满足$a >‘ b$.

2)构造集合FIRSTVT(P)的算法:

用下面两条规则构造集合FIRSTVT(P) :

a) 若有产生式P⟶a…或P⟶Qa,

则 a ∈FIRSTVT(P).

b) 若a ∈FIRSTVT(Q),且有产生式

P→Q…, 则a ∈FIRSTVT(P).

下面是求F[p,a]的值得算法:

```
proc  insert(p,a);
 if not  F[p,a]  then
begin     F[p,a]:=TRUE; 把(p,a)推进栈
end;
```

下面是主程序:

```
 begin
     for 每个非终结符P和终结符a  do
                F[P,a]:=FALSE;
     for  每个形如P→a…或P→Qa…的产生式 do
                insert(P,a);
     while  stack 非空     do
      begin
        把stack的栈顶项,记为 (Q,a),上托出去;
        for  每条形如P→Q…的产生式  do
               insert(P,a);
      end of while;
end
FIRSTVT(P)={a | F[P,a] = TRUE }
```

同理可以构造计算LASTVT(P)的算法.

3)　构造优先表的算法是:

```
for   每条产生式 P→X1X2…Xn do
   for  i:=1 to n-1  do
    begin
      if  Xi和Xi+1均为终结符  then 置 Xi=‘ Xi+1
      if  i<‘ n-2 且 Xi和Xi+2都为终结符
          但Xi+1为非终结符   then  置 X=‘ Xi+2;
      if  Xi 为终结符而Xi+1 为非终结符   then
          for  FIRSTVT(Xi+1)中的每个a   do
               置Xi<‘ a
      if  Xi 为非终结符而Xi+1 为终结符   then
          for  LASTVT(Xi+1)中的每个a   do
               置 a>’Xi+1
end
```

3 算符优先分析法的设计:

算符优先分析法的“可归约串”是最短素短语.

素短语:是指这样的一个短语,它至少含有一个终结符,并且除它自身之外不再含任何更小的素短语.

最左素短语:处于句型最左边的那个素短语.

例如: 考虑下述文法,的句型P*P+i的素短语.

(1) E⟶T|E+T

(2) T⟶F|T*F 的素短语是P*P和i.

(3) F⟶P|P^F 的最左素短语是P*P.

(4) P⟶(E) | i

一个算符优先文法的句型具有以下形式:

#N1a1N2a2 … Nn an Nn+1an+1# (*)

可以证明如下定理:

形如(*)的算符优先文法的句型是最左素短语应满足如下关系的最左子串:

Nj aj Nj+1aj+1 … Ni ai Ni+1

aj-1 <‘ aj

aj =‘ aj+1 =‘ … =‘ ai

ai >’ ai+1

下面给出算符优先分析法的算法思想

算法思想:

符号栈---S,寄存终结符和非终结符.

K---表示S的使用的深度.

(1) a为最左素短语的最右端: (>‘):

归约.

(2) a为最左素短语的左,中端:(<’ =‘):

入栈.

算法: (P92)

注意:算符优先分析一般不等于规范归约.

4 优先函数:

由 优先关系表 构造 优先函数.

优先函数的优点: 节省存储空间,便于执行比较运算.

有许多优先关系表 不存在 优先函数.

如果优先函数存在,从优先关系表 构造 优先函数的方法如下:

1) 对于每个终结符a(包括#)令其对应两个符号fa和ga,画一张以所有符号fa和ga为结点的方向图.

若有 a ’> ‘=b , 则画 由fa至gb的箭弧.

若有 a ’< ‘=b , 则画 由gb至fa的箭弧.

2) 给每个结点赋一个数,该数等于从每个结点出发所能到达的所有结点(包括出发点自身)的个数,赋给fa(gb)的数就是函数f(a)(g(b))的值.

3) 检查这些优先函数是否满足优先表,若满足,则它们是这个表的优先函数.

证明: 1) 若a ’= b, 则f(a)=g(b).

2) 若a ’> b, 则f(a)>g(b).

3) 若a ’< b, 则f(a)<g(b).

1) 可直接从构造算法中得到, 因为a’=b, 则既有从fa到gb的弧,又有从gb到fa的弧, 所以fa, gb所到达的结点的个数相同,即f(a)=g(b).

2)和3)只需证明其一.现证(2).

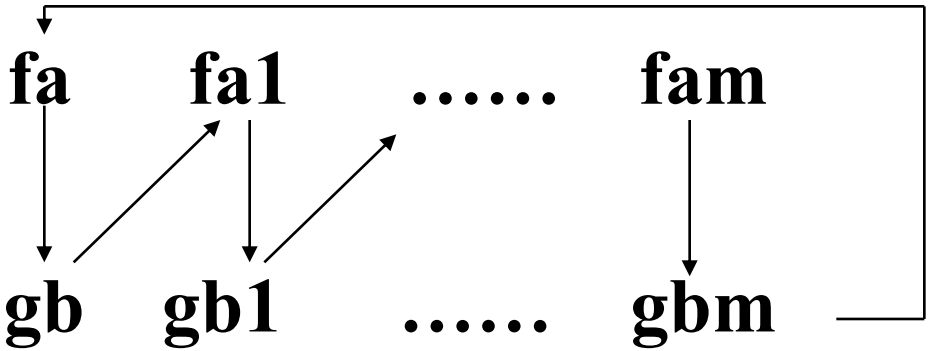
因为 a’>b 所以 有从fa到gb的弧,

故gb能到达的结点,fa一定能到达,

因此,至少有f(a)>=g(b), 现证f(a)=g(b)

不成立.否则不存在优先函数,

用反证法:假设f(a)=g(b), 必有回路:



从该图可得:

a’>b,a1<‘=‘b,a1’>’=b1,…am’>bm,a<‘=‘bm.

按定义可得:

f(a)>g(b)>=f(a1)>=g(b)……f(am)>=g(bm)>=f(a).

从而导致**f(a)>f(a),**产生矛盾.

因此,不存在优先函数**f**和**g.**

例如: 根据**P90 (**表**5.1)** 去掉**i**和**#**两个符号求其所对应的方向图,并构造优先函数表.

解答: **(**参看**P95** 图**5.2)**

从**(f+)**出发所能到达的结点: **f+ , g+, g) , f(,** 故 函数值**=4……**其它同理.

第4章:语法分析-自下而上分析

- 4.1 自底向上分析法
- 4.2 算符优先分析法
- 4.3 LR分析器
- 4.4 LR(0)分析表的构造
- 4.5 SLR分析表的构造
- 4.6 LR(1) 和LALR(1)分析表简介
- 4.7 软件工具Yacc

4.3 LR分析器

LR分析法 是自低向上的语法分析,L指的是从左向右扫描输入串,R指的是按照最右推导的逆过程进行归约,LR分析法在每次决定移进或归约之前,一般的说,需要向前查看k个输入符号,所以写成LR(K),通常K=1, 重点介绍LR(0),SLR(1),简单介绍LALR(1),
LR(K),语法分析程序的自动构造介绍YACC.

LR分析器

举例说明LR分析器的工作过程

例如:下述文法的一个LR分析表:(P101 图5.5).

(1) E→E+T　　(4) T → F

(2) E→T　　(5) F →(E)

(3) T→T*F　　(6) F → i

其中: 1) sj 把下一个状态j和现行输入符号a移进栈;

2) rj 按第j个产生式进行归约;

3) acc 接受;

4) 空白格 出错标志, 报错.

注意, 若a为终结符,则GOTO[S,a]的值已列在action[S,a]的sj之中(状态j).因此,GOTO表仅对所有非终结符A列出GOTO[S,A]的值.

利用这张分析表,假定输入串为:i*i+i,分析器的工作过程(即,三元式的变化过程)如图5.5分析表(书P102).

利用图5.5 关于 i*i+i的分析步骤 中加入 动作列 由分析表查出 (1) s5 (2) r6 (3) r4
(4) s7 (5) s5 (6) r6 (7) r3 (8) r2 (9) s6
(10) s5 (11) r6 (12) r4 (13) r1 (14) acc.

在分析过程中,跟踪分析细节:

(1)由于初态栈顶 状态为0,当前输入符号i, action[0,i]=s5,所以,移进i和状态5.此时输入符号变为* ,栈顶状态变成5

(2) 输入符号变为* ,栈顶 状态变成5,
由**action[5,*]=r6,**有产生式**6** 归约,从栈中弹出状态**5**和符号**i,**使栈顶状态变成**0,**归约出的非终结符为**F;**再由**goto[0,F]=3,**把**F**和状态**3**依次压入栈.于是,分析的内容如表**(3)**所示.
再由**action[3,*]=r4,**有产生式**4** 归约,从栈中弹出状态**3**和符号**F,**使栈顶状态变成**0,**归约出的非终结符为**T;**在由**goto[0,T]=3,**把**T**和状态**2**压入栈.
……继续分析……,

……继续分析……,

分析的内容如表(13)所示.

(3)再由action[9,#]=r1,有产生式1归约,从栈中弹出3个状态和3个符号,使栈顶状态又变成0,归约出的非终结符为E;再由goto[0,E]=1,把E和状态1压入栈.得到(14)再由action[1,#]=acc,接收 .

