第5章 自底向上语法分析

- 5.1 自底向上语法分析的关键问题
- 5.2 简单优先分析
- 5.3 LR分析

5.1 自底向上语法分析的关键问题

- 5.1.1 自底向上语法分析的关键问题
- 5.1.2 相关概念

5.1.1 自底向上语法分析的关键问题

- 自底向上分析,也称移入-归约分析,其思想是对输入符号串自左向右进行扫描，并将输入符逐个移入一个后进先出栈中，边移入边分析，一旦栈顶符号串形成某个句型的句柄或可归约串(该句柄或可归约串对应某产生式的右部)时，就用该产生式的左部非终极符代替相应右部的文法符号串，这称为一步归约。重复这一过程直到归约到栈中只剩文法的开始符号时则为分析成功，即确认输入串是文法的句子.关键问题是:
  - 如何找到句柄或可归约串?
  - 按照哪条产生式进行归约?
- 包含以下方法:
  - 简单优先法；算符优先法；LR类方法

5.1.2 相关概念

- 令G是一个文法, S是文法的开始符,αβγ是文法G的一个句型,
  - 若有$S \Rightarrow^{*} \alpha A\gamma$且$A \Rightarrow^{+} \beta$, 则称β为句型αβγ的*短语.*
  - 特别,如有$A \Rightarrow \beta$, 则称β为句型αβγ的*简单短语.*
  - 一个句型的最左简单短语称为该句型的*句柄.*
- 如果一个CFG无二义性, 则它的任意一个句型都有唯一的句柄.

5.1.2 相关概念

- 例5.1: 设有文法G的产生式如下，分别求句型AbBd和abBd的短语、简单短语、句柄.

- $S \rightarrow ABd$
- $A \rightarrow a$
- $B \rightarrow d$
- $B \rightarrow c$
- $B \rightarrow bB$

(1). 对于句型AbBd, 存在下面的推导序列:
$S \Rightarrow ABd \Rightarrow AbBd$
短语：AbBd，bB
简单短语：bB
句柄: bB

(2). 对于句型abBd, 存在下面的推导序列:
$S \Rightarrow ABd \Rightarrow AbBd \Rightarrow abBd$
短语：abBd，bB, a
简单短语：bB,a
句柄: a

5.1.2 相关概念

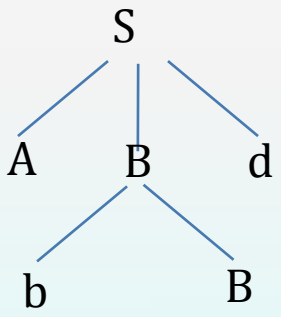
- 利用语法分析树寻找句型的短语、简单短语、句柄:

  子树的叶子结点上标记的符号联接而成的符号串构成短语，简单子树的叶子结点上标记的符号联接而成的符号串构成简单短语，最左简单子树的叶子结点上标记的符号联接而成的符号串构成句柄.

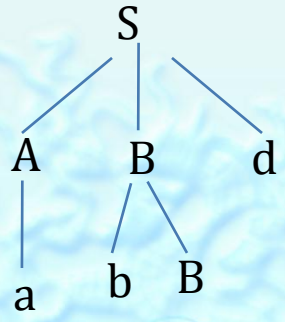
- 例5.2: 设有文法G的产生式如下，分别求句型AbBd和abBd的短语、简单短语、句柄.

- $S \rightarrow ABd$
- $A \rightarrow a$
- $B \rightarrow d$
- $B \rightarrow c$
- $B \rightarrow bB$

(1). 对于句型AbBd,
短语：AbBd，bB
简单短语：bB
句柄: bB

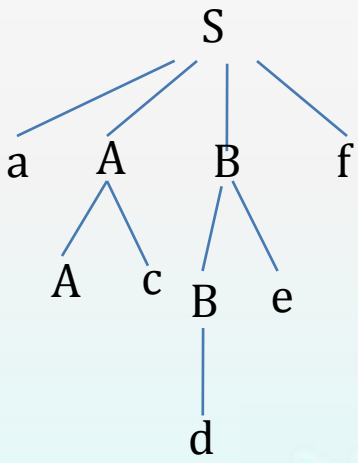


(2). 对于句型abBd,
短语：abBd，bB, a
简单短语：bB,a
句柄: a



5.1.2 相关概念

- 例5.3 设有文法G的产生式如下,画出句型aAcdef的语法分析树,并给出该句型的所有短语、简单短语、句柄.
- $S \rightarrow aABf$
- $A \rightarrow Ac \mid b$
- $B \rightarrow Be \mid d$
- 语法分析树如右图所示:
- 短语: aAcdef, Ac, d,de
- 简单短语: Ac, d
- 句柄: Ac
- 思考： aABf, Be是否是短语？No!



5.1.2 相关概念

- 推导: 对句型中的非终极符用产生式右部替换
- 归约: 推导的逆过程
- 规范推导: 一个句型的最右推导称为规范推导
- 规范归约: 规范推导的逆过程，称为规范归约(最左归约)
- 规范句型: 从开始符通过规范推导得到的句型
- 规范归约过程产生的句型仍是规范句型
- 规范归约过程也是对规范句型的最左简单短语(句柄)进行归约的过程

5.1.2 相关概念

- 前缀: 从一个符号串尾部去掉0个或多个符号后剩余的符号串
  - 符号串abc的前缀： abc,ab,a,ε
  - 符号串AbBd的前缀: AbBd, AbB, Ab, A, ε
- 规范前缀:一个规范句型的前缀，如果该前缀后面的符号串不包含非终极符
  - 规范句型AbBd的规范前缀: AbBd, AbB
- 规范活前缀: 一个规范句型的规范前缀不包含句柄或包含句柄但句柄在规范前缀的最右端
  - 规范句型AbBd的规范活前缀: AbB
  - 规范句型abcd的规范活前缀: ε,a
- 归约规范活前缀:包含句柄的规范活前缀
  - 规范句型AbBd的归约规范活前缀: AbB
  - 规范句型abcd的归约规范活前缀: a
- 规范活前缀决定了自底向上分析时应进行的动作:
- 若规范活前缀不包含句柄，则进行移入；若包含句柄，则进行归约.

