

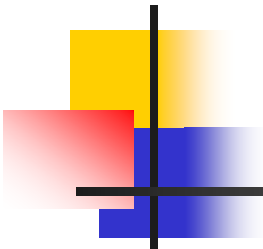


School of Computer Science & Technology
Harbin Institute of Technology

第五章 自底向上的语法分析

重点：自底向上分析的基本思想，算符优先分析法的基本思想，简单算符优先分析法。LR 分析器的基本构造思想，LR分析算法，规范句型活前缀及其识别器——DFA，LR(0)分析表的构造，SLR(1)分析表的构造，LR(1)分析表的构造。

难点：求FIRSTOP 和LASTOP，算符优先关系的确定，算符优先分析表的构造，素短语与最左素短语的概念。规范句型活前缀，LR(0)项目集闭包与项目集规范族，它们与句柄识别的关系，活前缀与句柄的关系，LR(1)项目集闭包与项目集规范族。



回顾：句型

- **定义2.20** 设文法 $G=(V, T, P, S)$, 对于
 $\forall \alpha \in (V \cup T)^*$, 如果 $S \xRightarrow{*} \alpha$, 则称 α 是 G 产生的
一个句型
- 对于任意文法 $G=(V, T, P, S)$, G 产生的句子 w 和句型 α 的区别在于句子满足 $w \in T^*$, 而句型满足 $\alpha \in (V \cup T)^*$



回顾：短语和句柄

- **定义2.27** 设有CFG $G=(V, T, P, S)$, $\exists \alpha, \beta$, $\gamma \in (V \cup T)^*$, $S \xRightarrow{*} \gamma A \beta$, $A \xRightarrow{+} \alpha$, 则称 α 是句型 $\gamma \alpha \beta$ 的相对于变量 A 的**短语**(phrase);
- **定义2.28** 设有CFG $G=(V, T, P, S)$, G 的句型的最左直接短语叫做**句柄**(handle)。



第5章 自底向上的语法分析

5.1 自底向上的语法分析概述

5.2 算符优先分析法

5.3 LR分析法

5.4 语法分析程序的自动生成工具Yacc

5.5 本章小结



5.1 自底向上的语法分析概述

- 所谓自底向上的语法分析，是指从给定的**输入符号串**出发，试图**自底向上**地为其建立一棵**语法分析树**



5.1 自底向上的语法分析概述

■ 思想

- 从输入串出发，反复利用产生式进行**归约**，如果最后能得到文法的**开始符号**，则输入串是相应文法的一个句子，否则输入串有语法错误。
- 在分析过程中，寻找**当前句型最左的和某个产生式的右部相匹配的子串（句柄）**，用该产生式的左部符号代替该子串（**最左归约**）。
- 如果每步都能正确选择子串，就可以得到输入串的**最左归约**，即规范归约过程



5.1 自底向上的语法分析概述

■ 核心问题

- 寻找句型中**当前归约对象**——“句柄”进行归约
- 用**不同方法寻找句柄**，就可获得不同的分析方法

例5.1 一个简单的归约过程

■ 设文法G为:

$S \rightarrow aABe$ $A \rightarrow Abc|b$ $B \rightarrow d$

句子分析: $abbcde$

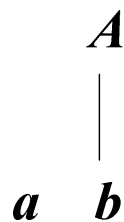
$a\underline{b}bcde$

$\Leftarrow a\underline{A}bcde$

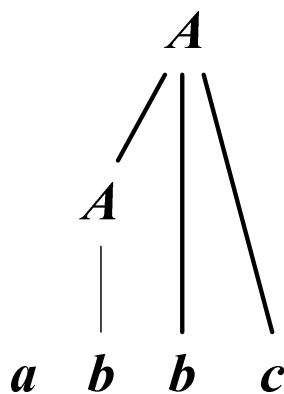
$\Leftarrow aA\underline{d}e$

$\Leftarrow a\underline{AB}e$

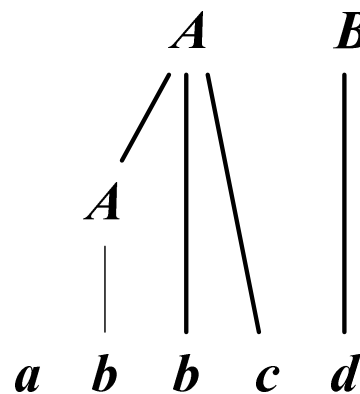
$\Leftarrow S$



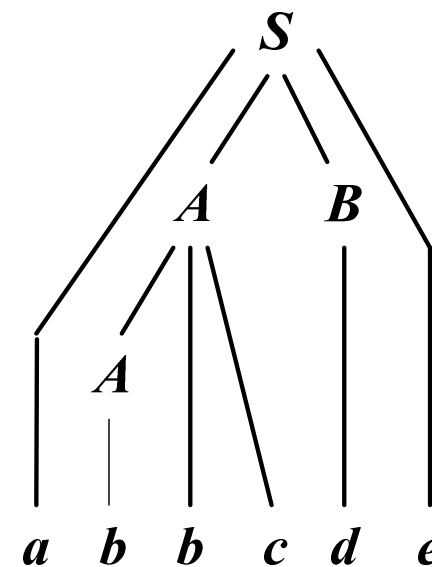
(a)



(b)



(c)



(d)

语法树的形成过程



5.1 自底向上的语法分析概述

- 根据基本思想，可以采用brute-force方法来执行自底向上语法分析的问题
 - 效率问题
 - 冲突问题
- 下面介绍常用的自底向上语法分析方法，即**移进-归约**分析法

5.1.1 移进-归约分析

■ 系统框架

- 采用**表驱动**的方式实现
- 输入缓冲区：保存输入符号串 w ，**初始时 $w\#$**
- 分析栈：保存语法符号，即**已经得到的分析结果**，栈底符号 $\#$ ，**初始状态 $\#$** ，即栈为空
- 分析表：存放不同情况下的**分析动作**
- 控制程序：控制分析过程，输出分析结果——产生式序列
- **格局：栈中符号串 + 输入缓冲区剩余内容**

5.1.1 移进-归约分析

■ 系统框架

- 采用**表驱动**的方式实现
- 输入缓冲区：保存输入符号串 w ，**初始时 $w\#$**
- 分析栈：保存语法符号，即**已经得到的分析结果**，栈底符号 $\#$ ，**初始状态 $\#$** ，即栈为空
- 分析表：存放不同情况下的移进-归约动作
- 控制程序：控制分析过程，产生式序列
- **格局**：栈中符号串+输入缓冲区剩余内容

如果输入符号串是语言的一个句子，那么该格局对应文法的一个句型

产生

5.1.1 移进-归约分析

■ 格局：栈+输入缓冲区剩余内容

句子分析：abbcde

a**b**bcde

←a**A**bcde

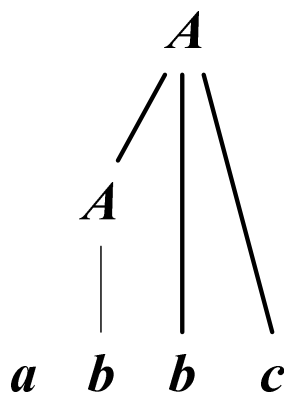
←a**A**de

←a**A**Be

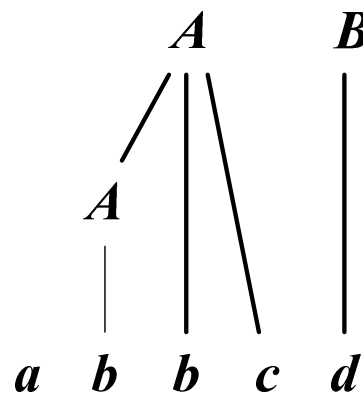
← S



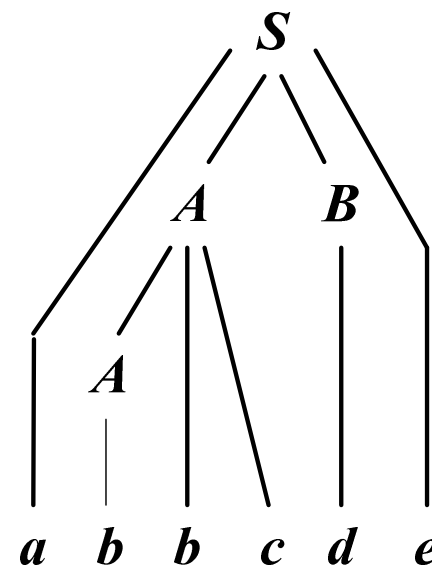
(a)



(b)

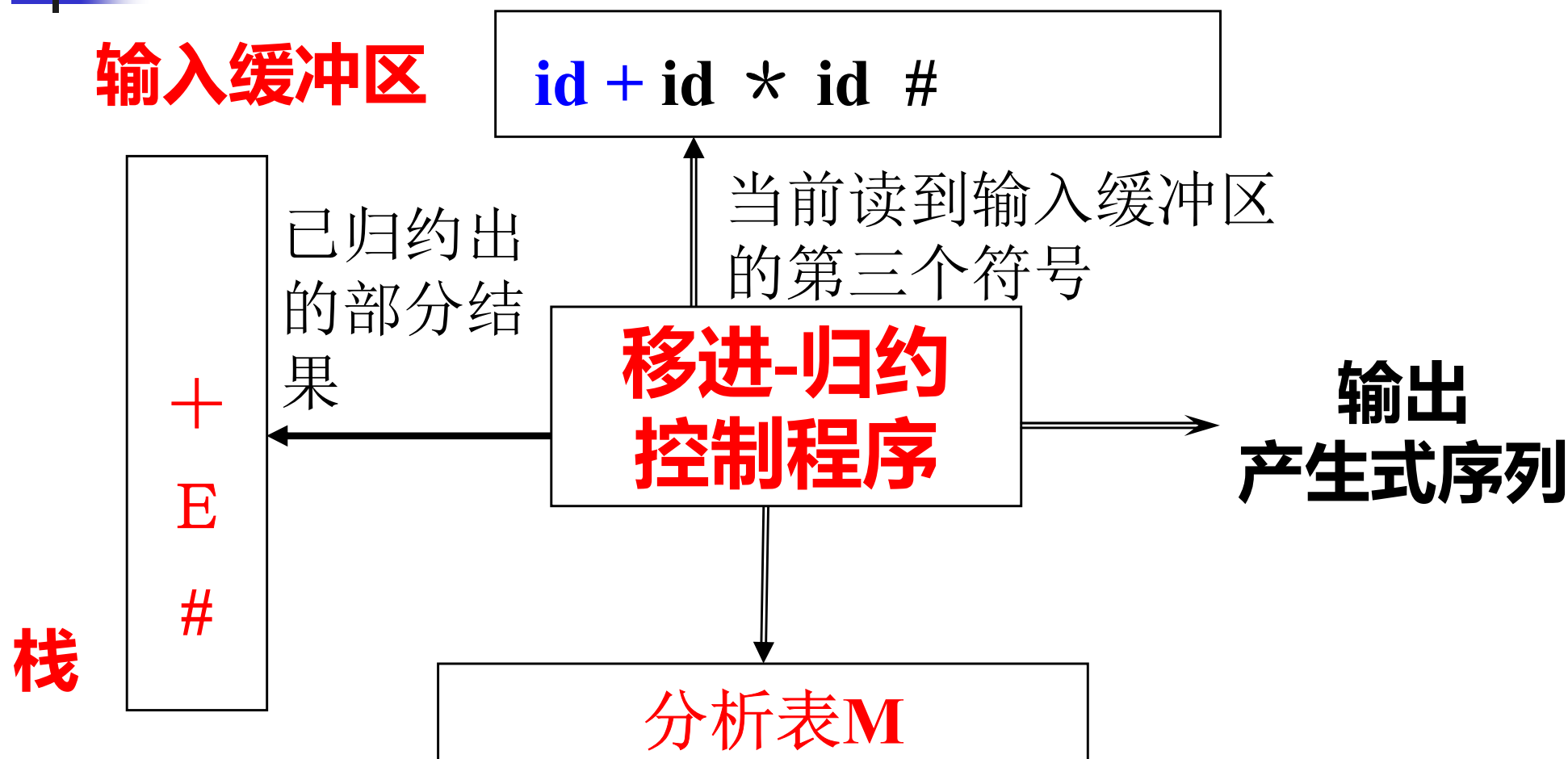


(c)



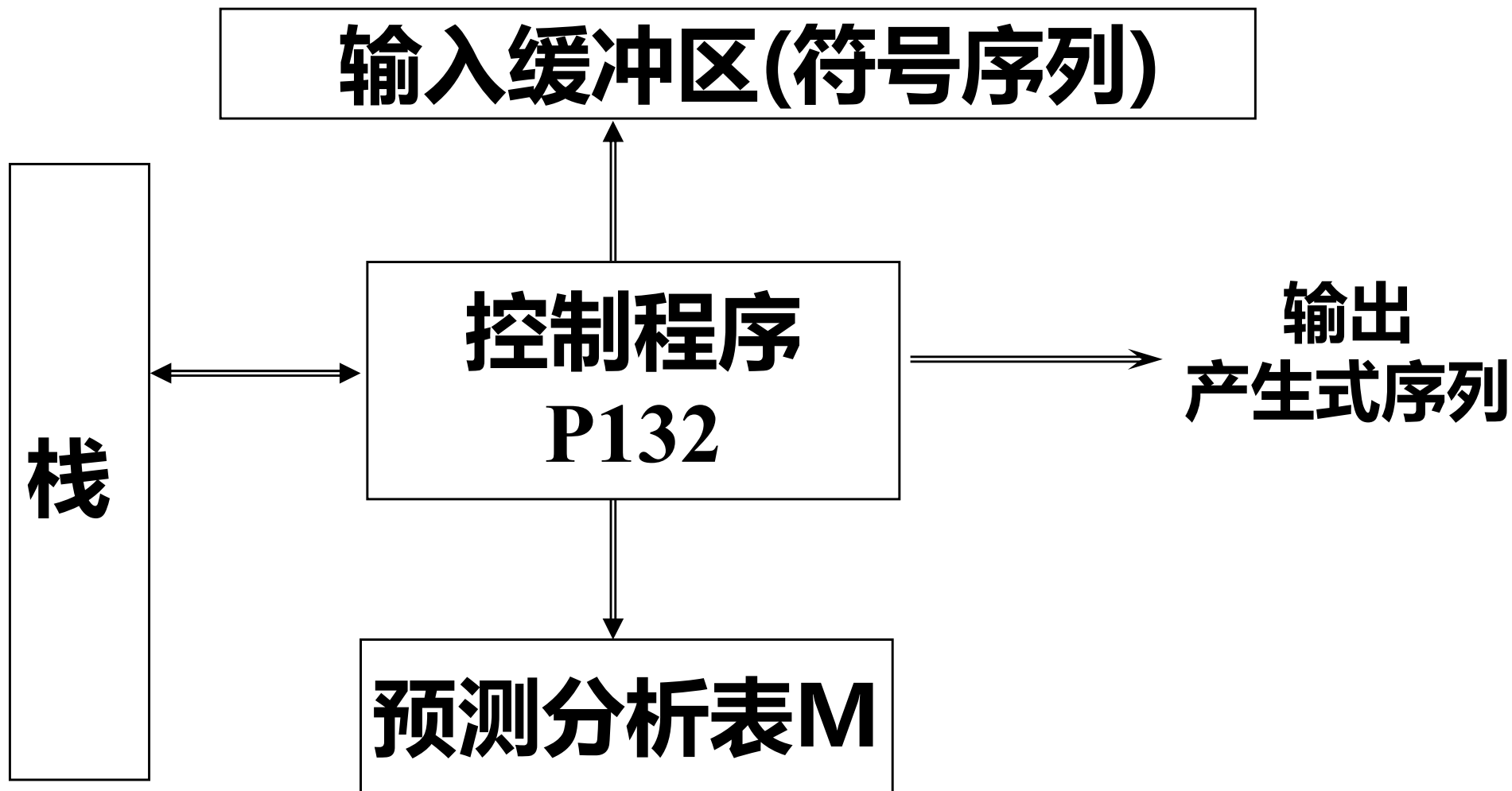
(d)

移进-归约语法分析器的总体结构



栈内容 + 输入缓冲区内容 = #当前句型#

与LL(1)的体系结构比较



移进-归约分析的工作过程

■ 系统运行

- 开始格局 (栈: #; 输入缓冲区: $w\#$)
- 自左向右地扫描输入串, 一边将输入符号移进分析栈内, 一边检查栈顶是否形成句柄 ($A \rightarrow \alpha$ 的右部)
- 如果栈顶出现句柄 α , 则将 α 替换为 A 。如果栈顶没有形成句柄, 则继续将输入符号移进栈内, 进行判断。
- 正常结束: 栈中为 $\#S$, 并且输入缓冲区只有

移进-归约分析的工作过程

■ 系统运行

- 开始格局 (栈: #; 输入缓冲区: w#)
- 自左向右地扫描输入串, 一边将**输入符号移进分析栈内**, 一边检查栈顶**是否形成句柄** ($A \rightarrow \alpha$ 的右部)
- 如果栈顶出现句柄 α , 则归约; 如果栈顶没有形成句柄, 则继续将输入符号移进栈内, 进行判断。
- 正常结束: 栈中为 #S, 并且输入缓冲区只有

问题: 系统如何发现句柄在栈顶形成?