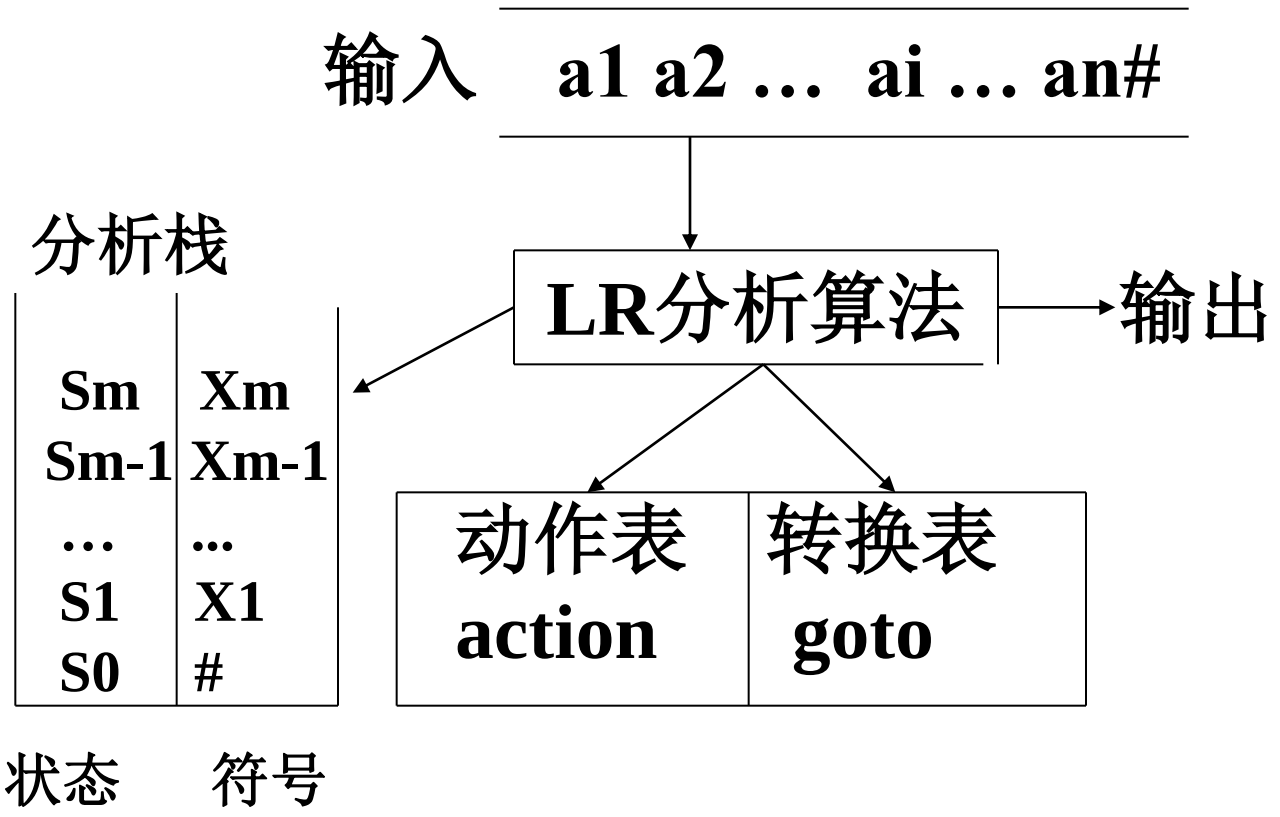
LR分析器的组成:

1) 分析栈:存放“状态”和移进,归约的文法符号:

S0#X1S1… Sm-1 Xm-1 Sm Xm

其中,Si表示状态,Xi表示文法符号;在实现中,文法符号不必进分析栈.

LR分析器示意图:


输入 a1 a2 … ai … an#
分析栈
LR分析算法
输出
Sm Xm
Sm-1 Xm-1
… ...
S1 X1
S0 #
动作表
action
转换表
goto
状态
符号

2)动作表: action[Sm,ai]表示下一个输入符号为ai, 栈顶状态为Sm时,分析算法应执行的动作;若action[sm,ai]=si,表示移进,然后栈顶状态为i;若action[sm,ai]= rj表示使用产生式(j)归约;若action[sm,ai]= acc表示接受输入串;若action[sm,ai]= err表示语法错误.

3)状态转换表: goto[Si,X]表示归约出非终结符号X之后,当前栈顶状态为Si时,分析栈应转换到的下一个状态,即栈顶的新状态.

4)LR分析算法:根据输入符号ai,栈顶状态Sm和动作表项action[Sm,ai]的值,决定当前分析应执行的动作:移进,归约,接受或出错;移进或归约之后要根据动作表或状态转换表设置分析栈的状态.

下面说明**LR**分析器的工作原理**.**

分析栈中的串和等待输入的符号串过程如下形式的三元组:

(S0S1 S2…Sm,#X1X2…Xm ,ai ai+1 …an #)

其初态为:

(S0, # ,a1a2 ...ai ai+1 …an #)

假定当前分析栈的栈顶为状态**Sm,**下一个输入符号为**ai,**分析器的下一个动作由动作表项**action[Sm,ai]**决定:

1)如果**action[Sm,ai]=**移进**S**且**s=GOTO[Sm,ai],**则分析器执行移进,三元组变成

(S0S1...SmS ,#X1…Xmai, ai+1…an #)

即分析器将输入符号**ai**和状态**S**移进栈,**ai+1**变成下一个符号.

(2)如果**action[Sm,ai]=**归约**A** β,则分析器执行归约,三元组变成

(S0S1...Sm-rS, #X1…Xm-rA, ai ai+1…an #)

此处**S=goto[Sm-r,A],r**为β的长度且**β=Xm-r+1Xm-r+2…Xm.**

(3) 若**action[Sm,ai]=acc,**则接收输入符号串**,**语法分析完成**.**

(4) 若**action[Sm,ai]=err,**则发现语法错误**,**调用错误恢复子程序进行处理**.**

一个**LR**分析器的工作过程就是一步一步的变换三元式**,**直至到“接受”或“报错”为止**.** 此方法同样适合于**SLR**分析器和**LALR**分析器**.**

状 态	ACTION（动作）						GOTO（转换）		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

图 5.5　LR 分析表

可见,LR分析表是进行LR分析的关键.一个文法,如果能为其构造LR分析表,则这个文法为LR文法.

第4章:语法分析-自下而上分析

- 4.1 自底向上分析法
- 4.2 算符优先分析法
- 4.3 LR分析器
- 4.4 LR(0)分析表的构造
- 4.5 SLR分析表的构造
- 4.6 LR(1) 和LALR(1)分析表简介
- 4.7 软件工具Yacc

4.4 LR(0)分析表的构造

1 LR(0)项目集族

定义:　　字的前缀:是指该字的任意首部.

例如,字abc的前缀有ε,a,ab或abc.

　　活前缀:是指规范句型的一个前缀,这种前缀不含句柄之后的任何符号.

之所以称为活前缀,是因为在右边增添一些终结符号之后,就可以使它成为一个规范句型

例如: 设有下列文法:

(1) S⟶var I : T

(2) I⟶I, id

(3) I⟶id

(4) T⟶ real

关于该文法的句型 var I ,id : real 其活前缀为: “var I”,”var I,id”和 “e”.但var I ,id : 不是.因为,“I,id”是句柄.

要想构造LR分析表,首先构造一个有限自动机DFA,它能识别G的所有活前缀.引入LR(0)项目代替活前缀,用来构造DFA,使DFA的每个状态和有穷个LR(0)项目的集合相关联.再由DFA构造LR分析表.构造DFA和LR分析表时,需要处理可能发生的冲突;策略不同,从而产生不同的LR分析表.

LR(0)项目(简称项目):文法G每个产生式的右部添加一个园点. 例如: 产生式

A ⟶ XYZ对应有四个项目:

A⟶.XYZ　　A⟶X.YZ

A⟶XY.Z　　A⟶XYZ.

但是,产生式 A⟶ε只对应一个项目A⟶ .,

例如: (P105 文法5.7)

这个文法的项目: 有18个

使用这些项目构造一个NFA. 方法:

(1) 项目1 为唯一的初态.任何项目为NFA的终态.

(2) 如状态i和j出自同一个产生式,且状态i

X⟶X1…Xi-1 . Xi…Xn

而状态j为

X⟶X1…Xi . Xi+1…Xn

那么,从状态i画一条标志为Xi的弧到状态j.假若状态i的圆点之后的那个符号为非终结符,如i为X⟶ α.Aβ,A为非终结符,则,就从状态i画ε弧到所有A⟶. γ状态(即,所有哪些圆点出现在最左边的A的项目).

例如:识别 (文法5.7)p105活前缀的NFA,如图5.6(p106)所示.

在计算机中,项目可用一对整数表示,相当于坐标 (产生式编号,圆点的位置).它反映分析某时刻能看到产生式多大部分. 是移进还是归约.

