重庆大学
CHONGQING UNIVERSITY

第四讲 语法分析（2）

重庆大学 计算机学院 张敏

思考

是否所有语言都有LL(1)文法?
能否将非LL(1) 文法变换为等价的LL(1)文法?
LL（1）文法的局限性?

4.6 自底向上语法分析概述

分析树的构造：自底向上

分析过程：

从输入符号串开始分析
查找当前句型的“可归约串”
使用规则，把它归约成相应的非终结符号
重复

关键：找出“可归约串”

常用方法：

优先分析方法
LR分析方法

4.6.1 自下而上语法分析的策略

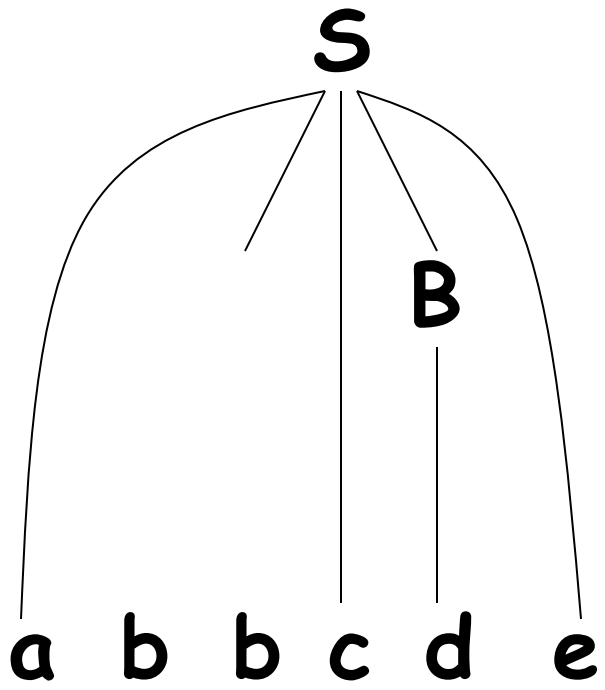
Let I = input string
repeat
pick a non-empty substring β of I
where X→ β is a production
if no such β, backtrack
replace one β by X in I
until I = “S” (the start symbol) or all possibilities are exhausted

移进-规约分析；

- **移进**就是将一个终结符推进栈
- **归约**就是将0个或多个符号从栈中弹出，根据产生式将一个非终结符压入栈

示例：

文法G[S]:
(1) S → aAcBe
(2) A → b
(3) A → Ab
(4) B → d



步骤	分析栈	输入符号串	动作
1）	#	abbcde#	移进
2）	#a	bbcde#	移进
3）	#ab	bcde#	归约(A→b)
4）	#aA	bcde#	移进
5）	#aAb	cde#	归约(A→Ab)
6）	#aA	cde#	移进
7）	#aAc	de#	移进
8）	# aAcd	e#	归约(B→d)
9）	#aAcB	e#	移进
10）	#aAcBe	#	归约(S→aAcBe)
11）	#S	#	接受

对输入串abbcde#的移进-规约分析过程

分析符号串abbcde是否G[S]的句子

$$S \Rightarrow aAcBe \Rightarrow aAcde \Rightarrow aAbcde \Rightarrow abbcde$$

4.6.1 自下而上语法分析的策略

特殊情况

实际应用中可能出现“冲突”

- 移进与归约都合法，产生移进-归约冲突
- 归约时可以适用两个不同的产生式，产生归约-归约冲突

结论:在自底向上的分析中,**可规约串**的识别是至关重要的.

如何决定什么时候移进，什么时候规约?

一般的移进-归约策略：若可规约串（句柄）在栈顶出现，则归约，否则移进

4.6.2 句柄的定义和识别方法

短语、直接短语、句柄的定义：

对于文法G[S]，

$S \overset{*}{\Rightarrow} xAy$且$A \overset{*}{\Rightarrow} b$则称b是句型xby相对于非终结符A的短语。

若有$A \Rightarrow b$则称b是句型xby相对于非终结符A的直接短语。

一个句型的最左直接短语称为该句型的句柄。

语法树与句柄的关系

最左简单子树的端末结点自左至右排列构成一句柄。

4.6.2 句柄的定义和识别方法

识别句柄必须解决的2个问题:

1)如何保证所找到的直接短语是最左的?

最左性

2)如何确定句柄在句型中的开始位置和结束位置,从而可以抽取它?

不同类型的文法;
判别句柄首尾的方法:
优先法(根据算符的优先级),
状态法(根据句柄形成的顺序来选择)

4.6.3 算符优先分析

某些文法具有“算符”特性，例如表达式运算符（优先级、结合性），可人为地规定其算符的优先顺序，即给出优先级别和同一级别的结合性。

算符文法定义:

如果不含ε产生式的上下文无关文法 G 中没有形如 U→...VW...的产生式，其中V,W∈V_N则称G 为算符文法（OG）。

性质1：在算符文法中任何句型都不包含两个相邻的非终结符.(数学归纳法)

性质2：如 Vx 或 xV 出现在算符文法的句型 α 中，其中V∈V_N,x∈V_T，则 α 中任何 含 x 的短语必含有V.

4.6.3 算符优先分析

在算符文法中定义算符优先关系

x = y

G中有形如.U→…xy…或U → …xVy..的产生式。

x < y

G中有形如.U → …xW…的产生式,而W $\overset{+}{\Rightarrow}$y….或W $\overset{+}{\Rightarrow}$ Vy…

x>y

G中有形如.U → …Wy…的产生式,而 W $\overset{+}{\Rightarrow}$…x或W $\overset{+}{\Rightarrow}$… xV

规定 若 S $\overset{+}{\Rightarrow}$x…或 S $\overset{+}{\Rightarrow}$Vx… 则 # < x

S $\overset{+}{\Rightarrow}$ …x 或 S $\overset{+}{\Rightarrow}$…xV 则 x > #

4.6.3 算符优先文法

在算符文法中定义算符优先关系

x = y

G中有形如.U→…xy…或U → …xVy..的产生式。

x < y

G中有形如.U → …xW…的产生式,而$W \overset{+}{\Rightarrow} y$….或$W \overset{+}{\Rightarrow} Vy$…

x>y

G中有形如.U → …Wy…的产生式,而 $W \overset{+}{\Rightarrow}$..x或$W \overset{+}{\Rightarrow}$… xV

规定 若 $S \overset{+}{\Rightarrow} x$…或 $S \overset{+}{\Rightarrow} Vx$… 则 # < x

$S \overset{+}{\Rightarrow}$ …x 或 $S \overset{+}{\Rightarrow}$ …xV 则 x > #

在算符文法G中，若任意两个终结符间**至多有一种**算符优先关系存在，则称G为**算符优先文法(OPG)**。算符优先文法是无二义的。

4.6.3 算符优先分析

算符优先分析法

只考虑终结符号之间的优先关系

分析速度快，不是规范归约，且只适用于算符优先文法

“可归约串”是句型的“最左素短语”。

素短语：句型的一个短语，至少含有一个终结符号，并且除它自身之外不再含有其他更小的素短语。

最左素短语：处于句型最左边的那个素短语。

4.6.3 算符优先分析

如何计算算符优先关系?

定义

FIRSTVT(B)={b|$B \overset{+}{\Rightarrow}$ b...或B $\overset{+}{\Rightarrow}$Cb...}

对于非终结符B，其往下推导所可能出现的最左终结符的集合

LASTVT(B)={a|$B \overset{+}{\Rightarrow}$...a或B $\overset{+}{\Rightarrow}$...aC}

对于非终结符B，其往下推导所可能出现的最右终结符的集合

4.6.3 算符优先分析

FIRSTVT(A),LASTVT(A)的计算方法：

（1）若有产生式A→a...，或A→Ba...,则将a加入FIRSTVT(A)，
（2）对于产生式A→B...，若b∈FIRSTVT(B)，则b∈FIRSTVT(A)；
重复（2）直至FIRSTVT(A)不再增大为止。

LASTVT(A)的计算方法类似,请自行补充完整的计算算法

4.6.3 算符优先分析

优先关系再定义：

1) ‘= ‘（同等优先）关系
直接看产生式的右部，若出现了A →...ab...或A →...aBb,则a=b

2)’ < ‘（后于）关系
求出每个非终结符B的FIRSTVT(B)，若A→...aB...,∀b∈FIRSTVT(B),a<b

3)’ >’ （先于）关系
求出每个非终结符B的LASTVT(B)，若A→...Bb...,则∀a∈LASTVT(B),a>b

4.6.3 算符优先分析

优先矩阵的计算算法：

- 手工填写
- 编程实现

例： 文法G（S）：S→（A）|a，　A→A+S|S

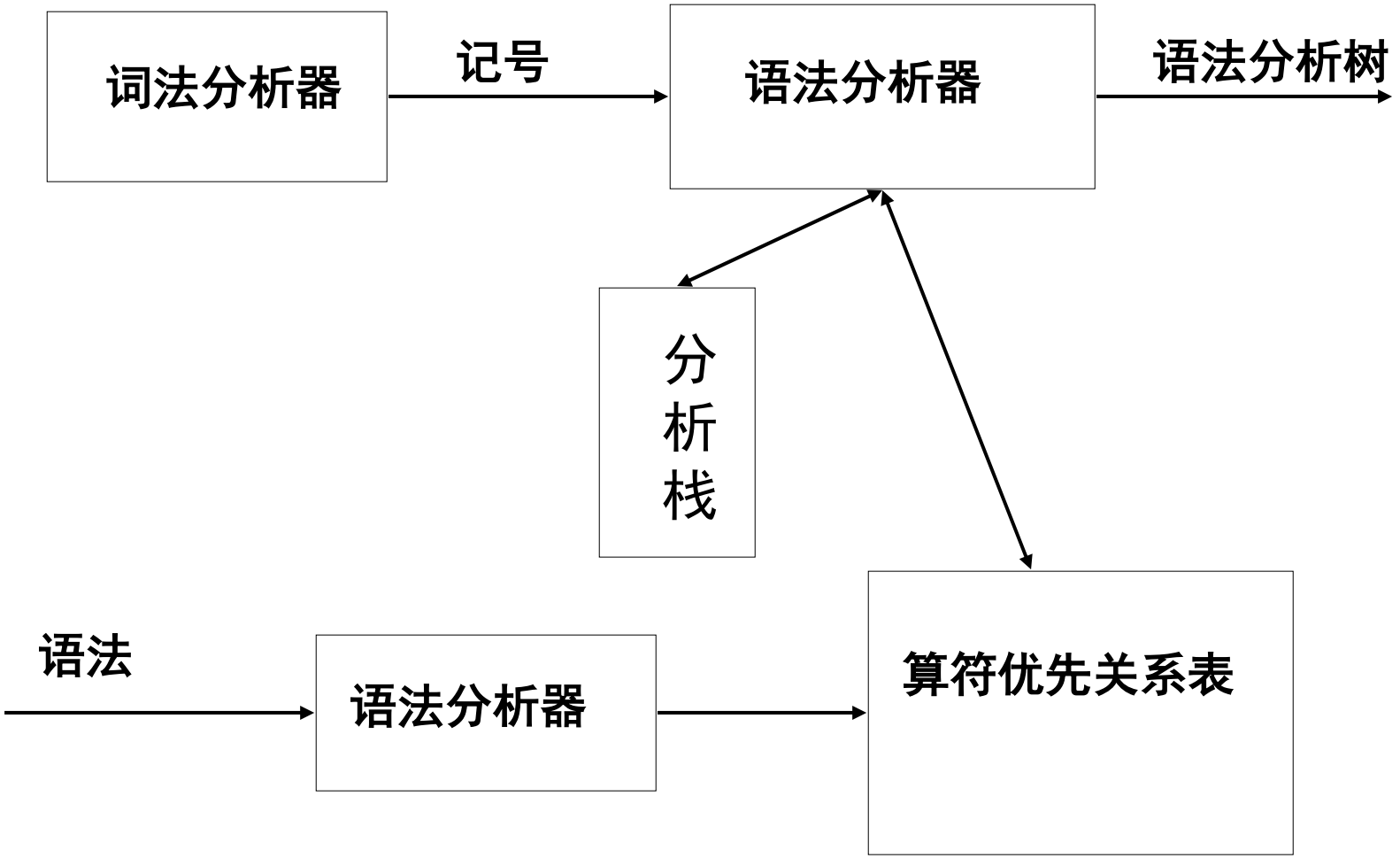
解：FIRSTVT（S）={a,(}
FIRSTVT（A）={+,a,(}
LASTVT(S)={a,)}
LASTVT(A)={+,a,)}

	a	+	(	)
a		>		>
+	<	>	<	>
(	<	<	<	=
)		>		>

A→A+S　　+<FIRSTVT(S)={a,(}

LASTVT(A)={a,+,)}>+

4.6.4 算符优先语法分析模型



4.6.4 算符优先语法分析模型

算符优先分析过程

栈顶形成可归约串的判断方法：

1 利用栈顶终结符和当前输入符号之间的优先关系>，找到可归约串的右端；
2 在栈内，利用<关系，找到可归约串的左端；
3 将<和>之间的符号串弹出栈，并将归约后的非终结符压入栈，完成一次归约

