基本思想

把一个输入符号串逐步归约到文法的开始符号。

用一个栈，将输入符号一个一个移进栈内，并检查栈顶符号串。当栈顶形成某个产生式的右部时，将栈顶的这一部分归约为它的左部符号。直到移进所有的输入符号串时，若栈内是开始符号，则符合语法。

对于**规范句型**来说，句柄的后面不会出现非终结符号。

LR分析方法

L：从左到右扫描

R：最右推导的逆过程

k：为了做出分析决定而向前看的输入符号的个数。

LR分析表

由两部分组成：分析动作表和状态转换表

- 分析动作表action

  以一个二维数组表示。行代表分析栈栈顶状态，列代表当前的输入符号，元素表示当前栈顶为状态S_i。当输入符号为a_j时，所执行的分析动作为$action[S_i, a_j]$

  动作有四种：

  - 移进：新状态S_i进状态栈，输入符号a_j进符号栈
  - 归约：用相应的产生式进行归约r_j。若产生式右边的符号串长度为n，则符号栈栈顶的n个符号为句柄，所以符号栈、状态栈栈顶的n个元素**出栈**，规约后的文法符号（非终结符）进栈，并根据goto表查规约后的文法符号所对应的**新状态进栈。**
  - 接受：当文法符号归约到只剩下开始符号，且输入串结束（当前输入为#）时，分析成功
  - 报错：状态栈栈顶为某一状态下，出现了不该出现的文法符号、

- 状态转换表

  用一个二维数组表示，行代表分析栈栈顶状态，列代表文法符号，元素表示当前栈顶为状态S_i面对文法符号为X_j时，应转去的新状态$Goto[S_i, X_j] = S_k$

算法过程

1. 将初始状态S_0和输入串的左边界#分别进分析栈
2. 根据状态栈栈顶和当前输入符号，查动作表，执行相应的动作

> 例：

有文法G[L]
(1)L→E , L

(2)L→E

(3)E→a

(4)E→b

state	Action表				Goto表	
	a	b	,	#	E	L
0	S_3	S_4			2	1
1				acc		
2			S_5	r_2		
3			r_3	r_3		
4			r_4	r_4		
5	S_3	S_4			2	6
6				r_1		

要求对输入串a,b,a进行分析，即分析#a,b,a#

51

(1)L→E,L
(2)L→E
(3)E→a
(4)E→b

动作序列中包含了一个规范归约，从而为输入符号串自底向上构造了一棵分析树

步骤	状态栈S	符号栈X	输入符号	动作
1	0	#	a,b,a#	
2	03	#a	,b,a#	
3	02	#E	,b,a#	r_3(E→a)
4	025	#E,	b,a#	
5	0254	#E,b	,a#	
6	0252	#E,E	,a#	r_4(E→b)
7	02525	#E,E,	a#	
8	025253	#E,E,a	#	
9	025252	#E,E,E	#	r_3(E→a)
10	025256	#E,E,L	#	r_2(L→E)
11	0256	#E,L	#	r_1(L→E,L)
12	01	#L	#	r_1(L→E,L)
13			分析成功	acc

一个上下文无关文法，若构造出无冲突的LR分析表，则说明该文法是LR文法。

LR分析表的构造

活前缀

活前缀是规范句型的一个前缀，这种前缀不含句柄之后的任何符号。即，对于规范句型$\alpha\beta\delta$，β为句柄，如果$\alpha\beta = u_1u_2\cdots u_r$，则符号串$u_1u_2\cdots u_i, 1 \le i \le r$是$\alpha\beta\delta$的活前缀。且$\delta$必为终结符串。

简单来说，活前缀是句柄的子集。

为了刻画分析过程中文法的每一个产生式的右部符号已有多大一部分被识别（已出现在栈顶），在产生式的右部加上一个圆点指示位置。

- 活前缀已含有句柄的全部符号

  $A \to \beta\cdot$

- 活前缀含有句柄的部分符号

  $A \to \beta_1 \cdot \beta_2$

- 活前缀不含有句柄的任何符号

  $A \to \cdot\beta$

构造识别活前缀的DFA

项目

要想知道活前缀有多大部分进栈了，可以为每个产生式构造一个自动机，由它的状态来记住当前情况，此状态称为项目。

可以理解为，项目是在分过程中的某一时刻，已经规约的部分和等待规约的部分。

一个产生式对应的项目个数是右部符号长度+1。

分类

- 移进项目：圆点后为终结符。表示栈内是句柄的一部分，期待从输入串中移进一个符号，以形成句柄。
- 待约项目：圆点后为非终结符。表示栈内是句柄的一部分，期待从剩余的输入串中进行归约而得到该终结符，以形成句柄。
- 归约项目：圆点在最后。表示栈顶的部分内容已经构成所期望的句柄，应使用产生式$A \to \alpha$进行归约
- 接受项目：圆点在最后，且使用产生式$S' \to \alpha$进行归约，表示整个句子已经分析完毕，分析成功。