(1) $S \rightarrow ABd$
(2) $A \rightarrow a$
(3) $B \rightarrow d$
(4) $B \rightarrow c$
(5) $B \rightarrow bB$

5.1.2 相关概念

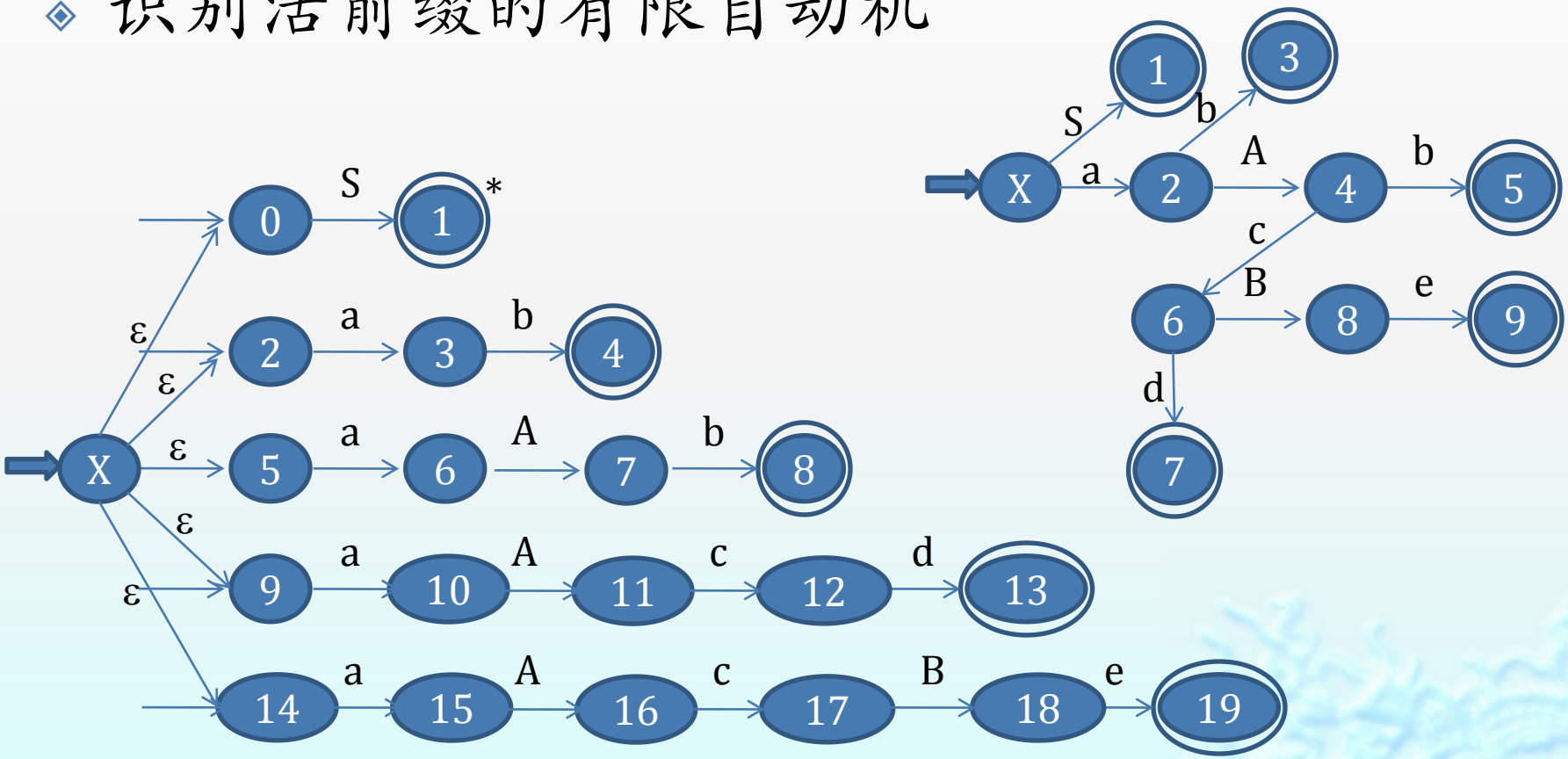
- 给定一个上下文无关文法，如何确定其所有的活前缀和归约活前缀呢?
- 派生定理: (1)文法的开始符是归约活前缀；(2) 若αAβ是归约活前缀, 且有产生式A→π, 则απ也是归约活前缀(其中α，β，π∈V*)
- 例5.4 有文法
- S → aAcBe
- A → b
- A → Ab
- B → d
- 则文法的归约活前缀有：S、aAcBe、ab、aAb、aAcd

5.1.2 相关概念

- 例5.5 有文法产生式如下, 则文法的归约活前缀有哪些?
- S→ aAc
- A → ABb
- A → a
- B → bB
- B → b
- 文法的归约活前缀有: S, aAc, aABb, aa, aAbB, aAb,aAbbB, aAbb, aAbbbB, aAbbb……