句子id+id*id$，分析过程如下

栈	关系	输入	动作	序号
$	<	id+id*id$	移进	0
$ id	>	+id*id$	归约	1
$ F	<	+id*id$	移进	2
$ F+	<	id*id$	移进	3
$ F+id	>	*id$	归约	4
$ F+F	<	*id$	移进	5
$ F+F *	<	id$	移进	6
$ F+F * id	>	$	归约	7
$ F+F * F	>	$	归约	8
$ F+T	>	$	归约	9
$ E		$	接受	10

自底向上分析基本分析过程/算符优先分析过程

0: S -> E
1: E -> E + T
2: | T
3: T -> T * F
4: | F
5: F -> n

1 + 2
F + 2
T + 2
E + 2
E + F
E + T
E
S

步骤	输入串	优先关系	动作	分析栈	所用产生式
1	i1+i2#2			#1	
2	+i2#2	#1<i1	移进i1	i1#1	
3	+i2#2	#1<i1>#2	归约i1	F#1	F->i
4	i2#2	#1<+	移进+	+F#1	
5	#2	+<i2	移进i2	i2+F#1	
6	#2	+<i2>#2	归约i2	F+F#1	F->i
7	#2	#2<+>#1	归约F+F	E#1	E->E+T

T->F

F+F像E+T

思考:为什么会出现这些问题?

4.6.5 语法架子树

算符优先只定义了终结符之间的优先关系,而没有定义非终结符之间的优先关系。

用这些优先关系不能识别单非终结符构成的句柄,例如:**T->F**

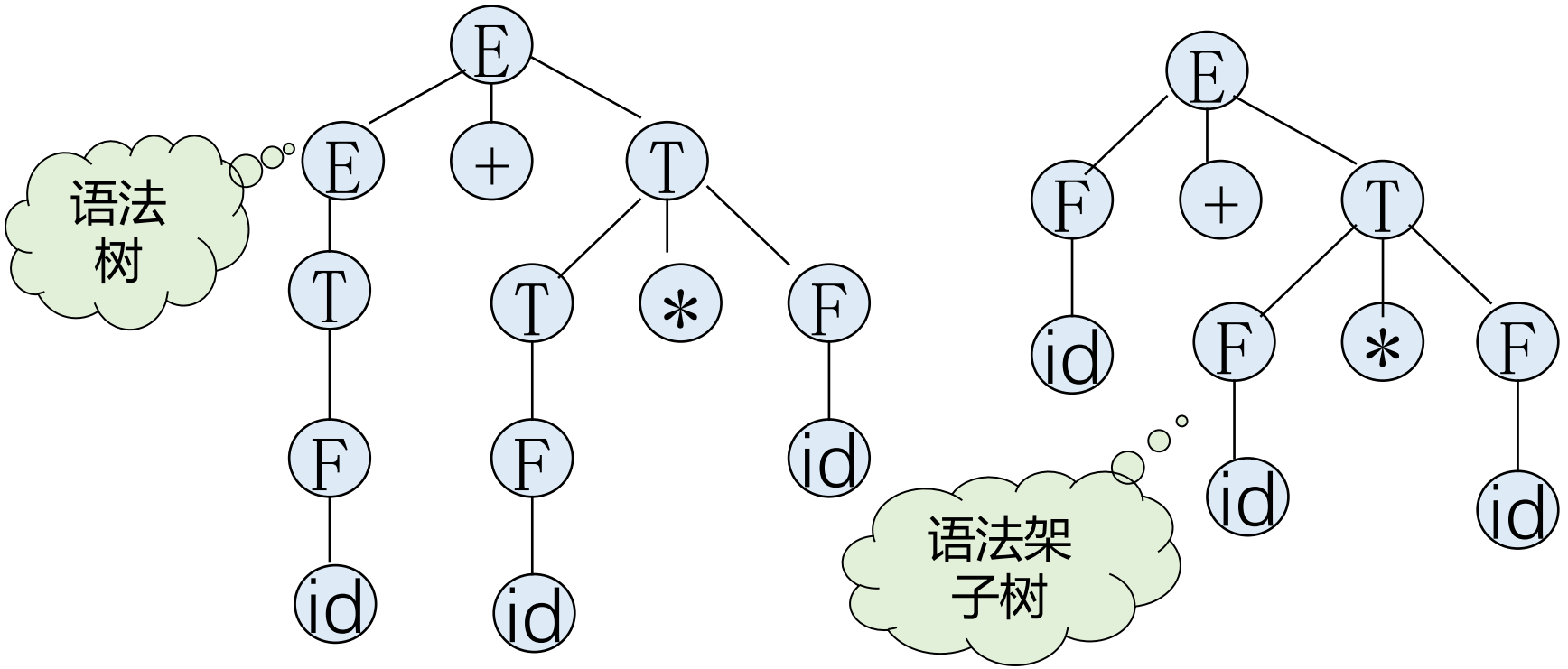
将算符优先分析过程中识别的”句柄”称为**最左素短语**

- 定义文法G的句型的素短语是一个短语，它至少包含一个终结符，且除自身外不再包含其他素短语。处于句型最左边的素短语为最左素短语
- 最左素短语是句型中由关系<和>所括起来的最左子串

4.6.5 语法架子树

语法树与语法架子树

G[E]: E→E+T | T
T →T*F | F
F →(E) | id



4.6.5 语法架子树

语法架子树的特点：

- 终结符的位置和语法树中对应，但非终结符位置不一定对应；
- 架子树的每个内节点都含有一个终结符的子节点。

思考:OPG每个句型的语法架子树是唯一的吗?

