
Lecture 6: Bottom-up Analysis

Xiaoyuan Xie 谢晓园

xxie@whu.edu.cn

计算机学院E301



6.1 Motivation Example

6.1 Motivation example

■ 自顶向下分析回顾：

P:

(1) $Z \rightarrow aBeA$

(2) $A \rightarrow Bc$

(3) $B \rightarrow d$

(4) $B \rightarrow bB$

(5) $B \rightarrow \epsilon$

a	b	e	c
---	---	---	---

读头 从左向右移动读头，读入字符串

输入 (读头所在)	栈内 符号	分析	执行推导	匹配
abec	Z	Z的哪一个产生式以a开头? -(1)	aBeA	a
bec	BeA	B的哪一个产生式以b开头? -(4)	bBeA	b
ec	BeA	B的哪一个产生式能够以e开头? -(5)	ϵ eA	e
c	A	A的哪一个产生式能够以c开头? -(2) (5)		

6.1 Motivation example

■ 自顶向下分析回顾(续)：

P:

(1) $Z \rightarrow aBeA$

(2) $A \rightarrow Bc$

(3) $B \rightarrow d$

(4) $B \rightarrow bB$

(5) $B \rightarrow \epsilon$

a	b	e	c
---	---	---	---

读头

从左向右移动读头，读入字符串

输入 (读头所在)	栈内 符号	分析	执行推导	匹配
c	A	A的哪一个产生式 能够以c开头? -(2)	Bc	
c	Bc	A的哪一个产生式 能够以c开头? -(5)	ϵc	c

6.1 Motivation example

- 自底向上分析过程是从所给**输入串**出发，对其进行**最左归约**的过程。

P:

(1) $Z \rightarrow ABb$

(2) $A \rightarrow a$

(3) $A \rightarrow b$

(4) $B \rightarrow b$

(5) $B \rightarrow c$

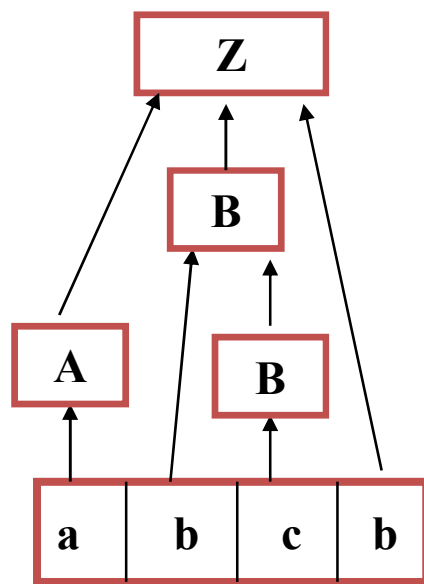
(6) $B \rightarrow bB$

a	b	c	b
---	---	---	---

符号栈	输入	动作
	abcb	移入
a	bcb	归约 (2)
A	bcb	移入
Ab	cb	移入
Abc	b	归约 (5)
AbB	b	归约 (6)
AB	b	移入
ABb		归约 (1)
Z		成功

6.1 Motivation example

- 自底向上归约的过程也是自底向上构建语法树的过程



归约过程

自底向上分析中归约过程的逆过程就是该句子的最右推导；

abcb
---> **A**bc**b**
---> **A**b**B**b
---> **A****B****b**
---> **Z**

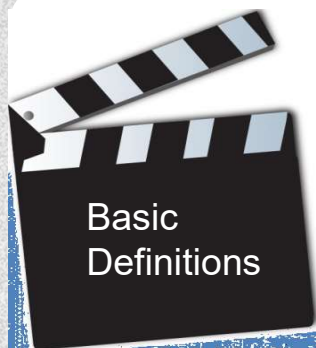
$Z \Rightarrow_{rm} A\underline{B}b$
 $\Rightarrow_{rm} A\underline{b}Bb$
 $\Rightarrow_{rm} A\underline{b}cb$
 $\Rightarrow_{rm} abcb$

6.1 Motivation example

- **从推导的角度看，语法分析的任务是**
 - 接受一个终结符号串作为输入，找出从文法的开始符号推导出这个串的方法
 - 自顶向下分析
- **从规约的角度看，语法分析的任务是**
 - 自底向上分析

6.1 Motivation example

- 自底向上分析主要思想:
 - 从输入串出发;
 - 尽可能地找到可归约子串并将其归约成一个非终极符;
 - 直到归约成文法的开始符或发现语法错误;
- 分析动作:移入(shift),归约(reduce)
- 包含以下方法:
 - 简单优先法; 算符优先法; LR 类的方法;
- 关键问题:
 - 什么时候进行归约, 按照哪条产生式进行归约;



6.2 基本概念

6.2 基本概念

■ 短语

- 一个句型形如 $\alpha\beta\gamma$, 如果存在一个句型 $\alpha A\gamma$, 而且 $A \Rightarrow^+ \beta$, 则称 β 为句型 $\alpha\beta\gamma$ 的短语;
- 例如句型 $AbBb$, 则 $bB, AbBb$ 是它的短语, 因为
 - 存在句型 ABb , $ABb \Rightarrow AbBb$, $\alpha = A$, $\gamma = b$;
 - 存在句型 Z , $Z \Rightarrow ABb \Rightarrow AbBb$, $\alpha = \varepsilon$, $\gamma = \varepsilon$;

(1) $Z \rightarrow ABb$

(2) $A \rightarrow a$

(3) $A \rightarrow b$

(4) $B \rightarrow d$

(5) $B \rightarrow c$

(6) $B \rightarrow bB$

■ 简单短语

- 一个句型形如 $\alpha\beta\gamma$, 如果存在一个句型 $\alpha A\gamma$, 而且 $A \Rightarrow \beta$, 则称 β 为句型 $\alpha\beta\gamma$ 的简单短语;

例如句型 $AbBb$, bB 是它的简单短语, $AbBb$ 不是它的简单短语

6.2 基本概念

- **句柄**: 一个句型的简单短语可能有多个, 称**最左简单短语**为**句柄**(handler);
 - 例如: 句型 $abBb$, 简单短语有两个: a, bB
 - $Z \Rightarrow ABb \Rightarrow aBb \Rightarrow abBb$
 - 最左简单短语 a 是句柄
- **句柄的唯一性**:
 - 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;