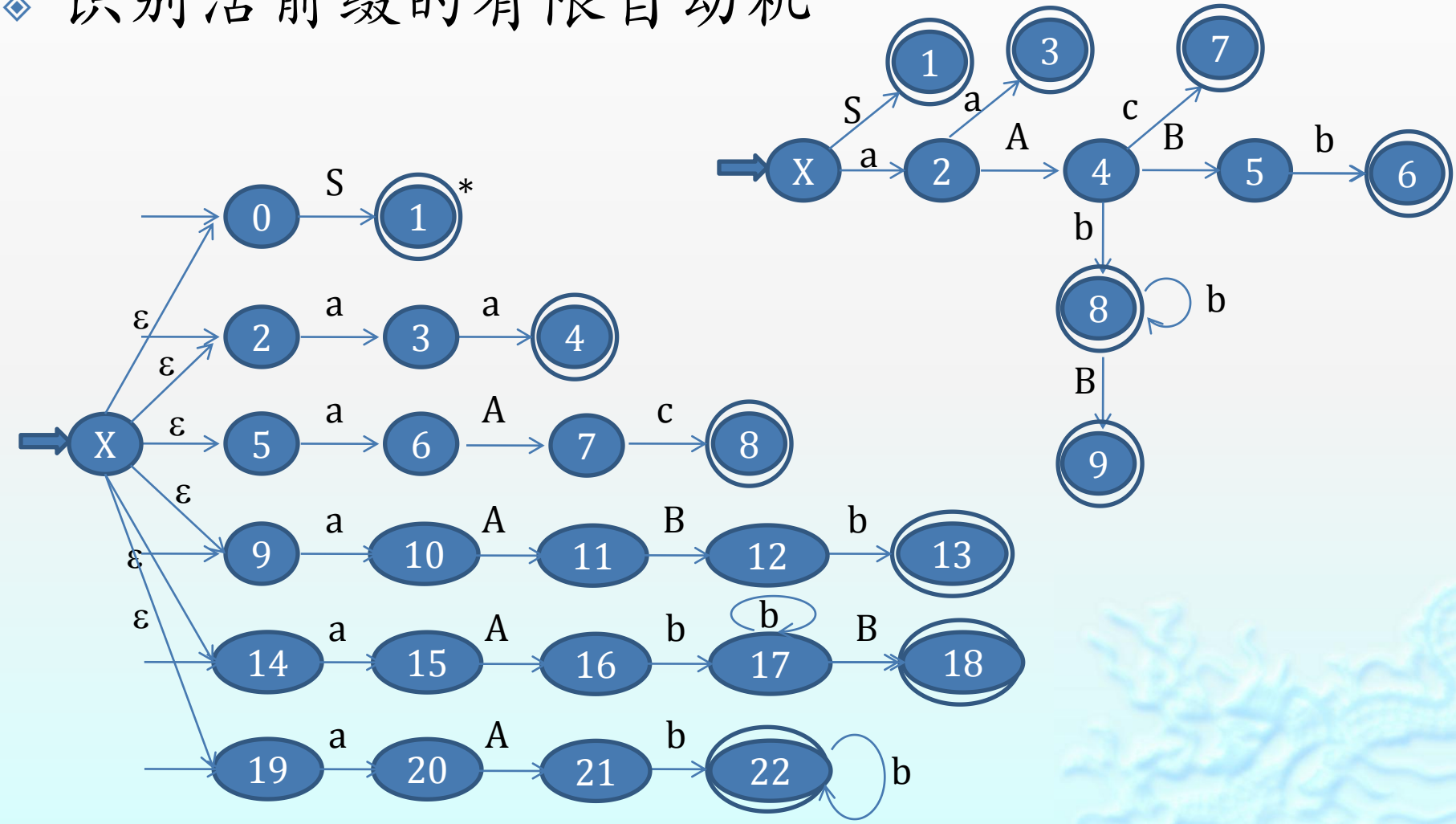
5.1.2 相关概念

- 识别活前缀的有限自动机

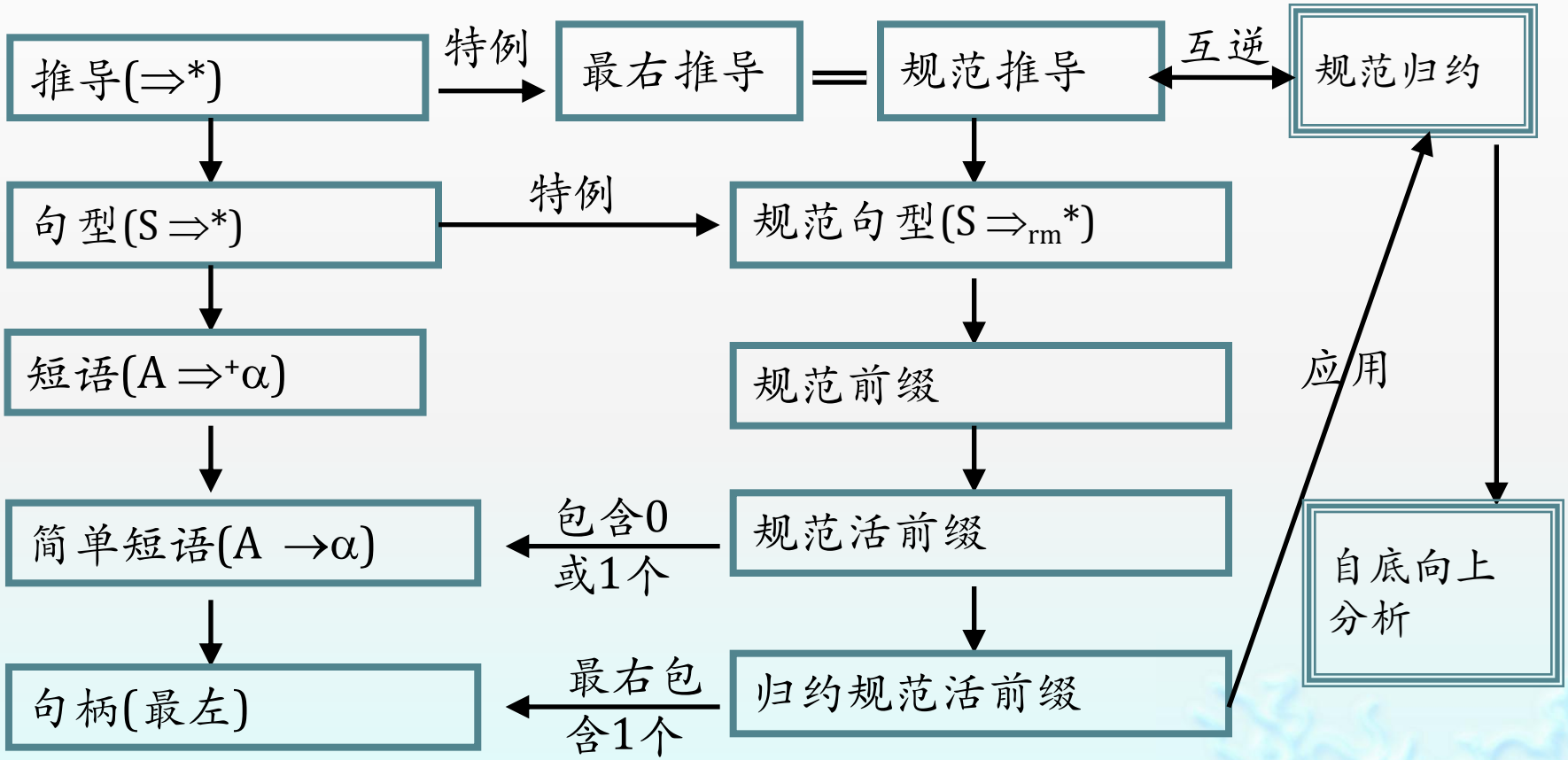


5.1.2 相关概念

- 识别活前缀的有限自动机



5.1.2 相关概念


推导(⇒*)
特例
最右推导
=
规范推导
互逆
规范归约
句型(S⇒*)
特例
规范句型(S⇒rm*)
短语(A⇒+α)
规范前缀
应用
简单短语(A →α)
包含0
或1个
规范活前缀
自底向上
分析
句柄(最左)
最右包
含1个
归约规范活前缀

5.2 简单优先分析

- 简单优先分析的基本思想是对一个文法按一定原则求出该文法所有符号即包括终结符和非终结符之间的优先关系，按照这种关系确定归约过程中的句柄，它的归约过程是一种规范归约。
- 文法G中任意两个符号(N,T)优先关系可能有≒、⋖和⋗三种，其定义如下：
  - $P \rightarrow \ldots XY \ldots$ ，则$X \doteq Y$: 表示X和Y的优先关系相等
  - $P \rightarrow \ldots XA \ldots$，$A \Rightarrow^{+} Y \ldots$,则 $X \lessdot Y$:表示X的优先性比Y的优先性小
  - $P \rightarrow \ldots AB \ldots$，$A \Rightarrow^{+} \ldots X$, $B \Rightarrow^{+} Y \ldots$,则 $X \gtrdot Y$:表示X的优先性比Y的优先性大

5.2 简单优先分析

- 一个文法中存在的全部简单优先关系可以用矩阵来表示，称作简单优先关系矩阵
- 例5.6 若有文法G,构造该文法的优先关系矩阵.
- $S \rightarrow bAb$
- $A \rightarrow (B \mid a$
- $B \rightarrow Aa)$

	S	A	B	a	b	(	)	#
S								$\doteq$
A				$\doteq$	$\doteq$			
B				$\doteq$	$\doteq$		$\doteq$	
a				$\doteq$	$\doteq$		$\doteq$	