输出结果表示:

例5.2 $E \rightarrow E + E | E * E | (E) | id$

用产生式序列表示语法分析树

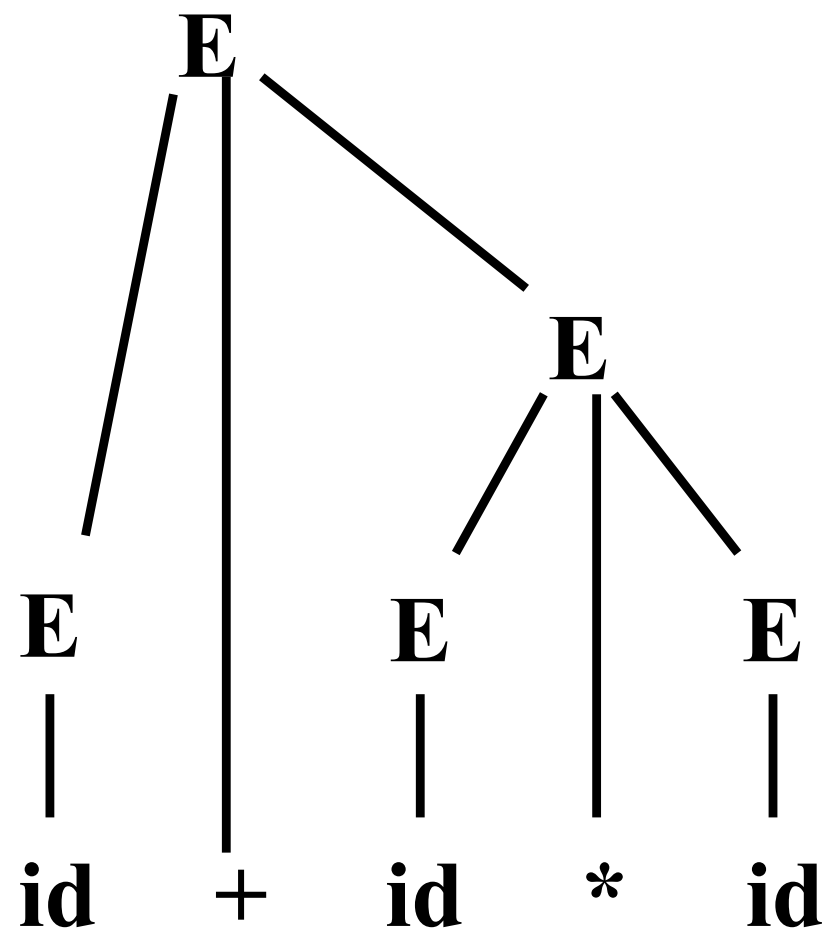
$E \rightarrow id$

$E \rightarrow id$

$E \rightarrow id$

$E \rightarrow E * E$

$E \rightarrow E + E$



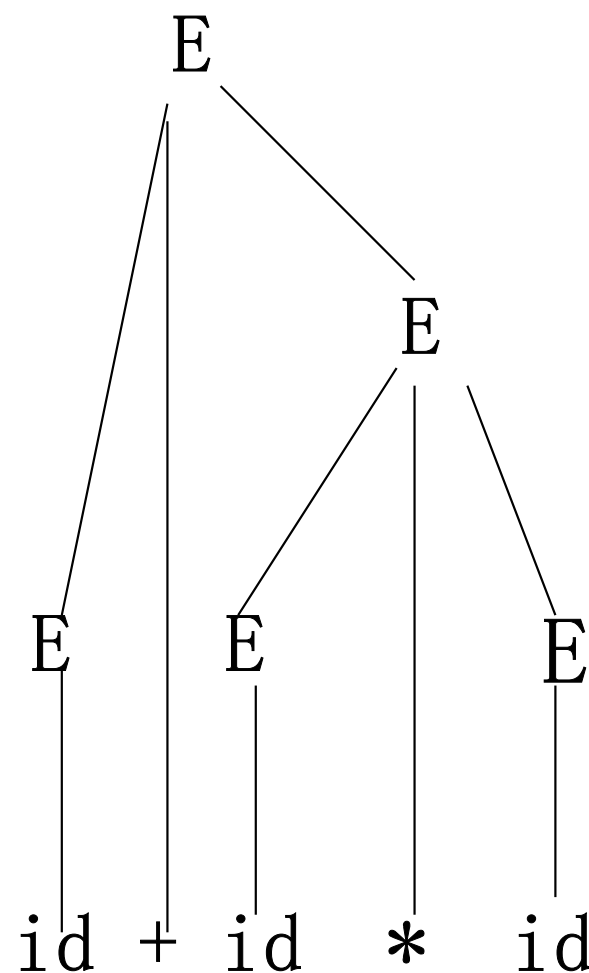
动作

栈

输入缓冲区

1)			#	$id_1 + id_2 * id_3 \#$
2)	移进		$\#id_1$	$+id_2 * id_3 \#$
3)	归约	$E \rightarrow id$	$\#E$	$+id_2 * id_3 \#$
4)	移进		$\#E +$	$id_2 * id_3 \#$
5)	移进		$\#E + id_2$	$*id_3 \#$
6)	归约	$E \rightarrow id$	$\#E + E$	$*id_3 \#$
7)	移进		$\#E + E *$	$id_3 \#$
8)	移进		$\#E + E * id_3$	$\#$
9)	归约	$E \rightarrow id$	$\#E + E * E$	$\#$
10)	归约	$E \rightarrow E * E$	$\#E + E$	$\#$
11)	归约	$E \rightarrow E + E$	$\#E$	$\#$
12)	接受			

例5.2 分析过程



分析器的四种动作

- 1) **移进**：将下一输入符号移入栈
- 2) **归约**：用产生式左侧的非终结符替换栈顶的句柄（某产生式右部）
- 3) **接受**：分析成功
- 4) **出错**：出错处理

对于这个问题的不同解决方法形成不同的移进-归约分析方法

**问题1：决定移进和归约的依据是什么，即：
如何确定栈顶是否已经形成句柄**

问题2：栈中是否会出现句型的句柄后的符号

5.1.2 优先法

- 基本思想：根据**归约的先后次序**为句型中**相邻**的文法符号规定**优先关系**
 - 句柄内相邻符号**同时归约**，是**同优先级**的
 - 句柄两端符号的优先级要高于句柄外与之相邻的符号
- $a_1 \dots a_{i-1} \prec a_i \equiv a_{i+1} \equiv \dots \equiv a_{j-1} \equiv a_j \succ a_{j+1} \dots a_n$
- 定义优先关系，语法分析程序可以通过 $a_{i-1} \prec a_i$ 和 $a_j \succ a_{j+1}$ 这两个关系来确定句柄的头和尾



5.1.2 优先法

- **简单优先文法**：如果各文法符号之间的**优先关系互不冲突**(即至多存在一种优先关系)，则可识别任意句型的句柄
- **算符优先文法**：仅对文法中可能在句型中**相邻的终结符**定义优先关系，并且**各终结符对**之间的优先关系互不冲突。
- **考虑算术表达式**： $\text{sum} = a + b + c$; $\text{total} = a + b * c$;



5.1.3 状态法

- **根据句柄的识别状态来识别句柄**
 - **句柄是逐步形成的，用状态来描述不同时刻形成的那部分句柄**
 - **因为句柄是产生式的右部，可用产生式来表示句柄的不同识别状态**

5.1.3 状态法

- 例如： $S \rightarrow bBB$ 可分解为如下识别状态

- $S \rightarrow .bBB$ 移进 b
- $S \rightarrow b.BB$ 等待归约出第一个 B
- $S \rightarrow bB.B$ 等待归约出第二个 B
- $S \rightarrow bBB.$ 归约

小黑点.左部的字符串表示已识别出的部分句柄符号

- 采用这种方法，语法分析程序根据当前分析状态就可以确定句柄的头和尾，并进行正确的归约。



移进-归约分析中的问题

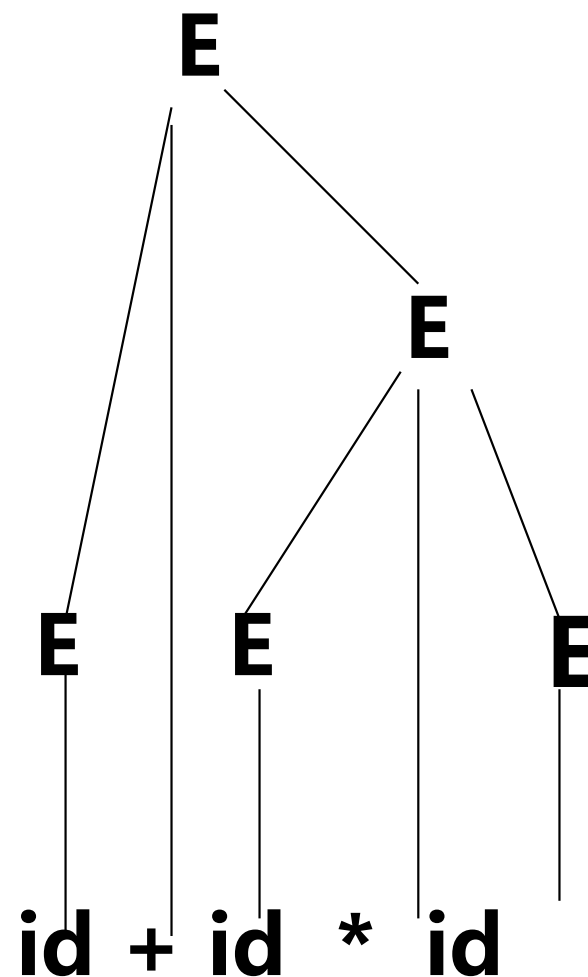
- 1) 移进归约冲突
 - 根据栈中的内容和下一个输入符号**不能确定是移进还是归约**

动作

栈 输入缓冲区

- 1) # $id_1 + id_2 * id_3 \#$
- 2) 移进 # id_1 + $id_2 * id_3 \#$
- 3) 归约 $E \rightarrow id$ # E + $id_2 * id_3 \#$
- 4) 移进 # $E +$ $id_2 * id_3 \#$
- 5) 移进 # $E + id_2$ * $id_3 \#$
- 6) 归约 $E \rightarrow id$ # $E + E$ * $id_3 \#$
- 7) 移进 # $E + E^*$ $id_3 \#$
- 8) 移进 # $E + E * id_3$ #
- 9) 归约 $E \rightarrow id$ # $E + E * E$ #
- 10) 归约 $E \rightarrow E * E$ # $E + E$ #
- 11) 归约 $E \rightarrow E + E$ # E #
- 12) 接受

例5.2分析过程





移进-归约分析中的问题

2) 归约归约冲突

- 存在两个可用的产生式，不能确定按哪一个产生式进行归约

$A \rightarrow id$

$B \rightarrow id$

.....

栈
(.....id

输入
, ...)



移进-归约分析中的问题

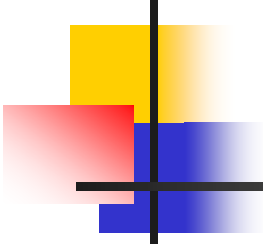
- 对于**移进归约冲突**和**归约归约冲突**，各种分析方法处理冲突的方法不同，可能方法包括**移进优先、文法改造**等。

5.2 算符优先分析法

- **Robert W.Floyd, Syntactic Analysis and Operator Precedence, Journal of the ACM (JACM), 1963, 10(3):316–333.**



1936年出生，1953年芝加哥大学文学学士学位，堆排序算法和Floyd-Warshall算法(解决任意两点间的最短路径的一种算法)的创始人，斯坦福大学教授，1978年图灵奖得主



5.2 算符优先分析法

- 分析各类**表达式**尤为有效
- 方法
 - 将句型中的**终结符号**当作“**算符**”，借助算符之间的**优先关系**确定句柄
- 算符优先关系的直观意义
 - $+$ \prec $*$ $+$ 的优先级低于 $*$
 - $($ \equiv $)$ $($ 的优先级等于 $)$
 - $+$ \succ $+$ 左侧 $+$ 优先级高于右侧 $+$ (左结合原则)



5.2.1 算符优先文法

■ 考虑 $A + B * C / D$ ，该表达式的计算步骤为

(1) $T_1 = B * C$

(2) $T_2 = T_1 / D$

(3) $T_3 = A + T_2$

■ 表达式的**运算次序和运算对象没有关系**，而只和**运算符**的优先级有关系



5.2.1 算符优先文法

- 先给出**算符文法**的定义

- 如果文法 $G = (V, T, P, S)$ 中不存在形如

$$A \rightarrow \alpha \mathbf{BC} \beta$$

的产生式，则称之为算符文法(OG —Operator Grammar)

即：如果文法 G 中**不存在具有相邻非终结符**的产生式，则称为**算符文法**。



5.2.1 算符优先文法

对 **id + id * id / id** 进行分析

有错误

■ $E \rightarrow E + E$

■ $E \rightarrow E - E$

■ $E \rightarrow E * E$

■ $E \rightarrow E / E$

■ $E \rightarrow (E)$

■ $E \rightarrow \text{id}$

原因在于算符文法本身**没有**规定算符之间的**优先关系**，所以语法分析程序只能根据**最左归约**的原则来做

希望使文法本身还有确定各终结符之间的**归约先后次序**的信息，从而引出**算符优先文法**的定义

5.2.1 算符优先文法

- 定义5.1 假设 G 是一个不含 ε -产生式的文法, A 、 B 和 C 均是 G 的语法变量, G 的任何一对终结符 a 和 b 之间的**优先关系定义**为:

- (1) $a \equiv b$, 当且仅当文法 G 中含有形如 $A \rightarrow \dots ab \dots$ 或 $A \rightarrow \dots aBb \dots$ 的产生式;

终结符 a 和 b 相邻, 且在对变量 A 进行分析时同时归约

5.2.1 算符优先文法

- 定义5.1 假设 G 是一个不含 ε -产生式的文法, A 、 B 和 C 均是 G 的语法变量, G 的任何一对终结符 a 和 b 之间的优先关系定义为:

- (1) $a \equiv b$, 当且仅当文法 G 中含有形如 $A \rightarrow \dots ab \dots$ 或 $A \rightarrow \dots aBb \dots$ 的产生式;
- (2) $a \prec b$, 当且仅当文法 G 中含有形如 $A \rightarrow \dots aB \dots$ 的产生式, 而且 $B \Rightarrow^+ b \dots$ 或 $B \Rightarrow^+ Cb \dots$;

终结符 a 和 b 相邻, 且在对变量 A 进行分析时, 终结符 b 比 a 先归约

5.2.1 算符优先文法

- 定义5.1 假设 G 是一个不含 ε -产生式的文法, A 、 B 和 C 均是 G 的语法变量, G 的任何一对终结符 a 和 b 之间的优先关系定义为:

- (1) $a \equiv b$, 当且仅当文法 G 中含有形如 $A \rightarrow \dots ab \dots$ 或 $A \rightarrow \dots aBb \dots$ 的产生式;
- (2) $a \prec b$, 当且仅当文法 G 中含有形如 $A \rightarrow \dots aB \dots$ 的产生式, 而且 $B \Rightarrow^+ b \dots$ 或 $B \Rightarrow^+ Cb \dots$;
- (3) $a \succ b$, 当且仅当文法 G 中含有形如 $A \rightarrow \dots Bb \dots$ 的产生式, 而且 $B \Rightarrow^+ \dots a$ 或 $B \Rightarrow^+ \dots aC$;

终结符 a 和 b 相邻, 且在对变量 A 进行分析时, 终结符 a 比 b 先归约

5.2.1 算符优先文法

- 定义5.1 假设 G 是一个不含 ε -产生式的文法, A 、 B 和 C 均是 G 的语法变量, G 的任何一对终结符 a 和 b 之间的优先关系定义为:

- (1) $a \equiv b$, 当且仅当文法 G 中含有形如 $A \rightarrow \dots ab \dots$ 或 $A \rightarrow \dots aBb \dots$ 的产生式;
- (2) $a \prec b$, 当且仅当文法 G 中含有形如 $A \rightarrow \dots aB \dots$ 的产生式, 而且 $B \Rightarrow^+ b \dots$ 或 $B \Rightarrow^+ Cb \dots$;
- (3) $a \succ b$, 当且仅当文法 G 中含有形如 $A \rightarrow \dots Bb \dots$ 的产生式, 而且 $B \Rightarrow^+ \dots a$ 或 $B \Rightarrow^+ \dots aC$;
- (4) a 与 b 无关系, 当且仅当 a 与 b 在 G 的任何句型中都不相邻。