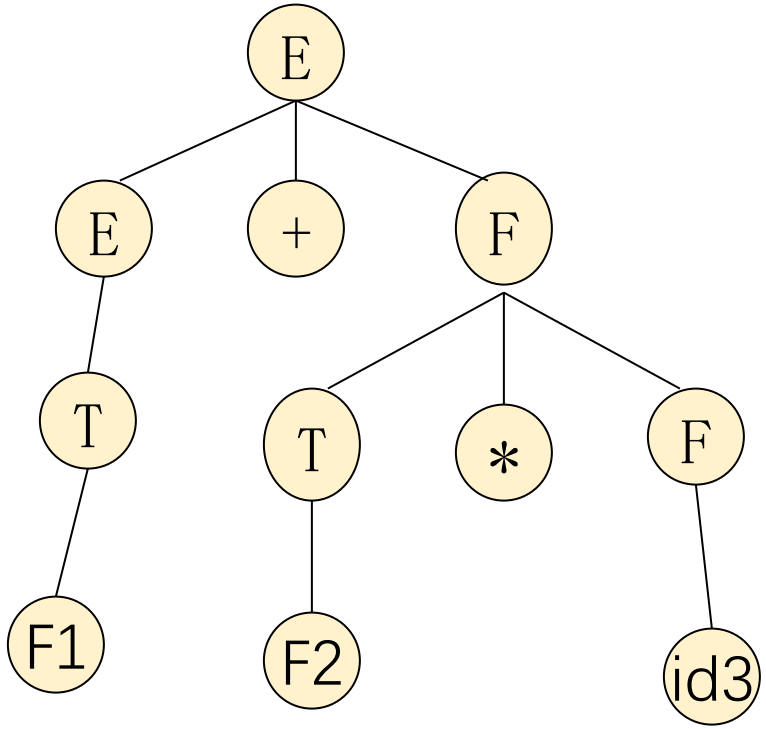
句型$F+F*id_3$

短语：F_1，F_2，id_3，F_2*id_3，$F_1+F_2*id_3$

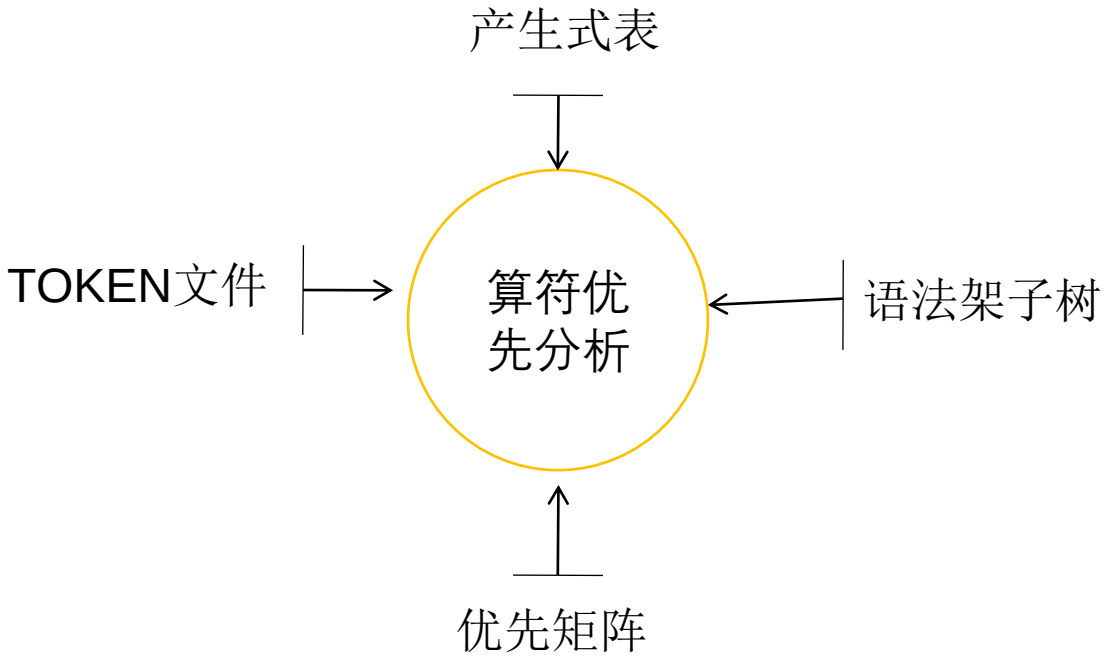
直接短语：F_1，F_2，id_3

句柄：F_1

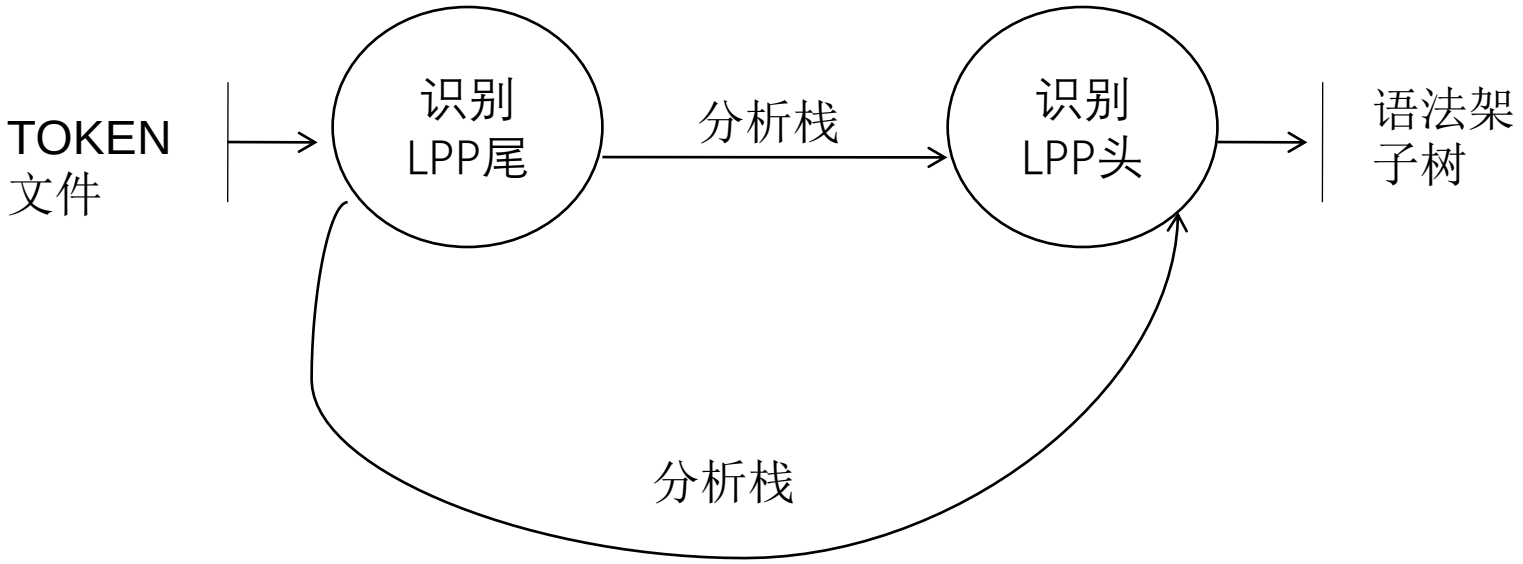
最左素短语：id_3



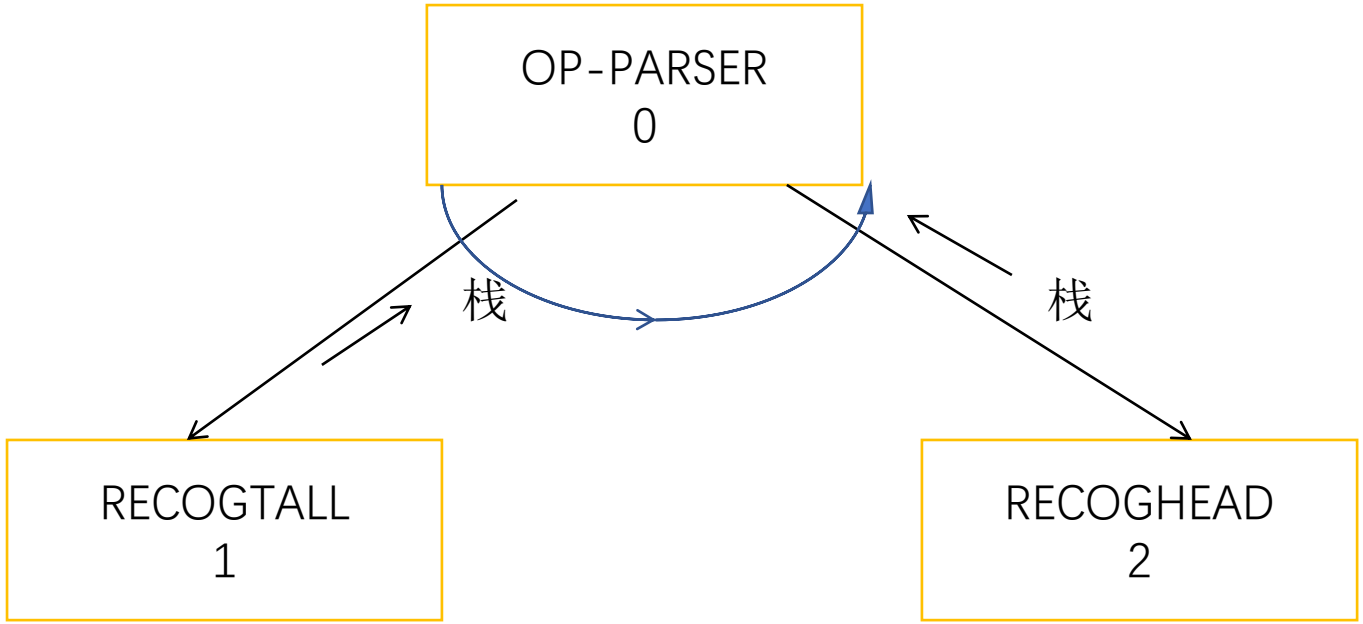
4.6.6 算符优先分析算法的实现



4.6.6 算符优先分析算法的实现



4.6.6 算符优先分析算法的实现



4.6.6 算符优先分析算法的实现

模块0：算法优先分析主程序

```
Program OP_PARSER;
begin
    TOP:=1;S[TOP]:= ‘#’ ;TOKEN:= ‘ ‘;
repeat
    call RECOGTAIL(TAIL);
    call RECOGHEAD(TAIL);
 until TOKEN= ‘#’ ;
 if TOP=2 且S[TOP]=文法开始符
    then 分析成功并输出带语义动作的语法架子树
    else PRINTERR（ ‘分析有错’ ）；
 end.
```

4.6.6 算符优先分析算法的实现

模块1：移进并识别LPP尾（RECOGTAIL）

输入：

输出：TAIL指向栈顶第一个终结符；

```
Procedure RECOGTAIL（TAIL）
begin
  repeat
      把下一个输入符读入TOKEN;
      if S[TOP]为V_T  then TAIL:=TOP
                          else TAIL:=TOP-1;
     if S[TAIL]<=TOKEN  then PUSH TOKEN;
 until S[TAIL]>TOKEN ;
 输入指针回退一个输入符;
 return
end;
```

模块2：识别LPP头并归约（RECOGHEAD）

输入：TAIL

输出：

```
Procedure RECOGHEAD（TAIL）；
  begin
    LOWER:=TAIL;
  repeat
       UPPER:=LOWER;    /*UPPER表示位于上方的终结符
       if S[LOWER-1] 为 V_T then LOWER:=LOWER-1
                                 else LOWER:=LOWER-2;
  until S[LOWER]< S[UPPER] ;
  LPP:= ‘S[LOWER+1] S[LOWER+2] … S[TOP]’ ;
  call MATCH(LPP,N,A)； /*N是归约符号，A是语义动作符号
  if N= ‘ ‘ then PRINTERR( ‘LPP无法归约’ )
           else
               begin
                 在架子树中构造节点N并附上语义动作A;
                  POP LPP;  PUSH N;
               end;
       return
   end;
```

思考

上述算符优先分析利用了一个分析栈实现，非终结符和终结符都放在里面，这种实现方式有什么不足之处?

如果改进，使用两个栈如何实现?

4.6.7 算符优先分析法的改进

- 优先函数
- 优先函数比优先矩阵节省空间（线性化）
- 优先函数的构造

4.6.7 算符优先分析法的改进

优先函数的构造

定义(算符优先函数)：

若定义在算符文法G的终结符号集合上的离散函数f和g

f, g：VT → N(整数)，满足:

(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)。

则称函数 f 和 g 为算符优先函数，f和g分别称为入栈优先函数和比较优先函数

4.6.7 算符优先分析法的改进

有向图构造优先函数算法

输入：一张算符优先关系表。
输出：该表的算符优先函数。
方法：

1、∀ a ∈VT∪{#} ，建立两个符号fa和ga。

2、将所有的 fa和ga 组成的集合分组，若a=b，则 fa和gb 之间有一条双向弧。

3、以第二步中建立的组为结点，对VT∪{#}中的任意符号a和b，若a<b，则gb从所在的组画一箭弧到 fa 所在的组；若a>b，则从 fa 所在的组画一箭弧到gb 所在的组。

4、每个节点从自身出发所能到达的节点(包括自身节点)的个数,作为该节点fa或ga的优先数。若构造出来的优先数g和g与优先矩阵无矛盾则f与g为所求,否则，不存在优先函数。

4.6.7 算符优先分析法的改进

若已知优先关系矩阵如下，求优先函数

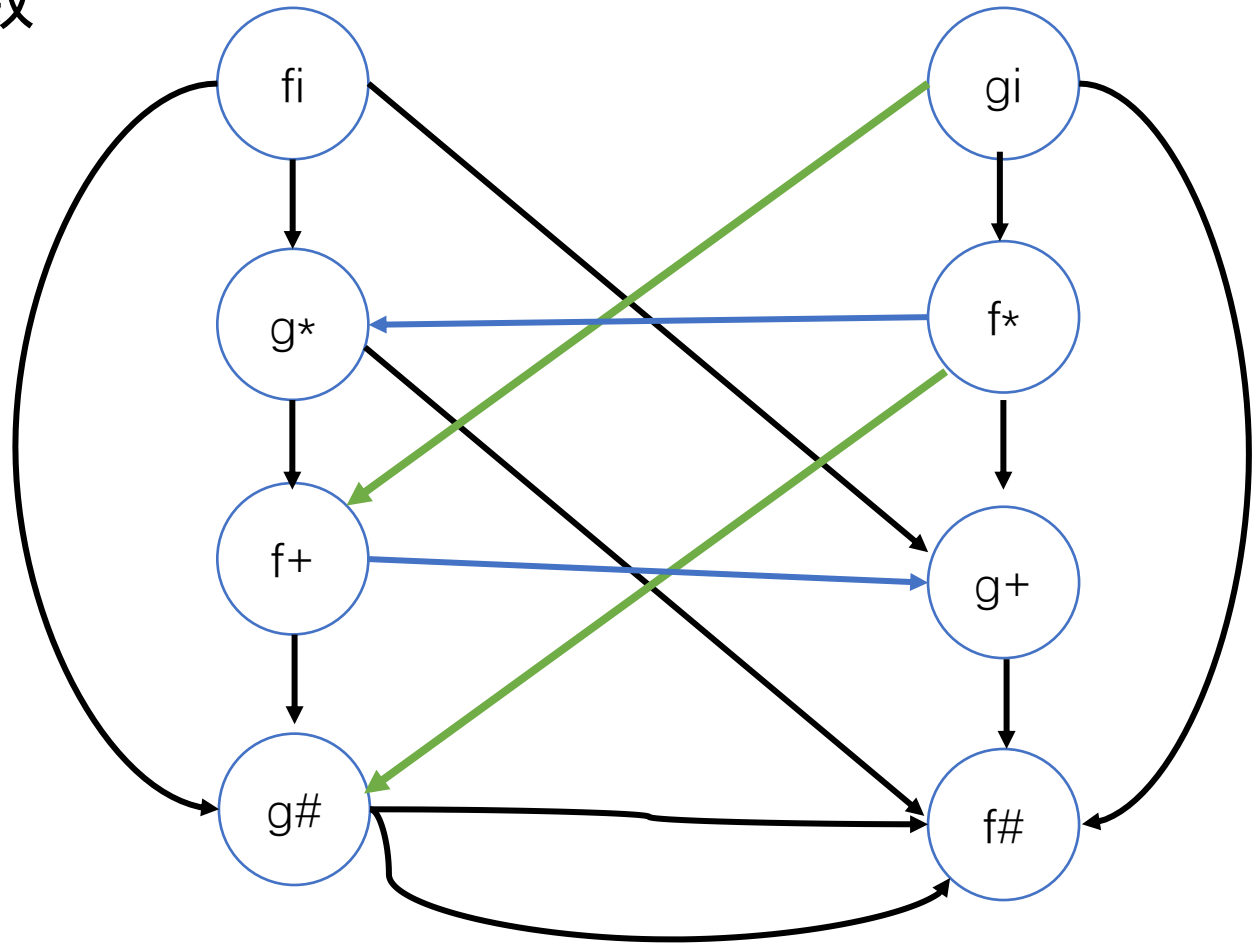
	i	*	+	#
i		>	>	>
*	<	>	>	>
+	<	<	>	>
#	<	<	<	=

4.6.7 算符优先分析法的改进

若已知优先关系矩阵如下，求优先函数

	i	*	+	#
i		>	>	>
*	<	>	>	>
+	<	<	>	>
#	<	<	<	=

	i	*	+	#
f	6	6	4	2
g	7	5	3	2



4.6.7 算符优先分析法的改进

优先函数可能带来什么问题?