6.2 基本概念

■ 例子

P:

- (1) $E \rightarrow T$
- (2) $E \rightarrow E + T$
- (3) $T \rightarrow F$
- (4) $T \rightarrow T * F$
- (5) $F \rightarrow (E)$
- (6) $F \rightarrow i$
- (7) $F \rightarrow n$

句型: $T + (E + T) * i$

句型的一个推导: (该句型没有最右推导)

$E \Rightarrow E + T \Rightarrow E + T * F \Rightarrow E + T * i \Rightarrow$
 $E + F * i \Rightarrow E + (E) * i \Rightarrow E + (E + T) * i$
 $\Rightarrow T + (E + T) * i$

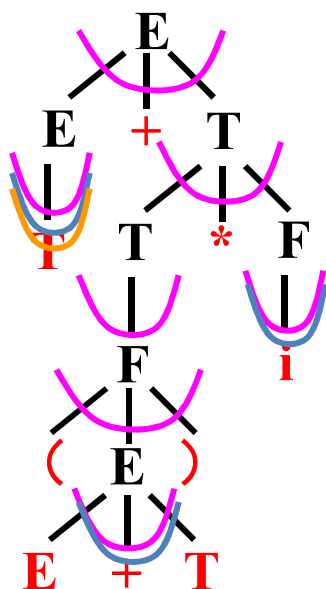
短语: $T + (E + T) * i, T, E + T, i,$
 $(E + T), (E + T) * i$

简单短语: $T, E + T, i$ 句柄: T

通过为所给句型建立语法分析树, 可以很容易地识别出该句型的所有短语、简单短语和句柄。

6.2 基本概念

■ 由语法分析树识别短语、简单短语和句柄



语法分析树的叶子节点构成句型:

$T + (E + T) * i$

每棵子树的叶子节点构成短语:

$T + (E + T) * i$ 、 T 、 $(E + T) * i$ 、 $(E + T)$ 、 $E + T$ 、 i

每棵简单子树(只有一层的子树)的叶子节点构成简单短语: T 、 $E + T$ 、 i

最左简单子树的叶子节点构成句柄: T

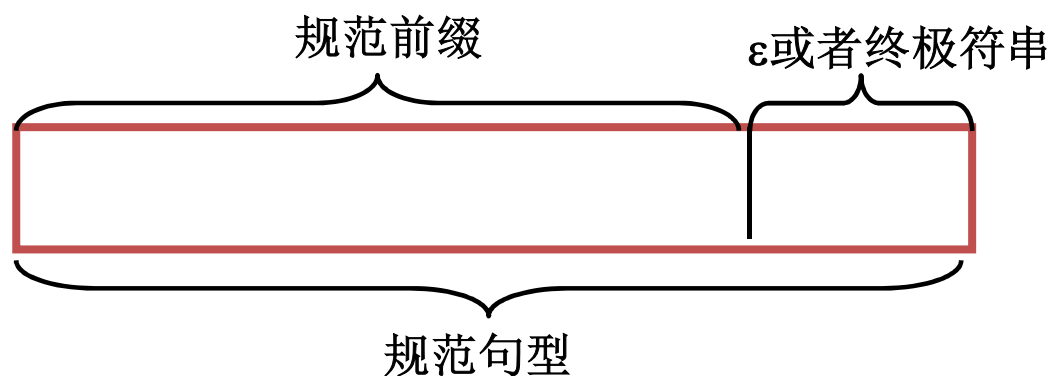
6.2 基本概念

- 自顶向下的语法分析方法中曾介绍过：
- 推导：对句型中的非终极符用产生式右部替换
推导的逆过程称为归约
- 规范推导：一个句型的最右推导称为该句型的规范推导；
规范推导的逆过程称为规范归约(最左归约)
- 规范句型(右句型)：从开始符通过规范推导得到的句型；
规范归约过程中产生的句型仍是规范句型
规范归约的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

6.2 基本概念

- **规范前缀**: 一个规范句型的一个前缀称为**规范前缀**, 如果该前缀后面的符号串不包含非终极符;

- $Z \Rightarrow ABb$: 规范前缀为 AB, ABb
- $Z \Rightarrow^+ Acb$: 规范前缀为 A, Ac, Acb



6.2 基本概念

- **规范活前缀**:满足如下条件之一的规范前缀称为规范活前缀:
 - 该规范前缀不包含简单短语; 或者
 - 该规范前缀只包含一个简单短语,而且是在该规范前缀的最后(这个简单短语就是句柄);
- 换句话说, 规范活前缀不含句柄之后的任何符号

- (1) $Z \rightarrow ABb$
- (2) $A \rightarrow a$
- (3) $A \rightarrow b$
- (4) $B \rightarrow d$
- (5) $B \rightarrow c$
- (6) $B \rightarrow bB$

$Z \Rightarrow ABb$

规范前缀为 AB, ABb

规范活前缀: AB (不包含简单短语)

ABb (包含一个简单短语且在最后)

6.2 基本概念

■ 规范活前缀例2：

(1) $Z \rightarrow ABb$

(2) $A \rightarrow a$

(3) $A \rightarrow b$

(4) $B \rightarrow d$

(5) $B \rightarrow c$

(6) $B \rightarrow bB$

$Z \Rightarrow^+ abcb$

规范前缀为 $a, ab, abc, abcb$

规范活前缀: a (包含一个简单短语且在最后)

$ab, abc, abcd$ 都不是规范活前缀

6.2 基本概念

■ 活前缀有两种类型

- **非归态(移入)活前缀**：规范活前缀不包含简单短语, 而且是在该规范活前缀的最后句柄尚未形成, 需要继续移进若干符号后才能形成句柄
- **归态活前缀**：只包含一个简单短语, 尾部正好是句柄之尾, 此时可以进行规约。规约后又成为另一句型的活前缀