5.2.1 算符优先文法

- 设 $G = (V, T, P, S)$ 为算符文法, 如果 $\forall a, b \in T, a \equiv b, a \prec b, a \succ b$ **至多有一个成立**, 则称之为**算符优先文法**(OPG — Operator Precedence Grammar)

换句话说, 在无 ε 产生式的算符文法 G 中, 如果**任意一对终结符之间至多有一种优先关系**, 则称为算符优先文法。

5.2.1 算符优先文法

- $E \rightarrow E + E$
- $E \rightarrow E - E$
- $E \rightarrow E * E$
- $E \rightarrow E / E$
- $E \rightarrow (E)$
- $E \rightarrow id$

该文法是不是算符优先文法？

$E \rightarrow E + E$, $E \rightarrow E - E$,

我们有 $+ \prec -$

$E \rightarrow E - E$, $E \rightarrow E + E$

我们有 $+ \succ -$



5.2.2 算符优先矩阵的构造

- 为什么需要算符优先矩阵
 - 在对算符优先文法进行分析时，语法分析程序**随时都要比较**两个相邻终结符之间的**优先关系**，以便识别句型的句柄
 - 为方便起见，通常用一个**矩阵**来存放文法中终结符之间的各种可能优先关系
 - **矩阵** $M[n*n]$ ， n 是终结符数量， $M[i, j]$ 维护第 i 个终结符和第 j 个终结符的优先关系
- 下面介绍如何来构造算符优先矩阵

5.2.2 算符优先矩阵的构造

- 优先关系的确定：根据优先关系的定义
 - $a \equiv b \Leftrightarrow A \rightarrow \dots ab \dots$ 或者 $A \rightarrow \dots aBb \dots$
 - $a \lessdot b \Leftrightarrow A \rightarrow \dots aB \dots \in P$ 且 $(B \Rightarrow^+ b \dots$ 或者 $B \Rightarrow^+ \text{C}b \dots)$, 需要求出非终结符B派生出的**最左终结符集**
 - $a \rhd b \Leftrightarrow A \rightarrow \dots Bb \dots \in P$ 且 $(B \Rightarrow^+ \dots a$ 或者 $B \Rightarrow^+ \dots a\text{C})$, 需要求出非终结符B派生出的**最右终结符集**

5.2.2 算符优先矩阵的构造

■ 优先关系的确定：根据优先关系的定义

■ $a \equiv b \Leftrightarrow A \rightarrow \dots ab \dots$ 或者 $A \rightarrow \dots aBb \dots$

■ $a \lessdot b \Leftrightarrow A \rightarrow \dots aB \dots \in P \text{ 且 } (B \Rightarrow^+ b \dots \text{或者 } B \Rightarrow^+ \text{C}b \dots)$
终结符集

如何求出非终结符可派生的最左终结符集和最右终结符集

■ $a \rhd b \Leftrightarrow A \rightarrow \dots a \text{或者 } B \Rightarrow^+ \dots a \text{C}),$ 需要求出非终结符B派生出的**最右终结符集**

5.2.2 算符优先矩阵的构造

- 设 $G = (V, T, P, S)$ 为算符优先文法, 则定义
 - **FIRSTOP(A)** = $\{b | A \Rightarrow^+ b \dots \text{或者} A \Rightarrow^+ Bb \dots, b \in T, B \in V\}$ (最左终结符集合)
 - **LASTOP(A)** = $\{b | A \Rightarrow^+ \dots b \text{或者} A \Rightarrow^+ \dots bB, b \in T, B \in V\}$ (最右终结符集合)

5.2.2 算符优先矩阵的构造

- $A \rightarrow \dots ab \dots ; A \rightarrow \dots aBb \dots$, 则 $a \equiv b$
- $A \rightarrow \dots aB \dots$, 则对 $\forall b \in \text{FIRSTOP}(B)$, $a \prec b$
- $A \rightarrow \dots Bb \dots$, 则对 $\forall a \in \text{LASTOP}(B)$, $a \succ b$
- if $A \rightarrow B \dots \in P$, then $\text{FIRSTOP}(B) \subseteq \text{FIRSTOP}(A)$,
即可从B推出的最左终结符肯定包括在可从A推出的最左终结符集合
- if $A \rightarrow \dots B \in P$, then $\text{LASTOP}(B) \subseteq \text{LASTOP}(A)$,
即可从B推出的最右终结符肯定包括在可从A推出的最右终结符集合
- 问题：求FIRSTOP、LASTOP

- $E \rightarrow E + E$

- $E \rightarrow E - E$

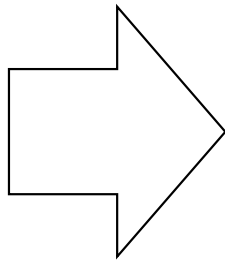
- $E \rightarrow E * E$

- $E \rightarrow E / E$

- $E \rightarrow (E)$

- $E \rightarrow \text{id}$

不是算符优先文法



改造为算符优先文法

- $E \rightarrow T$

- $E \rightarrow E + T$

- $E \rightarrow E - T$

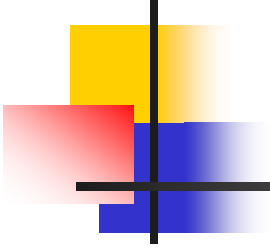
- $T \rightarrow F$

- $T \rightarrow T * F$

- $T \rightarrow T / F$

- $F \rightarrow (E)$

- $F \rightarrow \text{id}$

- 
-
- 1) $E \rightarrow T$ 先计算 \equiv 关系
 - 2) $E \rightarrow E + T$ (\equiv) 由产生式7
 - 3) $E \rightarrow E - T$
 - 4) $T \rightarrow F$
 - 5) $T \rightarrow T * F$
 - 6) $T \rightarrow T / F$
 - 7) $F \rightarrow (E)$
 - 8) $F \rightarrow \text{id}$



计算FIRSTOP

- 1) $E \rightarrow T$
- 2) $E \rightarrow E + T$
- 3) $E \rightarrow E - T$
- 4) $T \rightarrow F$
- 5) $T \rightarrow T * F$
- 6) $T \rightarrow T / F$
- 7) $F \rightarrow (E)$
- 8) $F \rightarrow id$

由产生式1, 2和3, 我们有
FIRSTOP(E)包括+、-和**FIRSTOP(T)**

由产生式4, 5和6, 我们有
FIRSTOP(T)包括*、/和**FIRSTOP(F)**

由产生式7和8, 我们有
FIRSTOP(F)包括 (和id

FIRSTOP(E) = {+, -, *, /, (, id}

FIRSTOP(T) = {*, /, (, id}

FIRSTOP(F) = {(, id}



计算LASTOP

- 1) $E \rightarrow T$
- 2) $E \rightarrow E + T$
- 3) $E \rightarrow E - T$
- 4) $T \rightarrow F$
- 5) $T \rightarrow T * F$
- 6) $T \rightarrow T / F$
- 7) $F \rightarrow (E)$
- 8) $F \rightarrow id$

由产生式1, 2和3, 我们有
LASTOP(E)包括+、-和LASTOP(T)

由产生式4, 5和6, 我们有
LASTOP(T)包括*、/和LASTOP(F)

由产生式7和8, 我们有
LASTOP(F)包括) 和id

LASTOP(E) = {+, -, *, /,), id}

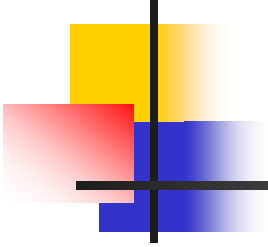
LASTOP(T) = {*, /,), id}

LASTOP(F) = {), id}



计算 \preceq 关系

- 1) $E \rightarrow T$
- 2) $E \rightarrow E + T$ $+ \preceq \text{FIRSTOP}(T)$
- 3) $E \rightarrow E - T$ $- \preceq \text{FIRSTOP}(T)$
- 4) $T \rightarrow F$
- 5) $T \rightarrow T * F$ $* \preceq \text{FIRSTOP}(F)$
- 6) $T \rightarrow T / F$ $/ \preceq \text{FIRSTOP}(F)$
- 7) $F \rightarrow (E)$ $(\preceq \text{FIRSTOP}(E)$
- 8) $F \rightarrow \text{id}$



计算 \triangleright 关系

- 1) $E \rightarrow T$
- 2) $E \rightarrow E + T$ $\text{LASTOP}(E) \triangleright +$
- 3) $E \rightarrow E - T$ $\text{LASTOP}(E) \triangleright -$
- 4) $T \rightarrow F$
- 5) $T \rightarrow T * F$ $\text{LASTOP}(T) \triangleright *$
- 6) $T \rightarrow T / F$ $\text{LASTOP}(T) \triangleright /$
- 7) $F \rightarrow (E)$ $\text{LASTOP}(E) \triangleright)$
- 8) $F \rightarrow \text{id}$

例 5.6 表达式文法的算符优先关系

	+	-	*	/	()	id	#
+	\succ	\succ	\prec	\prec	\prec	\succ	\prec	\succ
-	\succ	\succ	\prec	\prec				
*	\succ	\succ	\succ	\succ				
/	\succ	\succ	\succ	\succ				
(\prec	\prec	\prec	\prec				
)	\succ	\succ	\succ	\succ		\succ		\succ
id	\succ	\succ	\succ	\succ		\succ		\succ
#	\prec	\prec	\prec	\prec	\prec		\prec	acc

#的优先关系通过增加语法变量 S' 和产生式 $S' \rightarrow \#S\#$ 来完成

算法5.1 计算FIRSTOP集合

输入：文法 $G = (V, T, P, S)$

输出： $\forall A \in V, \text{FIRSTOP}(A)$

步骤：

begin

for $\forall (A, a) \in V \times T$ do $F[A, a] = \text{false}$

for $\forall A \rightarrow a \dots \in P$ 或 $\forall A \rightarrow Ba \dots \in P$

insert(A, a)

while 栈非空 do

begin

pop(A, a)

for $\forall C \rightarrow A \dots \in P$ do

insert(C, a)

end

for $\forall A \in V$ do

$\text{FIRSTOP}(A) = \emptyset$

for $\forall (A, a) \in V \times T$ do

begin

if $F[A, a]$ then

$\text{FIRSTOP}(A) = \text{FIRSTOP}(A) \cup \{a\}$

end

end

$F[A, a]$ 表示 a 是否属于
 $\text{FIRSTOP}(A)$ 的指示器

procedure insert(A, a)
if not $F[A, a]$ then
 $F[A, a] = \text{true}$
 push(A, a)
end

算法: 计算LASTOP集合

输入: 文法 $G = (V, T, P, S)$

输出: $\forall A \in V, \text{LASTOP}(A)$

步骤:

begin

for $\forall (A, a) \in V \times T$ do $F[A, a] = \text{false}$

for $\forall A \rightarrow \dots a \in P$ 或 $\forall A \rightarrow \dots aB \in P$

insert(A, a)

while 栈非空 do

begin

pop(A, a)

for $\forall C \rightarrow \dots A \in P$ do

insert(C, a)

end

for $\forall A \in V$ do

$\text{LASTOP}(A) = \emptyset$

for $\forall (A, a) \in V \times T$ do

begin

if $F[A, a]$ then

$\text{LASTOP}(A) = \text{LASTOP}(A) \cup \{a\}$

end

end

5.2.3 算符优先分析算法

■ 原理

- 移入-归约分析程序的一种，不断移入符号，通过**优先关系**来识别句柄并归约
- 各种优先关系存放在算符优先分析表
- 利用 \succ 识别句柄尾，利用 \prec 识别句柄头
- 分析栈存放已识别部分，**比较栈顶终结符号和下一输入符号的关系**
- 如果是句柄尾，则**沿栈顶向下**寻找句柄头，找到后弹出句柄，归约为非终结符。

**例5.7 $E \rightarrow E+T|E-T|T$ $T \rightarrow T * F|T / F|F$ $F \rightarrow (E)|id$,
试利用算符优先分析法对 $id+id$ 进行分析**

步骤	栈	输入串	优先关系	动作
1	#	$id_1+id_2\#$	$\# \lessdot id_1$	移进 id_1
2	# id_1	$+id_2\#$	$\# \lessdot id_1 \gtrdot +$	用 $F \rightarrow id$ 归约
3	#F	$+id_2\#$	\lessdot	移进+
4	#F+	$id_2\#$	\lessdot	移进 id_2
5	#F+ id_2	#	$+ \lessdot id_2 \gtrdot \#$	用 $F \rightarrow id$ 归约
6	#F+F	#	$\# \lessdot + \gtrdot \#$	用 $E \rightarrow E+T$ 归约
7	#E	#		

**例5.7 $E \rightarrow E+T|E-T|T$ $T \rightarrow T*F|T/F|F$ $F \rightarrow (E)|id$,
试利用算符优先分析法对 $id+id$ 进行分析**

步骤	栈	输入串	优先关系	动作
1	#	id ₁		
2	# id ₁			归约
3	#F	+		
4	#F+	id ₂		
5	#F+ id ₂			id归约
6	#F+F	#	# < + > #	用 $E \rightarrow E+T$ 归约
7	#E	#		

问题

1. 有时未归约真正的句柄
2. 不是严格的最左归约
3. 归约的符号串有时和相应产生式的右部不同



5.2.3 算符优先分析算法

- 有时未归约真正的句柄
- 不是严格的最左归约
- 归约的符号串有时与产生式右部不同
- 仍能正确识别句子的原因
 - 算符优先文法未定义非终结符之间的优先关系，不能识别由单非终结符组成的句柄
 - 为和严格意义上的“句柄”区分，定义算符优先分析过程识别的“句柄”为最左素短语
LPP (Leftmost Prime Phrase)



5.2.3 算符优先分析算法

- **素短语**是一个短语，**至少包含一个终结符**，除自身外不再包括其他含终结符的短语
- $S \Rightarrow^* \alpha A \beta$ and $A \Rightarrow^+ \gamma$ ， γ **至少含一个终结符**，且不含更小的含终结符的短语，则称 γ 是句型 $\alpha\gamma\beta$ 的相对于变量 A 的素短语(Prime Phrase)

5.2.3 算符优先分析算法

■ 例: $E \rightarrow E+T \mid T$ $T \rightarrow T * F \mid F$ $F \rightarrow (E) \mid id$

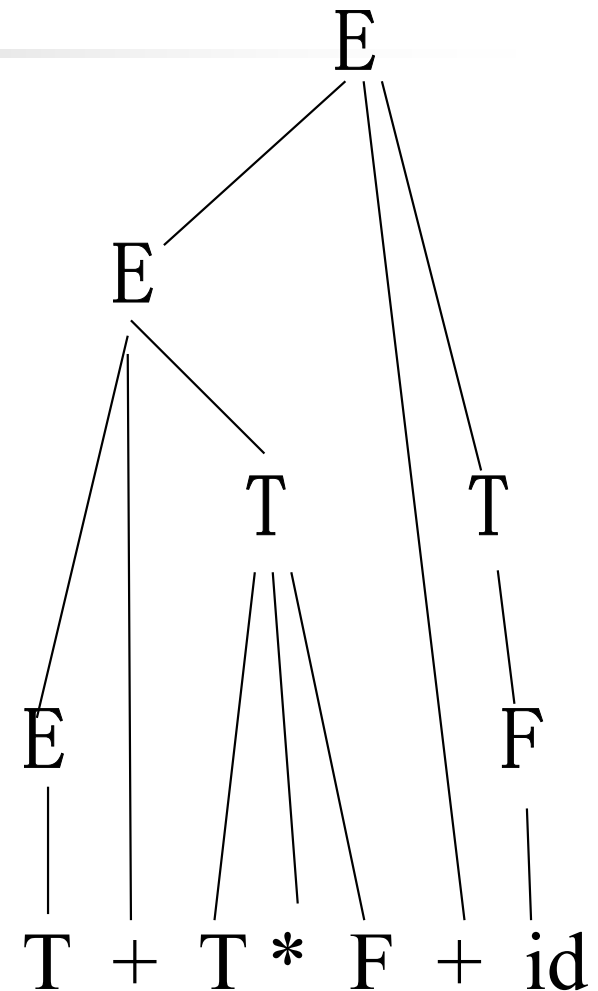
句型 $T+T * F+id$ 的短语有

T $T * F$ id $T+T * F$ $T+T * F+id$

其中的素短语为

$T * F$ id

$T * F$ 为最左素短语, 是被归约的对象



5.2.3 算符优先分析算法

文法: $E \rightarrow E + E \mid E * E$

$E \rightarrow (E) \mid \text{id}$

句型 $\text{id} + E * \text{id} + \text{id}$

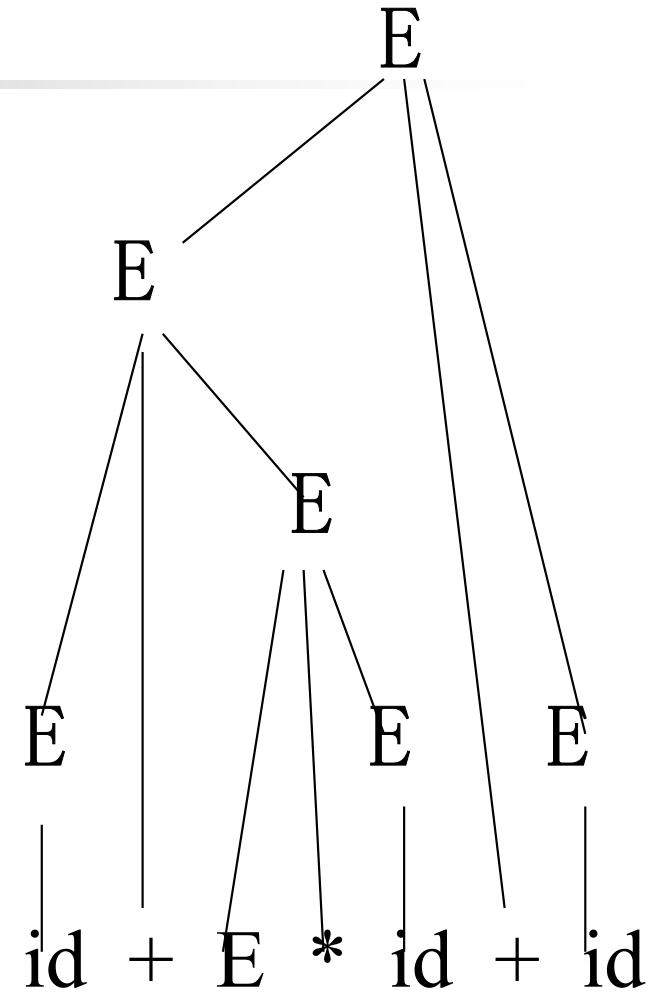
其中的短语为

$\text{id} \quad \text{id} \quad E * \text{id} \quad \text{id}$

$\text{id} + E * \text{id} \quad \text{id} + E * \text{id} + \text{id}$

其中的素短语为

$\text{id} \quad \text{id} \quad \text{id}$



问题: 归约过程中如何发现 “中间句型”
的最左素短语?