b		$\doteq$		$\lessdot$		$\lessdot$		$\doteq$
(		$\lessdot$	$\doteq$	$\lessdot$		$\lessdot$		
)				$\doteq$	$\doteq$			
#	$\lessdot$				$\lessdot$			

#表示句子括号#α#,
#≤所有符号,所有符号≥#, 当然只对与#相邻的符号有意义.

5.2 简单优先分析

- 简单优先文法: 若一个文法是简单优先文法必须满足如下条件:
  - (1) 在文法符号集V中,任意两个符号之间最多只有一种优先关系成立.
  - (2)在文法中任意两个产生式没有相同右部.
- 简单优先分析步骤: (已建好优先矩阵,符号栈S)
  - 将输入符号 串#$a_1a_2...a_n$#依次逐个存入符号栈S,直到栈顶符号a_i的优先性>下一个输入符号为止.
  - 栈顶当前符号a_i为句柄尾,由此向左在栈中找句柄的头符号a_k,即找到$a_{k-1}<\cdot a_k$为止.
  - 由句柄$a_k...a_i$在文法的产生式中查找右部为$a_k...a_i$的产生式,若找到则用相应左部代替句柄,若找不到则为出错,这时可断定输入串不是该文法的句子.
- 简单优先分析准确、规范，但分析效率低，实际使用价值不大.

5.2 简单优先分析

- 例: 符号串b(aa)b是否是例5.6中文法的句子?

S栈	输入符号串	优先关系	分析动作
#	b(aa)b#	# ⋖ b	移入
# ⋖ b	(aa)b#	b ⋖ (	移入
# ⋖ b ⋖ (	aa)b#	(⋖ a	移入
# ⋖ b ⋖ (⋖ a	a)b#	a ⋗ a	归约 a，产生式A→a
# ⋖ b ⋖ (⋖ A	a)b#	A ≐ a	移入
# ⋖ b ⋖ (⋖ A ≐ a	)b#	a ≐)	移入
# ⋖ b ⋖ (⋖ A ≐ a ≐)	b#	) ⋗ b	归约Aa)，产生式B →Aa)
# ⋖ b ⋖ (≐ B	b#	B ⋗ b	归约(B，产生式A →(B
# ⋖ b ≐ A	b#	A ≐ b	移入
# ⋖ b ≐ A ≐ b	#	b ⋗ #	归约bAb，产生式S →bAb
# ⋖S	#	S ⋗ #	成功!