$Z \Rightarrow ABb$ 规范前缀为 AB, ABb

规范活前缀: AB (不包含简单短语) --- 移入型规范活前缀

ABb (包含一个简单短语) --- 归约规范活前缀



6.3 LR分析法概述

6.3 LR分析法概述

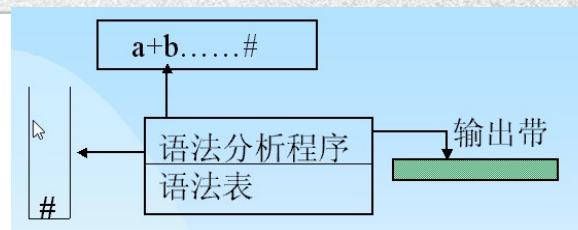
■ LR 方法

■ 主要思想

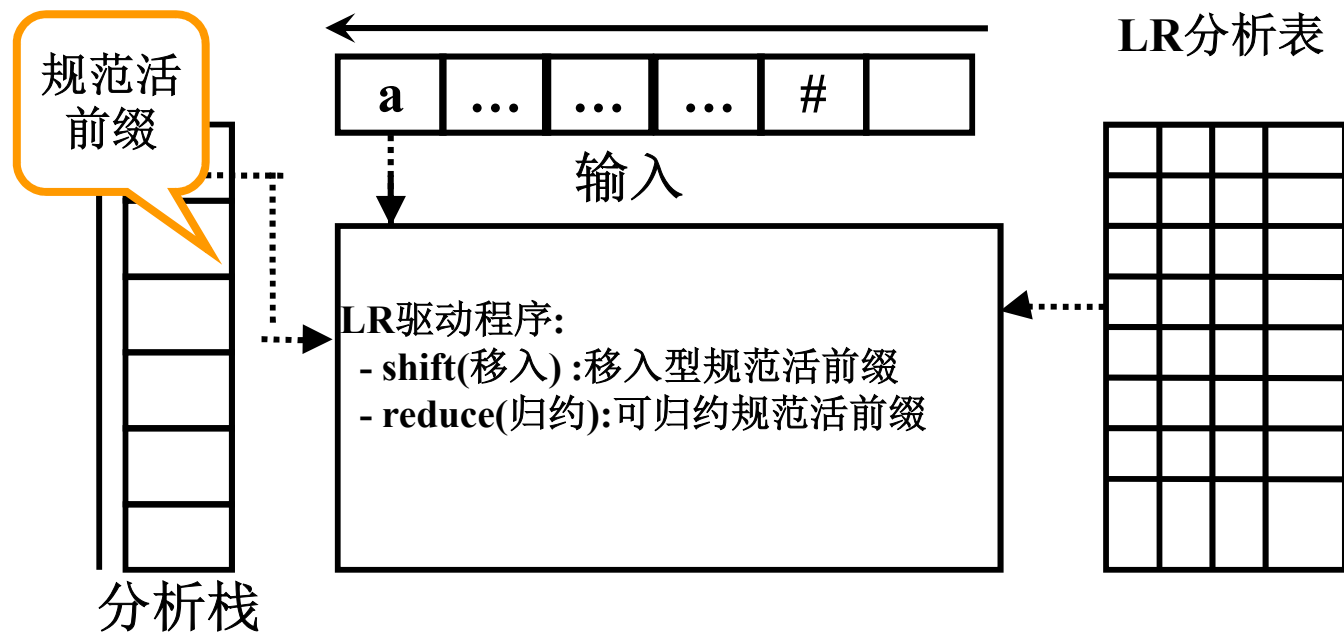
- **L**表示从左至右读入输入串;
- **R**表示构造一个最右推导的逆过程，即每次找到句柄 π (归约规范活前缀 $\alpha\pi$)来进行归约;

■ 工作方式：移进-规约

- 自左至右将输入串符号一个个移进栈，移进过程中不断查看栈顶符号，一旦形成了某个句型的句柄，就将此句柄用相应产生式左部替换（规约）
- 规约后若再形成句柄，则继续替换（规约）；直到栈顶不再形成句柄为止。然后继续移进符号，重复上面过程，直至栈顶只剩下文法的开始符号，输入串读完为止 --- 句子识别成功



6.3 LR分析法概述



6.3 LR分析法概述

■ LR 文法：

- 某一CFG能够构造一张分析表，使得表中每一元素至多只有一种明确动作，则称该文法为LR文法
- 注：并非所有CFG都是LR文法，但对于多数程序设计语言，一般都可以用LR文法描述；和LL(1)相比，LR分析适应的文法范围要广一些

■ 不同的 LR 方法

- 不同的LR方法对CFG的要求不一样，能够分析的CFG多少也不一样， $LR(0) \subseteq SLR(1) \subseteq LALR(1) \subseteq LR(1)$

6.3 LR分析法概述

■ LR 关键问题

- 如何判定规范活前缀?
- 如何判定移入型规范活前缀?
- 如何判定归约规范活前缀以及用哪一个产生式归约?
- 如何判定分析结束?
- **构造一个判定归约规范活前缀的自动机 -- LR自动机**
- **由LR自动机构造LR分析表指导LR分析**



6.4 LR(0) 分析

6.4 LR(0) 分析法

■ 构造LR(0)自动机识别活前缀

- LR(0) 项目: 为每个产生式添加圆点, 圆点只能出现在产生式的右部符号串的任意位置
 - 产生式 $S \rightarrow aAc$, 有4个LR(0)项目:
 - $S \rightarrow \bullet aAc$
 - $S \rightarrow a \bullet Ac$
 - $S \rightarrow aA \bullet c$
 - $S \rightarrow aAc \bullet$
 - 特别地, 空产生式: $S \rightarrow \varepsilon$, 对应LR(0)项目 $S \rightarrow \bullet$
 - 右部长度为n的产生式有n+1个项目;

6.4 LR(0) 分析法

■ 构造LR(0)自动机识别活前缀

■ 圆点理解为栈内栈外分界线

■ 例如： $S \rightarrow aAb, A \rightarrow XYZ$

找句柄 \rightarrow 找归态活前缀,

$aXYZb$ 无法直接看出句柄，但是如果加上dot, 就能立即发现句柄

例如，产生式 $A \rightarrow XYZ$ 对应四个项目

- $A \rightarrow \bullet XYZ$ 预期要归约的句柄是XYZ，但都未进栈
- $A \rightarrow X \bullet YZ$ 预期要归约的句柄是XYZ，仅X进栈
- $A \rightarrow XY \bullet Z$ 预期要归约的句柄是XYZ，仅XY进栈
- $A \rightarrow XYZ \bullet$ 已处于归态活前缀，XYZ可进行归约，这个项目是归约项目。