5.2.3 算符优先分析算法

给定算符优先文法，其句型的一般形式为

$\#N_1a_1N_2a_2\cdots N_na_nN_{n+1}\#$ ($N_i \in V \cup \{\varepsilon\}, a_i \in T$)

它的最左素短语是满足下列条件的**最左子串**

$$N_ia_iN_{i+1}a_{i+1}\cdots N_ja_jN_{j+1}$$

其中： $a_{i-1} \lessdot a_i$

$$a_i \equiv a_{i+1} \equiv \cdots \equiv a_{j-1} \equiv a_j$$

$$a_j \rhd a_{j+1}$$

5.2.3 算符优先分析算法

$N_1=F, a_1=+, N_2=F, N_1a_1N_2$
就是选定的最左素短语

6	#F+F	#	# $\leftarrow + \rightarrow$ #	用 $E \rightarrow E+T$ 归约
7	#E	#		

5.2.3 算符优先分析算法

设句型的一般形式为 $\#N_1a_1 N_2a_2 \dots N_na_n \#$, 它的最左素短语 $N_ia_i N_{i+1}a_{i+1} \dots N_ja_j N_{j+1}$, 规约为哪个语法变量:

1) 在文法中寻找这样的产生式, 其右部形如

$U_ia_i U_{i+1}a_{i+1} \dots U_ja_j U_{j+1}$ 的结构

2) 每个终结符与最左素短语中对应位置的终结符相同, 而每个语法变量只要与最左素短语相对应位置的语法变量对应即可 (看作匿名语法变量)

5.2.3 算符优先分析算法

$E \rightarrow E+T$ 的右部与 $F+F$ 对应，
所以 $F+F$ 被归约为 E

6	#F+F	#	# $\leftarrow + \rightarrow$ #	用 $E \rightarrow E+T$ 归约
7	#E	#		



5.2.3 算符优先分析算法

算符优先分析的实现

- **系统组成：移进归约分析器 + 优先关系表**
- **分析算法：根据输入串、优先关系表，完成一系列归约，生成语法分析树—输出产生式**

算法5.3 算符优先分析算法。

输入：文法 G ，输入字符串 w 和优先关系表 f ， S 是用到的分析栈；

输出：如果 w 是一个句子则输出一个分析树架子，否则指出错误；

步骤：

begin $S[1] := \text{'\#'}; i := 1;$

repeat 将下一输入符号读入 R ;

if $S[i] \in T$ then $j := i$ else $j := i - 1$;

while $S[j] \succ R$ do

begin

repeat $Q := S[j]$;

if $S[j-1] \in T$ then $j := j - 1$ else $j := j - 2$

until $S[j] \preccurlyeq Q$;

将 $S[j+1] \dots S[i]$ 归约为 N ; $i := j + 1; S[i] := N$

end;

移入操作

if $S[j] \preccurlyeq R$ or $S[j] \equiv R$ then begin $i := i + 1; S[i] := R$ end

else error

until $i = 2$ and $R = \text{'\#'}$ end;

$S[i]$ 表示栈顶符号，
 $S[j]$ 表示栈顶终结符号

表示栈内已出现最左素短语

完成归约操作， $S[i]$ 指向当前的栈顶变量

5.2.4 优先函数

- 给定文法G，如果G含有 n 个终结符，那么其算符优先分析表的表项数量是 n^2 ，当 n 较大时，空间消耗较大，优先级比较的效率也较低
- 在寻找最左素短语的尾时，比较当前栈顶算符的优先级（可看作算符在**栈内的优先级**）和当前待读入算符的优先级（可看作算符在**栈外的优先级**）



5.2.4 优先函数

- 为了节省存储空间 ($n^2 \rightarrow 2n$) 和便于执行比较运算, 用两个优先函数 f 和 g (f 是栈内优先数, g 是栈外优先数), 来表示算符之间的优先关系。
- 对于终结符号 a 和 b 选择 f 和 g , 使之满足:
 - 如果 $a \prec b$, 则 $f(a) < g(b)$
 - 如果 $a \equiv b$, 则 $f(a) = g(b)$
 - 如果 $a \succ b$, 则 $f(a) > g(b)$ 。
- 损失: 错误检测能力降低, 如: $id \succ id$ 不存在, 但 $f(id) > g(id)$ 可比较

例 5.6 表达式文法的算符优先关系

	+	-	*	/	()	id	#
+	\succ	\succ	\prec	\prec	\prec	\succ	\prec	\succ
-	\succ	\succ	\prec	\prec	\prec	\succ	\prec	\succ
*	\succ	\succ	\succ	\succ	\prec	\succ	\prec	\succ
/	\succ	\succ	\succ	\succ	\prec	\succ	\prec	\succ
(\prec	\prec	\prec	\prec	\prec	\equiv	\prec	
)	\succ	\succ	\succ	\succ		\succ		\succ
id	\succ	\succ	\succ	\succ		\succ		\succ
#	\prec	\prec	\prec	\prec	\prec		\prec	acc

5.2.4 优先函数

对应的优先函数:

	+	-	*	/	()	id	#
f	2	2	4	4	0	4	4	0
g	1	1	3	3	5	0	5	0

- 1) 构造优先函数的算法不是唯一的。
- 2) 存在一组优先函数，那就存在无穷组优先函数。

算法5.4 优先函数的构造。

输入：算符优先矩阵；

输出：表示输入矩阵的优先函数，或指出其不存在；

步骤：

1. 对 $\forall a \in T \cup \{\#\}$ ，建立以 fa 和 ga 为标记的顶点；
2. 对 $\forall a, b \in T \cup \{\#\}$ ，若 $a \succ b$ 或者 $a \equiv b$ ，则从 fa 至 gb 画一条有向弧；若 $a \prec b$ 或者 $a \equiv b$ ，则从 gb 至 fa 画一条有向弧；
3. 如果构造的有向图中有环路，则说明不存在优先函数；如果没有环路，则对 $\forall a \in T \cup \{\#\}$ ，将 $f(a)$ 设为从 fa 开始的最长路径的长度，将 $g(a)$ 设为从 ga 开始的最长路径的长度。

例5.10

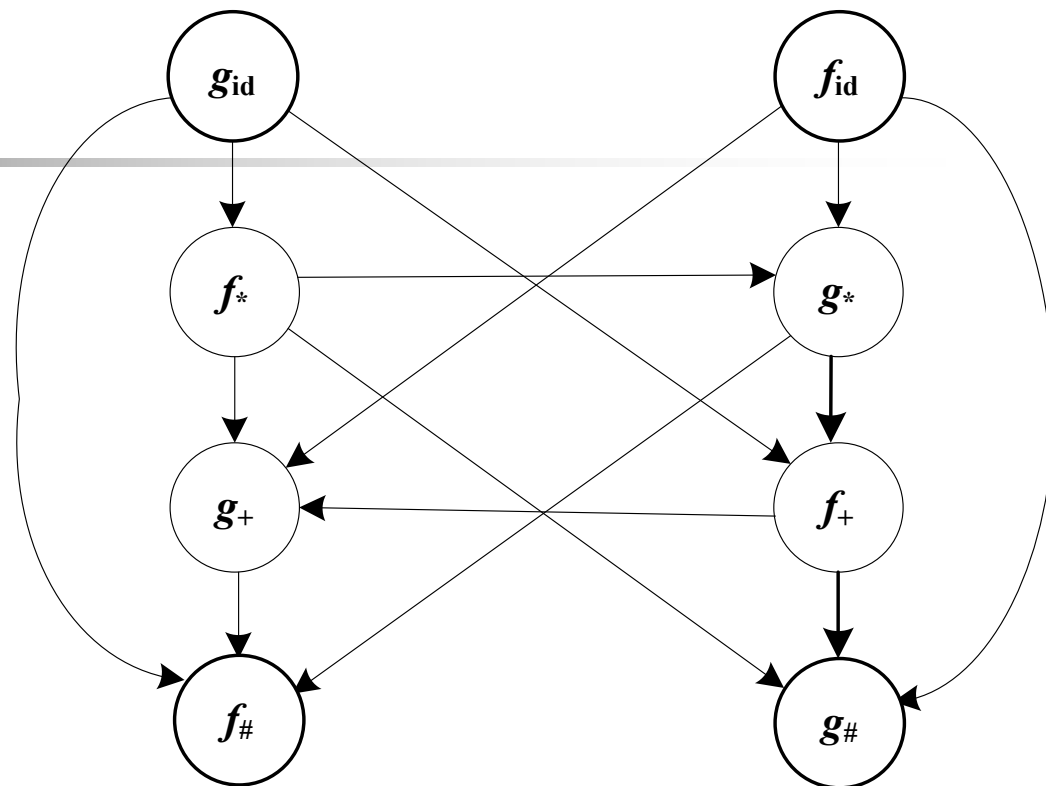
$G_{es} : E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow \text{id}$

	+	*	id	#
+	\triangleright	\triangleleft	\triangleleft	\triangleright
*	\triangleright	\triangleright	\triangleleft	\triangleright
id	\triangleright	\triangleright		\triangleright
#	\triangleleft	\triangleleft	\triangleleft	

G_{es} 的优先矩阵



	+	*	id	#
f	2	4	4	0
g	1	3	5	0

根据 G_{es} 的优先矩阵建立的
有向图和优先函数

5.2.5 算符优先分析的出错处理

- (1) 栈顶终结符号和当前输入符号不存在任何优先关系;**
- (2) 发现被“归约对象”，但该“归约对象”不能满足归约要求，即不是任何产生式的右部。**
- 对于第(1)种情况，为了进行错误恢复，必须修改栈、输入或两者都修改。对于优先矩阵中的每个空白项，必须指定一个出错处理程序，而且同一程序可用在多个地方。**
- 对于第(2)种情况，由于找不到与“归约对象”匹配的的产生式右部，分析器可以继续将这些符号弹出栈，而不执行任何语义动作。**



算符优先分析法小结

■ 优点

- 简单、效率高
- 能够处理部分二义性文法

■ 缺点

- 文法书写限制大（算符之间优先关系的唯一性）
- 占用内存空间大
- 不规范、存在查不到的语法错误
- 算法在发现最左素短语的尾时，需要回头寻找对应的头



5.3 LR分析法

- **LR(k)分析法: 有效的自底向上语法分析技术**
 - **L: 自左到右扫描输入符号**
 - **R: 构造最右推导的逆过程 (即最左归约)**
 - **k: 超前读入的字符个数以便确定归约用的产生式**
 - **k=1时可以满足绝大多数高级语言编译程序的需要, 着重介绍LR(0)、SLR(1)和LR(1)方法**



5.3 LR分析法

■ LR分析的特点

- 规范规约
- 适用范围广，可识别所有上下文无关文法描述的
程序设计语言
- 分析速度快，准确定位错误
- 缺点：构造的工作量大，多采用自动生成（YACC）



5.3.1 LR分析算法

■ 基本原理

- 分析器将句柄的识别过程分为**若干状态**，根据当前的状态，至多向前查看**k个输入符号**，就可以确定是否找到**句柄**
- 如果找到句柄，则执行**归约**操作，进入下一个状态
- 如果未找到句柄，则**移进**输入符号，也进入下一个状态



5.3.1 LR分析算法

- **状态是句柄识别程度的描述**，每个状态识别句柄的一部分符号
- 一个长为 n 的句柄的识别需要 $n+1$ 个状态

5.3.1 LR分析算法

- 符号串的**前缀**是指符号串的任意首部
- 每个状态所识别的那一部分符号正好是**当前句型的一个前缀**，**这个前缀不含相应句型的句柄右部的任何符号**，将其称为规范句型的**活前缀**
- 如果已得到待规约的句柄，该活前缀也称为**可归前缀**

例：规范句型 $a\underline{Ab}cde$ (下划线为句柄)的可归前缀为 aAb ，活前缀为： ε, a, aA, aAb



5.3.1 LR分析算法

- **所有活前缀构成一个正则语言，对文法的识别就变成对规范句型的活前缀的识别**
- **利用有穷状态自动机识别**



5.3.1 LR分析算法

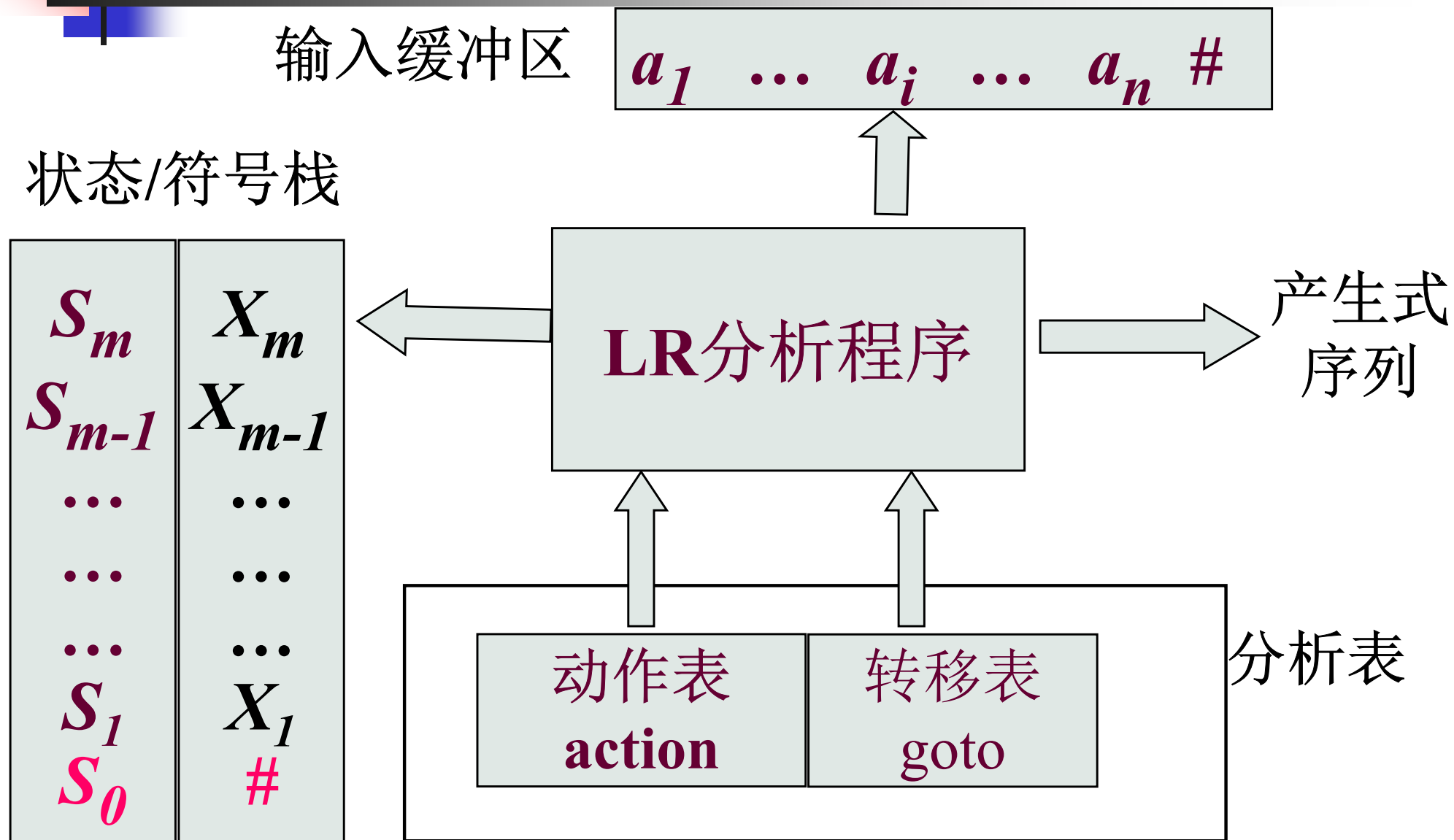
- 有穷状态自动机的状态转移函数包括两个表：动作表(action表)和转移表(goto表)，两个表统称为LR分析表
 - ACTION表示当前状态面临输入符号时应采取的动作
 - GOTO表示当前状态面临文法符号时应转向的下一个状态。