“归约”项目:凡圆点在最右端的项目,如A→α．,

“接受”项目:一种特殊的归约项目,对文法开始符号S’的归约项目,如S’→α．,

“移进”项目:形如A→α．aβ的项目,其中a为终结符,

“待约”项目:形如A→α . Bβ的项目,其中B为非终结符.

LR(0)项目集规范族的构造

1. ε-CLOSURE(闭包)的办法构造.

假定I是文法G’的任意项目集,

1) I的任何项目都属于CLOSURE(I);

2) 若A⟶α . Bβ属于CLOSURE(I),那么,对任何关于B的产生式B⟶γ,项目B → .γ也属于CLOSURE(I);

3)重复执行上述两步骤直至CLOSURE(I)不再增大为止.

例如:文法(5.7), 假若I = {S’→ .E},那么,CLOSURE(I)所含的项目为:

S’→ .E

E → .aA

E→ .bB

2. 状态转换函数GO的构造.

GO(I,X)=CLOSURE(J)

其中: J={任何形如A→ αX.β的项目 | A→ α.Xβ属于I}.

例如:项目集0:{S’→ .E, E→ .aA, E→ .bB } 那么,GO(I,a)就是该项目集2: {E→ a.A, A→.cA, A→.d}

是检查I中所有那些圆点之后紧跟着a的项目.

3. LR(0)项目集规范族的构造

通过函数CLOSURE和GO构造一个文法G的拓广的文法G‘的LR(0)项目集规范族.

算法:

```
proce  itemsets(G');
begin
      c:={ closure( {S'→.S})};
      repeat
        for   c 中的每个项目集I和G'的每个符号X   do
          if  GO(I,X) 非空且不属于c   then
                GO(I,X)   放入c族中
      until   c不再增大
end.
```

例如: (P105 文法5.7)

文法的项目 (如 P105 所示)

其LR(0)项目集规范族. (p106如图5.7所示)

12个集合, 转换函数GO把这些集合联成一张DFA转换图.

2 有效项目

定义: 项目A→β1.β2对活前缀αβ1是有效的,其条件是存在规范推导

$S' \overset{*}{\Longrightarrow} \alpha A \omega \overset{*}{\Longrightarrow} \alpha\beta 1\beta 2\omega$.

事实上,活前缀γ有效项目集,是从上述的DFA的初态出发,经读出γ后而得到的那个项目集(状态).

<u>定理: (LR分析理论)</u>

在任何时候,分析栈中的活前缀X1X2…Xm的有效项目集 正是栈顶状态Sm所代表的那个集合.

例如:(文法5.7) p106图5.7 DFA 的一个活前缀bc,使DFA 到达 5 , 5有三个项目

B⟶ c.B

B⟶ . cB

B⟶ . d

下面说明项目集对bc是有效的. 考虑下面三个推导:

(1) S’=>E=>bB=>bcB

(2) S’=>E=>bcB =>bccB

(3) S’=>E =>bB =>bcB =>bcd

(1)推导表明B⟶ c.B的有效性.

(2)推导表明B⟶ . cB的有效性

(3)推导表明B⟶ . d的有效性

3 LR(0)分析表的构造

定义: (LR(0)文法)

LR(0)文法的规范族的每个项目集不包含任何冲突项目.这个文法称为LR(0)文法.

构造LR(0)分析表

算法思想: 根据DFA和转换函数GO构造action[ ]和goto[ ]表.

算法: 有识别活前缀的DFA构造LR(0)分析表.

输入: 识别LR(0)文法G的活前缀的DFA.

输出:文法G的LR(0)分析表,包括action表

和goto表.

方法:对每个状态Ii按下列步骤执行:

(1)if 状态Ii中存在形如$A \longrightarrow \alpha.X\beta$的项目,

且状态转换函数go(Ii,X)=Ij then

```
begin
    if X=a 是终结符号 then action[I,a]=sj;
    if X 是非终结符号  then goto[i,X]=j;
end;
```

(2) if 状态**Ii**中存在形如**A⟶ α .**的归约项目

then 对任何终结符**a, action[i,a]=rj;**

(3) if 状态**Ii**中存在接受项目**S'⟶ S.**

then action[i,#] = acc;

(4)分析表中凡不能用规则 **(1),(2)**或**(3)**填入信息的空白元素都置为“**err**”.

按上述方法构造的分析表的每个入口都是唯一的.称此分析表是一个**LR(0)**表.使用**LR(0)**表的分析器叫做一个**LR(0)**分析器.

例如:文法**5.7** 的**LR(0)**分析表如表**5.4 (P109).**

例如: 设有下列文法:

(0) S' → S

(1) S → var I : T

(2) I → I, id

(3) I → id

(4) T → real

构造它的LR(0)分析表.

解: 1) 关于文法的全部LR(0)项目如下:

(0) S' → . S \\待约

(1) S' → S . \\接受

(2) S → . var I : T \\移进

(3) S ⟶ var . I : T \\待约

(4) S ⟶ var I . : T \\移进

(5) S ⟶ var I : . T \\待约

(6) S ⟶ var I : T . \\句柄

(7) I ⟶ . I, id \\待约

(8) I ⟶ I . , id \\移进

(9) I ⟶ I , . id \\移进

(10) I ⟶ I , id . \\句柄

(11) I ⟶ . id \\移进

(12) I ⟶ id . \\句柄

(13) T ⟶ . real \\移进

(14) T ⟶ real . \\句柄

2) 应用算法**closure,**找出其等价项目,
使每个项目集表示一个状态**Ii.**各项目对应的项目集如下**:**

I0={ (0) S’⟶ . S

(2) S⟶ . var I : T }

I1={ (1) S’⟶ S . }

I2={ (3) S⟶ var . I : T

(7) I⟶ . I, id

(11) I⟶ . id }

I3={ (4) S⟶ var I . : T

(8) I⟶ I . , id }

I4={ (12) I⟶ id . }

I5= { (5) S ⟶ var I : . T

(13) T⟶ . real }

I6= { (9) I ⟶ I , . Id }

I7= { (10) I ⟶ I , id . }

I8= { (6) S ⟶ var I : T . }

I9= { (14) T ⟶ real . }

每个等价项目集表示DFA的一个状态.下面构造识别文法规范句型的活前缀的DFA.DFA的开始状态为包含初态项目的状态I0,其状态转换函数为go,对状态Ii中的任意项目A ⟶ α . Xβ,

其中X为任意的一个文法符号,当移进或归约出X之后,应转换到其后继项目A⟶ αX.β所在的状态Ij,记为:

go (Ii , X) = Ij

于是,得到:

go(I0, S) =I1　　go(I0,var) =I2

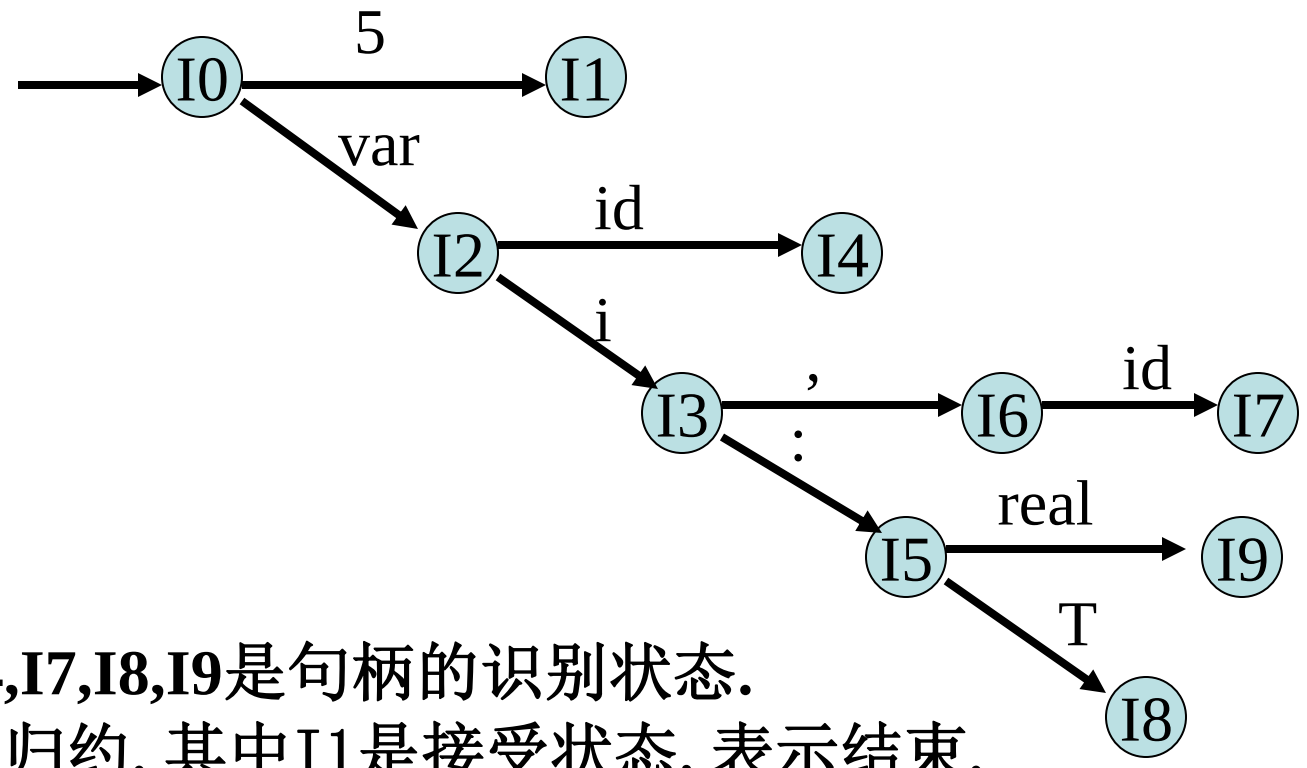
go(I2, I) =I3　　go(I2, id) =I4

go(I3, :) =I5　　go(I3, ,) =I6

go(I6, id) =I7　　go(I5, T) =I8

go(I5, real) =I9

于是,画出识别文法的规范句型的活前缀的DFA如下图所示:



I1,I4,I7,I8,I9是句柄的识别状态.
可以归约. 其中I1是接受状态, 表示结束.