不能区分两个终结符间没有优先关系的情况

出错时不能准确定位

算符优先分析法的错误诊断恢复策略

请分析在算法优先分析法中会出现错误的情况，并给出相应的错误诊断恢复策略。

使用算符优先法进行语法分析时，出现下面两种情况，则发生错误：

(1) 若栈顶算符与当前输入符号间不存在优先关系；

(2) 若找到某一句柄，但不存在任一产生式，其右部与此句柄形式吻合。

算符优先分析法的错误诊断恢复策略

(1)若栈顶算符与当前输入符号间不存在优先关系；

在第一种出错情况下，即栈顶算符与当前输入符号间不存在优先关系时，可采用改变、插入或删除符号的方法来纠正错误。

eg1：

表中的终结符号对(，#无优先关系，当栈顶算符为（，当前输入符号为#，说明表达式中有（，但无）便结束了，此时调用此程序将(从栈顶删除，继续分析。给出诊断信息；“缺少)”

eg2：

当栈顶算符为)或 i，当前输入符号为i或(时出错，)或 i不能直接跟 i或(，它们之间应有运算符，此时调用此程序在输入串当前符号前插入+，继续分析。给出诊断信息：“缺少运算符”

算符优先分析法的错误诊断恢复策略

若找到某一句柄，但不存在任一产生式，其右部与此句柄形式吻合。

e6:
句柄中含*时，若该句柄正确，则它应与产生式 T →T*F右部形式吻合，即两边各出现一个非终结符，否则出错，调用此程序将句柄归约为非终结符 N，继续分析。
给出诊断信息："缺表达式。"

e7:
句柄中含i时，若该句柄正确，则它应与产生式 F→ i右部形式吻合，即i左边或右边不能出现非终结符，否则出错，调用此程序将句柄归约为非终结符 N，继续分析。
给出诊断信息："表达式间无运算符连接。"

算符优先分析法的错误诊断恢复策略

请完善算符优先分析法的错误诊断恢复策略，并对下列输入串进行分析：

1+）1#

4.6.7 算符优先分析法的局限

- 一般语言的文法很难满足算符优先文法的条件
- 难以完全避免把错误的句子正确的规约
- 仅适用于表达式的语法分析

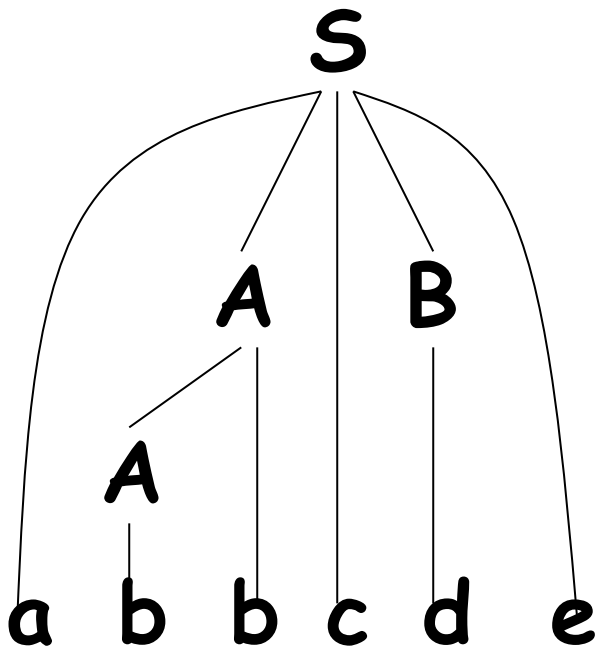
4.6.7 算符优先分析法的应用

扩展阅读：

汉语自动句法分析的算符优先文法模型
利用算符优先移进-归约算法实现自定义财务指标的分析与计算

思考

文法G[S]:
(1) S → aAcBe
(2) A → b
(3) A → Ab
(4) B → d



步骤	符号栈	输入符号串	动作
1）	#	abbcde#	移进
2）	#a	bbcde#	移进
3）	#ab	bcde#	归约(A→b)
4）	#aA	bcde#	移进
5）	#aAb	cde#	归约(A→Ab)
6）	#aA	cde#	移进
7）	#aAc	de#	移进
8）	# aAcd	e#	归约(B→d)
9）	#aAcB	e#	移进
10）	#aAcBe	#	归约(S→aAcBe)
11）	#S	#	接受

对输入串abbcde#的移进-规约分析过程

在步骤3中，用A→b归约
在步骤5中，用A→Ab归约
问题:
何时移进？何时归约?

分析符号串abbcde是否G[S]的句子

$$S \Rightarrow aAcBe \Rightarrow aAcde \Rightarrow aAbcde \Rightarrow abbcde$$

思考

文法G[S]:
(1) S → aAcBe
(2) A → b
(3) A → Ab
(4) B → d

在步骤3中，用A→b归约
在步骤5中，用A→Ab归约
问题:
何时移进？何时归约?

3）	#ab	bcde#	归约(A→b)
4）	#aA	bcde#	移进
5）	#aAb	cde#	归约(A→Ab)
6）	#aA	cde#	移进

分析:

已分析过的部分在栈中的前缀不同，而且移进和归约后栈中的状态会发生变化

4.7 LR分析法

规范归约的定义

定义：假定α是文法G的一个句子，我们称右句型序列 α_n，α_{n-1}，…，α_1，α_0 是α的一个规范归约，如果序列满足：

(1) $\alpha_n=\alpha$，$\alpha_0=S$

(2) 对任何i($0<i\leq n$)，α_{i-1}是经过把α_i的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α的一个最右推导的逆过程，因此规范归约也称为最左归约。
- abbcde的一个规范归约是如下的右句型序列：abbcde，aAbcde，aAcde，aAcBe，S。

4.7 LR分析法

句柄的最左性

- 规范句型：最右推导得到的句型
- 规范句型的特点：句柄之后没有非终结符号
- 利用句柄的最左性：与符号栈的栈顶相关
- 不同的最右推导，其逆过程也是不同

例：考虑文法 E→E+E|E*E|(E)|id的句子id+id*id

E⇒E+E⇒E+E*E⇒E+E*id⇒E+id*id⇒id+id*id

E⇒E*E⇒E*id⇒E+E*id⇒E+id*id⇒id+id*id

句柄：E+E
产生式：E→E+E

句柄：id
产生式：E→id

4.7 LR分析法

LR(k)的含义：

L 表示自左至右扫描输入符号串

R 表示为输入符号串构造一个最右推导的逆过程

k 表示为作出分析决定而向前看的输入符号的个数。

LR分析方法的基本思想

“历史信息”：记住已经移进和归约出的整个符号串；

“预测信息”：根据所用的产生式推测未来可能遇到的输入符号；

根据“历史信息”和“预测信息”，以及“现实”的输入符号，确定栈顶的符号串是否构成相对于某一产生式的句柄。

4.7 LR分析法

LR分析法的特点

LR分析技术是一种比较完备的技术：

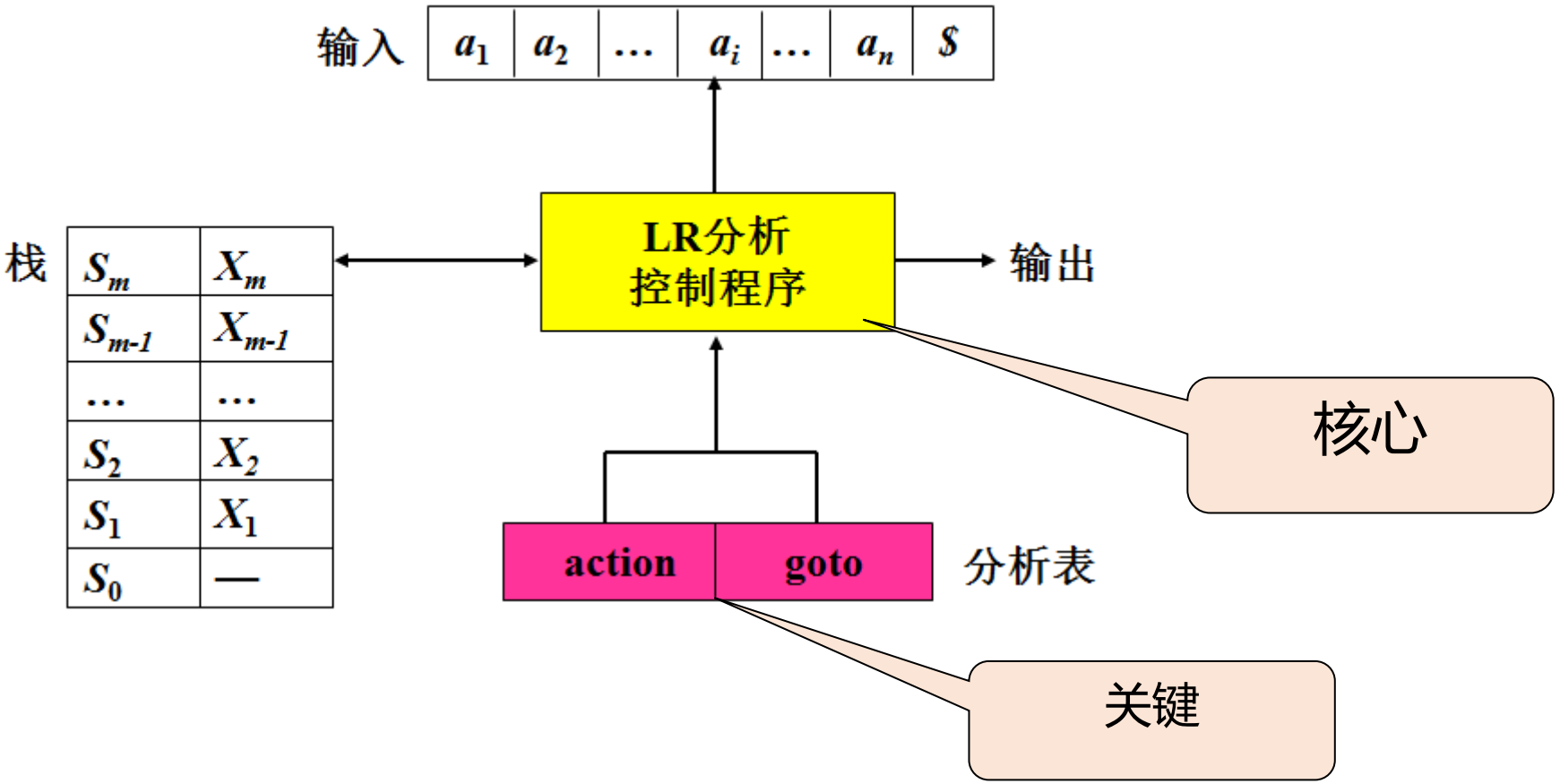
可以分析所有能用上下文无关文法书写的程序设计语言的结构；
最一般的无回溯的“移进-归约“方法；
能分析的文法类是预测分析方法能分析的文法类的真超集；
分析过程中，能及时发现错误，快到自左至右扫描输入的最大可能。

LR分析方法的不足之处：

手工编写LR分析程序的工作量太大
需要专门的工具，即LR分析程序生成器（如YACC）

4.7.1 LR分析程序的模型及基本原理

LR分析程序的模型



4.7.1 LR分析程序的模型及基本原理

分析表

LR分析控制程序工作的依据

goto[S_m，X]：状态S_m经X转移的后继状态

action[S_m，a_i]：状态S_m面临输入符号a_i时应采取的分析动作

移进：把当前输入符号a_i及由S_m和a_i所决定的下一个状态S=goto[S_m，a_i]推进栈，向前扫描指针前移。

归约：用某产生式A→β进行归约，若β的长度为r，归约的动作从栈顶起向下弹出r项，使S_{m-r}成为栈顶状态，然后把文法符号A及状态S=goto[S_{m-r}，A]推进栈。

接受：宣布分析成功，停止分析。

出错：调用出错处理程序，进行错误恢复。

4.7.1 LR分析程序的模型及基本原理

LR分析法的基本原理

LR分析法严格执行最左归约，每次都归约真正的句柄。
将识别句柄的过程划分为由若干状态控制，每个状态控制识别出句柄的一个符号。

句柄识别态：
识别一个长为n的句柄需要n+1个状态，识别句柄尾的状态称为句柄识别态。
活前缀：
一个规范句型的一个前缀，如果不含句柄之后的任何符号，则称它为该句型的一个活前缀。
分析过程中 ($S_0X_1S_1X_2...X_mS_m$，$a_ia_{i+1}...a_n\$$)
$X_1X_2...X_ma_ia_{i+1}...a_n$是一个右句型，
$X_1X_2...X_m$是它的一个活前缀。

➢结论:对句柄的识别转变成对活前缀的识别。

4.7.1 LR分析程序的模型及基本原理

示例

句型aAbcde的句柄是Ab

该句型的活前缀有ε、a、aA、aAb；

句型aAcde的句柄是d

它的活前缀有：ε、a、aA、aAc、aAcd。

文法G[S]：
(1) S → aAcBe
(2) A → b
(3) A → Ab
(4) B → d

4.7.1 LR分析程序的模型及基本原理

构造识别给定文法所有活前缀的DFA

- 活前缀与句柄之间的关系
  - 活前缀不含有句柄的任何符号
  - 活前缀只含有句柄的部分符号
  - 活前缀已经含有句柄的全部符号
- 分析过程中，分析栈中出现的活前缀
  - 第一种情况，期望从剩余输入串中能够看到由某产生式A→α的右部α所推导出的终结符号串；
  - 第二种情况，某产生式A→α1α2的右部子串α1已经出现在栈顶，期待从剩余的输入串中能够看到α2推导出的符号串；
  - 第三种情况，某一产生式A→α的右部符号串α已经出现在栈顶，用该产生式进行归约。