5.3.1 LR分析算法

- LR分析器包括一个**分析栈**
 - 文法符号栈：已经识别出的符号串
 - 状态符号栈：栈里的状态序列

5.3.1 LR分析算法



5.3.1 LR分析算法

- LR分析器的核心是action表和goto表组成的分析表
 - $\text{action}[S, a]$: 当栈顶状态为 S , 当前输入符号为 a 时分析器应执行的动作, 可能动作包括: 移进、归约、接受和报错
 - $\text{goto}[S, X]$: 当栈顶状态为 S , 且分析器刚归约出语法变量 X 时要转向的下一个状态

5.3.1 LR分析算法

约定:

s_j 表示将符号 a 、状态 j 压入栈

r_k 表示用第 k 个产生式进行归约

LR(0)、SLR(1)、
LR(1)、LALR(1)
将以不同的原则
构造这张分析表

状态	动作表 action			转移表 goto	
	a	b	#	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

LR分析器的工作过程 (5.3.1)

■ 分析器的格局 (三元组)

$(s_0s_1\dots s_m, \#X_1X_2\dots X_m, a_ia_{i+1}\dots a_n\#)$

□ $s_0s_1\dots s_m$ 表示状态符号栈里的状态序列

□ $\#X_1X_2\dots X_m$ 表示文法符号栈里已经识别出的符号串

□ $a_ia_{i+1}\dots a_n\#$ 表示剩余的输入符号串

如果输入句子是所分析语言的一个句子，那么 $X_1X_2\dots X_m a_ia_{i+1}\dots a_n$ 代表一个规范句型

□ $X_1X_2\dots X_m$ 是该规范句型的活前缀

LR分析器的工作过程 (5.3.1)

■ 分析器的格局 (三元组)

$(s_0s_1 \dots s_m, \#X_1X_2 \dots X_m, a_ia_{i+1} \dots a_n\#)$

■ 以分析栈的格式表示为

$s_0s_1 \dots s_m$

$\#X_1 \dots X_m$

$a_ia_{i+1} \dots a_n\#$

LR分析器的工作过程 (5.3.1)

1. 初始化, 初始状态 s_0 和#压入栈中

s_0

$a_1 a_2 \dots a_n$ # 对应 “句型” $a_1 a_2 \dots a_n$

2. 在一般情况下, 假设分析器的格局如下:

$s_0 s_1 \dots s_m$

$X_1 \dots X_m a_i a_{i+1} \dots a_n$ # 对应 “句型” $X_1 \dots X_m a_i a_{i+1} \dots a_n$

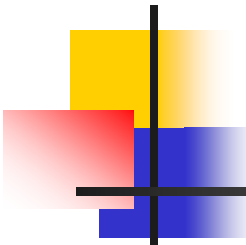
分析器根据当前输入符号 a_i 和栈顶状态 s_m 查分析表 $\text{action}[s_m, a_i]$, 包括四种不同动作

(5.3.1节)


$$s_0 s_1 \dots s_m$$
$$\#X_1 \dots X_m \quad a_i a_{i+1} \dots a_n \#$$

① If $\text{action}[s_m, a_i] = sj$ (移进) then 格局变为

$$s_0 s_1 \dots s_m \quad j$$
$$\#X_1 \dots X_m a_i \quad a_{i+1} \dots a_n \#$$



$$s_0 s_1 \dots s_m$$

$$\#X_1 \dots X_m \quad a_i a_{i+1} \dots a_n \#$$

(5.3.1节)

② If $\text{action}[s_m, a_i] = r_j$ (归约) then 表示用第j个产生式 $A \rightarrow X_{m-(k-1)} \dots X_m$ 进行归约, 格局变为

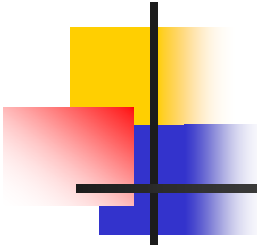
$$s_0 s_1 \dots s_{m-k}$$

$$\#X_1 \dots X_{m-k} A \quad a_i a_{i+1} \dots a_n \#$$

查goto表, 如果 $\text{goto}[s_{m-k}, A] = z$ then 格局变为

$$s_0 s_1 \dots s_{m-k} z$$

$$\#X_1 \dots X_{m-k} A \quad a_i a_{i+1} \dots a_n \#$$



$s_0 s_1 \dots s_m$
 $\#X_1 \dots X_m \quad a_i a_{i+1} \dots a_n \#$

(5.3.1节)

③ If $\text{action}[s_m, a_i] = \text{acc}$ then 分析成功

④ If $\text{action}[s_m, a_i] = \text{err}$ then 出现语法错误

LR分析算法

(5.3.1节)

算法5.5 LR分析算法。

输入：文法 G 的LR分析表和输入串 w ;

输出：如果 $w \in L(G)$ ，则输出 w 的自底向上分析，否则报错;

步骤：

1. 将 $\#$ 和初始状态 S_0 压入栈，将 $w\#$ 放入输入缓冲区;

2. 令输入指针 ip 指向 $w\#$ 的第一个符号;

3. 令 S 是栈顶状态， a 是 ip 所指向的符号;

4. repeat

5. if $action[S,a]=S_i$ then /* S_i 表示移进 a 并转入状态 i */

6. begin

7. 把符号 a 和状态 i 先后压入栈;

8. 令 ip 指向下一输入符号

9. end

```
10.  elseif action[S,a]=rk then
/* ri表示按第k个产生式 $A \rightarrow \beta$ 归约 */
11.      begin
12.          从栈顶弹出 $2*|\beta|$ 个符号;
13.          令 $S'$ 是现在的栈顶状态;
14.          把 $A$ 和 $goto[S',A]$ 先后压入栈中;
15.          输出产生式  $A \rightarrow \beta$ 
16.      end
17.  elseif action[S,a]= acc then
18.      return
19.  else
20.      error();
```



例5.12

分析表

文法

- 1) $S \rightarrow BB$
- 2) $B \rightarrow aB$
- 3) $B \rightarrow b$

识别bab 的过程
初始状态看作状态0

状态	动作表 action			转移表 goto	
	a	b	#	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



栈 输入 动作说明

0
bab# action(0,b)=s4
04
#b ab# action(4,a)=r3
0
#B ab# goto(0,B)=2
02
#B ab# action(2,a)=s3
023
#Ba b# action(3,b)=s4
0234
#Bab # action(4,#)=r3
023
#BaB # goto(3,B)=6

bab 的分析过程:

- 1) $S \rightarrow BB$
- 2) $B \rightarrow aB$
- 3) $B \rightarrow b$

0236
#BaB # action(6,#)=r2
02
#BB # goto(2,B)=5
025
#BB # action(5,#)=r1
0
#S # goto(0,S)=1
01
#S # action(1,#)=acc



5.3.2 LR(0)分析表的构造

- LR(0)分析法**不需要**向前查看输入符号，只需要根据当前的**栈顶状态**就可以确定下一步所应采取的动作

5.3.2 LR(0)分析表的构造

- 分析栈中内容+剩余输入符号=规范句型
 - 分析栈中内容为某一句型的前缀
- 来自分析栈的活前缀(Active Prefix)
 - 不含句柄右侧任意符号的规范句型前缀
- 例: $id + id * id$ 的分析中
 - 句型 $E + id . * id$ 和 $E + E * . id$
 - 活前缀
 - 活前缀

5.3.2 LR(0)分析表的构造

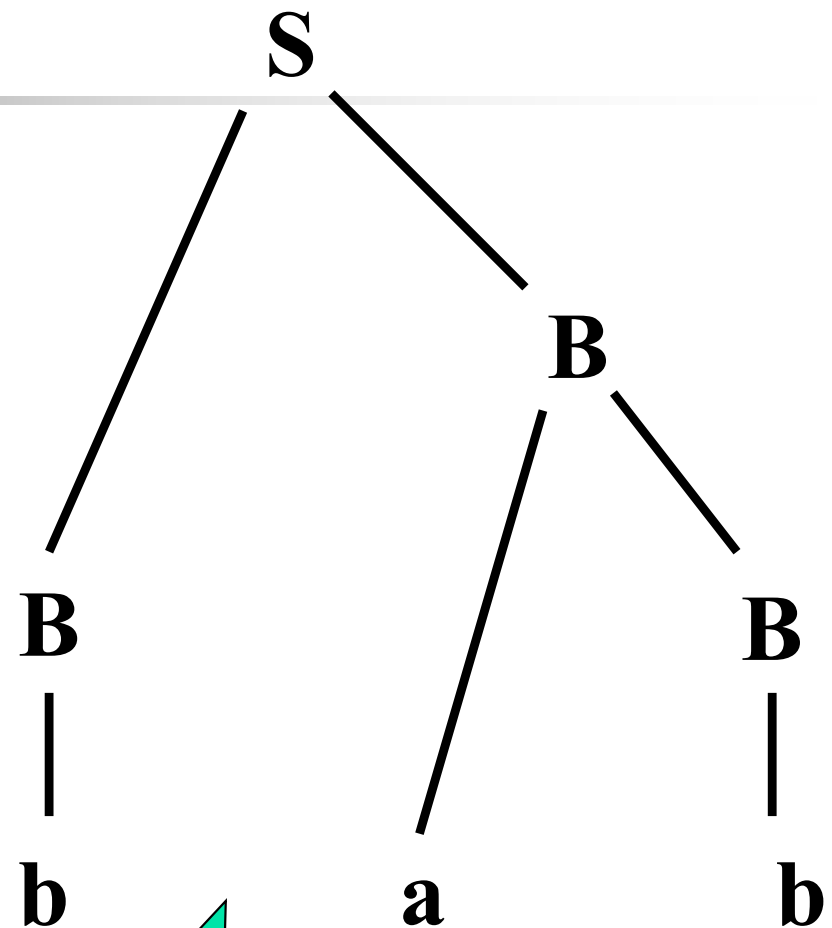
- 规范句型的活前缀是出现在**分析栈中的符号串**，并且不会出现句柄之后的任何字符
- 相应的后缀正是输入串中还未处理的终结符号串
- 活前缀与句柄的关系
 - 包含句柄 $A \rightarrow \beta$.
 - 包含句柄的部分符号 $A \rightarrow \beta_1 \cdot \beta_2$
 - 不含句柄的任何符号 $A \rightarrow \cdot \beta$

5.3.2 LR(0)分析表的构造

- LR(0)项目——从产生式寻找归约方法
 - 定义5.3 右部某个位置标有圆点的产生式称为相应文法的**LR(0)项目** (Item)
 - 例 $S \rightarrow .bBB$ $S \rightarrow b.BB$ $S \rightarrow bB.B$ $S \rightarrow bBB.$
 - 归约 (Reduce) 项目: $S \rightarrow bBB.$
 - 移进 (Shift) 项目: $S \rightarrow .bBB$
 - 待约项目: $S \rightarrow b.BB$ $S \rightarrow bB.B$

项目的意义

- 项目表示分析的进程
(即句柄的识别状态)
- 方法：在产生式右部
加一圆点来分割已获
取的内容和待获取的
内容



$S \rightarrow B . B$
 $B \rightarrow . a B$

5.3.2 LR(0)分析表的构造

- 用LR(0)项目构造识别规范句型活前缀的DFA
- 为了保证文法开始符号只出现在一个产生式的左边, **保证只有一个接受状态**, 需要对文法进行**拓广**, 增加一个0号产生式 $S' \rightarrow S$
- 文法 $G = (V, T, P, S)$ 的**拓广文法** G' :
 - $G' = (V \cup \{S'\}, T, P \cup \{S' \rightarrow S\}, S')$
 - $S' \notin V$
 - 对应 $S' \rightarrow .S$ (分析开始) 和 $S' \rightarrow S.$ (分析成功)



5.3.2 LR(0)分析表的构造

■ 例5.13

0) $S' \rightarrow S$

1) $S \rightarrow BB$

2) $B \rightarrow aB$

3) $B \rightarrow b$

求其全部LR(0)项目。



5.3.2 LR(0)分析表的构造

- **问题：如何设计一个装置，能够指导分析器运行，并且能够根据当前状态（栈顶）确定句柄**

**构造识别文法G的所有规范句型
活前缀的DFA**



5.3.2 LR(0)分析表的构造

每个项目集闭包对应分析器的一个状态

定义**5.5** (闭包): 设**I**是文法**G = (V, T, P, S)**的一个**LR(0)**项目集, 则**I**的闭包(**closure**)定义如下:

CLOSURE(I)

$= I$

$\cup \{A \rightarrow \cdot \alpha \mid \exists B \rightarrow \gamma \cdot A\beta \in CLOSURE(I) \& A \rightarrow \alpha \in P, \alpha, \beta, \gamma \in (V \cup T)^*, A, B \in V\}$



5.3.2 LR(0)分析表的构造

计算闭包的函数

function CLOSURE(I)

begin

J := I

repeat

for J中每一个形如 $\mathbf{B} \rightarrow \alpha.A\beta$ **的项目 do**

for G'中每一个形如 $\mathbf{A} \rightarrow \gamma$ **的产生式 do**

if $\mathbf{A} \rightarrow .\gamma$ **不在J中**

将 $\mathbf{A} \rightarrow .\gamma$ **加入J中**

until J不再增大

return J

end;



5.3.2 LR(0)分析表的构造

例5.13

- 0) $S' \rightarrow S$
- 1) $S \rightarrow BB$
- 2) $B \rightarrow aB$
- 3) $B \rightarrow b$

I_0 :表示**DFA**的初始状态，记为状态**0**，其包括**LR(0)**项目 **$S' \rightarrow .S$** ，表示等待归约出**S**，但是目前尚未得到**S**的任何符号， **$I_0 = \text{CLOSURE}(S' \rightarrow .S)$**

如何确定从状态**0**可能转移到的下一个状态呢？



5.3.2 LR(0)分析表的构造

- 后继项目 (Successive Item)

- $A \rightarrow \alpha.X\beta$ 的后继项目是 $A \rightarrow \alpha X.\beta$

- 闭包之间的转移

- 项目集 I 的关于 X 的后继项目集

$$GO(I, X) = CLOSURE(\{A \rightarrow \alpha X.\beta \mid A \rightarrow \alpha.X\beta \in I\})$$



5.3.2 LR(0)分析表的构造

- 确定在某状态遇到一个文法符号后的状态转移目标

```
function GO(I, X);  
begin  
  J:= $\emptyset$ ;  
  for I中每个形如 $A \rightarrow \alpha.X\beta$ 的项目do  
    begin J:=J  $\cup$  { $A \rightarrow \alpha X.\beta$ } end;  
  return CLOSURE(J)  
end;
```

5.3.2 LR(0)分析表的构造

- 识别文法 $G = (V, T, P, S)$ 的拓广文法 G' 的所有规范句型活前缀的DFA :

$$M = (C, V \cup T, GO, I_0, C)$$

- $I_0 = \text{CLOSURE}(\{S' \rightarrow \cdot S\})$
- $C = \{I_0\} \cup \{I \mid \exists J \in C, X \in V \cup T, I = GO(J, X)\}$

称为 G' 的**LR(0)项目集规范族**

5.3.2 LR(0)分析表的构造

计算LR(0)项目集规范族 C

即：分析器状态集合

begin

$C := \{\text{closure}(\{ S' \rightarrow .S \})\};$

repeat

for $\forall I \in C, \forall X \in V \cup T$

if $GO(I, X) \neq \Phi$ & $GO(I, X) \notin C$ then

$C = C \cup \{GO(I, X)\}$

until C不变化

end.