每个产生式都能获得 $n+1$ 个项目
我们现在需要将所有项目串起来，形成一个完整的，可以识别活前缀的LR(0)自动机

6.4 LR(0) 分析法

■ 构造LR(0)自动机-Naïve方法

- 首先将文法进行拓广，增加产生式 $S' \rightarrow S$ ，令 $S' \rightarrow \bullet S$ 作为初态项
- 为什么要拓广：保证文法开始符号不出现在任何产生式右部
 - CFG可能会导致 S 出现在右部，但是为了避免混淆，保证开始符号不出现在右部 → 每当识别出 S ，就确定完成句子识别
 - 例如 $S \rightarrow aSb$, $S \rightarrow c$ ，给定 abc ， c 可以规约为 S ，结束分析？

6.4 LR(0) 分析法

■ 构造LR(0)自动机-Naïve方法

■ 接着由项目集构造NFA

如果 X_i 是 V_N ，那么从 i 项目转到 j 项目，必须要识别一整个由 X_i 定义的范畴

(3) 设项目 i 为 $X \rightarrow X_1 \dots X_{i-1} \bullet X_i \dots X_n$ ，项目 j 为 $X \rightarrow X_1 \dots X_i \bullet X_{i+1} \dots X_n$ ，则从项目 i 画一弧线射向 j ，标记为 X_i ，若 X_i 是终结符则项目 i 称为移进项目， X_i 是非终结符则称项目 i 为待约项目；

(4) 若项目 i 为 $X \rightarrow \alpha \bullet A \beta$ ，其中 A 是非终结符，则从 i 项目画 ϵ 弧射向所有 $A \rightarrow \bullet \gamma$ 的项目， $\gamma \in V^*$

■ 圆点在最右端的称为终态项目或规约项目，特别地， $S' \rightarrow S \bullet$ 称为接受项目

6.4 LR(0) 分析法

■ 构造LR(0)自动机-Naïve方法

- 构造出的NFA是包含 ϵ 串的NFA，使用子集法使其确定化，成为一个以项目集为状态的DFA ---LR分析的基础
 - DFA每个状态是一个项目集，称为LR(0)项目集，整个状态集称为LR(0)项目集规范簇
 - DFA的一个状态对应的项目集内，每个项目是等价的（从期待规约的角度看是相同的）
- 有一个唯一的初态和唯一的接受态，但有若干规约态，表示有若干种活前缀的识别状态
- 状态反映了识别句柄的情况，即句柄多大部分已进栈 (历史情况已知)

6.4 LR(0) 分析法

■ 构造LR(0)自动机-Naïve方法

■ 示例

例如：有一已拓广的文法G6.1构造识别活前缀的NFA

$S' \rightarrow E$ $E \rightarrow aA|bB$ $A \rightarrow cA|d$ $B \rightarrow cB|d$

解：1、这个文法的项目有：

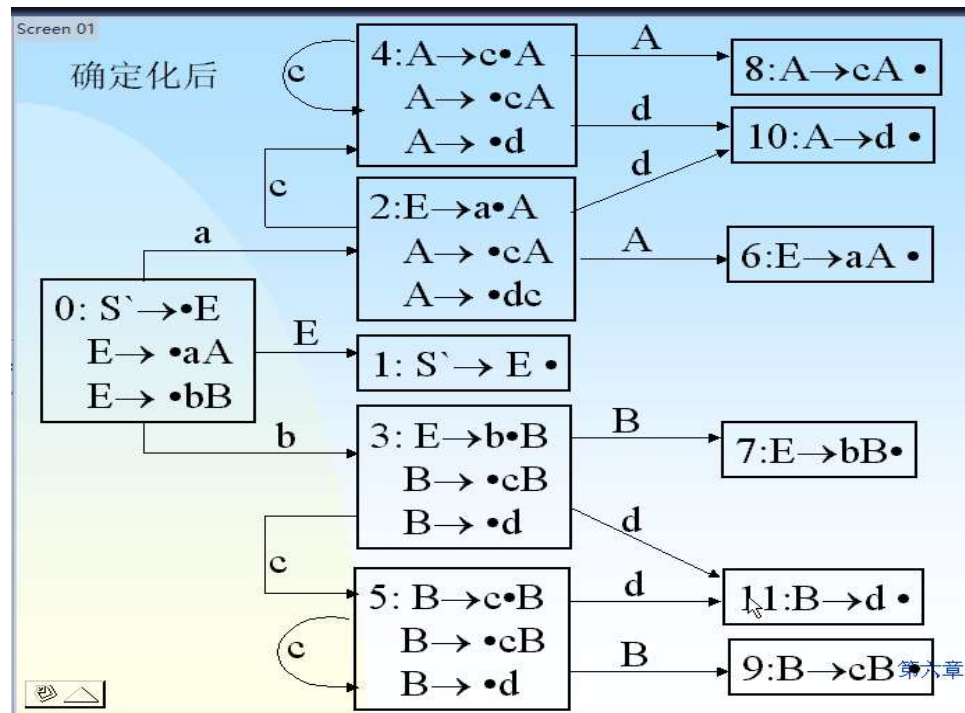
- | | | |
|--------------------------------|--------------------------------|---------------------------------|
| 1. $S' \rightarrow \bullet E$ | 2. $S' \rightarrow E \bullet$ | 3. $E \rightarrow \bullet aA$ |
| 4. $E \rightarrow a \bullet A$ | 5. $E \rightarrow aA \bullet$ | 6. $A \rightarrow \bullet cA$ |
| 7. $A \rightarrow c \bullet A$ | 8. $A \rightarrow cA \bullet$ | 9. $A \rightarrow \bullet d$ |
| 10. $A \rightarrow d \bullet$ | 11. $E \rightarrow \bullet bB$ | 12. $E \rightarrow b \bullet B$ |
| 13. $E \rightarrow bB \bullet$ | 14. $B \rightarrow \bullet cB$ | 15. $B \rightarrow c \bullet B$ |
| 16. $B \rightarrow cB \bullet$ | 17. $B \rightarrow \bullet d$ | 18. $B \rightarrow d \bullet$ |