4.7.1 LR分析程序的模型及基本原理

构造识别给定文法所有活前缀的DFA

为方便标记语法分析器已读入了多少输入，可以引入一个点记号“•”

0: S -> E
1: E -> E + T
2: | T
3: T -> T * F
4: | F
5: F -> n

•1 + 2 * 3
F •+ 2 * 3
T •+ 2 * 3
E •+ 2 * 3
E + •2 * 3
E + 2• * 3
E + F • * 3
E + T • * 3
E + T * • 3
E + T * 3•
E + T * F•
E + T•
E•
S•

4.7.1 LR分析程序的模型及基本原理

LR项目

文法G的每个产生式右部的某个位置添加一个“·”，项目圆点的左部表示分析过程的某个时刻用该产生式归约时句柄已识别的部分，圆点右部表示待识别的部分,用于替代活前缀直接构造所需的DFA。

4.7.1 LR分析程序的模型及基本原理

LR(0)项目

右部某个位置上标有圆点的产生式称为文法G的一个LR(0)项目。

产生式A→XYZ对应有4个LR(0)项目

A→·XYZ

A→X·YZ

A→XY·Z

A→XYZ·

归约项目：圆点在产生式最右端的LR(0)项目

接受项目：对文法开始符号的归约项目

待约项目：圆点后第一个符号为非终结符号的LR(0)项目

移进项目：圆点后第一个符号为终结符号的LR(0)项目

产生式A→ε，只有一个LR(0)归约项目 A→·。

4.7.1 LR分析程序的模型及基本原理

拓广文法

为保证文法开始符号出现在一个产生式的左边，需要把文法加以拓广。

文法G的拓广文法：

任何文法$G=(V_T,V_N,S,\varphi)$，都有等价的文法 $G'=(V_T,V_N\cup\{S'\},S',\varphi\cup\{S'\rightarrow S\})$，称G′为G的拓广文法。

拓广文法G′的接受项目是唯一的(即$S'\rightarrow S\cdot$)

示例：求如下文法的LR(0)项目

G[S]:

<S> → var <IDS>:<TYPE>

<IDS> → <IDS>,i

<IDS> → I

<TYPE> → real

示例：求如下文法的LR(0)项目

重庆大学 CHONGQING UNIVERSITY

第1步：拓广文法

G[S']: (略简化并缩写)

0. S' ⟶ S
1. S ⟶ vI:T
2. I ⟶ I,i
3. I ⟶ i
4. T ⟶ r

第2步：构造LR（0）项目

LR(0)项目

0. S' ⟶ •S
1. S' ⟶ S•
2. S ⟶ • vI:T
3. S ⟶ v•I:T
4. S ⟶ vI•:T
5. S ⟶ vI:•T
6. S ⟶ vI:T•
7. I ⟶ •I,i
8. I ⟶ I•,i
9. I ⟶ I,•i
10. I ⟶ I,i•
11. I ⟶ •i
12. I ⟶ i•
13. T ⟶ •r
14. T ⟶ r•

4.7.1 LR分析程序的模型及基本原理

LR有效项目

项目$A\to\beta_1\cdot\beta_2$对活前缀$\gamma=\alpha\beta_1$是有效的，如果存在一个规范推导：

$$S\overset{*}{\Rightarrow}\alpha A\omega\Rightarrow\alpha\beta_1\beta_2\omega$$

推广：若项目$A\to\alpha\cdot B\beta$对活前缀$\gamma=\delta\alpha$是有效的，并且$B\to\eta$是一个产生式，则项目$B\to\cdot\eta$对活前缀$\gamma=\delta\alpha$也是有效的。

LR(0)项目$S'\to\cdot S$是活前缀ε的有效项目（这里取$\gamma=\varepsilon$，$\beta_1=\varepsilon$，$\beta_2=S$，$A=S'$）。

4.7.1 LR分析程序的模型及基本原理

LR(0)有效项目集和LR(0)项目集规范族

文法G的某个活前缀γ的所有LR(0)有效项目组成的集合称为γ的LR(0)有效项目集。

文法G的所有LR(0)有效项目集组成的集合称为G的LR(0)项目集规范族。

如何构造LR(0)项目集族?

4.7.1 LR分析程序的模型及基本原理

定义4.11：闭包(closure)

设I是文法G的一个LR(0)项目集合，closure(I)是从I 出发，用下面的方法构造的项目集：

(1) I中的每一个项目都属于closure(I)；
(2) 若项目A→α·Bβ属于closure(I)，
　　且B→η是G的一个产生式，
　　若B→·η不属于closure(I)，
　　则将B→·η加入closure(I)；
(3) 重复规则(2)，直到closure(I)不再增大为止

输入：项目集合I。
输出：集合J=closure(I)。
方法：

```
J=I;
do {
    J_new=J;
    for ( J_new中的每一个项目A→α·Bβ
          和文法G的每个产生式B→η)
        if (B→·η∉J)  把B→·η加入J;
} while (J_new!=J).
```

4.7.1 LR分析程序的模型及基本原理

转移函数—GO(I, X)函数

若I是文法G的一个LR(0)项目集，X是一个文法符号，定义

go(I,X)=closure(J)

其中：J={ A→αX·β|当 A→α·Xβ 属于 I 时 }，go(I,X)称为转移函数，项目A→αX·β称为A→α·Xβ的后继

直观含义：若I中的项目A→α·Xβ是某个活前缀 γ=δα的有效项目，J中的项目A→αX·β是活前缀δαX（即γX）的有效项目。

4.7.1 LR分析程序的模型及基本原理

算法 构造文法G的LR(0)项目集规范族

输入：文法G
输出：G的LR(0)项目集规范族C
方法：

```
C={closure({S′→·S})};
do
   for (对C中的每一个项目集I和每一个文法符号X)
      if  (go(I,X)不为空，且不在C中)
          把go(I,X)加入C中;
while (没有新项目集加入C中);
```

这里closure({S′→·S})是活前缀 ε 的有效项目集

4.7.1 LR分析程序的模型及基本原理

示例

构造如下文法G的LR(0)项目集规范族：S→aA|bB　A→cA|d　B→cB|d

4.7.1 LR分析程序的模型及基本原理

构造如下文法G的LR(0)项目集规范族：S→aA|bB　A→cA|d　B→cB|d

1、拓广文法G′：S′→S　　S→aA|bB　　A→cA|d　　B→cB|d

2、活前缀ε的有效项目集 I_0=closure({S′→·S})={ S′→·S, S→·aA, S→·bB }

从I_0出发的转移有

I_1=go(I_0,S)=closure({S′→S·})={S′→S·}

I_2 =go(I_0,a)=closure({S→a·A})={S→a·A, A→·cA, A→·d}

I_3 =go(I_0,b)=closure({S→b·B})={S→b·B, B→·cB, B→·d}

从I2出发的转移有

I4=go(I2,A)=closure({S→aA·})={ S→aA·}

I5=go(I2,c)=closure({A→c·A})={ A→c·A, A→·cA, A→·d}

I6=go(I2,d)=closure({A→d·})={A→d·}

从I3出发的转移有

I7=go(I3,B)=closure({S→bB·})={ S→bB·}

I8=go(I3,c)=closure({B→c·B})={ B→c·B, B→·cB, B→·d}

I9=go(I3,d)=closure({B→d·})={B→d·}

4.7.1 LR分析程序的模型及基本原理

构造如下文法G的LR(0)项目集规范族：S→aA|bB　A→cA|d　B→cB|d

从I5出发的转移有

I10=go(I5,A)=closure({A→cA·})={ A→cA·}

go(I5,c)=closure({A→c·A})=I5

go(I5,d)=closure({A→d·})=I6

从I8出发的转移有

I11=go(I8,B)=closure({B→cB·})={ B→cB·}

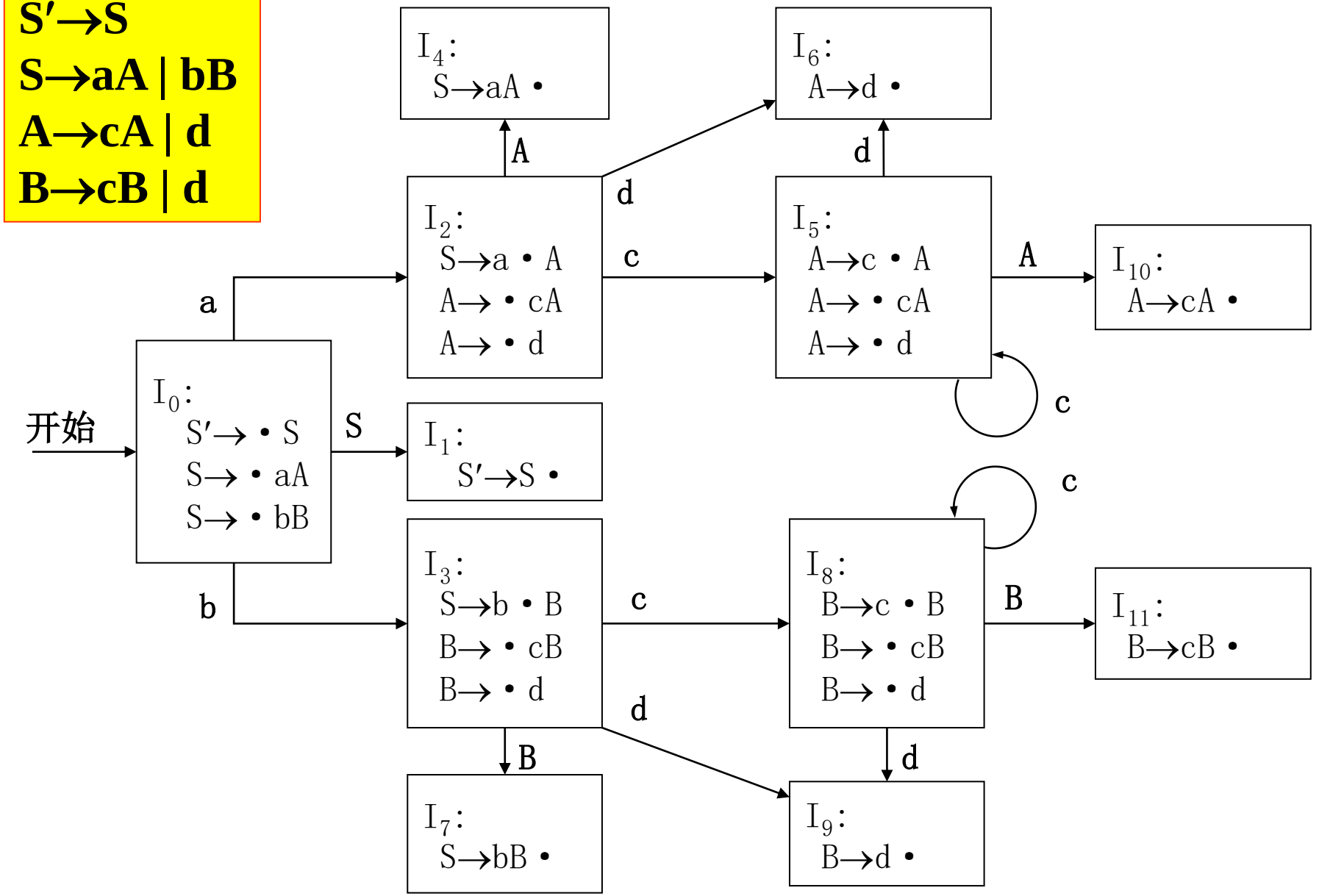
go(I8,c)=closure({B→c·B})=I8

go(I8,d)=closure({B→d·})=I9

文法G'的LR(0)项目集规范族C={I0、I1、…、I11}

识别文法G′的所有活前缀的DFA

S′→S
S→aA | bB
A→cA | d
B→cB | d



课堂讨论

如果在LR(0)项目的同一项目集中出现如下情况会怎样?