识别拓广文法所有规范句型活前缀的DFA

例5.13

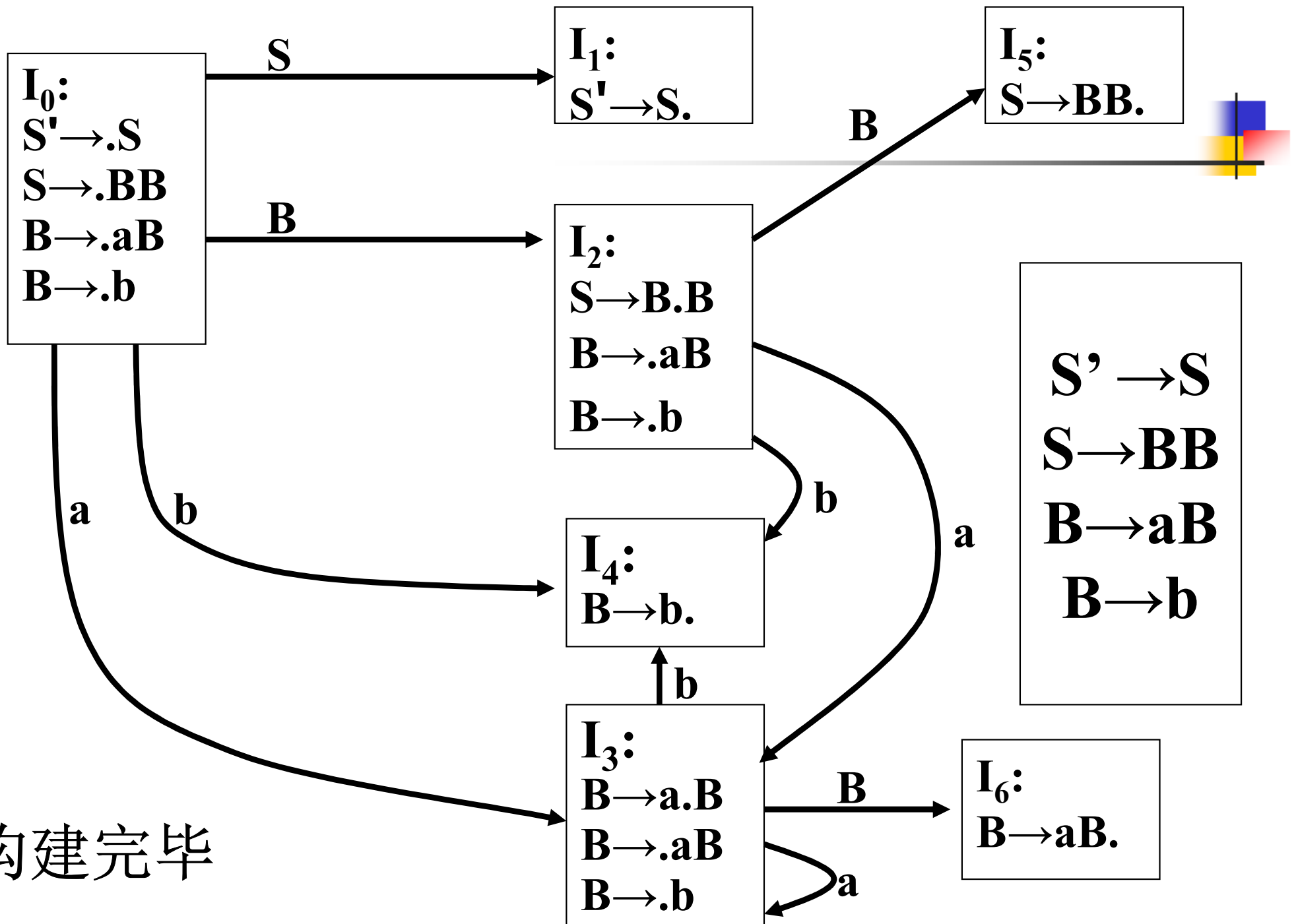
0) $S' \rightarrow S$

1) $S \rightarrow BB$

2) $B \rightarrow aB$

3) $B \rightarrow b$

如何构建识别该文法所有活前缀的**DFA**?



构建完毕

LR(0)分析表的构造算法

算法5.6 LR(0)分析表的构造。

输入：文法 $G=(V, T, P, S)$ 的拓广文法 G' ;

输出： G' 的LR(0)分析表，即action表和goto表;

步骤:

1. 令 $I_0 = \text{CLOSURE}(\{S' \rightarrow .S\})$ ，构造 G' 的LR(0)项目集规范族 $C = \{I_0, I_1, \dots, I_n\}$

2. 让 I_i 对应状态 i ， I_0 对应状态0，0为初始状态。

3. for $k=0$ to n do begin

(1) if $A \rightarrow \alpha.a\beta \in I_k$ & $a \in T$ & $\text{GO}(I_k, a) = I_j$ then $\text{action}[k, a] := Sj$;

(2) if $A \rightarrow \alpha.B\beta \in I_k$ & $B \in V$ & $\text{GO}(I_k, B) = I_j$ then $\text{goto}[k, B] := j$;

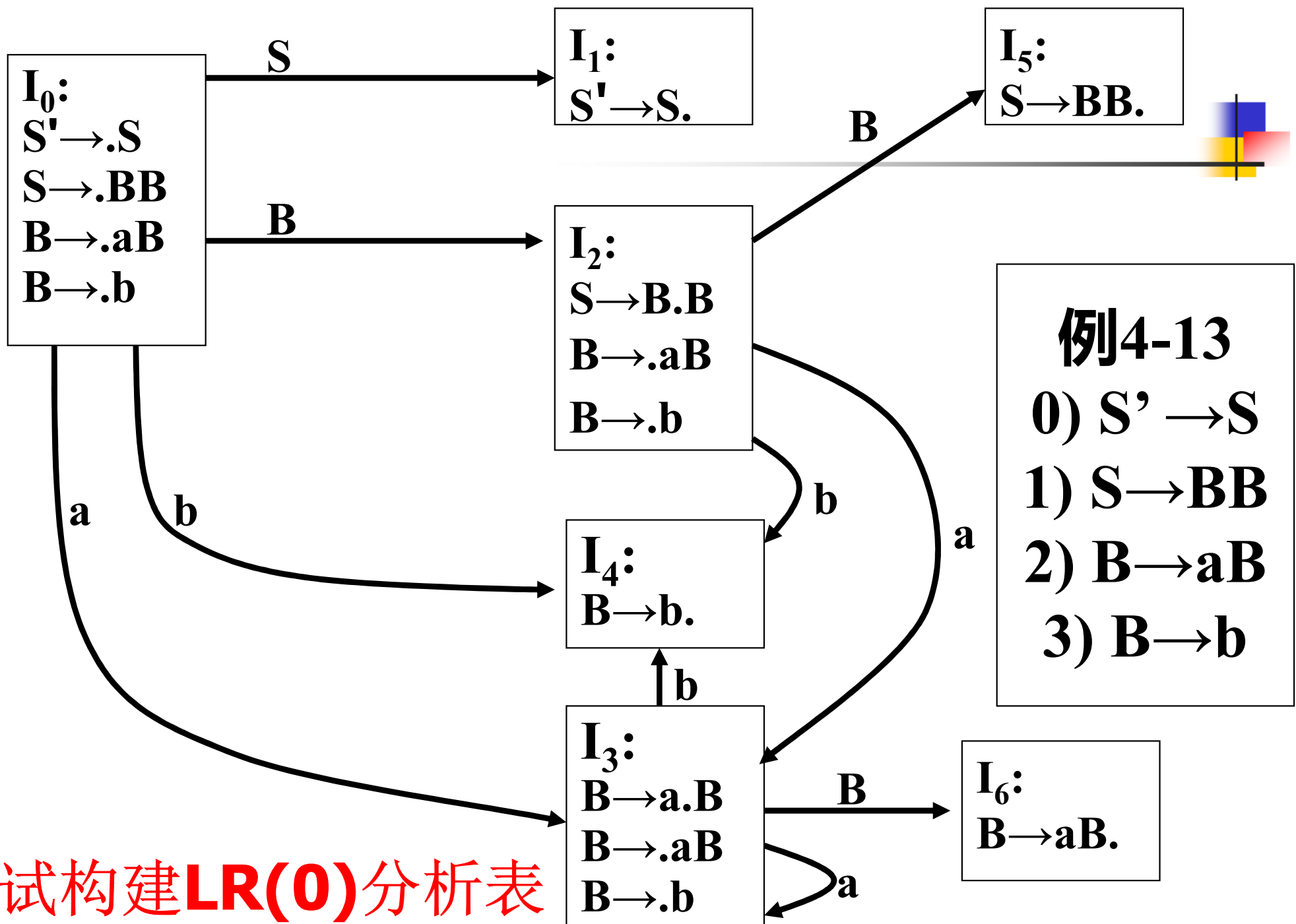
(3) if $A \rightarrow \alpha. \in I_k$ & $A \rightarrow \alpha$ 为 G 的第 j 个产生式 then

for $\forall a \in T \cup \{\#\}$ do $\text{action}[k, a] := rj$;

(4) if $S' \rightarrow S. \in I_k$ then $\text{action}[k, \#] := \text{acc}$ end;

4. 上述(1)到(4)步未填入信息的表项均置

只要栈内出现句柄，
无论当前输入符号如何，
执行归约操作



例4-13

- 0) $S' \rightarrow S$
- 1) $S \rightarrow BB$
- 2) $B \rightarrow aB$
- 3) $B \rightarrow b$

尝试构建**LR(0)**分析表

例5.12

分析表

文法

- 1) $S \rightarrow BB$
- 2) $B \rightarrow aB$
- 3) $B \rightarrow b$

状态	动作表 action			转移表 goto	
	a	b	#	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



LR(0)不是总有效的

($S' \rightarrow S$)

1) $S \rightarrow A|B$

2) $A \rightarrow aAc$

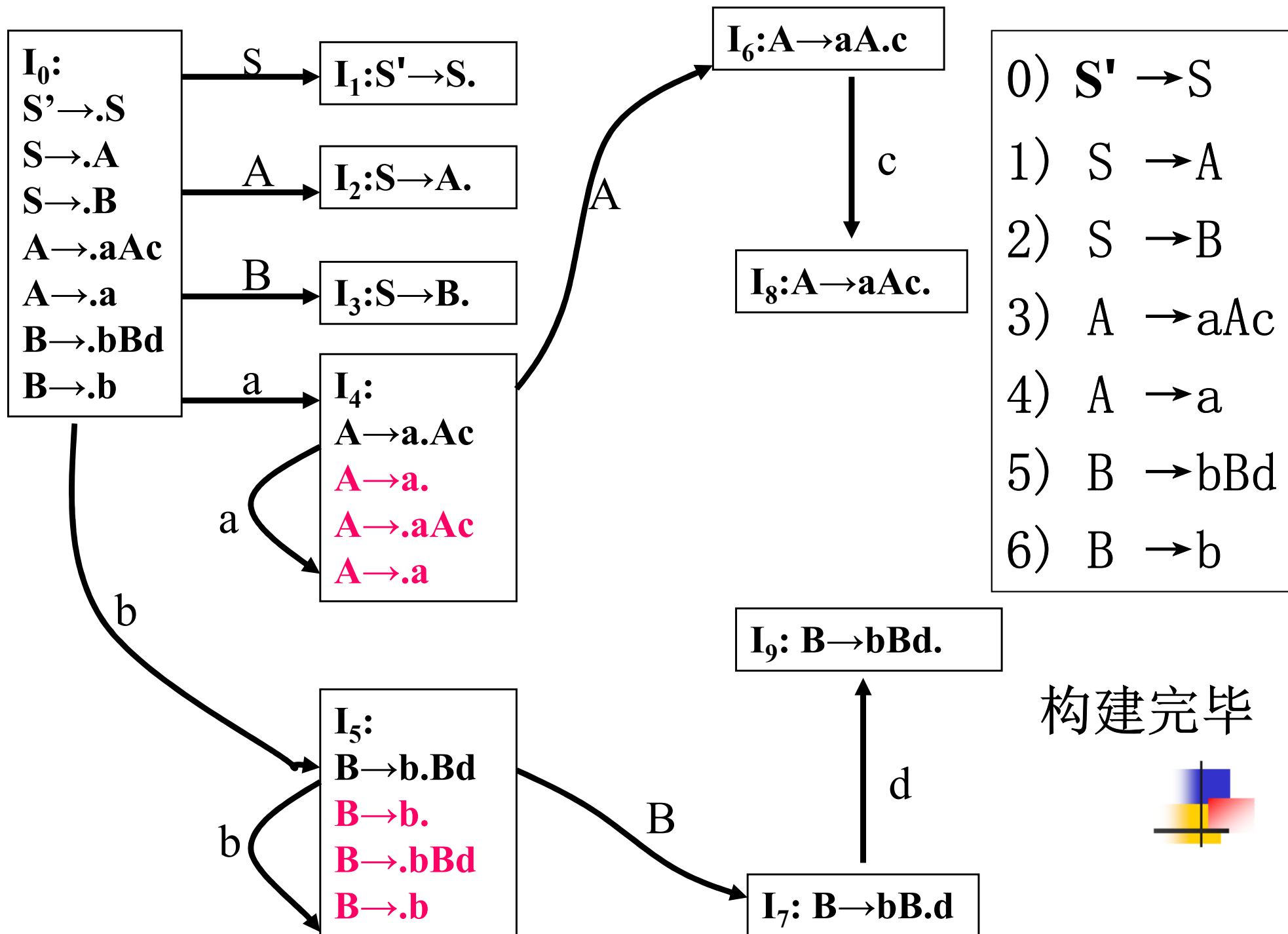
3) $A \rightarrow a$

4) $B \rightarrow bBd$

5) $B \rightarrow b$

不是所有上下文无关文法都能用LR(0)方法进行分析

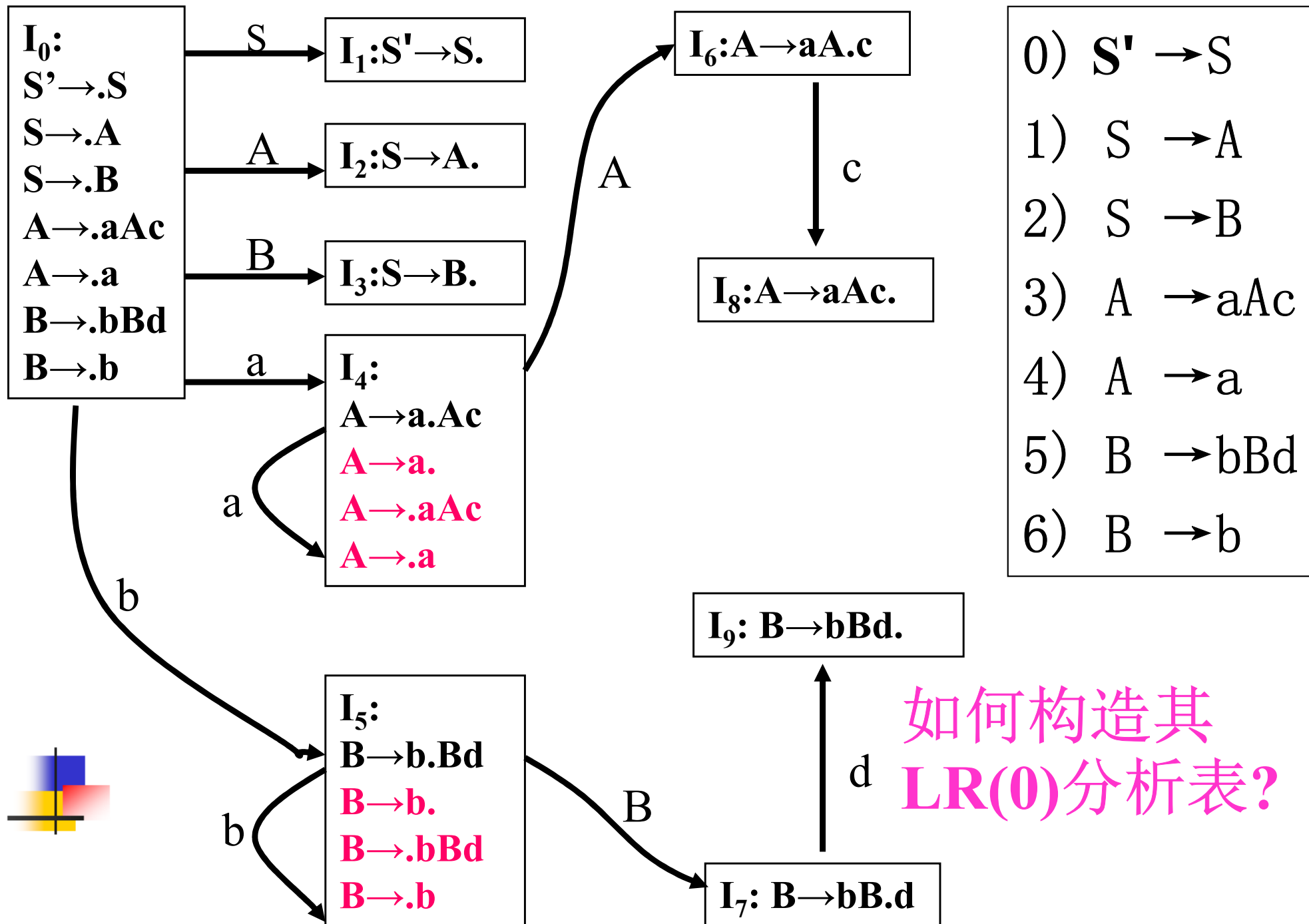
CFG不总是LR(0)文法.