5.3 LR分析

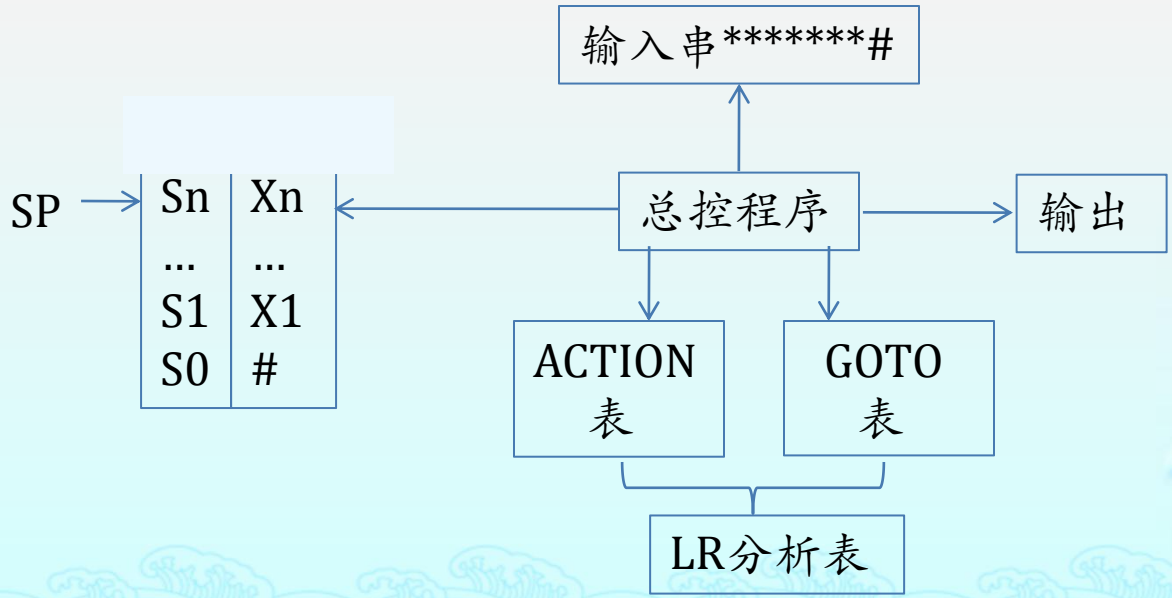
- 自底向上分析法的关键问题是在分析过程中如何确定句柄。LR分析法正是给出一种能根据当前栈中符号串(通常以状态表示)和向右顺序查看输入串的k个(k≥0)符号就可惟一地确定分析器的动作是移进还是归约和用哪个产生式进行归约.
- LR分析法的归约过程是规范推导的逆过程, LR分析过程是一种规范归约过程.
- LR分析方法比起自顶向下的LL(k)分析方法和自底向上的优先方法对文法的限制要少得多，对于大多数的无二义性的上下文无关文法描述的语言都可以用相应的LR分析器进行识别. 该方法还具有分析速度快、能准确、即时地指出出错位置的优点.该方法的主要缺点是对于一个实用语言文法的分析器的构造工作量相当大，k越大构造越复杂，实现比较困难.

5.3 LR分析

- 5.3.1 LR分析概述
- 5.3.2 LR(0)分析
- 5.3.3 LR(1)分析
- 5.3.4 LALR(1)分析

5.3.1 LR分析概述

- 一个LR分析器由3个部分组成:
- （1）总控程序:所有LR分析器的总控程序都是相同的
- （2）分析表或分析函数:不同的文法分析表将不同，同一个文法采用的LR分析器不同，分析表也不同. 分析表可分为动作表ACTION和状态转换表GOTO两个部分
- （3）分析栈:包括文法符号栈和相应的状态栈.



ACTION(Si, a) =
移进，归约，
接受acc，报错

5.3.2 LR(0)分析

- LR(0)分析是只根据分析栈或符号栈的内容确定句柄或分析动作的方法
- LR(0)活前缀有限自动机(简称LR0自动机):
  - LR(0)项目:带圆点•的产生式(•可出现在产生式右部的任何位置)
  - 例如产生式S→aAc**对应的LR(0)项目可以有:**
    - **S→ •aAc**
    - **S → a • Ac**
    - **S → a A • c**
    - **S → a Ac •**
  - 特别，对于空产生式S→ε,只有**S → •** 一个LR(0)项目

5.3.2 LR(0)分析

- LR(0)自动机：
- LR(0)项目集IS关于符号X的投影$IS_{(X)}$
  - X是一个符号(终极符或非终极符)；
  - $IS_{(X)} = \{S \rightarrow \alpha X \bullet \beta \mid S \rightarrow \alpha \bullet X\beta \in IS, X \in V_T \cup V_N\}$
  - 即IS(x)只对IS中圆点后面是X的项目起作用，所起的作用就是将圆点从X的前面移到X的后面
- 例：$IS = \{A \rightarrow A\bullet Bb, B \rightarrow a\bullet, B \rightarrow b\bullet B, Z \rightarrow \bullet cB\}$
- $X = B$
- $IS_{(B)} = \{A \rightarrow AB\bullet b, B \rightarrow bB\bullet\}$

5.3.2 LR(0)分析

- LR(0)自动机:
  - LR(0)项目集的闭包Closure(IS):
  - 按照下面的步骤计算:
    - [1]初始, CLOSURE(IS) = IS
    - [2] 对于CLOSURE(IS)中没有处理的LR(0)项目,
      如果该项目的圆点后面是一个非终极符A,
      而且A的全部产生式是{A→α1, ⋯, A→αn}
      则增加如下LR(0)项目到CLOSURE(IS)
      {A→ •α1, ⋯, A→ •αn}
    - [3] 重复[2]直到 CLOSURE(IS)收敛;

5.3.2 LR(0)分析

- LR(0)自动机：
  - 状态转换函数goto按照下面的公式计算:
  - goto(IS, X) = CLOSURE($IS_{(x)}$)
- 例5.7：构造下面文法的LR(0)自动机
- S→ aAc
- A → Abb
- A → Ba
- B → b
- 结论：一个上下文无关文法的LR(0)自动机接受的是该文法的LR(0)归约规范活前缀集

5.3.2 LR(0)分析

- LR(0)自动机的相关概念:
- 移入项目：形如 $A \rightarrow \alpha \bullet a\beta, a \in T, \alpha,\beta \in V^*$
- 待约项目：形如 $A \rightarrow \alpha \bullet B\beta, B \in N, \alpha,\beta \in V^*$
- 归约项目：形如 $A \rightarrow \alpha \bullet , \alpha \in V^*$
- 接受项目：形如 $A \rightarrow \alpha \bullet , \alpha \in V^*$，且$A \rightarrow \alpha$是增广产生式
- 移入状态：包含移入项目的状态
- 归约状态：包含归约项目的状态
- 冲突状态：若同一状态中，既有移入项目又有归约项目，则称该状态存在移入-归约冲突；若同一状态中，同时有多个归约项目，则称该状态存在归约-归约冲突