6.4 LR(0) 分析法

■ 构造LR(0)自动机-Naïve方法

太复杂，不用！

■ 示例



6.4 LR(0) 分析法

■ 构造LR(0)自动机-方法2：

- IS 是一个LR(0)项目的集合;
- X 是一个符号(终极符或非终极符);
- $IS_{(X)}$ 表示项目集IS关于符号X的投影: $IS_{(X)} = \{S \rightarrow \alpha X \bullet \beta \mid S \rightarrow \alpha \bullet X \beta \in IS, X \in V_T \cup V_N\}$ --- $IS_{(X)}$ 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

$$IS = \{A \rightarrow A \bullet Bb, B \rightarrow a \bullet, B \rightarrow b \bullet B, Z \rightarrow \bullet cB\}$$

$$X = B$$

$$IS_{(B)} = \{A \rightarrow AB \bullet b, B \rightarrow bB \bullet\}$$

6.4 LR(0) 分析法

■ 构造LR(0)自动机-方法2：

- IS 是 LR(0)项目的集合; **CLOSURE(IS)**也是一个LR(0)项目集合, 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 $\{A \rightarrow \alpha_1, \dots, A \rightarrow \alpha_n\}$

则增加如下LR(0)项目到CLOSURE(IS)

$\{A \rightarrow \bullet \alpha_1, \dots, A \rightarrow \bullet \alpha_n\}$

[3] 重复[2]直到 CLOSURE(IS)收敛;

6.4 LR(0) 分析法

■ 构造LR(0)自动机-方法2 :

■ CLOSURE(IS)示例

$V_T = \{a, b, c\}$
 $V_N = \{S, A, B\}$
 $S = S$
P:
{ $S \rightarrow aAc$
 $A \rightarrow ABb$
 $A \rightarrow Ba$
 $B \rightarrow b$
}

$IS = \{S \rightarrow \bullet aAc\}$
 $CLOSURE(IS) = \{S \rightarrow \bullet aAc\}$

$IS = \{S \rightarrow a\bullet Ac\}$
 $CLOSURE(IS)$
 $= \{S \rightarrow a\bullet Ac,$
 $A \rightarrow \bullet ABb, A \rightarrow \bullet Ba,$
 $B \rightarrow \bullet b\}$

6.4 LR(0) 分析法

■ 构造LR(0)自动机-方法2：

- goto函数 (DFA中的弧, 表示状态变迁)
 - IS 是 LR(0)项目集合;
 - X 是一个符号(V_T or V_N 终极符或非终极符);
 - $\text{goto}(IS, X) = \text{CLOSURE}(IS_{(x)})$
- 其含义是对任意项目集IS, 由于IS中有 $A \rightarrow \alpha \bullet X \beta$ 项目, $IS(x)$ 中有 $A \rightarrow \alpha X \bullet \beta$ 项目, 表示识别活前缀又移进了一个符号X

6.4 LR(0) 分析法

■ 构造LR(0)自动机-方法2的完整算法

- CFG $G=\{V_T, V_N, S, P\}$ 的LR(0)自动机
 - $(\Sigma, SS, S0, TS, \Phi)$
 - 是否需要 增广产生式 $Z \rightarrow S$?
 - $\Sigma = V_T \cup V_N \cup \{\#\}$
 - $S0 = \text{CLOSURE}(Z \rightarrow \bullet S)$
 - SS
 - TS
 - Φ
- 构造 LR(0)自动机的过程中逐步生成的

6.4 LR(0) 分析法

■ 构造LR(0)自动机-方法2的完整算法

- [1] 增广产生式 $Z \rightarrow S$
- [2] $\Sigma = V_T \cup V_N \cup \{\#\}$
- [3] $S_0 = \text{CLOSURE}(Z \rightarrow \bullet S)$
- [4] $SS = \{S_0\}$
- [5] 对于SS中的每一个项目IS, 和每个符号 $X \in \Sigma$,
 计算 $IS' = \text{goto}(IS, X)$,
 如果 IS' 不为空, 则建立 $IS \xrightarrow{X} IS'$,
 如果 IS' 不为空且 IS' 不属于SS, 则把 IS' 加入SS;
- [6] 重复[5]直到SS收敛;
- [7] 终止状态: 包含形如 $A \rightarrow \alpha \bullet$ 项目的项目集合;

6.4 LR(0) 分析法

■ 构造LR(0)自动机-方法2，例1

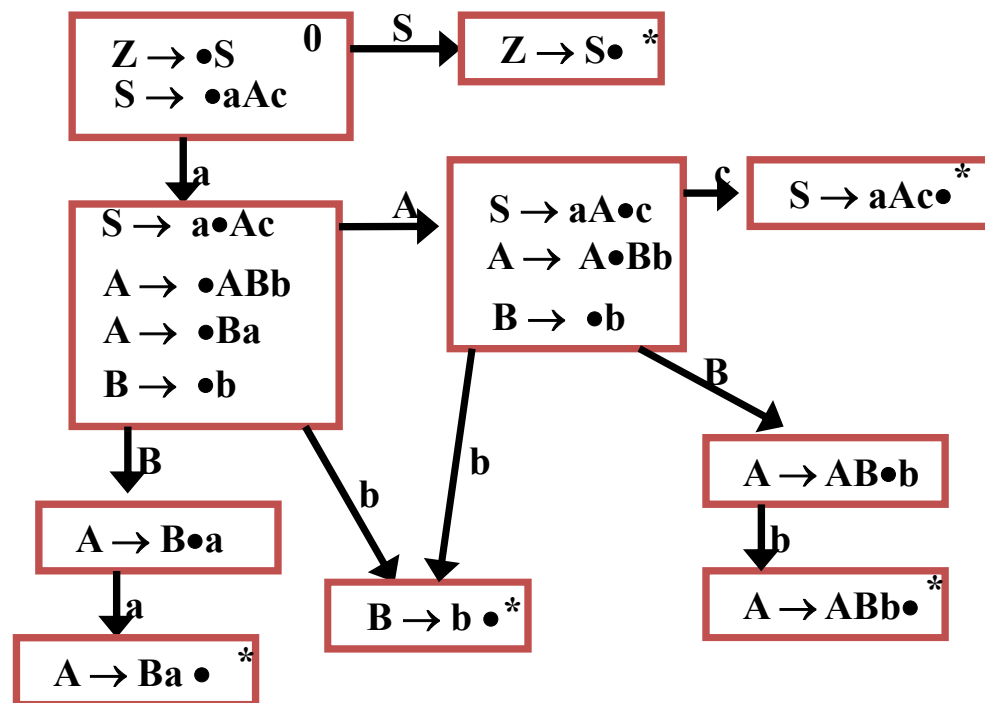
$V_T = \{a, b, c\}$

$V_N = \{S, A, B\}$

$S = S$

P:

$\{$
 $S \rightarrow aAc$
 $A \rightarrow ABb$
 $A \rightarrow Ba$
 $B \rightarrow b$
 $\}$



6.4 LR(0) 分析法

■ 构造LR(0)自动机-方法2，例2

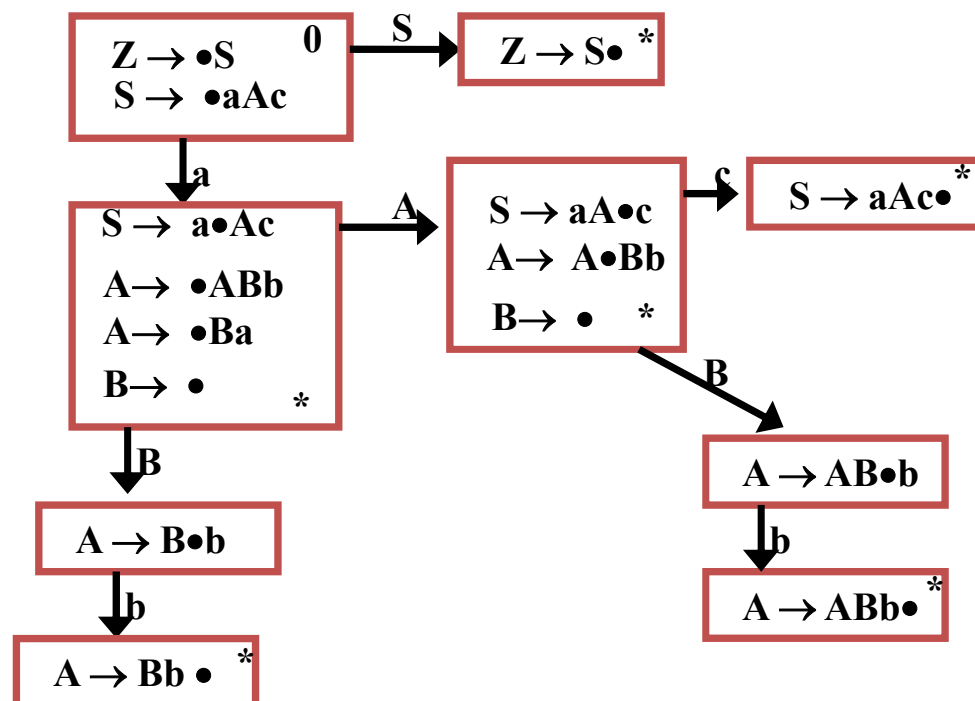
$V_T = \{a, b, c\}$

$V_N = \{S, A, B\}$

$S = S$

P:

$\{$
 $S \rightarrow aAc$
 $A \rightarrow ABb$
 $A \rightarrow Ba$
 $B \rightarrow \varepsilon$
 $\}$



6.4 LR(0) 分析法

■ 构造LR(0)自动机-方法2，例3

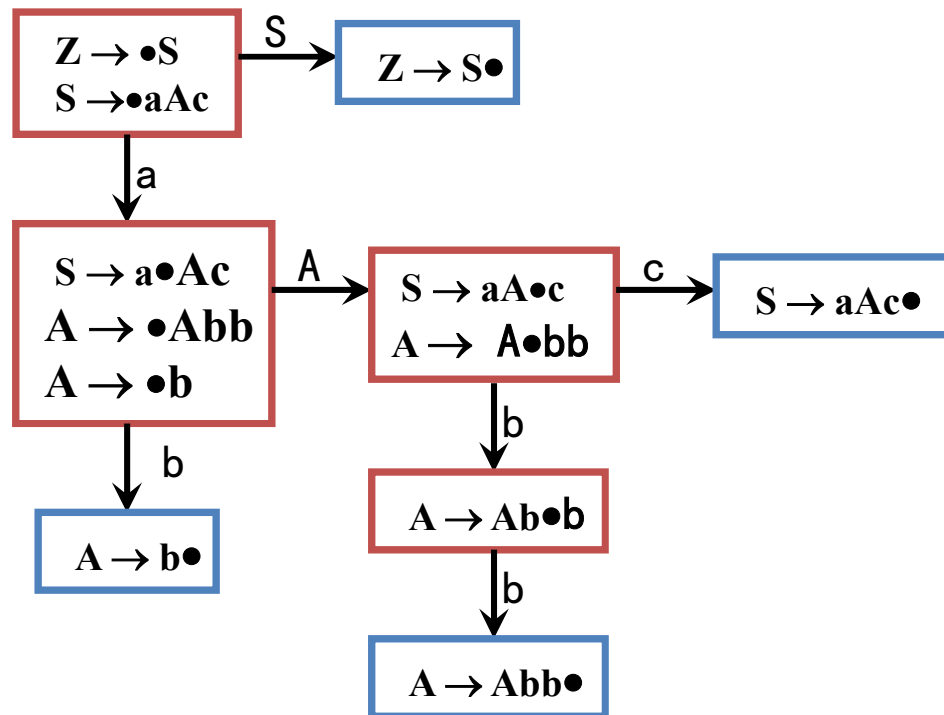
$V_T = \{a, b, c\}$

$V_N = \{S, A\}$

$S = S$

P:

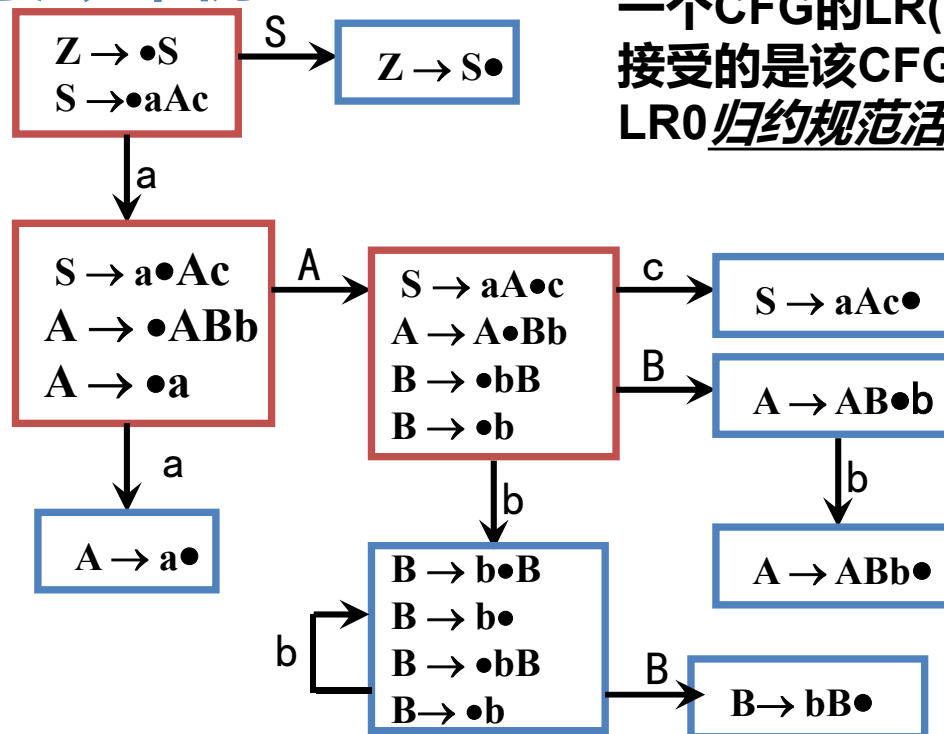
$\{$
 $S \rightarrow aAc$
 $A \rightarrow Abb$
 $A \rightarrow b$
 $\}$



6.4 LR(0) 分析法

■ 构造LR(0)自动机-方法2，例4

$V_T = \{a, b, c\}$
 $V_N = \{S, A, B\}$
 $S = S$
P:
{ $S \rightarrow aAc$
 $A \rightarrow ABb$
 $A \rightarrow a$
 $B \rightarrow bB$
 $B \rightarrow b$
}



一个CFG的LR(0)自动机
接受的是该CFG的所有
LR0 归约规范活前缀,

6.4 LR(0) 分析法

■ LR(0)分析法,相关概念

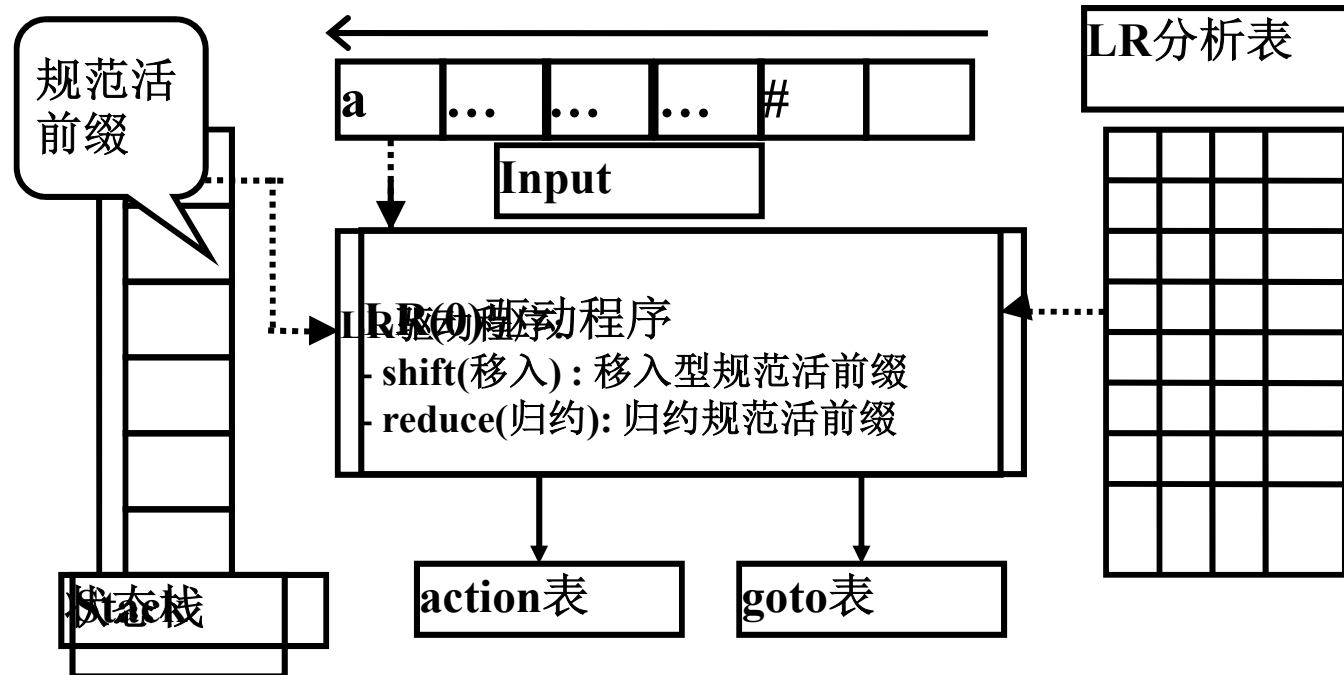
- 移入项目: $A \rightarrow \alpha \bullet a \beta$, $a \in V_T$
- 归约项目: $A \rightarrow \alpha \bullet$,
- 接受项目: $Z \rightarrow S \bullet$, ($Z \rightarrow S$ 是增广产生式)
- 移入状态:包含移入项目的状态
- 归约状态:包含归约项目的状态
- 冲突状态(同一个状态中):
 - 包含不同的归约项目,则称该状态存在归约-归约冲突 ;
 - 既包含移入项目,又包含归约项目,则称该状态存在移入-归约冲突

6.4 LR(0) 分析法

■ 给定一个上下文无关文法 G

- LR_0 是 G 的 LR(0) 自动机
- 如果 LR_0 的任意一个状态都不存在冲突(归约-归约冲突、移入-归约冲突), 则 G 称为 **LR(0) 文法**;
- 可以推知: 在 LR(0) 文法中 ,
 - 任意状态或者是移入状态, 或者是归约状态
 - 如果是归约状态, 一定存在一个**唯一**的归约项目, 该归约项目对应一个产生式 p , 因此, 该归约状态称为 p -归约状态