  > 拓广文法：添加一条文法$S' \to S$

求项目集闭包

对于圆点右侧为非终结符的，将其对应产生式加入当前项目集。

直接构造DFA（判断是否LR0）

在每个项目集中，按照圆点右侧的符号分支，新的项目集要求闭包，并将圆点移进一位。从而构造DFA。

例：

G[S]:

S -> aAcBe

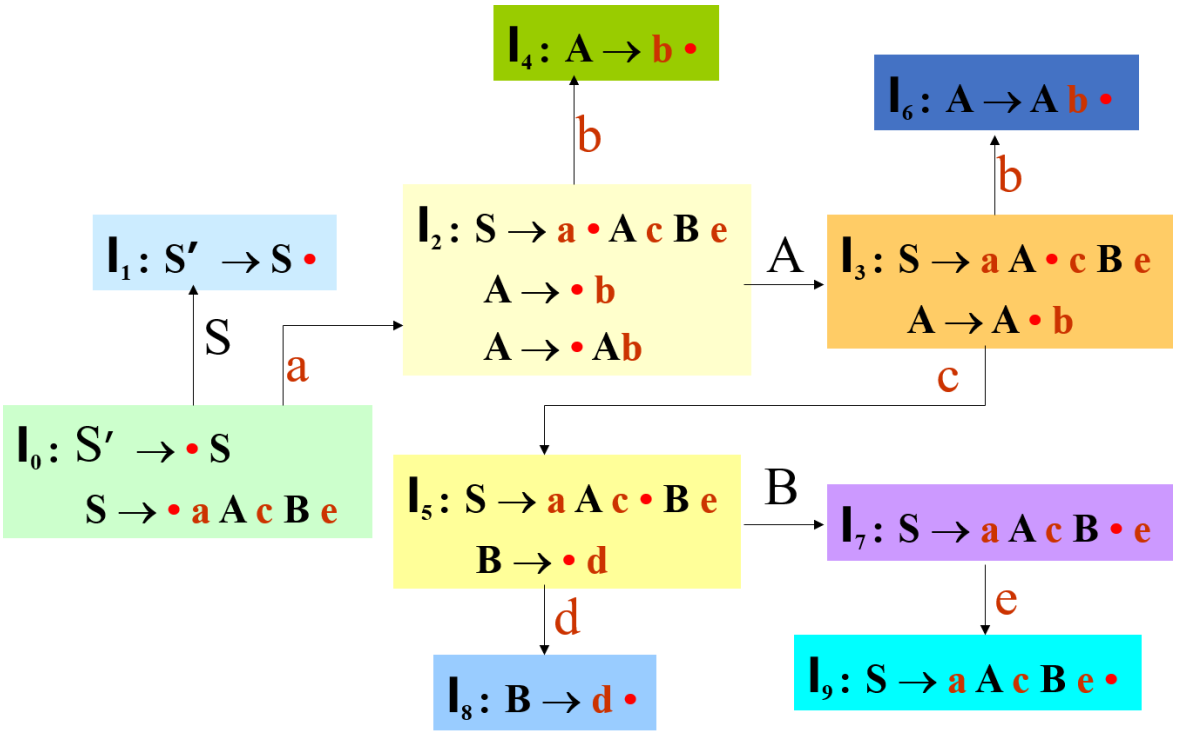
A -> b

A -> Ab

B -> d

1. 通过项目集规范族构造识别活前缀的DFA

   构造拓广文法，$S' \to S$。



2. 构造LR(0)分析表

根据上面的DFA，I的下标就是状态。

终结符的弧是移进S，填入ACTION

非终结符的弧是GOTO

归约项目r填入ACTION，r下标和文法序号对应。

	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			

3. abbce#是否为G[S]句子

Step	states.	Syms.	The rest of input	action	goto
1	0	#	abbce#	s2	
2	02	#a	bbce#	s4	
3	024	#ab	bce#	r2	3
4	023	#aA	bce#	s6	
5	0236	#aAb	ce#	r3	3
6	023	#aA	ce#	s5	
7	0235	#aAc	e#	出错	

判断SLR

只有当文法是LR(0)文法时，才能构造出LR(0)分析表。

一个项目集中，存在移进-归约冲突/归约-归约冲突，会导致LR(0)分析表具有多重入口，分析动作不唯一。

对LR(0)进行改造，**只在发生冲突时**，从左到右向前看一个符号，从而确定应该采用什么动作，得到SLR(1)方法。

只要在一个项目集中，**规约成的非终结符（产生式左侧）的FOLLOW集**与**移进符号集**不相交，就可以区分开，从而避免冲突。