构造LR(0)分析表:

状态	action						goto		
	var	id	,	:	real	#	S	I	T
0	s2						1		
1						acc			
2		s4						3	
3			s6	s5					
4	r3	r3	r3	r3	r3	r3			
5					s9				
6		s7							8
7	r2	r2	r2	r2	r2	r2			
8	r1	r1	r1	r1	r1	r1			
9	r4	r4	r4	r4	r4	r4			

第4章:语法分析-自下而上分析

- 4.1 自底向上分析法
- 4.2 算符优先分析法
- 4.3 LR分析器
- 4.4 LR(0)分析表的构造
- 4.5 SLR分析表的构造
- 4.6 LR(1) 和LALR(1)分析表简介
- 4.7 软件工具Yacc

4.5 SLR分析表的构造

LR(0)文法是一类非常简单的文法.但这种文法的使用范围十分小.如果这种文法的活前缀识别自动机的某一个状态含冲突性的项目,则需研究另一种文法SLR文法.

例如:假定一个LR(0)规范族中含有如下的一个项目集I,

I={ $X \longrightarrow \alpha . b\beta$, \\移进

$A \longrightarrow \alpha .$, \\归约

$B \longrightarrow \alpha .$, } \\归约

这三个项目说明发生“移进-归约”冲突.解决方法:分析说有含A或B的句型,考察句型中可以直接跟在A或B之后的终结符,即FOLLOW(A)和FOLLOW(B),如果这两个集合不相交,而且都不包含b,那么,当状态I面临任何输入符号a时,可以决定“移进”或是“归约”.

1. 若a=b,则移进;
2. 若a∈FOLLOW（A）,
   则用产生式A $\longrightarrow$α进行归约.
3. 若a∈FOLLOW（B）,
   则用产生式B $\longrightarrow$ α进行归约.

定义: (合法项目)

如果存在最右推导

$S' =*\Rightarrow \alpha Aw =*\Rightarrow \alpha\beta 1\beta 2w$

则称 $A \longrightarrow \beta 1.\beta 2$对活前缀$\alpha\beta 1$是一个合法项目.

事实上,当在栈顶发现$\alpha\beta 1$时,可以作出移进或归约的决定.特别是,如果$\beta 2 <> \varepsilon$,则认为栈顶还没形成句柄,所以移进;如果$\beta 2 = \varepsilon$,则$\beta 1$看成句柄,所以用产生式$A \rightarrow \beta 1$归约.

SLR分析表解决冲突的方法,采用通过查看下一个输入符号来解决.即简单“展望”材料的LR分析法,(SLR法).

例题5.11 (P111)是表达式的拓广文法.

(1) 其LR(0)等价项目集 I1 -- I11{如(P111)所示}.

(2) 由以上项目集的转换函数GO表示的{(P112)图5.8所示}的DFA.即:文法的活前缀识别自动机.

(3)在这12个项目集中,I1,I2和I9含有“移进-归约”冲突. 用SLR(1)可以解决冲突.

例如:考虑I2　　E ——→T .

T——→T . * F

由于FOLLOW(E)={ # ,) , + }, 所以,当状态I2面临输入符号为 # ,) , + 时,使用产生式E——→T进行归约; 当面临*时,应移进;若面临其它符号则报错.

一般的说,假定拓广文法G'的LR(0)项目集I中有m可移进项目

A1——→α . a1β1

A2 ——→α . a2β2　　…….

Am ——→α . amβm

和n个归约项目

$B1 \longrightarrow \alpha .$

$B2 \longrightarrow \alpha .$ ……

$Bn \longrightarrow \alpha .$

如果集合

{a1,a2,…am},FOLLOW(B1), …

FOLLOW(Bn)两两不相交,并且不与输入结束符#相交,则,

(1) 若a是某个ai , i=1,2,…,m,则移进;

(2) 若a∈FOLLOW（Bi）, i=1,2,…n,

则用产生式$Bi \longrightarrow \alpha$进行归约;

(3) 此外,报错. 称SLR(1) 办法.

算法: (构造SLR(1)分析表)

输入: 拓广文法G'和识别其活前缀的DFA.

输出:文法G的SLR(1)分析表,包括action表

和goto表.

方法:对每个状态Ii按下列步骤执行:

(1)if 状态Ii中存在形如A⟶α.Xβ的项目,

且状态转换函数go(Ii,X)=Ij then

```
begin
    if X=a 是终结符号 then action[i,a]=sj;
    if X 是非终结符号  then goto[i,X]=j;
end;
```

(2) if 状态Ii中存在形如A⟶α.的归约项目

then 对任何终结符a∈FOLLOW（A），

action[i,a]=rj;

(3) if 状态Ii中存在接受项目S'⟶S.

then action[i,#] = acc;

(4)分析表中凡不能用规则 (1),(2)或(3)填入信息的空白元素都置为“err”.

按上述方法构造的分析表的每个入口都是唯一的.称此分析表是一个SLR(1)表.使用SLR(1)表的分析器叫做一个SLR(1)分析器.

例如: 文法5.8(P112)是表达式的拓广文法

其SLR（1）的分析表（P101）图5.5 。

(0) S′→E

(1) E→E + T

(2) E→T

(3) T→T * F

(4) T→F

(5) F→(E)

(6) F→i

注意:SLR(1)文法都是无二义性的.但是,无二义性的文法并不都是SLR(1)的.

状 态	ACTION（动作）						GOTO（转换）		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

图 5.5　LR 分析表

第4章:语法分析-自下而上分析

- 4.1 自底向上分析法
- 4.2 算符优先分析法
- 4.3 LR分析器
- 4.4 LR(0)分析表的构造
- 4.5 SLR分析表的构造
- 4.6 LR(1) 和LALR(1)分析表简介
- 4.7 软件工具Yacc

4.6 LR(1) 和LALR(1)分析表简介

- 例如: 文法4.4 (P112)
  考虑I2,第一个项目使action[2,=]为s6,第二个项目,由于FOLLOW(R)含有=(因有S=>L=R=>*R=R) ,使action[2,=]为 “用R-->L

归约”.因此,状态2当前面临输入符号=时,存在“移进-归约”冲突.因此,SLR(1)无法分析.

LR(1)分析表和SLR(1)的区别:

在SLR(1)构造算法中,没有考虑 当前符号左侧的上下文,一般地,要认真地考虑当前符号左侧的上下文,甚至于一个符号是否是合法的后记符号也要依赖左边的上下文.因此需要重新定义项目集.使之能够包含后记符号,后记符号是可能进行归约时的超前搜索符.

LR(1)项目的一般形式为:

[A⟶α . β , a] 其中, A⟶α . β 是一个产生式, a表示一个终结符或者输入串的结束标志#(后继符号).

后继符号是可能合法地跟在非终结符后面的符号,因此,当项目集相同(在SLR(1)意义下),而后记符号不同时,他们是不同的状态.故LR(1)分析表中的状态数比LR(0)和SLR(1)分析表的状态多许多,但它能分析所有的LR(1)语言.