I->i·

I->i· a

4.7.1 LR分析程序的模型及基本原理

LR（0）项目集规范族的项目类型分为如下四种：

1）移进项目 A →α·aβ

2）待约项目 A →α·Bβ

3）归约项目 A →α·

4）接受项目 S ‘→ S·

一个项目集可能包含多种项目

a) 移进和归约项目同时存在。移进-归约冲突

b) 归约和归约项目同时存在。归约-归约冲突

4.7.2 LR(0)分析法及SLR(1)分析法

如果文法G′(文法G的拓广文法)的项目集规范族的每个项目集中不存在下述任何冲突项目：

① 移进项目和归约项目并存

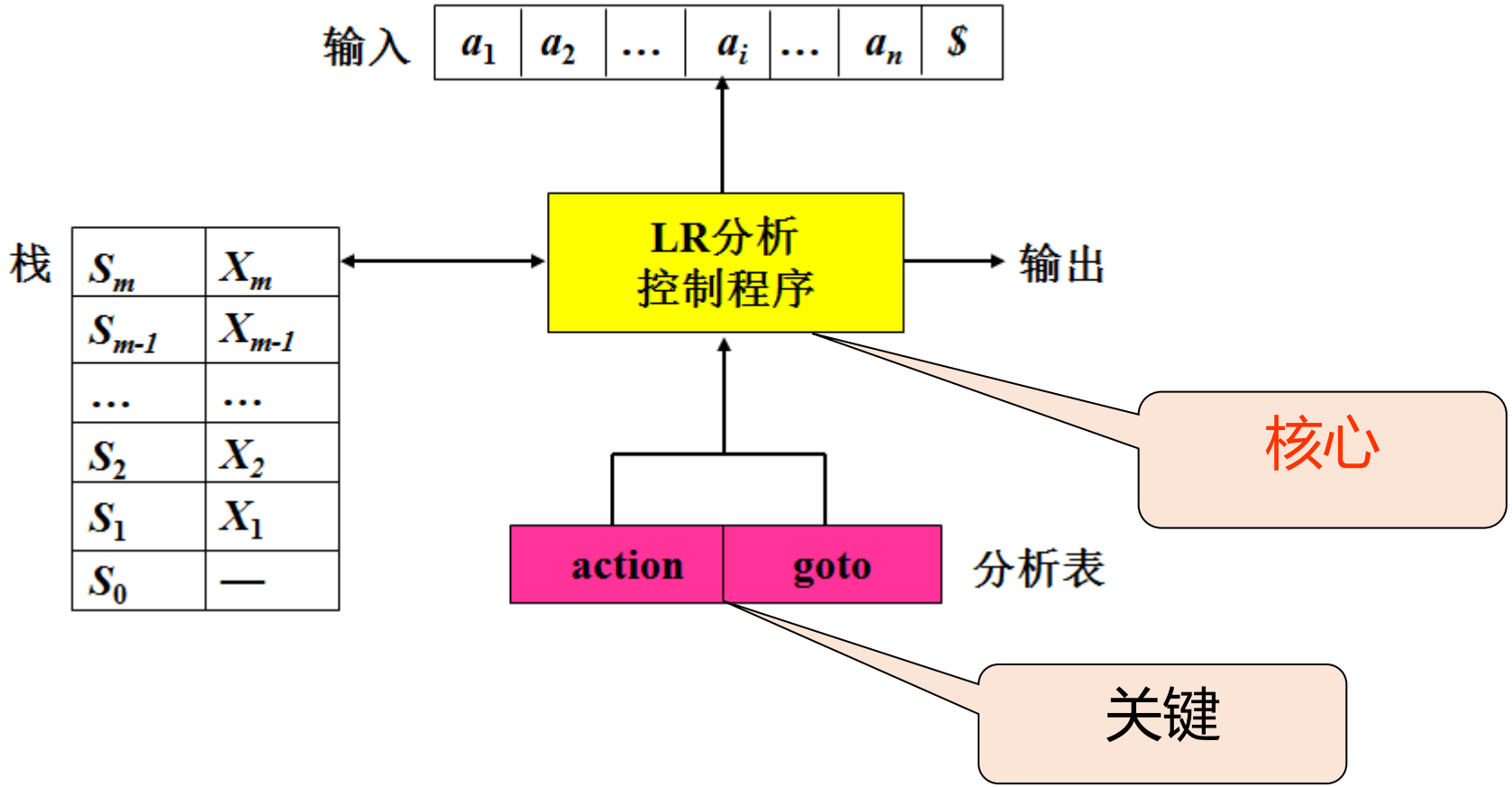
② 多个归约项目并存

则称文法G′为LR(0)文法。

(LR(0)文法：若其LR(0)项目集规范族不存在移进-归约，或归约-归约冲突，称为LR(0)文法。)

4.7.2 LR(0)分析法及SLR(1)分析法

LR分析程序的模型及工作过程



4.7.2 LR(0)分析法及SLR(1)分析法

LR(0)分析表

LR分析法的核心，它包含两部分：动作表（ACTION）和状态转换表（GOTO）

动作表动作规定如下：

$$action[s_m,a_i]=\begin{cases}\text{移进} & a_i \text{ 和} s=action[s_m,a_i]\text{进栈}\\ \text{归约} & r_j: A\rightarrow X_{m-r+1}X_{m-r+2}\ldots X_m\\ \text{接受} & s=goto[s_{m-r}, A]\\ \text{出错} & \end{cases}$$

GOTO[s,a]=后继状态,表示在当前状态下.栈顶归约为某非终结符时,所指向的下一个状态,用状态号表示

4.7.2 LR(0)分析法及SLR(1)分析法

LR分析控制程序

LR分析控制程序的工作过程：

分析开始时，初始的二元式为：$(S_0, a_1a_2\cdots a_n\$)$

分析过程中每步的结果，均可表示为如下的二元式：$(S_0S_1S_2\cdots S_m, a_ia_{i+1}\cdots a_n\$)$

若action$[S_m, a_i]$=shift S，且S=goto$[S_m, a_i]$，则二元式变为：$(S_0S_1\cdots S_mS, a_{i+1}\cdots a_n\$)$

若action$[S_m, a_i]$=reduce by $A\rightarrow\beta$，则二元式变为：$(S_0S_1\cdots S_{m-r}S, a_ia_{i+1}\cdots a_n\$)$

若action$[S_m, a_i]$=accept（接受），则分析成功，二元式变化过程终止。

若action$[S_m, a_i]$=error（出错）发现错误，调用错误处理程序。

4.7.2 LR(0)分析法及SLR(1)分析法

LR(0)分析表的构造

①若项目$A\rightarrow\alpha\cdot X\beta\in Ik_i$且GO(Ik, x)=$Ij_j$，若$X\in V_T$，且X=a,则置ACTION[i, a]=Sj，即“将状态j、符号x移进栈”；但若$x\in V_N$，则仅置GOTO[i, X]=j。

②若项目$A\rightarrow\alpha\cdot\in Ik$，对于任何输入符号$a\in(V_T\cup\{\#\})$，则置ACTION[i,a]=rj，即“用第j条产生式$A\rightarrow\alpha$进行归约”

③若项目$S'\rightarrow S\cdot\in Ik$，则置ACTION[k, #]=“acc”。

④分析表中凡不能用规则①~③填入信息的元素均填上”报错”（用空白表示）。

4.7.2 LR(0)分析法及SLR(1)分析法

构造LR(0)分析表的步骤小结

①写出给定文法G的拓广文法G′；

②写出文法G′的基本LR(0)项目——G′的LR(0)项目集；

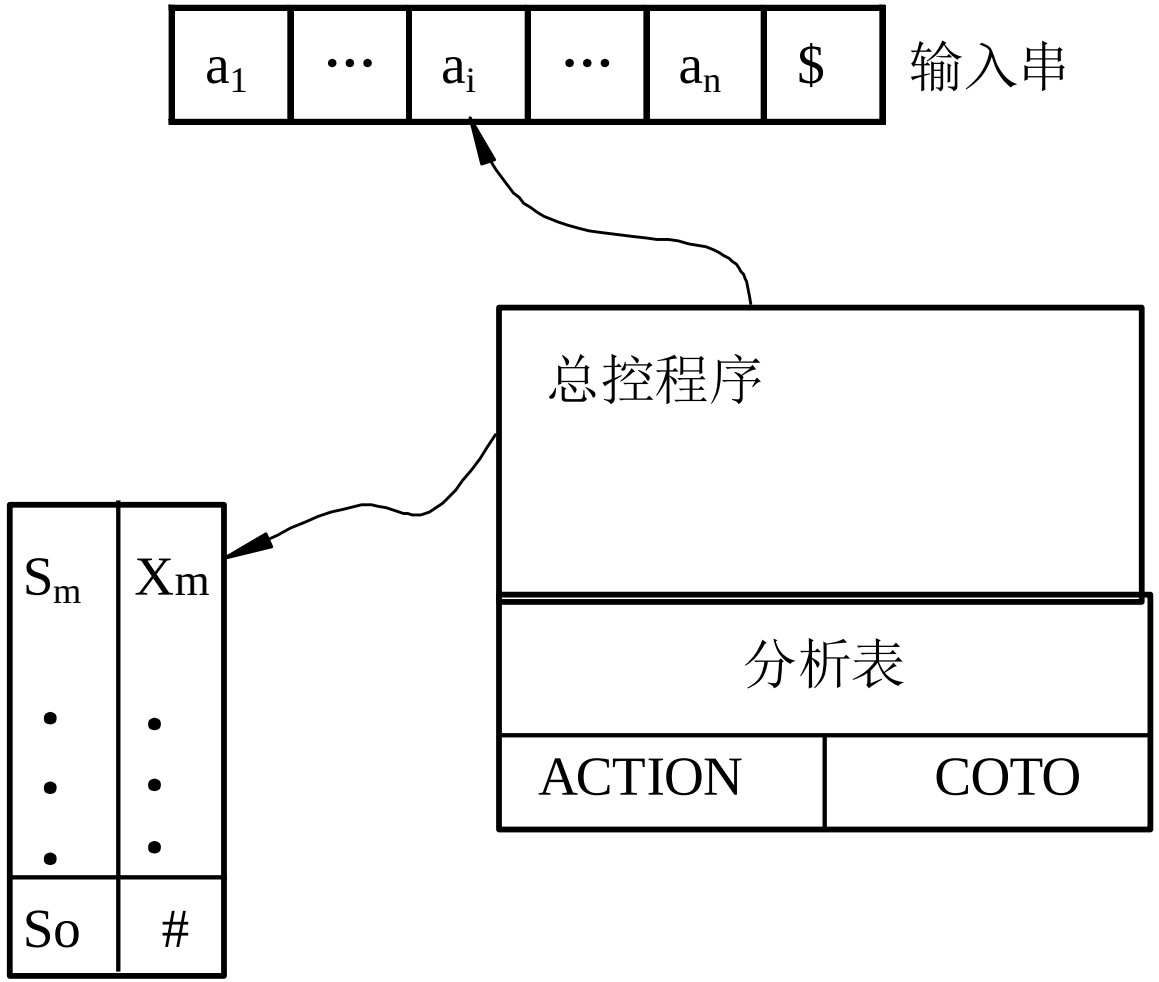
③利用CLOSURE和GO函数，求出相应的LR(0)项目集规范族C；

④构造识别该文法全部活前缀的DFA；

⑤根据算法构造LR(0)分析表。

4.7.2 LR(0)分析法及SLR(1)分析法

LR(0)分析过程



4.7.2 LR(0)分析法及SLR(1)分析法

算法4.3 LR分析控制程序

输入：文法G的一张分析表和一个输入符号串ω

输出：若ω∈L(G)，得到ω的自底向上的分析，否则报错

方法：开始时，初始状态S_0在栈顶，ω$在输入缓冲区中，置ip指向ω$的第一个符号；

```
do {
    令S是栈顶状态，a是ip所指向的符号
    if (action[S，a]==shift S′) {
        把a和S′分别压入符号栈和状态栈;
        推进ip，使它指向下一个输入符号;
    };
    else if (action[S，a]==reduce by A→β) {
            从栈顶弹出|β|个符号; （令S′是现在的栈顶状态）
            把A和goto[S′，A]分别压入符号栈和状态栈;
            输出产生式A→β;
        };
        else if (action[S，a]==accept) return;
            else  error();
} while(1).
```