项目集 I 的相容

- 如果 I 中至少含两个归约项目，则称 I 有**归约—归约冲突** (Reduce/Reduce Conflict)
- 如果 I 中既含归约项目，又含移进项目，则称 I 有**移进—归约冲突** (Shift/Reduce Conflict)
- 如果 I 既没有归约—归约冲突，又没有移进—归约冲突，则称 I 是**相容的** (Consistent)，否则称 I 是不相容的
- 对文法 G，如果 $\forall I \in C$ 都是相容的，则称 G 为**LR(0) 文法**



如何构造其
LR(0)分析表?

LR(0)分析表的构造算法

算法5.6 LR(0)分析表的构造。

输入：文法 $G=(V, T, P, S)$ 的拓广文法 G' ;

输出： G' 的LR(0)分析表，即action表和goto表;

步骤：

1. 令 $I_0 = \text{CLOSURE}(\{S' \rightarrow .S\})$ ，构造 G' 的LR(0)项目集规范族 $C = \{I_0, I_1, \dots, I_n\}$

2. 让 I_i 对应状态 i ， I_0 对应状态0，0为初始状态。

3. for $k=0$ to n do begin

(1) if $A \rightarrow \alpha.a\beta \in I_k$ & $a \in T$ & $\text{GO}(I_k, a) = I_j$ then $\text{action}[k, a] := Sj$;

(2) if $A \rightarrow \alpha.B\beta \in I_k$ & $B \in V$ & $\text{GO}(I_k, B) = I_l$ then $\text{goto}[k, B] := l$;

(3) if $A \rightarrow \alpha. \in I_k$ & $A \rightarrow \alpha$ 为 G 的第 j 个产生式

for $\forall a \in T \cup \{\#\}$ do $\text{action}[k, a] := rj$;

(4) if $S' \rightarrow S. \in I_k$ then $\text{action}[k, \#] := \text{acc}$

4. 上述(1)到(4)步未填入信息的表项均

问题出现原因：在LR(0)分析表的构造过程中，只要当前状态出现可以归约的项目，无论后面遇到什么符号都进行归约

LR(0)分析表的构造算法

算法5.6 LR(0)分析表的构造。

输入：文法 $G=(V, T, P, S)$ 的拓广文法 G' ;

输出： G' 的LR(0)分析表，即action表和goto表;

步骤：

1. 令 $I_0 = \text{CLOSURE}(\{S' \rightarrow .S\})$ ，构造 G' 的LR(0)项目集规范族 $C = \{I_0, I_1, \dots, I_n\}$

2. 让 I_i 对应状态 i ， I_0 对应状态0，0为初始状态。

3. for $k=0$ to n do begin

(1) if $A \rightarrow \alpha.a\beta \in I_k$ & $a \in T$ & $\text{GO}(I_k, a) = I_j$ then $\text{action}[k, a] := Sj$;

(2) if $A \rightarrow \alpha.B\beta \in I_k$ & $B \in V$ & $\text{GO}(I_k, B) = I_j$ then $\text{goto}[k, B] := j$;

(3) if $A \rightarrow \alpha. \in I_k$ & $A \rightarrow \alpha$ 为 G 的第 j 个产生式 then

for $\forall a \in T \cup \{\#\}$ do $\text{action}[k, a] := rj$;

(4) if $S' \rightarrow S. \in I_k$ then $\text{action}[k, \#] := \text{acc}$

4. 上述(1)到(4)步未填入信息的表项均

在产生移进-归约冲突时，确定应该优先读入符号还是优先进行归约

5.3.3 SLR(1)分析表的构造算法

输入：文法 $G=(V, T, P, S)$ 的拓广文法 G' ;

输出： G' 的 $LR(0)$ 分析表，即 $action$ 表和 $goto$ 表;

步骤：

1. 令 $I_0 = \text{CLOSURE}(\{S' \rightarrow .S\})$ ，构造 G' 的 $LR(0)$ 项目集规范族
 $C = \{I_0, I_1, \dots, I_n\}$

2. 让 I_i 对应状态 i ， I_0 对应状态0，0为初始状态

3. for $k=0$ to n do begin

(1) if $A \rightarrow \alpha.a\beta \in I_k$ & $a \in T$ & $\text{GO}(I_k, a) = I_j$ then

(2) if $A \rightarrow \alpha.B\beta \in I_k$ & $B \in V$ & $\text{GO}(I_k, B) = I_j$ then

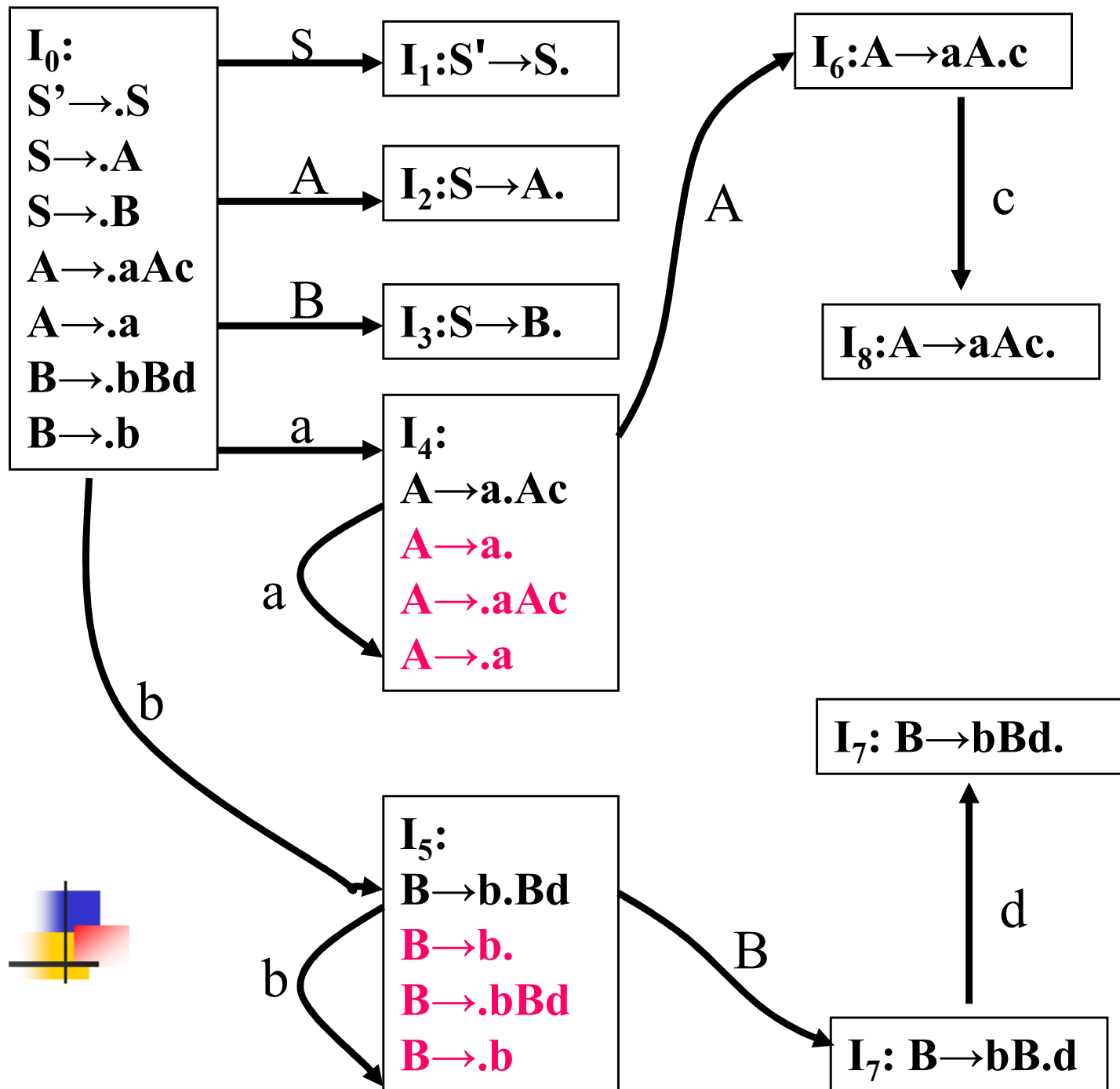
(3) if $A \rightarrow \alpha. \in I_k$ & $A \rightarrow \alpha$ 为 G 的第 j 个产生式 then

for $\forall a \in \text{FOLLOW}(A)$ do $action[k, a] := rj$;

(4) if $S' \rightarrow S. \in I_k$ then $action[k, \#] := acc$ end;

4. 上述(1)到(4)步未填入信息的表项均置为error。

解决办法：向前查看一个输入符号，判断当前是否可以归约



- 0) $S' \rightarrow S$
- 1) $S \rightarrow A$
- 2) $S \rightarrow B$
- 3) $A \rightarrow aAc$
- 4) $A \rightarrow a$
- 5) $B \rightarrow bBd$
- 6) $B \rightarrow b$

如何构造
其SLR(1)
分析表?



识别表达式文法的所有活前缀的DFA

拓广文法

0) $E' \rightarrow E$

1) $E \rightarrow E + T$

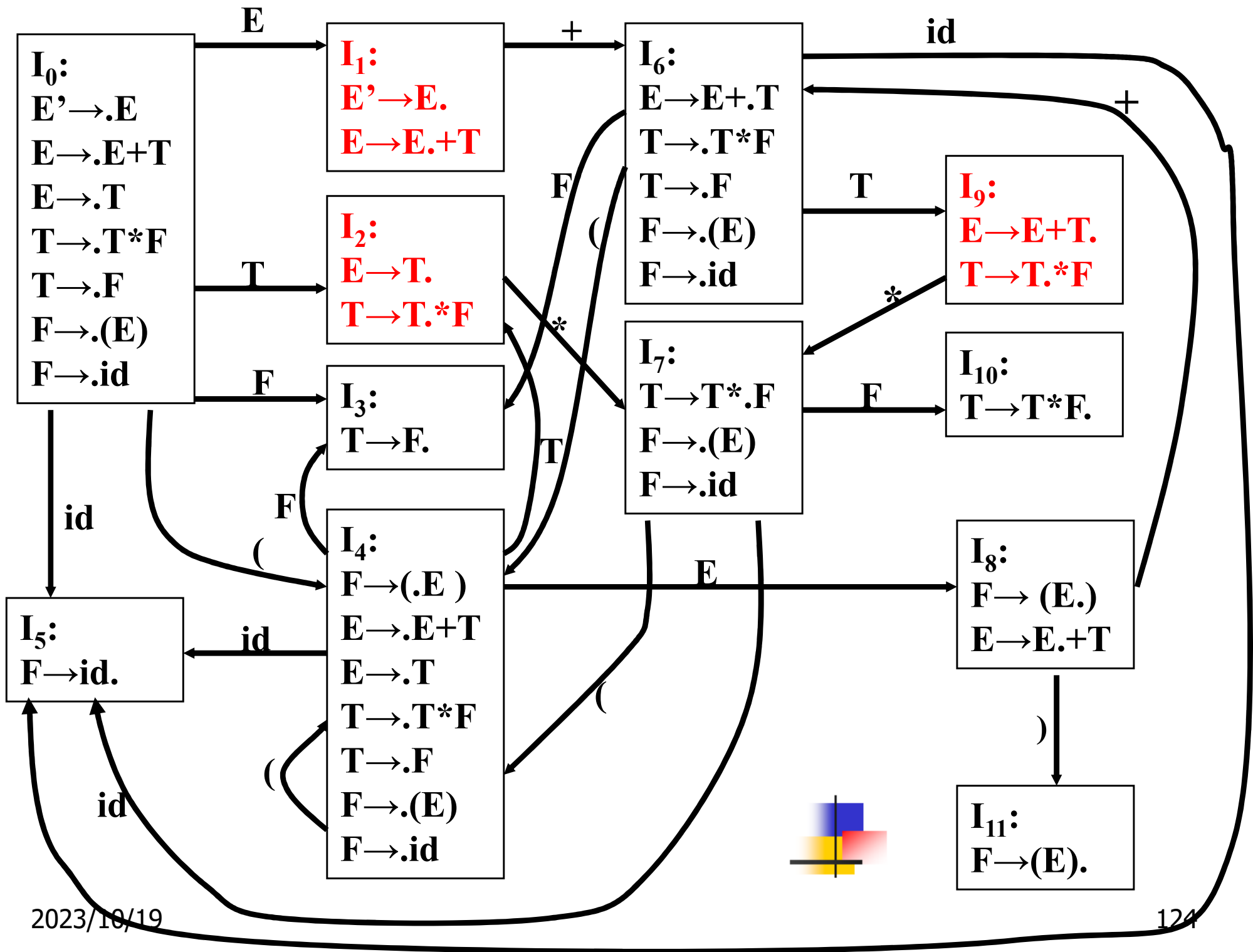
2) $E \rightarrow T$

3) $T \rightarrow T * F$

4) $T \rightarrow F$

5) $F \rightarrow (E)$

6) $F \rightarrow \text{id}$



表达式文法的 LR(0)分析表含有冲突

状态	ACTION					
	id	+	*	()	#
0	...					
1	r0	r0/s6	r0	r0	r0	r0
2	r2	r2	r2/s7	r2	r2	r2
3	r4	r4	r4	r4	r4	r4
	...					
5	r6	r6	r6	r6	r6	r6
	...					
9	r1	r1	r1/s7	r1	r1	r1
10	r3	r3	r3	r3	r3	r3
11	r5	r5	r5	r5	r5	r5

- 在状态1、2、9 采用归约，出现移进归约冲突

表达式文法的SLR(1)分析表

■ 求非终结符 FOLLOW 集

- $\text{FOLLOW}(E') = \{ \# \}$
- $\text{FOLLOW}(E) = \{), +, \# \}$
- $\text{FOLLOW}(T) = \{), +, \#, * \}$
- $\text{FOLLOW}(F) = \{), +, \#, * \}$