5.3.2 LR(0)分析

- LR(0)文法：若一个文法的LR(0)自动机不存在冲突状态，则称该文法为LR(0)文法.
  - 当一个文法满足LR(0)文法的条件时才能用LR(0)方法分析.
- LR(0)分析过程:
  - 增广文法
  - 构造LR(0)自动机,判断是否满足LR(0)文法的条件，若满足，则:
  - 构造LR(0)分析表
  - 调用LR分析器的总控程序(驱动程序)对输入串进行分析

5.3.2 LR(0)分析

- LR(0)分析表：是二维表格,行标为LR(0)自动机的状态号,列标为文法符号和"#".分析表由两部分组成，一部分为动作(ACTION)表,一部分是转换(GOTO)表.样例如下：
- 表项内容根据LR(0)自动机填写
-

ACTION				GOTO		
	a	b	#	A	B	C
q_0	*Si*					
q_1		*accept*			1	
q_2				3		
…	*Rp*					2
q_n		*error*				

5.3.2 LR(0)分析

- 例5.8: 判断下面文法是否是LR(0)文法. 若是则构造该文法的LR(0)分析表,并给出符号串abac的分析过程.
- (1) S→ aAc
- (2) A → Abb
- (3) A → Ba
- (4) B → b

5.3.2 LR(0)分析

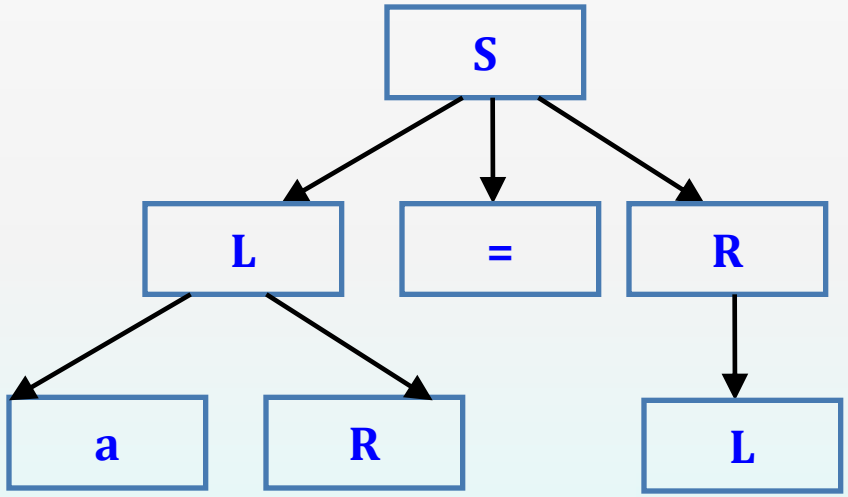
- 例5.9: 判断下面文法是否是LR(0)文法并说明理由.
- S→ rD
- D → D, i
- D → i

5.3.2 LR(0)分析

- 一般的文法在构造LR(0)自动机的时候极易产生冲突，特别是当文法存在空产生式或两个产生式的右部相同的情况下. 因此LR(0)文法对产生式的限制较严格，LR(0)分析不是一种实用的方法.
- LR(0)自动机极易产生冲突的原因是在确定可归约串(归约活前缀)时没有考虑输入串的信息.

5.3.3 LR(1)分析

- 有文法G，其句型aR=L的语法树如下：
- $S \rightarrow L = R$
- $S \rightarrow R$
- $L \rightarrow aR$
- $L \rightarrow b$
- $R \rightarrow L$



- 对非终极符的每个不同出现求其后继符集(Follow)，也叫展望符集,基于此思想的LR分析称为LR(1)分析.

5.3.3 LR(1)分析

- LR(1)自动机
  - 状态是LR(1)项目集
  - LR(1)项目=LR(0)项目+{展望符}
- LR(1)文法
  - LR(1)自动机不存在移入-归约或归约-归约冲突的文法是LR(1)文法
  - LR(1)冲突的判断要考虑展望符，与LR(0)略有不同
- LR(1)分析表
  - 与LR(0)分析表的区别在归约动作上
- LR(1)控制程序=LR(0)控制程序