6.4 LR(0) 分析法



6.4 LR(0) 分析法

■ 构造LR(0)分析表

■ Action表

状态 \ 终极符	a_1	\dots	$\#$
S_1			
\dots			
S_n			

$\text{action}(S_i, a) = S_j$, 如果 S_i 到 S_j 有 a 输出边

$\text{action}(S_i, c) = R_p$, 如果 S_i 是 p -归约状态, $c \in V_t \cup \{\#\}$

$\text{action}(S_i, \#) = \text{accept}$, 如果 S_i 是接受状态

$\text{action}(S_i, a) = \text{error}$, 其他情形

6.4 LR(0) 分析法

■ 构造LR(0)分析表

■ GOTO表

状态 \ 非终极符	A_1	\dots	$\#$
S_1			
\dots			
S_n			

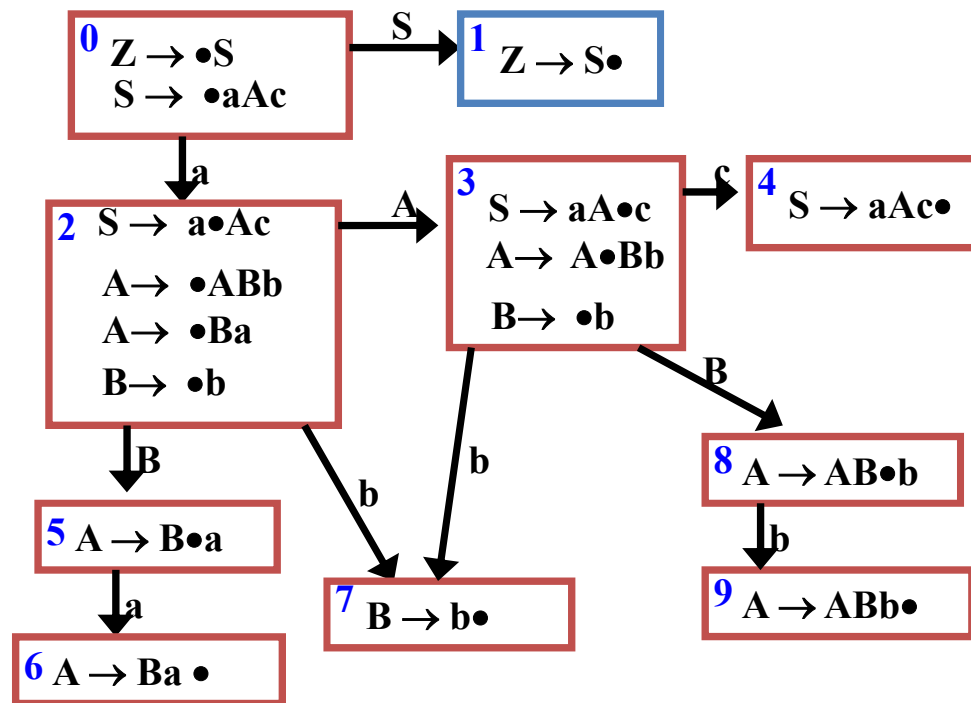
$\text{goto}(S_i, A) = S_j$, 如果 S_i 到 S_j 有 A 输出边
 $\text{goto}(S_i, A) = \text{error}$, 如果 S_i 没有 A 输出边

6.4 LR(0) 分析法

■ 例子

$V_T = \{a, b, c\}$
 $V_N = \{S, A, B\}$
 $S = S$
P:

- { (1) $S \rightarrow aAc$
- (2) $A \rightarrow ABb$
- (3) $A \rightarrow Ba$
- (4) $B \rightarrow b$
- }



6.4 LR(0) 分析法

先前做法

P:

(0) $Z \rightarrow S$

(1) $S \rightarrow aAc$

(2) $A \rightarrow ABb$

(3) $A \rightarrow Ba$

(4) $B \rightarrow b$

a

b

a

c

符号栈	输入流	分析动作
	abac#	移入
a	bac#	移入
ab	ac#	归约 (4)
aB	ac#	移入
aBa	c#	归约(3)
aA	c#	移入
aAc	#	归约(1)
S	#	归约(0)
Z	#	接受

6.4 LR(0) 分析法

action表					goto表		
	a	b	c	#	S	A	B
0	S2				1		
1				accept			
2		S7				3	5
3		S7	S4				8
4	R1	R1	R1	R1			
5	S6						
6	R3	R3	R3	R3			
7	R4	R4	R4	R4			
8		S9					
9	R2	R2	R2	R2			

6.4 LR(0) 分析法

■ LR(0)分析驱动程序

■ 符号约定：

- S0: 开始状态
- Stack: 状态栈
- Stack(top): 栈顶元素
- P: 产生式
- |P|: 产生式P右部符号个数;
- P_A: 产生式P左部非终极符;
- Push(S): 把状态S压入stack;
- Pop(n): 从stack弹出n个栈顶元素;

初始化: push(S0); a = readOne();

L: Switch *action*(stack(top), a)

Case **error**: error();

Case **accept**: return true;

Case **Si**: push(Si), a=readOne(); goto L;

Case **R_P**: pop(|P|);

push(*goto*(stack(top), P_A));
goto L;

6.4 LR(0) 分析法

完整过程

a	b	a	c
---	---	---	---

P: (0) $Z \rightarrow S$; (1) $S \rightarrow aAc$; (2) $A \rightarrow ABb$;
(3) $A \rightarrow Ba$; (4) $B \rightarrow b$

action表					goto表		
	a	b	c	#	S	A	B
0	S2				1		
1				accept			
2		S7				3	5
3		S7	S4				8
4	R1	R1	R1	R1			
5	S6						
6	R3	R3	R3	R3			
7	R4	R4	R4	R4			
8		S9					
9	R2	R2	R2	R2			

状态栈	输入流	分析动作
0	abac#	S2
02	bac#	S7
027	ac#	R4, Goto(2, B)=5
025	ac#	S6
0256	c#	R3, Goto(2, A)=3
023	c#	S4
0234	#	R1, Goto(0, S)=1
01	#	Accept

作业

- 教材P153: 4.5.2, 4.5.3



Thank you!