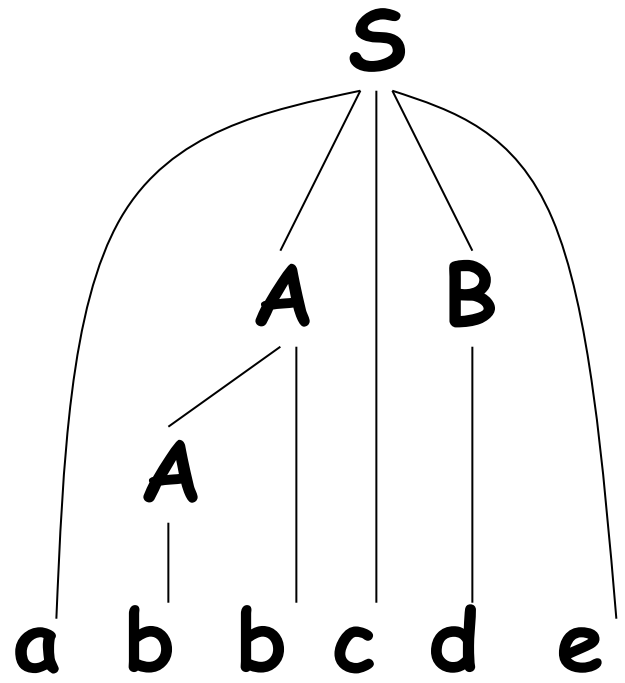
示例：文法G[S]

(1) S → aAcBe

(2) A → b

(3) A → Ab

(4) B → d



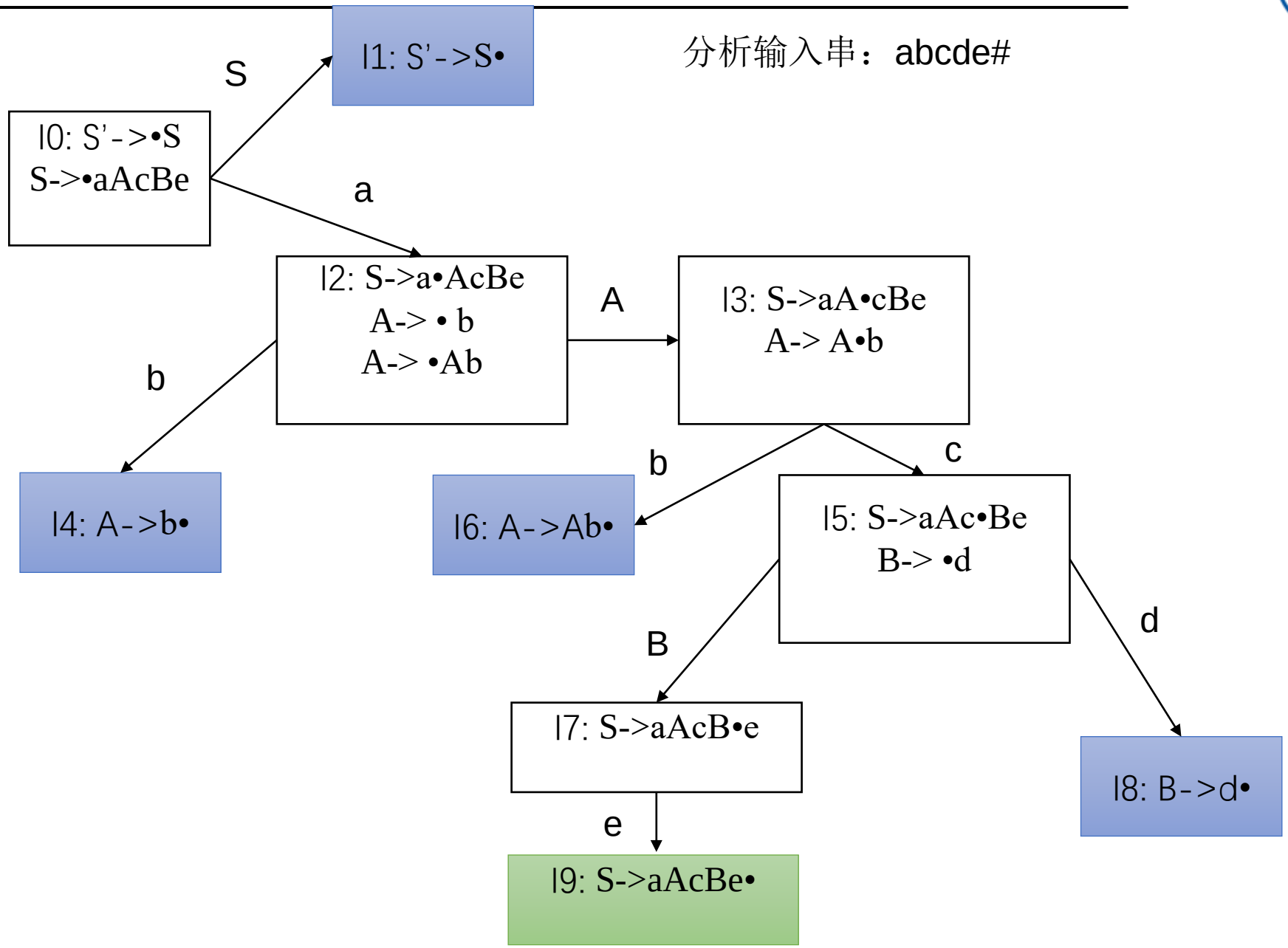
步骤	符号栈	输入符号串	动作
1）	#	abbcde#	移进
2）	#a	bbcde#	移进
3）	#ab	bcde#	归约(A→b)
4）	#aA	bcde#	移进
5）	#aAb	cde#	归约(A→Ab)
6）	#aA	cde#	移进
7）	#aAc	de#	移进
8）	# aAcd	e#	归约(B→d)
9）	#aAcB	e#	移进
10）	#aAcBe	#	归约(S→aAcBe)
11）	#S	#	接受

对输入串abbcde#的移进-规约分析过程

分析符号串abbcde是否G[S]的句子

$$S \Rightarrow aAcBe \Rightarrow aAcde \Rightarrow aAbcde \Rightarrow abbcde$$

4.7.2 LR(0)分析法及SLR(1)分析法



分析输入串：abcde#

4.7.2 LR(0)分析法及SLR(1)分析法

abbcde#

	ACTION						GOTO		
	a	c	e	b	d	#	S	A	B
0	S_2						1		
1						acc			
2				S_4				3	
3		S_5		S_6					
4	r_2	r_2	r_2	r_2	r_2	r_2			
5					S_8				7
6	r_3	r_3	r_3	r_3	r_3	r_3			
7			S_9						
8	r_4	r_4	r_4	r_4	r_4	r_4			
9	r_1	r_1	r_1	r_1	r_1	r_1			

4.7.2 LR(0)分析法及SLR(1)分析法

LR(0)分析的特点

一旦栈顶形成句柄,都可立即归约,且不会发生错误。

思考:LR(0)文法的要求每个项目集中不存在冲突项,如果存在会如何?
若文法不是LR(0)文法,如何解决冲突?

向前查看一个符号来判断当时所处的环境,然后决定是执行移进还是归约这种做法将能满足一些文法的需要，因为只对有冲突的状态才向前查看一个符号，以确定做那种动作，因而称这种分析方法为**简单的LR(1)分析法**，用SLR(1)表示。

4.7.2 LR(0)分析法及SLR(1)分析法

SLR(1)分析方法

如果文法的有效项目集中有冲突动作，多数冲突可通过考察有关非终结符号的FOLLOW集合而得到解决。

如项目集：$I=\{X\rightarrow\alpha\cdot b\beta,\ A\rightarrow\alpha\cdot,\ B\rightarrow\alpha\cdot\}$存在移进-归约冲突及归约-归约冲突。

冲突的解决方法，查看FOLLOW(A)和FOLLOW(B)，如果$FOLLOW(A)\cap FOLLOW(B)=\Phi$，$b\notin FOLLOW(A)$并且$b\notin FOLLOW(B)$，则采取以下决策：

- 当a=b时，把b移进栈里；
- 当$a\in FOLLOW(A)$时，用产生式$A\rightarrow\alpha$进行归约；
- 当$a\in FOLLOW(B)$时，用产生式$B\rightarrow\alpha$进行归约。

4.7.2 LR(0)分析法及SLR(1)分析法

示例：构造如下文法的SLR（1）分析表：

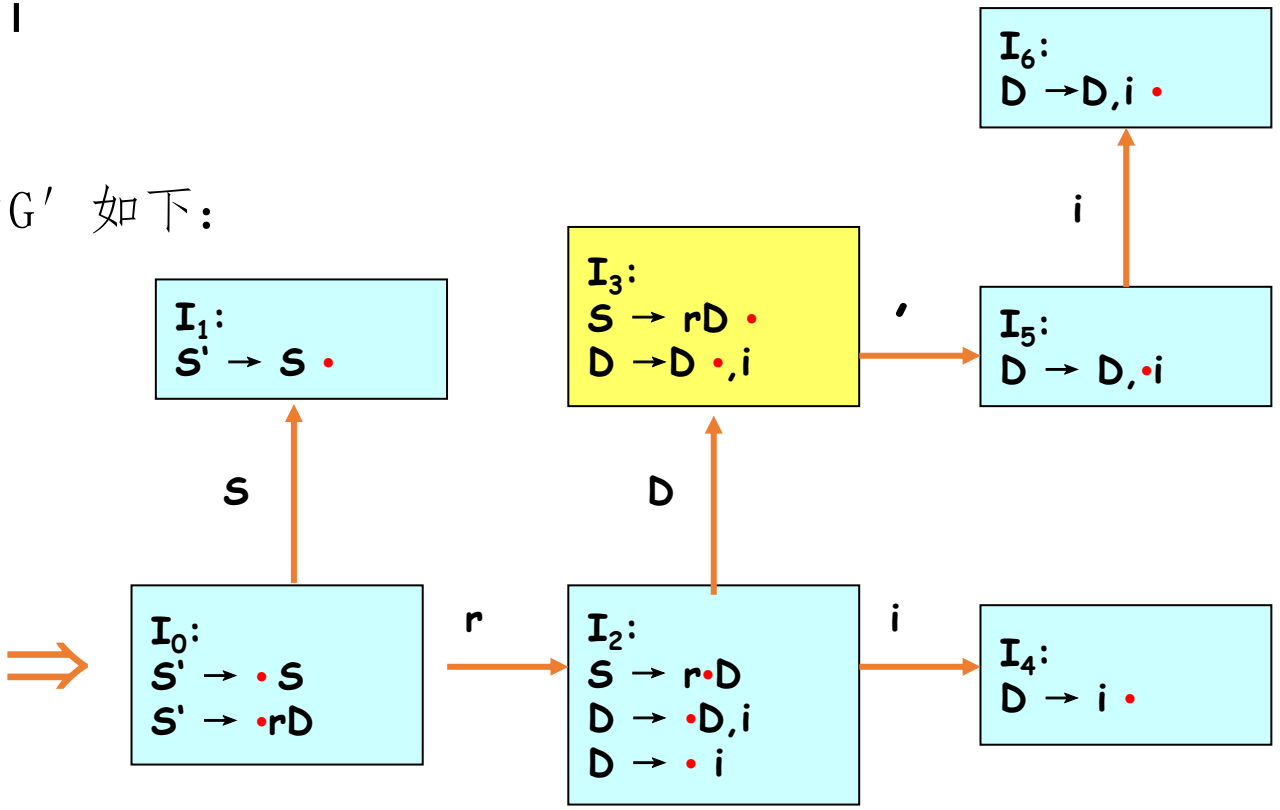
<实型变量说明> → real <标识符表>
<标识符表> → <标识符表> , i
<标识符表> →i

解答：将该文法缩写后并拓广为G′如下：

(0) S′ →S
(1) S→rD
(2) D→D,i
(3) D→i

I3中含项目：

S→rD· 为归约项目
D→D·, i 为移进项目
显然该文法不是LR(0)文法



4.7.2 LR(0)分析法及SLR(1)分析法

I_3:
S → rD ·
D →D ·,i

FOLLOW(S)={#}

向前查看一个符号，看其是否是S的后跟符号（FOLLOW(S)），
若否则移进，是则归约.

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S_2				1	
1				acc		
2			S_4			3
3		S_5		r_1		
4	r_3	r_3	r_3	r_3		
5			S_6			
6	r_2	r_2	r_2	r_2		

4.7.2 LR(0)分析法及SLR(1)分析法

SLR(1)文法

若一个文法的LR(0)分析表中所含有的动作冲突都能用SLR(1)方法解决，则称这个文法是**SLR(1)文法**.

实际上大多数实用适用的程序设计语言的文法也不能满足SLR(1)文法的条件，但是SLR(1) 分析法是为了解LR(1)分析法做准备的.