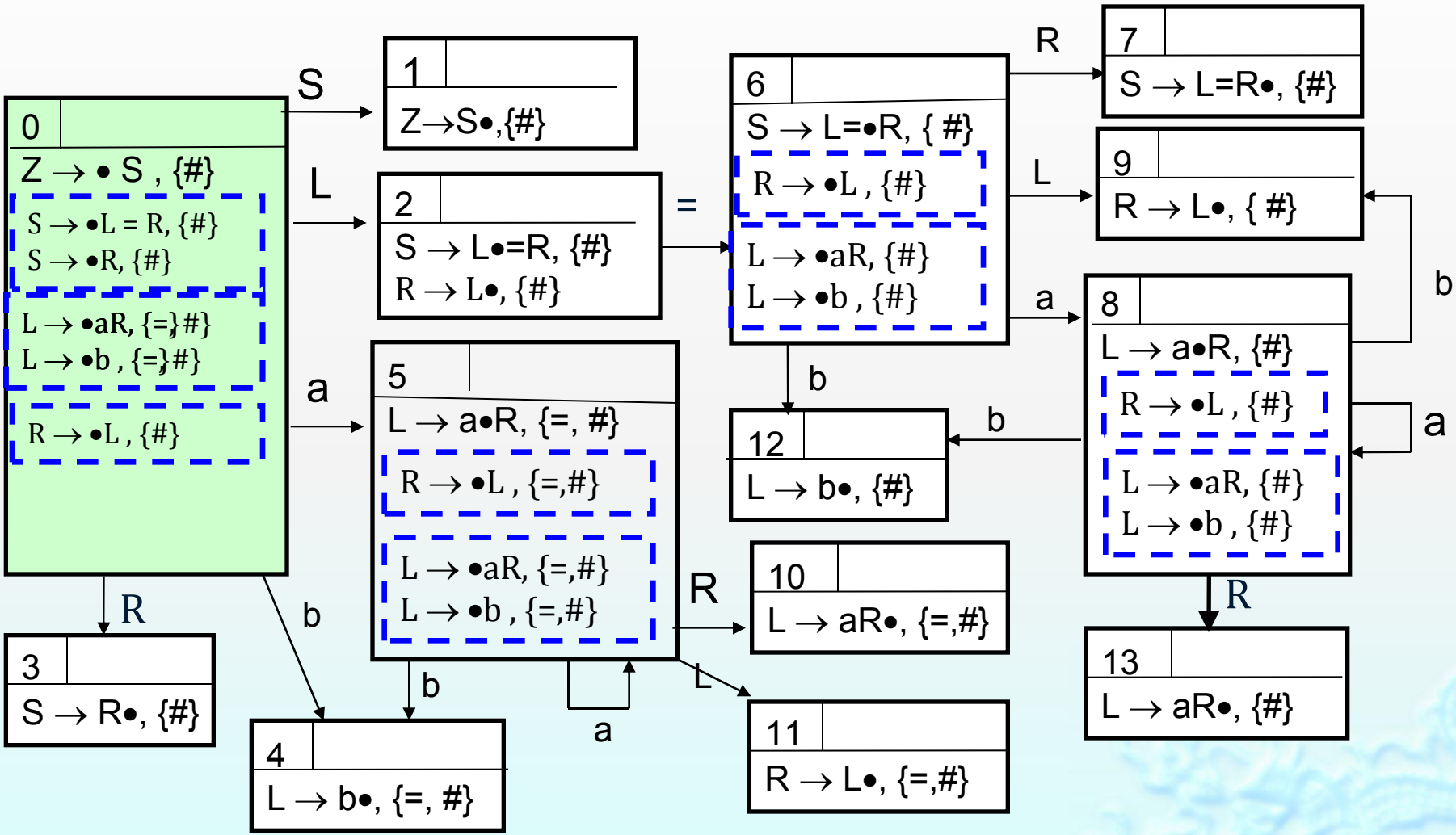
5.3.3 LR(1)分析

- LR(1)自动机
  - 状态是LR(1)项目集
  - LR(1)项目=LR(0)项目+{展望符}
  - 构造过程:
  - 例5.10:
    - $S \rightarrow L = R$
    - $S \rightarrow R$
    - $L \rightarrow aR$
    - $L \rightarrow b$
    - $R \rightarrow L$

增广产生式，并给产生式加编号:

$Z \rightarrow S$	[1]
$S \rightarrow L = R$	[2]
$S \rightarrow R$	[3]
$L \rightarrow aR$	[4]
$L \rightarrow b$	[5]
$R \rightarrow L$	[6]

5.3.3 LR(1)分析


0
Z → • S , {#}
S → •L = R, {#}
S → •R, {#}
L → •aR, {=,#}
L → •b , {=,#}
R → •L , {#}
1
Z→S•,{#}
2
S → L•=R, {#}
R → L•, {#}
3
S → R•, {#}
4
L → b•, {=, #}
5
L → a•R, {=, #}
R → •L , {=,#}
L → •aR, {=,#}
L → •b , {=,#}
6
S → L=•R, { #}
R → •L , {#}
L → •aR, {#}
L → •b , {#}
7
S → L=R•, {#}
8
L → a•R, {#}
R → •L , {#}
L → •aR, {#}
L → •b , {#}
9
R → L•, { #}
10
L → aR•, {=,#}
11
R → L•, {=,#}
12
L → b•, {#}
13
L → aR•, {#}
S
L
R
a
b
=

LR(1)分析表:ACTION表和GOTO表

	ACTION 表					GOTO 表		
	a	b	=	#		S	L	R
0	S5	S4				1	2	3
1				Accept				
2			S6	R6				
3				R3				
4			R5	R5				
5	S5	S4					11	10
6	S8	S12					9	7
7				R2				
8	S8	S12					9	13
9				R6				
10			R4	R4				
11			R6	R6				
12				R5				
13				R4				

LR(1)分析过程

- 根据LR(1)分析表，符号串b=b#的LR(1)分析过程如下：

状态栈	输入流	分析动作
0	b=b#	S4
04	=b#	R5, GOTO(0,L)=2
02	=b#	S6
026	b#	S12
026(12)	#	R5, GOTO(6, L)=9
0269	#	R6 , GOTO(6, R)=7
0267	#	R2, GOTO(0, S)=1
01	#	Accept