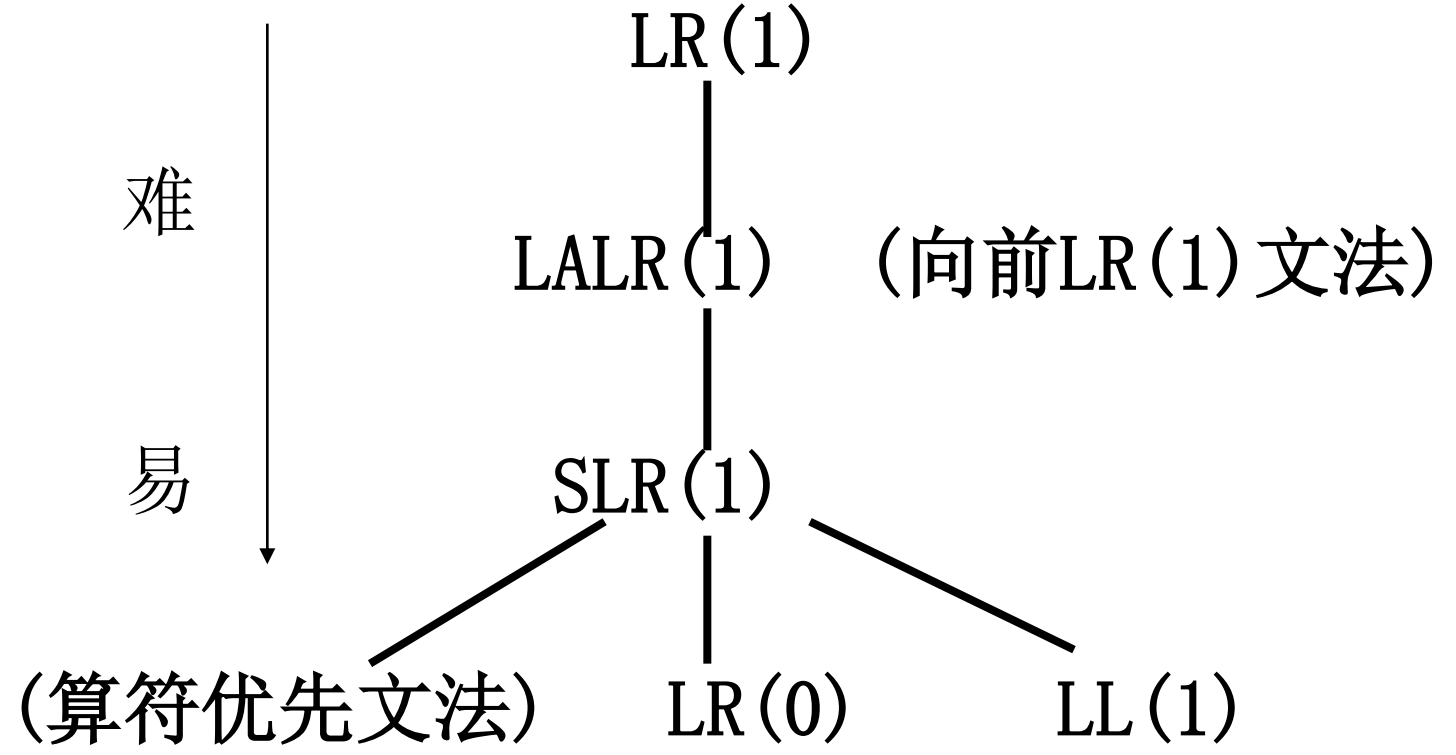
事实上,程序设计语言大部分为简单的SLR(1)的.

LALR(1)分析表:

由于LR(1)分析表的状态很多,构造规范LR分析表工作令人望而却步;所以,LR(1)分析表目前还不便于推广使用.为了发挥其优点,对它进行压缩,减少状态个数,从而出现LALR分析表.

LALR(1)分析表,对一个文法产生的状态数和SLR分析表相同.常用的程序设计语言结构一般都可以使用LALR文法描述.

各种文法的等级关系为:



构造一个算法,先从LR(0)开始,若没有遇到冲突就是LR(0)的; 如不成功,就用

SLR(1),若没有遇到冲突就是SLR(1)的;否则就用LALR(1),若没有遇到冲突就是LALR(1)的;否则就用LR(1),若解决了所有的矛盾就是LR(1)的;

一个程序设计文法几乎总是SLR(1)或LALR(1)的.所以,不要从LR(1)开始做.

不同点:

1)LR(1)文法对文法的形势要求部分严格,很少做文法转换;相比之下,LL(1)问法学要仔细推敲,而且文法中的产生式较多.

2)在时间和空间上,LL(1)的体积小;平均时间上,LL(1)的速度快.

3)如果做语法分析程序的自动生成器,必须使用LR(1)文法.

4) LR(1)甚至可以处理二义文法.

规范LR分析

- 例： G[S]: (0) S`→S (1) S→L=R (2) S→R (3) L→ *R (4) L→id (5) R→L

I0: S' –> •S
S –> •L = R
S –> •R
R –> •L
L –> •id
L –> •*R

I1: S' –> S•

I2: S –> L• = R
R –> L•

I3: S –> R•

I4: L –> *•R
R –> •L
L –> •*R
L –> •id

I5: L –> id•

I6: S –> L =•R
R –> •L
L –> •*R
L –> •id

I7: L –> *R•

I8: R –> L•

I9: S –> L=R•

考虑分析表达式 id = id时，I_2 处已经把第一个 id 归约到 L了，看到下一个输入 = 要作决策，第一个项目要设置 Action[2,=] 为S6，但 =也是属于 Follow(R) 的. 第二个项目要用 R–>L归约.出现 移进-归约冲突.

虽然在栈顶的符号序列可以归约到 R,但我们不能要这个选择，因为不可能有规范句型以 R = ...开头 (有以 *R = ... 开头的规范句型).

LR(1)项目

- LR（1）项目的一般形式[A →α.β，a]
  - A →α.β是LR(0)项目,a为终结符，称为向前搜索符
  - 向前搜索符只对圆点在最后的项目，即归约项目起作用
  - [A → αβ •, a]意味着处在栈中是α β的相应状态，但只有当下一个输入符是a时才能进行归约.
- LR(1)项目对某个活前缀有效
  - 一个LR(1)项目[A →α . β,a]对活前缀r=δα 是有效的,如果存在一个规范推导S => δ Aω => δ α β ω,

    a 是ω的第一个符号,或ω=ε而a为#

构造LR(1)项目集族与GO函数

- LR(1)项目集族的构造：针对初始项目S'→•S,#求闭包后再用转换函数逐步求出整个文法的LR(1)项目集族。
- 构造LR（1）项目集的闭包函数

  a)I的项目都在CLOSURE(I)中

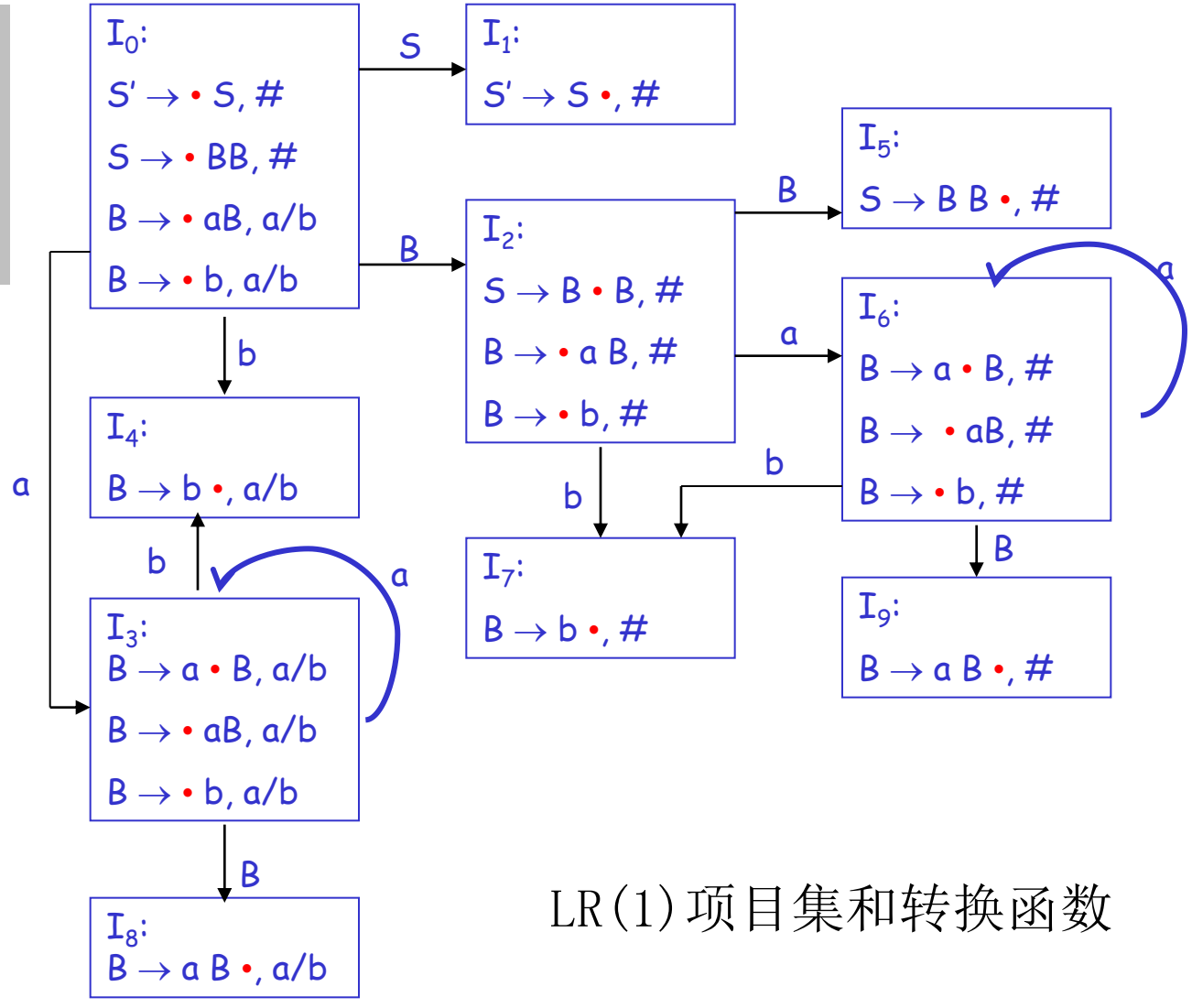
  b)若[A→α • Bβ,a]属于CLOSURE(I)， B→ γ是文法的产生式，β∈V*，b∈FIRST(βa),则[B→•γ,b]也属于CLOSURE(I)