4.7.2 LR(0)分析法及SLR(1)分析法

算法: 构造SLR(1)分析表

输入：拓广文法G′　　输出：G′的SLR分析表　　方法如下：

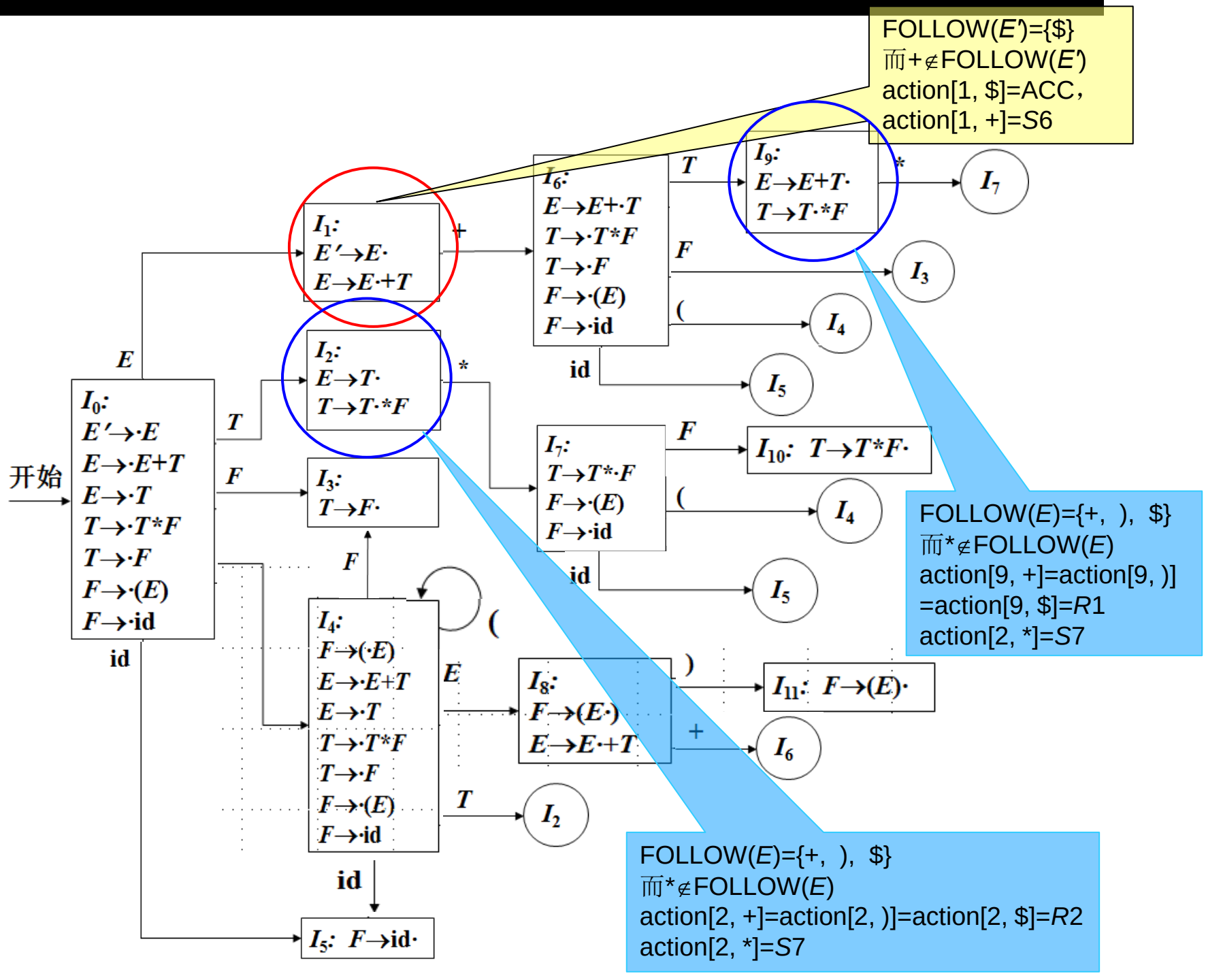
1. 构造G′的LR(0)项目集规范族C={$I_0, I_1, \cdots I_n$}。
2. 对于状态i（对应于项目集I_i的状态）的分析动作如下
   a) 若$A\rightarrow\alpha\cdot a\beta\in I_i$，且$go(I_i,a)=I_j$，则置action[i,a]=sj
   b) 若$A\rightarrow\alpha\cdot\in I_i$，则对所有$a\in FOLLOW(A)$，置action[i,a]=r $A\rightarrow\alpha$
   c) 若$S'\rightarrow S\cdot\ \in I_i$，则置action[i,$]=acc，表示分析成功
3. 若$go(I_i,A)=I_j$，A为非终结符号，则置goto[i,A]=j
4. 分析表中凡不能用规则(2)、(3)填入信息的空白表项，均置为出错标error。
5. 分析程序的初态是包含项目$S'\rightarrow S$的有效项目集所对应的状态。

4.7.2 LR(0)分析法及SLR(1)分析法

示例：判断下列文法是LR(0)文法，还是SLR(1)文法

(0) E′→E　　(1) E→E+T　　(2) E→T　　(3) T→T*F

(4) T→F　　(5) F→(E)　　(6) F→id


FOLLOW(E')={$}
而+∉FOLLOW(E')
action[1, $]=ACC，
action[1, +]=S6
开始
I0:
E′→·E
E→·E+T
E→·T
T→·T*F
T→·F
F→·(E)
F→·id
I1:
E′→E·
E→E·+T
I2:
E→T·
T→T·*F
I3:
T→F·
I4:
F→(·E)
E→·E+T
E→·T
T→·T*F
T→·F
F→·(E)
F→·id
I5: F→id·
I6:
E→E+·T
T→·T*F
T→·F
F→·(E)
F→·id
I7:
T→T*·F
F→·(E)
F→·id
I8:
F→(E·)
E→E·+T
I9:
E→E+T·
T→T·*F
I10: T→T*F·
I11: F→(E)·
FOLLOW(E)={+,), $}
而*∉FOLLOW(E)
action[9, +]=action[9,)]
=action[9, $]=R1
action[2, *]=S7
FOLLOW(E)={+,), $}
而*∉FOLLOW(E)
action[2, +]=action[2,)]=action[2, $]=R2
action[2, *]=S7
编译原理

课堂讨论

- 思考：SLR(1)文法是二义的吗?

每一个SLR(1)文法都是无二义的文法,但并非无二义的文法都是SLR(1)文法。

例：有文法G具有如下产生式：

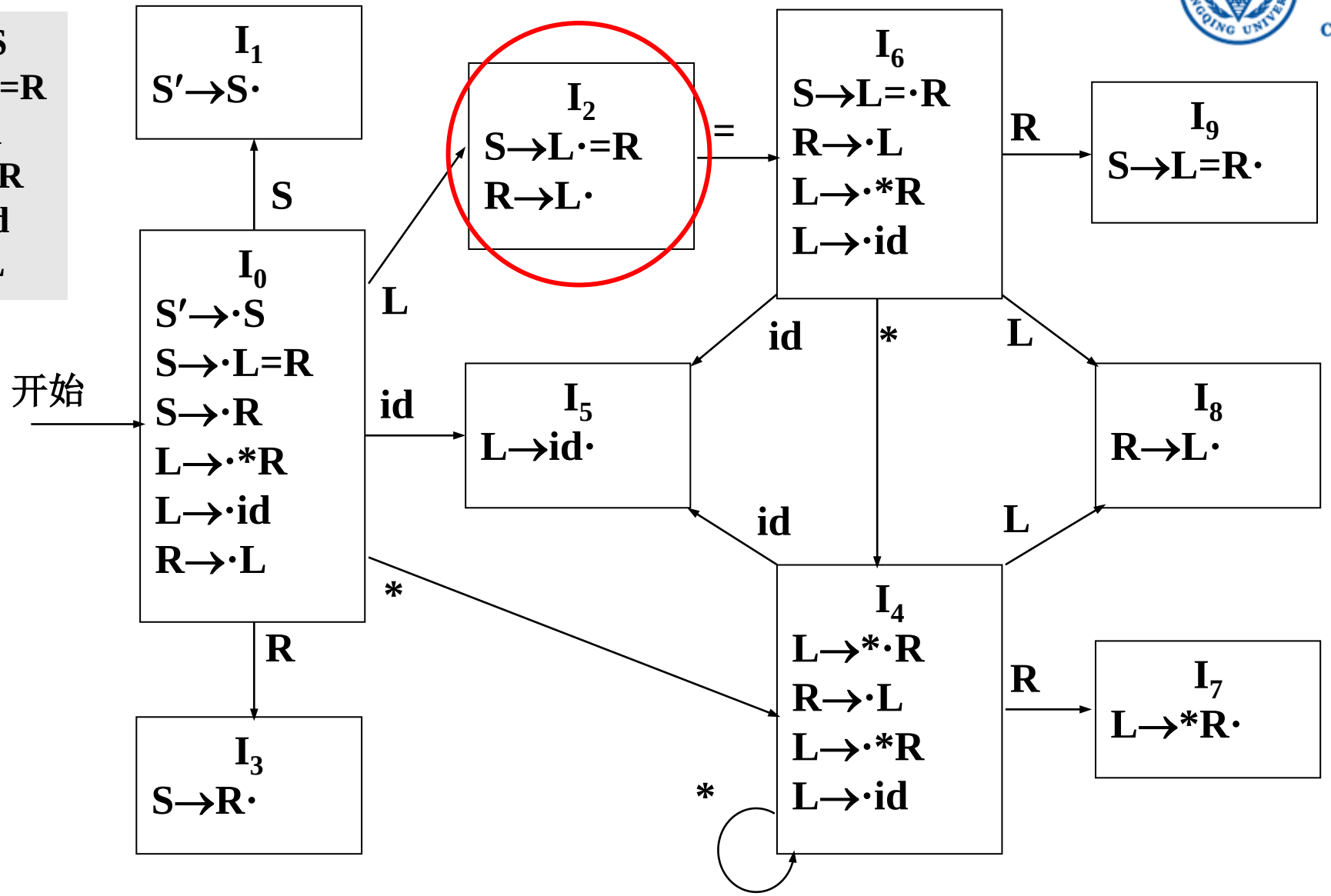
S→L=R
S→R
L→*R
L→id
R→L　　（文法4.7）

该文法无二义性，但不是SLR(1)文法。
拓广文法G′的产生式如下：

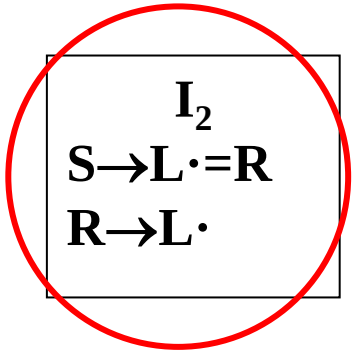
(0) S′→S　　(1) S→L=R　　(2) S→R
(3) L→*R　　(4) L→id　　(5) R→L

构造文法G′的LR(0)项目集规范族及识别活前缀的DFA

S'→S
S→L=R
S→R
L→*R
L→id
R→L



4.7.2 LR(0)分析法及SLR(1)分析法



为之构造SLR分析表

考察I_2:

项目S→L·=R与R→L·存在移进-归约冲突,

根据第一个项目,有: action[2,=]=s6

根据第二个项目,由于FOLLOW(R)={=,$}所以有:

action[2,$]=action[2,=]=r5, 存在“移进-归约”冲突。

SLR分析表中未包含足够多的预测信息，仍有许多文法构造的LR(0)项目集规范族存在的动作冲突不能用SLR(1)方法解决.

课后作业 1

已知文法：

G[E]: E ->E+T|E-T|T
T ->T*F|T/F|F
F ->i|(E)

试利用算符优先分析法分析句子(1+2)*3/4

课后作业 2

请用算法优先分析法，对于带括号的整数四则运算表达式语句进行语法分析，并编程实现。

输入：算数表达式

输出：运算结果及语法架子树

要求：可手工构造算符优先关系表，能识别错误表达式并给出相应报错信息。提交源码及相应文档（包括文法，算符优先分析表及测试用例）

课后作业 3

文法： (1) E→E+T (2) E→T (3) T→T*F (4) T→F (5) F→(E) (6) F→id

对输入串i+i*i分别给出SLR(1)和算符优先分析过程,并对两种方法进行比较,说明它们各自的优缺点