5.3.4 LALR(1)分析

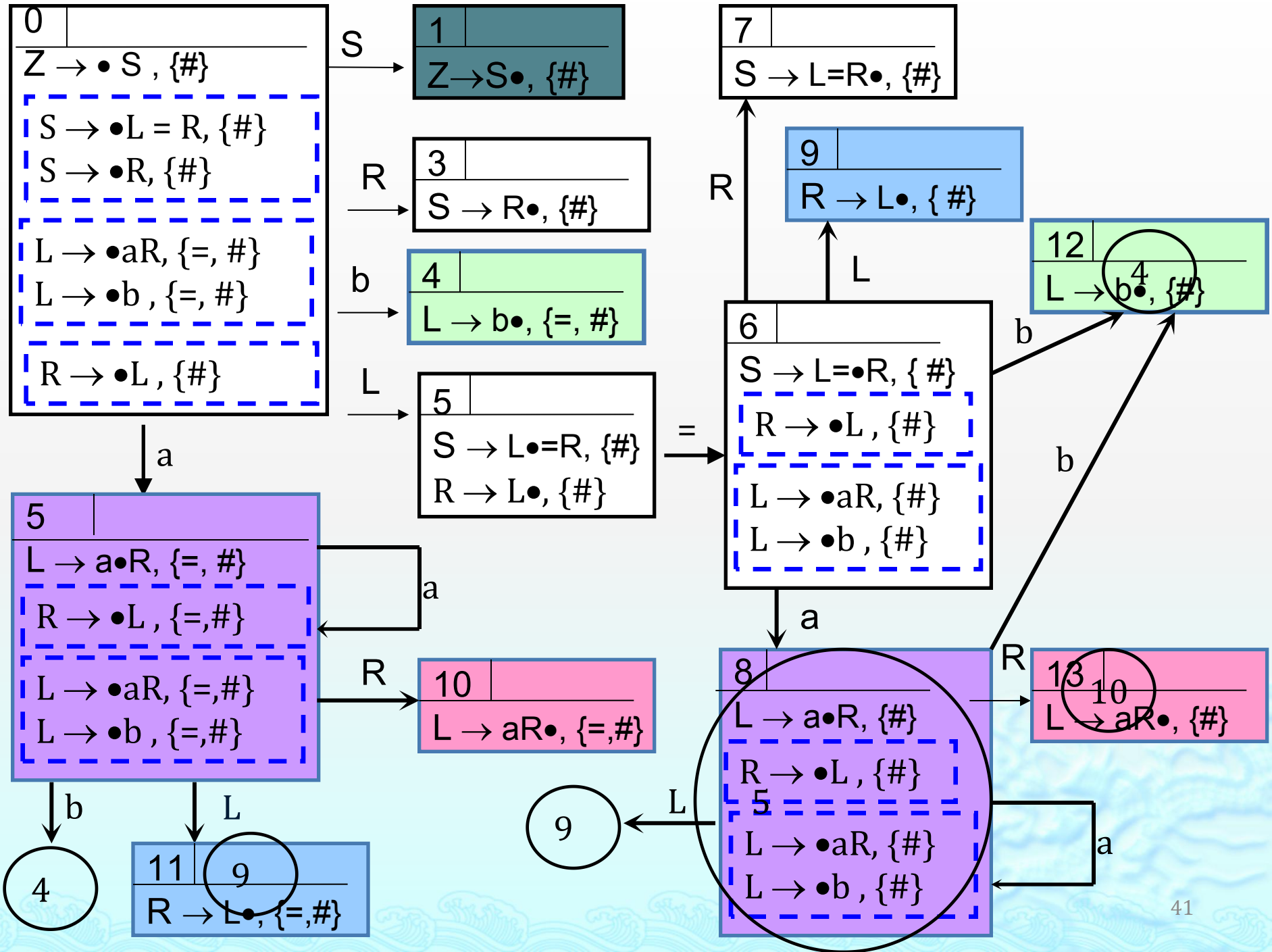
- LR(1)自动机为区分不同位置的非终极符引入了很多的状态,会导致分析效率的下降,因此也不是一种实用的方法.
- 事实上,在LR(1)自动机中,有很多状态的LR(0)项目部分是相同的,只有展望符不同.
- LR(1)项目的心: LR(0)项目
- LR(1)自动机状态的心:状态中所有LR(1)项目的心,即每个状态中所有的LR(0)项目.
- 同心状态:若两个状态的心相同,则称两个状态为同心状态.

5.3.4 LALR(1)分析

- LALR(1)文法
  - 将文法的LR(1)自动机的所有同心状态合并后得到的自动机，称为LALR(1)自动机.
  - 若文法的LALR(1)自动机中没有冲突,则该文法为LALR(1)文法.
  - 合并同心状态时，若有冲突则一定是归约-归约冲突,不可能是移入-归约冲突.
- LALR(1)分析过程
  - 构造LALR(1)自动机
  - 构造LALR(1)分析表(原理与LR(1)分析表相同)
  - LALR(1)驱动程序=LR驱动程序

5.3.4 LALR(1)分析

- 例5.11: 判断例5.10中的文法是否是LALR(1)文法,给出判断理由.
- 解:
  - (1) 首先构造该文法的LR(1)自动机;
  - (2) 该自动机存在四个同心状态，合并同心状态得到LALR(1)自动机如下;
  - (3) 该自动机不存在冲突状态，所以该文法是LALR(1)文法.


0
Z → • S , {#}
S → •L = R, {#}
S → •R, {#}
L → •aR, {=, #}
L → •b , {=, #}
R → •L , {#}
S
1
Z→S•, {#}
R
3
S → R•, {#}
b
4
L → b•, {=, #}
L
5
S → L•=R, {#}
R → L•, {#}
=
6
S → L=•R, { #}
R → •L , {#}
L → •aR, {#}
L → •b , {#}
7
S → L=R•, {#}
R
9
R → L•, { #}
L
12
4
L → b•, {#}
b
a
5
L → a•R, {=, #}
R → •L , {=,#}
L → •aR, {=,#}
L → •b , {=,#}
a
R
10
L → aR•, {=,#}
b
L
4
11
9
R → L•, {=,#}
a
8
L → a•R, {#}
R → •L , {#}
5
L → •aR, {#}
L → •b , {#}
R
13
10
L → aR•, {#}
L
9
a
b

5.3.4 LALR(1)分析

- 例5.12: 判断下面文法是否是LALR(1)文法, 若是，构造该文法的LALR(1)分析表, 并给出符号串(a#和(a,a#的LALR(1)分析过程.
- $S \rightarrow a \mid \wedge \mid (T)$
- $T \rightarrow T , S \mid S$

5.3.4 LALR(1)分析

- LR(0)、LR(1)、LALR(1)的比较
- 状态数: LR(0) = LALR(1) ≤ LR(1)
- 向前看输入符:
  - LR(0)不看
  - LR(1)、LALR(1)看1个
- 展望符的确定:
  - LR(0)不计算展望符
  - LR(1)计算非终极符在不同位置的follow集
  - LALR(1)合并同心项的展望符集
- 分析能力:
  - LR(0) ⊂ LALR(1) ⊂ LR(1)

作 业

- 1.判断下面文法是否是LR(0)文法，LR(1)文法，LALR(1)文法.若是构造相应分析表,并对输入串ab#给出分析过程.
  - $A \rightarrow aAd \mid aAb \mid \varepsilon$
- 2.若有定义二进制数的文法如下:
  - $S \rightarrow L.L \mid L$
  - $L \rightarrow LB \mid B$
  - $B \rightarrow 0 \mid 1$

  (1)试为该文法构造LR分析表,并说明属哪类LR分析表.

  (2)给出输入串101.110的分析过程.

作 业

- 3. 证明下面文法是LR(1)但不是LALR(1).
  - S → Aa | bAe | Be | bBa
  - A → d
  - B → d
- 4. 设文法为:
  - S → AS | ε
  - A → aA | b
- (1) 证明该文法是LR(1)文法;
- (2) 构造它的LR(1)分析表;
- (3) 给出符号串abab#的分析过程.