  c)重复b)直到CLOSURE(I)不再扩大
- 转换函数的构造
  - GOTO（I，X）= CLOSURE（J）

    其中：I为LR(1)的项目集，X为一文法符号

    J={任何形如A→αX • β,a的项目| A→α • X β,a属于I}

文法G'：
(0) S' → S
(1) B → aB
(2) S → BB
(3) B → b



LR(1)项目集和转换函数

LR(1)分析表的构造

1) 若项目[A→α·aβ,b]属于I_k，且转换函数GO(I_k,a)= I_j ，当a为终结符时，则置ACTION[k,a]为S_j

2) 若项目[A→α·,a]属于I_k ，则对a为任何终结符或‘#’，置ACTION[k,a] = r_j ，j为产生式在文法G‘中的编号

3) 若GO(I_k,A)= I_j ，则置GOTO[k,A]=j，其中A为非终结符，j为某一状态号

4) 若项目[S‘→S·,#]属于I_k ，则置ACTION[k,#] = acc

5) 其它填上“报错标志”

按上述算法构造的含有ACTION和GOTO两部分的分析表，如果每个入口不含多重定义，则称它为文法G的一张规范的LR(1)分析表。具有规范的LR(1)表的文法G称为一个LR（1）文法。

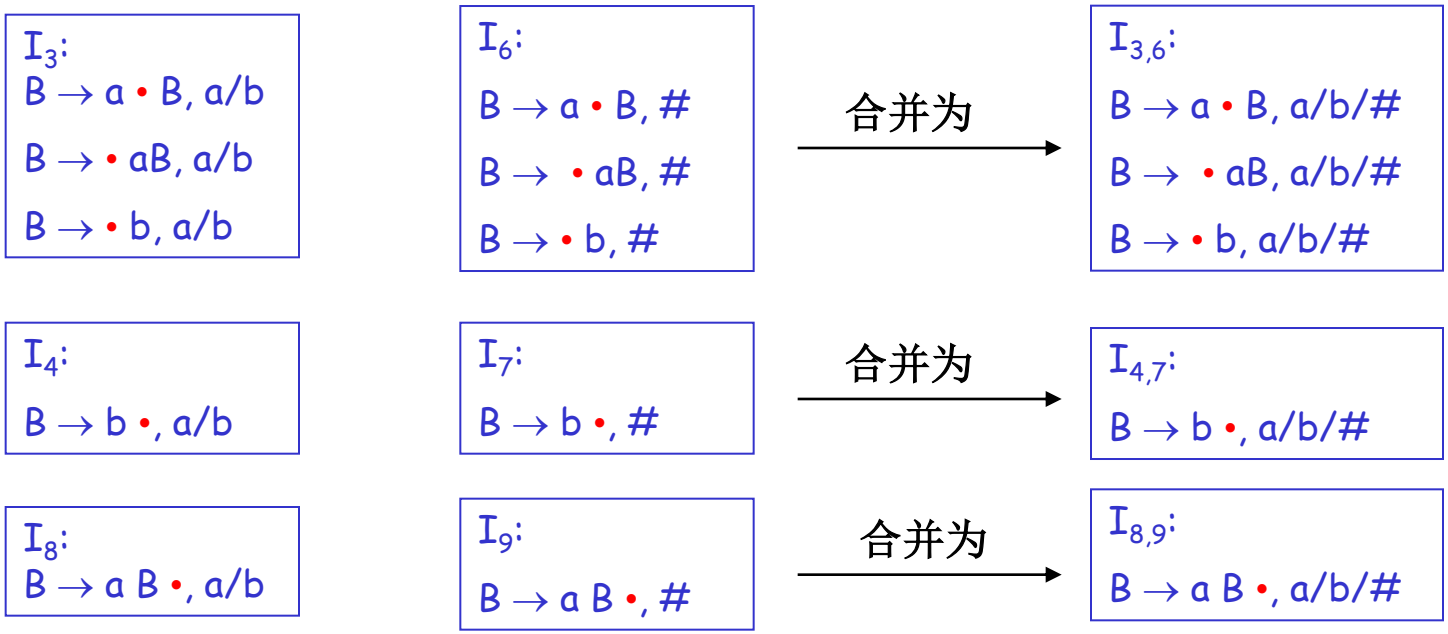
例：LR(1)分析表

状态	ACTION			GOTO	
	a	b	#	S	B
• 0	S3	S4		1	2
• 1			acc		
• 2	S6	S7			5
• 3	S3	S4			8
• 4	r3	r3			
• 5			r1		
• 6	S6	S7			9
• 7			r3		
• 8	r2	r2			
• 9			r2		

LALR(lookahead LR)

分析发现：

- I_3和I_6，I4和I_7，I_8和I_9分别为同心集
- LR(1)的两个项目集如果除去搜索符后相同，则称两个集合同心



合并同心集的几点说明

- 同心集合并后心仍相同，只是超前搜索符集合为各同心集超前搜索符的和集
- 合并同心集后转换函数自动合并
- LR(1)文法合并同心集后也只可能出现归约-归约冲突，而没有移进-归约冲突
- 合并同心集后可能会推迟发现错误的时间，但错误出现的位置仍是准确的

合并同心集后产生归约-归约冲突

例，文法G

S' –> S

S –> aBc | bCc | aCd | bBd

B –> e

C –> e

LR(1) 项目集规范族：

I0: S' –> •S, #
S –> •aBc, #
S –> •bCc, #
S –> •aCd, #
S –> •bBd, #
I1: S' –> S•, #
I2: S –> a•Bc, #
S –> a•Cd, #
B –> •e, c
C –> •e, d
I3: S –> b•Cc, #
S –> b•Bd, #
C –> •e, c
B –> •e, d
I4: S –> aB•c, #
I5: S –> aC•d, #
I6: B –> e•, c
C –> e•, d
I7: S –> bC•c, #
I8: S –> bB•d, #
I9: B –> e•, d
C –> e•, c
I10: S –> aBc•, #
I11: S –> aCd•, #
I12: S –> bCc•, #
I13: S –> bBd•, #

- I69: C –> e •, c/d
B –> e •, d/c

构造 LALR(1)分析表

- 方法1
  - 1.构造文法G的规范 LR(1) 状态.
  - 2.合并同心集的状态.
  - 3.新 LALR(1) 状态的GO函数是合并的同心集状态的GO函数的并.
  - 4. LALR(1)分析表的action 和 goto 登录方法与LR(1)分析表一样

  经上述步骤构造的表若不存在冲突，则称它为G的LALR(1)分析表。

  存在这种分析表的文法称为LALR（1）文法。

LR(1)分析表

状态	ACTION			GOTO	
	a	b	#	S	B
• 0	S3	S4		1	2
• 1			acc		
• 2	S6	S7			5
• 3	S3	S4			8
• 4	r3	r3			
• 5			r1		
• 6	S6	S7			9
• 7			r3		
• 8	r2	r2			
• 9			r2		

LALR(1)分析表

状态	ACTION			GOTO	
	a	b	#	S	B
• 0	$S_{3,6}$	$S_{4,7}$	1	2	
• 1			acc		
• 2	$S_{3,6}$	$S_{4,7}$		5	
• 3,6	$S_{3,6}$	$S_{4,7}$		8,9	
• 4,7	r_3	r_3	r_3		
• 5			r_1		
• 8,9	r_2	r_2	r_2		

状态	ACTION			GOTO	
	a	b	#	S	B
0	S_3	S_4		1	2
1			acc		
2	S_6	S_7			5
3	S_3	S_4			8
4	r_3	r_3			
5			r_1		
6	S_6	S_7			9
7			r_3		
8	r_2	r_2			
9			r_2		

合并同心集后

状态	ACTION			GOTO	
	a	b	#	S	B
0	$S_{3,6}$	$S_{4,7}$		1	2
1			acc		
2	$S_{3,6}$	$S_{4,7}$			5
3,6	$S_{3,6}$	$S_{4,7}$			8,9
4,7	r_3	r_3	r_3		
5			r_1		
8,9	r_2	r_2	r_2		

对输入串ab#用LR(1)分析的过程

步骤	状态栈	符号栈	输入串	ACTION	GOTO
• 1	0	#	ab#	S_3	
• 2	03	#a	b#	S_4	
• 3	034	#ab	#	出错	

对输入串ab#用LALR(1)分析的过程

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	ab#	$S_{3,6}$	
2	0(3,6)	#a	b#	$S_{4,7}$	
3	0(3,6)(4,7)	#ab	#	r_3	(8,9)
4	0(3,6)(8,9)	#aB	#	r_2	2
5	02	#B	#	出错	

构造 LALR(1)分析表*

- 方法二: 用核构造分析表
  - 优点：节省存储空间
  - 核：圆点不在产生式右部最左边的项目称为核，唯一的例外是S’ → • S。因此用GOTO（I, X）转换函数得到的J为转向后状态所含项目集的核
  - 构造LALR(1)的项目集的核
    - 构造思想：为LR（0）项目集核中的每个项目都配一个搜索符，使之成为LALR（1）的项目集核

构造 LALR(1)分析表*

- 搜索符的产生途径: 假定[B →β.Cγ,b]属于LR(1)项目集I的核

  K，$C \overset{*}{\Rightarrow} A\ \delta$, 则GO(I,X)核中[A →X.ρ ， a]里的搜索符a可能的途径为
  - 自生：a∈FIRST(δ γ),a<>b
  - 传播： $\delta\ \gamma\ \overset{\Rightarrow}{\varepsilon}$, 即a＝b
- 考察搜索符产生途径的算法：I为一个LR(0)集，K是I的核，X为一个文法符号

```
Procedure SPONSOR (I,X);
 FOR I的核中的每个项目 B →γ.δ DO
    BEGIN
        J:= CLOSURE({[B →γ.δ,® ]});
        IF [A →α.Xρ ， a]∈J and a<>®
            THEN GO(I,X) 核中的A →αX . P ， a 的搜索符a是自生的;
      IF [A →α.Xρ ， ®]∈J
            THEN GO(I,X) 核中的A →αX.ρ ， a 的搜索符®是从K中的B →γ.δ传播
        过来的;
    END
```

构造 LALR(1)分析表*

- 构造LALR(1)项目族步骤
  - 1. 对拓广文法G，构造所有的LR(0)集的核
  - 2. 用SPONSOR算法，对每个LR(0)集I的核K和每个文法符号X，确定出GO(I,X)核中每个项目所有自生的搜索符，以及哪些项目将接收K传播的搜索符
  - 3. 对每个项目集的每个核项目，初始化一个表，列出每个项目的自生搜索符（由2得到）
  - 4. 反复传播所有集合的核项目。对项目i，利用2的结果，找到i传播它的搜索所那达到的那些核项目。把i的当前搜索符集合附加到这些核项目中。继续该过程，直到没有新的搜索符传播为止。

构造 LALR(1)分析表*

- 例： G[S]:

  (0) S`→S
  (1) S→L=R
  (2) S→R
  (3) L→ *R
  (4) L→I
  (5) R→L

Step1：
I0: S' –> •S
I1: S' –> S•
I2: S –> L• = R
R –> L•
I3: S –> R•
I4: L –> *•R
I5: L –> i•
I6: S –> L =•R
I7: L –> *R•
I8: R –> L•
I9: S –> L=R•

构造 LALR(1)分析表*

Step2：
计算closure({[S‘ –> •S， ＃]})，得
S‘ –> •S， ＃
S –> •L = R， ＃
S –> •R， ＃
R –> •L ， ＃
L –> •i ， ＃/=
L –> •*R ， ＃ /=

[L –> •*R ， =]使“＝”成为I4: L –> *•R
的自生搜索符；
[L –> •i ， =]使“＝”成为I5: L –> i•
的自生搜索符

from	to
I0: S' –> •S	I1: S' –> S• I2: S –> L• = R R –> L• I3: S –> R• I4: L –> *•R I5: L –> i•
I2: S –> L• = R	I6: S –> L =•R
I4: L –> *•R	I4: L –> *•R I5: L –> i• I7: L –> *R• I8: R –> L•
I6: S –> L =•R	I4: L –> *•R I5: L –> i• I8: R –> L• I9: S –> L=R•

构造 LALR(1)分析表*

- Step3&Step4

项目表	搜索符			
	初始	第一遍	第二遍	第三遍
I0: S' –> •S	#	#	#	#
I1: S' –> S•		#	#	#
I2: S –> L• = R		#	#	#
I2: R –> L•		#	#	#
I3: S –> R•		#	#	#
I4: L –> *•R	=	=/#	=/#	=/#
I5: L –> i•	=	=/#	=/#	=/#
I6: S –> L =•R			#	#
I7: L –> *R•		=	=/#	=/#
I8: R –> L•		=	=/#	=/#
I9: S –> L=R•				#

LR(0),SLR(1),LR(1),LR(k),LALR(1)

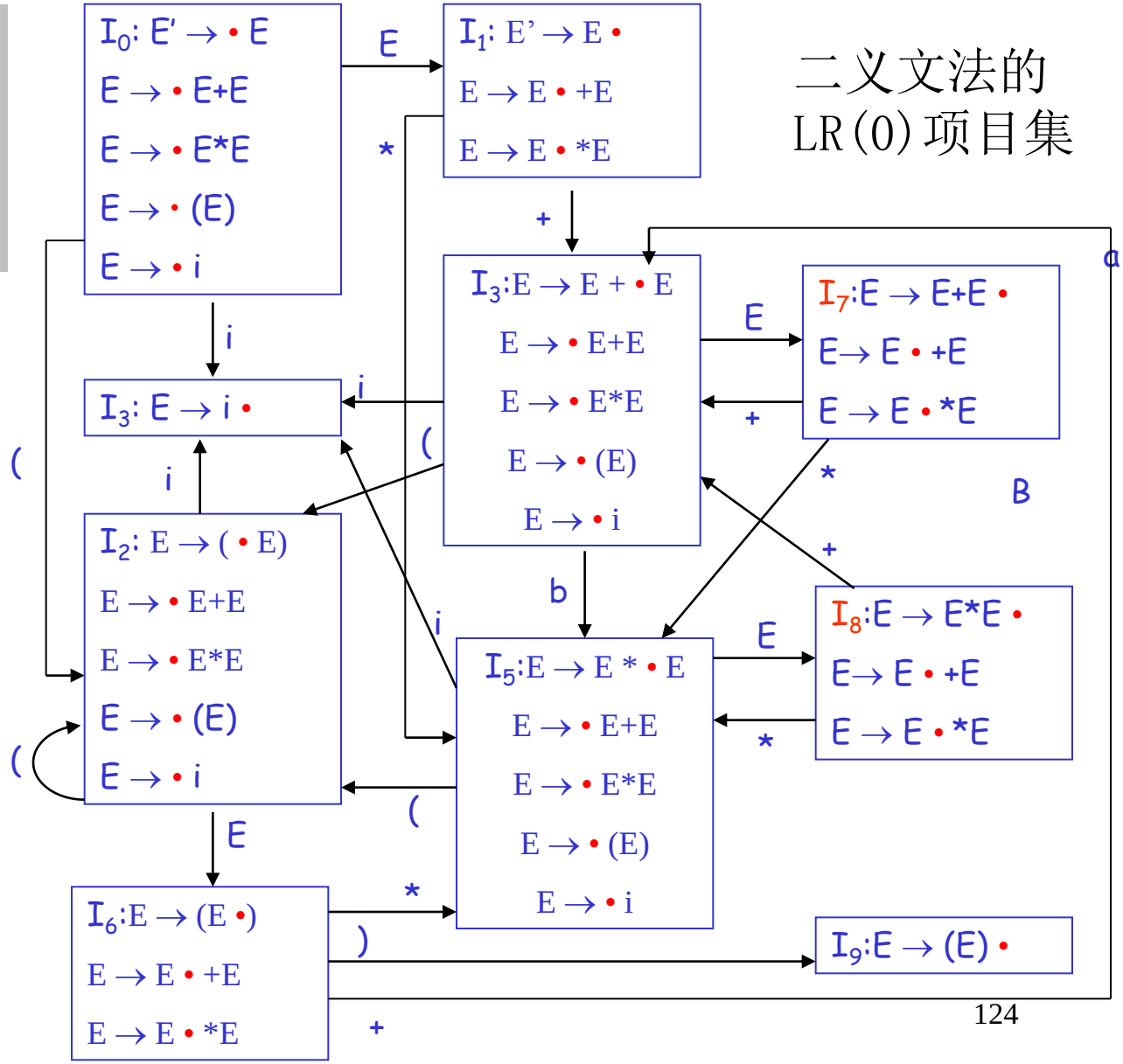
- LR(0)
- SLR(1): 生成的LR(0)项目集如有冲突，则根据非终结符的FOLLOW集决定
- LR(1): 项由 心与向前搜索符组成，搜索符长度为1
- LALR(1): 对LR(1)项目集规范族合并同心集

二义性文法在LR分析中的应用

- LR,LALR,SLR文法不是二义文法
- 对于某些二义文法，可以人为地给出优先性和结合性的规定，从而可以构造出比相应非二义性文法更优越的LR分析器

文法：
(1) E → E+E
(2) E → (E)
(3) E →E*E
(4) E → i

二义文法的LR(0)项目集



二义文法在LR分析中的应用

- 定义*优先于+; *、＋服从左结合，得到二义文法的LR分析表

状态	ACTION						GOTO
	i	+	*	(	)	#	E
0	S_3			S_2			1
1		S_4	S_5			Acc	
2	S_3			S_2			6
3		r_4	r_4		r_4	r_4	
4	S_3			S_2			7
5	S_3			S_2			8
6		S_4	S_5		S_9		
7		r_1	S_5		r_1	r_1	
8		r_2	r_2		r_2	r_2	
9		r_3	r_3		r_3	r_3	

LR分析中的出错处理

- 错误：
  - 不能移入
  - 不能归约
- 处理方法：
  - 插入，删除或修改
  - 出错处理子程序

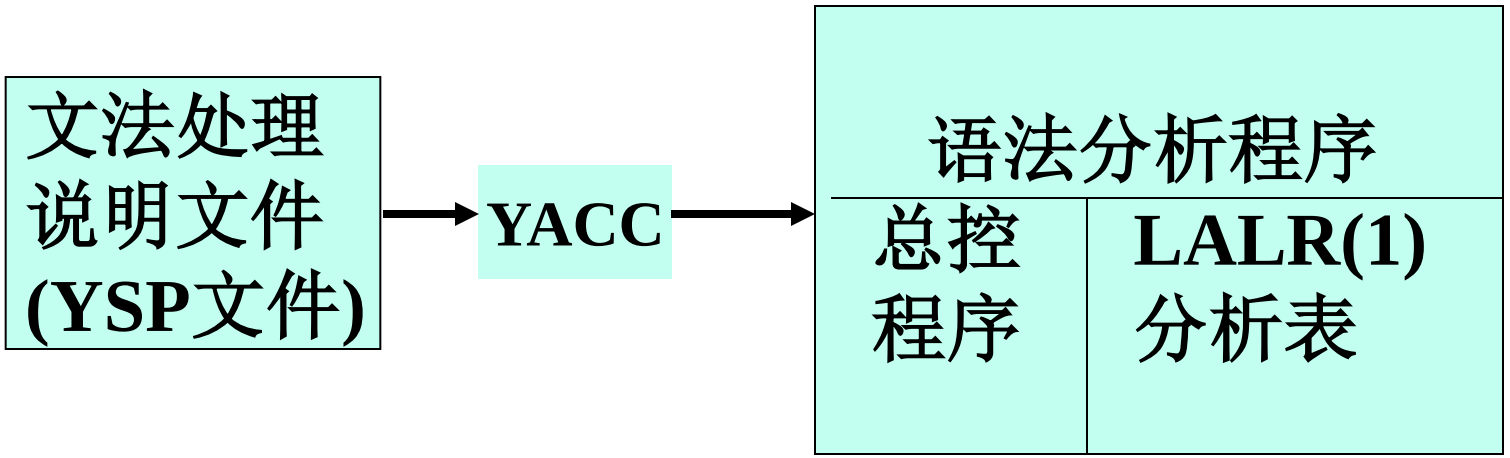
第4章:语法分析-自下而上分析

- 4.1 自底向上分析法
- 4.2 算符优先分析法
- 4.3 LR分析器
- 4.4 LR(0)分析表的构造
- 4.5 SLR分析表的构造
- 4.6 LR(1) 和LALR(1)分析表简介
- 4.7 软件工具Yacc

4.7 软件工具Yacc

软件工具Yacc是编译程序自动生成器,是在UNIX中运行的一个实用程序.该程序读用户提供的关于语法分析器的规格说明,基于LALR(1)语法分析的原理,自动构造一个语法分析器;并且能根据规格说明中给出的语义动作建立规定的翻译.

用YACC生成语法分析程序



Yacc要求用户输入处理功能说明:

1)描述输入结构的一组说明;

2)识别出这些结构后要调用的动作;

3)小型辅助说明程序.

Yacc的输出为分析程序.

Yacc的输入中,最重要的是规则.

Yacc语言程序的组成

Yacc语言程序,与LEX规格说明类似,由说明部分,翻译规则和辅助过程三部分组成.

各部分之间用双百分号分隔.即:

说明部分---可有可无

%%

翻译规则 (必须有)

%%

辅助过程---可有可无

说明部分:包括变量说明,标识符常量说明和正规定义.

翻译规则:是具有如下形式的语句序列:

P1　　{动作1}

P2　　{动作2}

……

Pn　　{动作n}

辅助过程:对翻译规则的补充,翻译规则里某些动作需要调用过程,如C语言的库程序.

例如:台式计算器读一个算术表达式进行求值,然后打印其结果.

表达式的文法:

E ⟶ E+T|T

T ⟶ T*F|F

F ⟶ (E)|digit

其中digit表示0…9的数字,按这一文法写出的Yacc规格说明如下:

```
(1)  %{
(2)     # include  <ctype.h>
(3)     %}
(4)     token DIGIT
(5)  %%
```

```
(6)   line     :  expr  '\n'   { printf("%d\n",$1); }
(7)              ;
(8)   expr     :  expr   '+'  term    { $$ = $1 + $3; }
(9)              |  term
(10)             ;
(11)   term    :   term   '*'  factor    { $$ = $1 * $3; }
(12)             |   factor
(13)             ;
(14) factor  :  '('  expr  ')'          { $$ = $2 ; }
(15)             |  DIGIT
(16)             ;
(17)  %%
```

```
(18) yylex( ) {
(19)          int  c ;
(20)          c = getchar ( );
(21)          if  (  isdigit ( c ))  {
(22)              yylval=c-’0’;
(23)              return DIGIT;
(24)          }
(25)          return  c;
(26)     }
```

程序说明:

a) 第(1)-(4)行是说明部分,其中包括可供选择的两部分.用%{和%}括起来的部分是C语言程序的正规说明,可以说明翻译规则和辅助过程里使用的变量和函数的类型.

这里用第(2)行的蕴含控制行代替全部说明,具体内容在 ctype.h 里,第(4)行指出DIGIT是token类型的词汇,供后面使用.

b)　　第(6)-(16)行是翻译规则,每条规则由文法的产生式和相关的语义动作组成.
形如

左部⟶右部1|右部2|…|右部n

的产生式,在YACC规格说明里写成

```
左部 :    右部1    {语义动作1}
      |    右部2    {语义动作2}
      ……
      |    右部n    {语义动作n}
      ;
```

在YACC里,用单引号括起来的单个字符看成是终结符号. 语义动作是C语言的语句序列.语义动作中,$$表示和左部非终结符相关的属性值,$1表示和产生式第一个文法符号相关的属性值,例如: 语义动作

```
line    : expr '\n'  { printf("%d\n",$1); }
```

C) (18)-(26)行是辅助过程,每个辅助过程都是C语言的函数,对翻译规则的补充,并且,其中必须包含 名为yylex的词法分析器.调用名

为yylex( )的函数得到一个词汇. 该词汇包括两部分的属性,通过Yacc定义的全程变量yylval传递给语法分析器,返回词汇的属性.

(22) yylval=c-'0'; yylex( )返回词汇的值.(23) yylex( )返回词汇DIGIT;第(25)行,除了数之外的任何字符, yylex( )返回该字符本身.

例如:扩充前面的台式计算器的规格说明:

1)允许输入几个表达式,每个表达式占一行.

2)表达式中的数由不止一个的数字组成,可含小数点和负号.

3)增加减法和除法运算.

表达式的文法:

E⟶E+E|E-E|E*E|E/E|(E)|-E|number

按这一文法写出的Yacc规格说明如下:

```
(1)   %{
(2)      # include  <ctype.h>
(3)      # include  <stdio.h>
(4)      # define    YYSTYPE    double
                 /*double type for Yacc stack */
(5)      %}
(6)      token NUMBER
(7)     % left  '+'   '-'
(8)     % left  '*'   '/'
(9)    % right  UMINUS
(10)   %%
```

```
(11) lines  :  lines   expr  '\n'   { printf("%d\n",$2); }
(12)        |  lines   '\n'
(13)        |  /*    ε   */
(14)        ;
(15) expr   :  expr   '+'  expr    { $$ = $1 + $3; }
(16)        |   expr   '-'  expr    { $$ = $1  - $3; }
(17)        |   expr   '*'  expr    { $$ = $1 * $3; }
(18)        |   expr   '/'  expr    { $$ = $1  / $3; }
(19)        |  '('  expr  ')'         { $$ = $2 ; }
(20)        |  '-' expr %prec UMINUS   {$$ = - $ 2 ; }
(21)        |  NUMBER
(22)        ;
(23)  %%
```

```
(24) yylex( ) {
(25)      int  c ;
(26)      while ( (c = getchar ( )) == ‘ ’);
(27)      if  ( ( c==‘.’) || isdigit ( c ))  {
(28)           ungetc(c,stdin);
(29)           scanf(“%lf”,&yylval);
(30)           return NUMBER;
(31)          }
(32)          return  c;
(33)      }
```

Yacc处理冲突的规则:

1)产生“归约-归约”.冲突时,按照规格说明中产生式的排列顺序,选择排列在前面的产生式进行归约.

2)当产生“移进-归约”冲突时,选择执行移进动作.

3) Yacc在规格说明部分里,可以规定终结符号的优先顺序和结合性.

优先顺序:按说明终结符的次序,后说明的具有最高的优先顺序.“%prec”说明其后的终结符具有最高的优先顺序.

结合性:“%right”(右结合),

“%left” (左结合),

“%nonassoc”(不具有结合性)

4) 应用这些机制,对二义文法,用户可以提供附加信息Yacc可以解决冲突.