0) $E' \rightarrow E$

1) $E \rightarrow E + T$

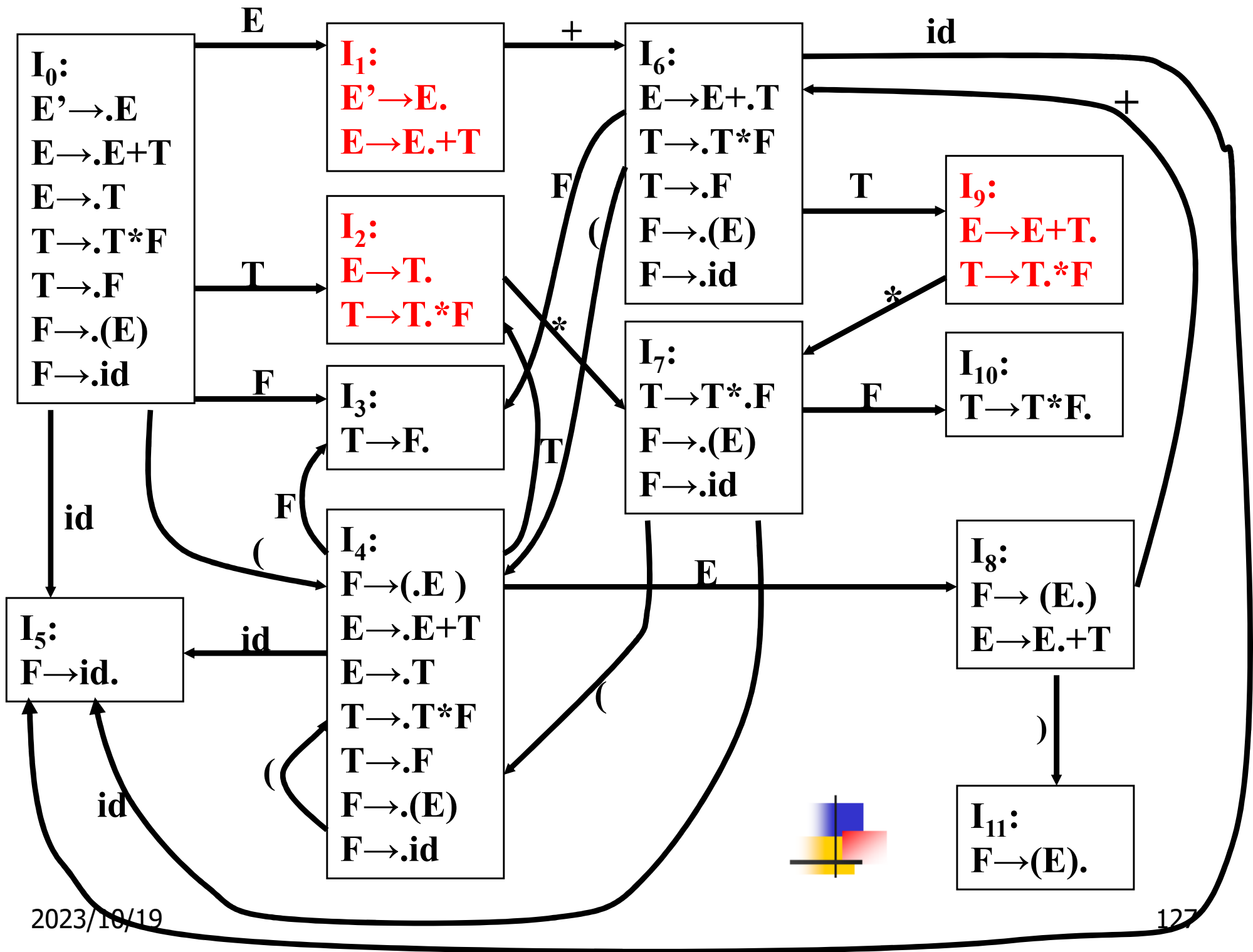
2) $E \rightarrow T$

3) $T \rightarrow T * F$

4) $T \rightarrow F$

5) $F \rightarrow (E)$

6) $F \rightarrow \text{id}$



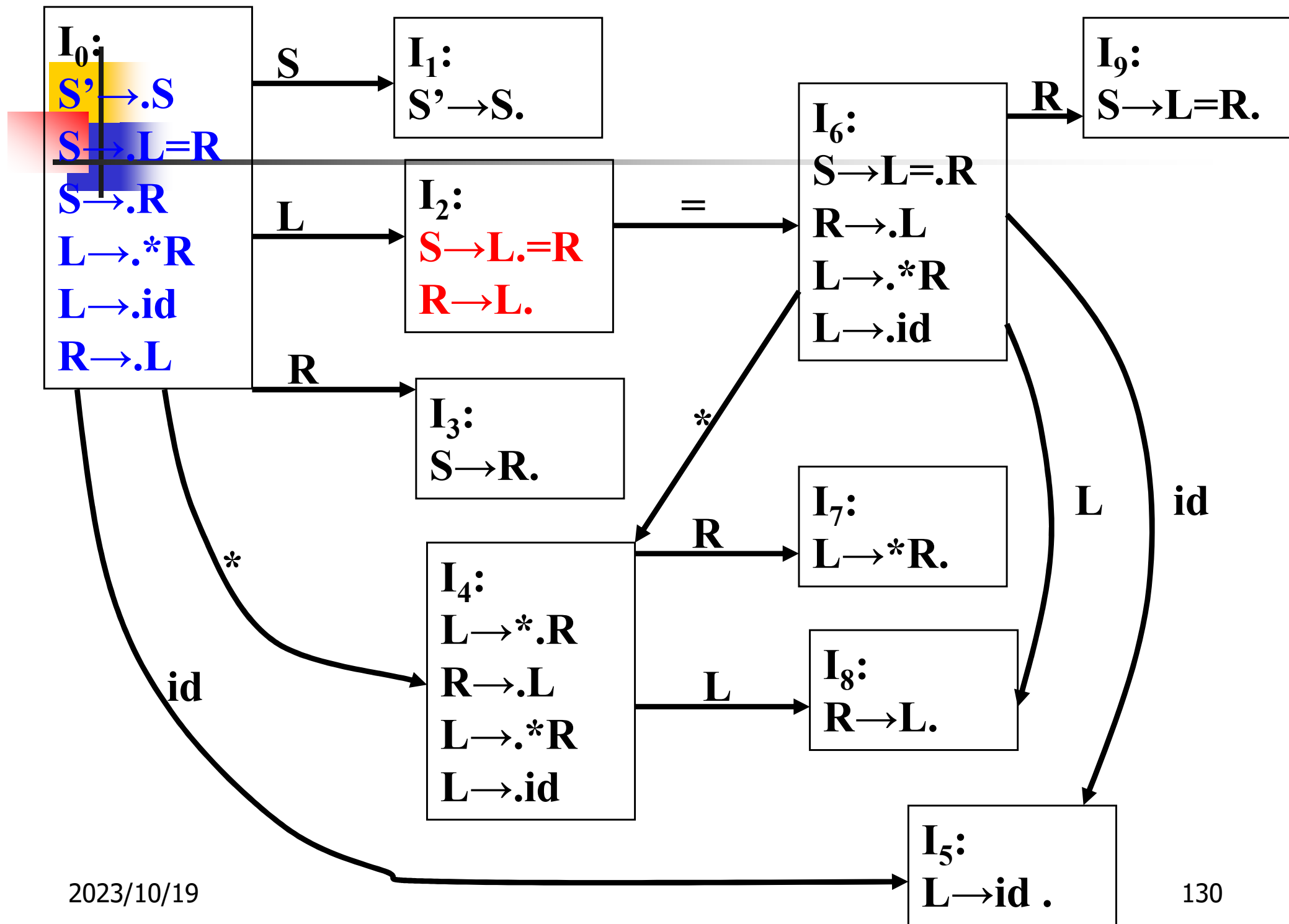


状态	ACTION						GOTO		
	id	+	*	()	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			



SLR(1)分析的局限性

- **SLR(1) 文法：SLR(1)分析表无冲突的CFG**
- **如果 SLR(1) 分析表仍有多重入口（移进归约冲突或归约归约冲突），则说明该文法不是 SLR(1) 文法；**
- **说明仅使用 LR(0) 项目集和 FOLLOW 集还不足以分析这种文法**



SLR分析中的冲突——需要更强的分析方法

$$I_2 = \{S \rightarrow L.=R, R \rightarrow L.\}$$

- 输入符号为 = 时，出现了移进归约冲突：

$$S \rightarrow L . = R \in I_2 \text{ and } \text{go}(I_2, =) = I_6$$

$$\Rightarrow \text{action}[2, =] = \text{Shift } 6$$

$$R \rightarrow L . \in I_2 \text{ and } = \in \text{FOLLOW}(R) = \{=, \# \}$$

$$\Rightarrow \text{action}[2, =] = \text{Reduce } R \rightarrow L$$

- 说明该文法不是SLR(1)文法，分析这种文法需要更多的信息。

SLR分析中存在冲突的原因

- SLR(1)只孤立地考察输入符号是否属于归约项目 $A \rightarrow \alpha$. 相关联的集合 $FOLLOW(A)$, 而没有考察符号串 α 所在规范句型的“**上下文**”。
- 试图用某一产生式 $A \rightarrow \alpha$ 归约栈顶符号串 α 时, 不仅要向前扫描一个输入符号, 还要查看**栈中的符号串 $\delta\alpha$** , 只有当 $\delta A \alpha$ 的确构成文法某一规范句型的活前缀时才能用 $A \rightarrow \alpha$ 归约。**亦即要考虑归约的有效性。**
- **问题: 怎样确定 $\delta A \alpha$ 是否是文法某一规范句型活前缀**



5.3.4 LR(1)分析表的构造

- LR(0)不考虑后继符(也叫搜索符)
- SLR(1)仅在归约时考虑后继符，对后继符所含信息量的利用有限，**未考虑栈中内容。**
- LR(1)希望在构造状态时就考虑后继符的作用

5.3.4 LR(1)分析表的构造

■ 定义5.11

- $[A \rightarrow \alpha.\beta, a_1a_2\dots a_k]$ 为 **LR(k)项目**，其中 $[A \rightarrow \alpha.\beta]$ 是一个 LR(0)项目， $a_1a_2\dots a_k$ 表示此 LR(k)项目的 **搜索符串**，**只对归约项目有意义**
- 归约项目: $[A \rightarrow \alpha., a_1a_2\dots a_k]$
- 移进项目: $[A \rightarrow \alpha.a\beta, a_1a_2\dots a_k]$
- 待约项目: $[A \rightarrow \alpha.B\beta, a_1a_2\dots a_k]$
- 利用 LR(k)项目进行 LR(k)分析，当 $k=1$ 时，为 LR(1)项目，相应的分析叫 LR(1)分析

5.3.4 LR(1)分析表的构造

- 形式上，称LR(1)项目 $[A \rightarrow \alpha.\beta, a]$ 对活前缀 $\delta\alpha$ 是有效的，如果存在规范推导 $S \Rightarrow^* \delta A w \Rightarrow \delta \alpha \beta w$ ，其中 a 为 $w\#$ 的首字符，如果 $w = \varepsilon$ ，则 $a = \#$
- 与LR(0)文法类似，识别文法全部活前缀的DFA的每一状态用一个LR(1)项目集来表示
- 为保证分析时每一步都在栈中得到规范句型的活前缀，应使每一个LR(1)项目集所包含的项目对相应活前缀都有效，即从开始只构建有效的LR(1)项目集。



5.3.4 LR(1)分析表的构造

■ LR(1) 项目集族的求法

- **CLOSURE(I):** 给定LR(1)项目I, 求I的闭包, 目的是为了合并某些状态, 节省空间
- **GO(I, X):** 转移函数

5.3.4 LR(1)分析表的构造

闭包的计算

- 如果 $[A \rightarrow \alpha.B\beta, a]$ 对 $\gamma = \delta\alpha$ 有效

/*即存在 $S \Rightarrow^* \delta A a x \Rightarrow \delta \alpha B \beta a x$ */

- 假定 $\beta a x \Rightarrow^* \mathbf{b} y$, 则对任意的 $B \rightarrow \eta$ 有: $[B \rightarrow \cdot \eta, \mathbf{b}]$ 对 $\gamma = \delta\alpha$ 也是有效的

5.3.4 LR(1)分析表的构造

闭包的计算

$J := I;$

repeat

$J = J \cup \{ [B \rightarrow \cdot \eta, b] \mid [A \rightarrow \alpha \cdot B \underline{\beta}, a] \in J,$

$b \in \underline{\text{FIRST}}(\beta a) \}$

until J 不再扩大

- 当 $\beta \Rightarrow^+ \epsilon$ 时, 此时 $b=a$ 叫**继承**的后继符, 否则叫**自生**的后继符



5.3.4 LR(1)分析表的构造

状态 I 和文法符号 X 的转移函数

$go(I, X) =$

$\text{closure}([A \rightarrow \alpha X \beta, \mathbf{a}] \mid [A \rightarrow \alpha \cdot X \beta, \mathbf{a}] \in I)$

5.3.4 LR(1)分析表的构造

$C = \{I_0\} \cup \{I \mid \exists J \in C, X \in V \cup T, I = \text{go}(J, X)\}$ 称为 G' 的 LR(1)项目集规范族 (算法: P185)

begin

$C := \{\text{closure}(\{S' \rightarrow \cdot S, \#\})\};$

repeat

for $\forall I \in C, \forall X \in V \cup T$

if $\text{go}(I, X) \neq \Phi$ & $\text{go}(I, X) \notin C$ then

$C = C \cup \text{go}(I, X)$

until C 不变化

end.

计算LR(1)项目集
规范族 C

5.3.4 LR(1)分析表的构造

识别活前缀的LR(1)文法的DFA

- 识别文法 $G = (V, T, P, S)$ 的拓广文法 G' 的所有活前缀的DFA $M = (C, V \cup T, go, I_0, C)$
 - $I_0 = \text{CLOSURE}(\{S' \rightarrow \cdot S, \#\})$
- 如果CFG G 的LR(1)分析表无冲突则称 G 为LR(1)文法

1. 令 $I_0 = \text{CLOSURE}(\{S' \rightarrow .S, \#\})$, 构造 $C = \{I_0, I_1, \dots, I_n\}$, 即 G' 的 LR(1) 项目集规范族。

2. 从 I_i 构造状态 i , 0 为初始状态。

for $k=0$ to n do

(1) if $[A \rightarrow \alpha.a\beta, b] \in I_k$ & $a \in T$ & $\text{GO}(I_k, a) = I_j$ then
 $\text{action}[k, a] := S_j$;

(2) if $\text{GO}(I_k, B) = I_j$ & $B \in V$ then $\text{goto}[k, B] := j$;

(3) if $[A \rightarrow \alpha., \mathbf{a}] \in I_k$ & $A \rightarrow \alpha$ 为 G' 的第 j 个产生式 then
 $\text{action}[k, \mathbf{a}] := r_j$;

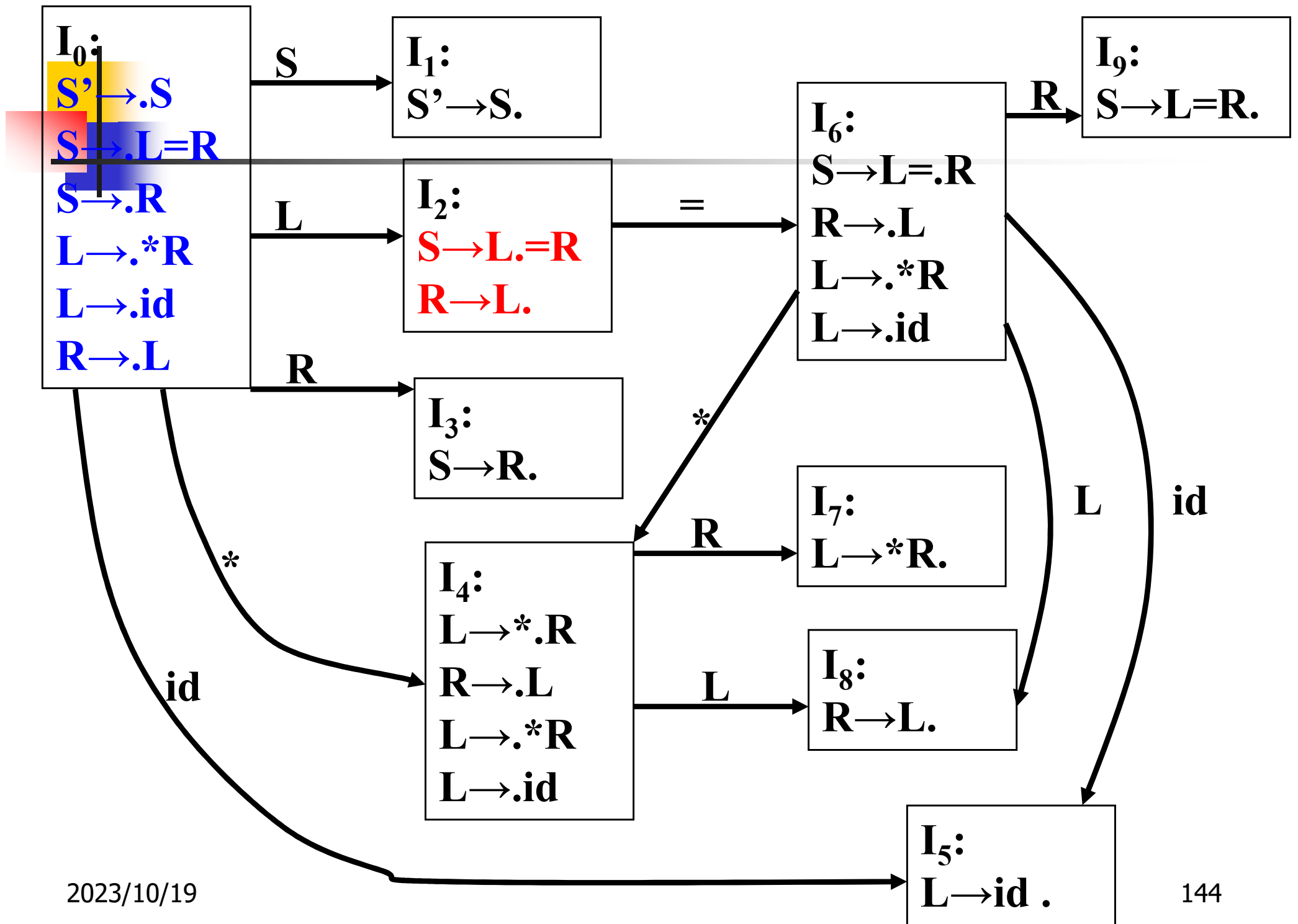
(4) if $[S' \rightarrow S., \#] \in I_k$ then $\text{action}[k, \#] := \text{acc}$;

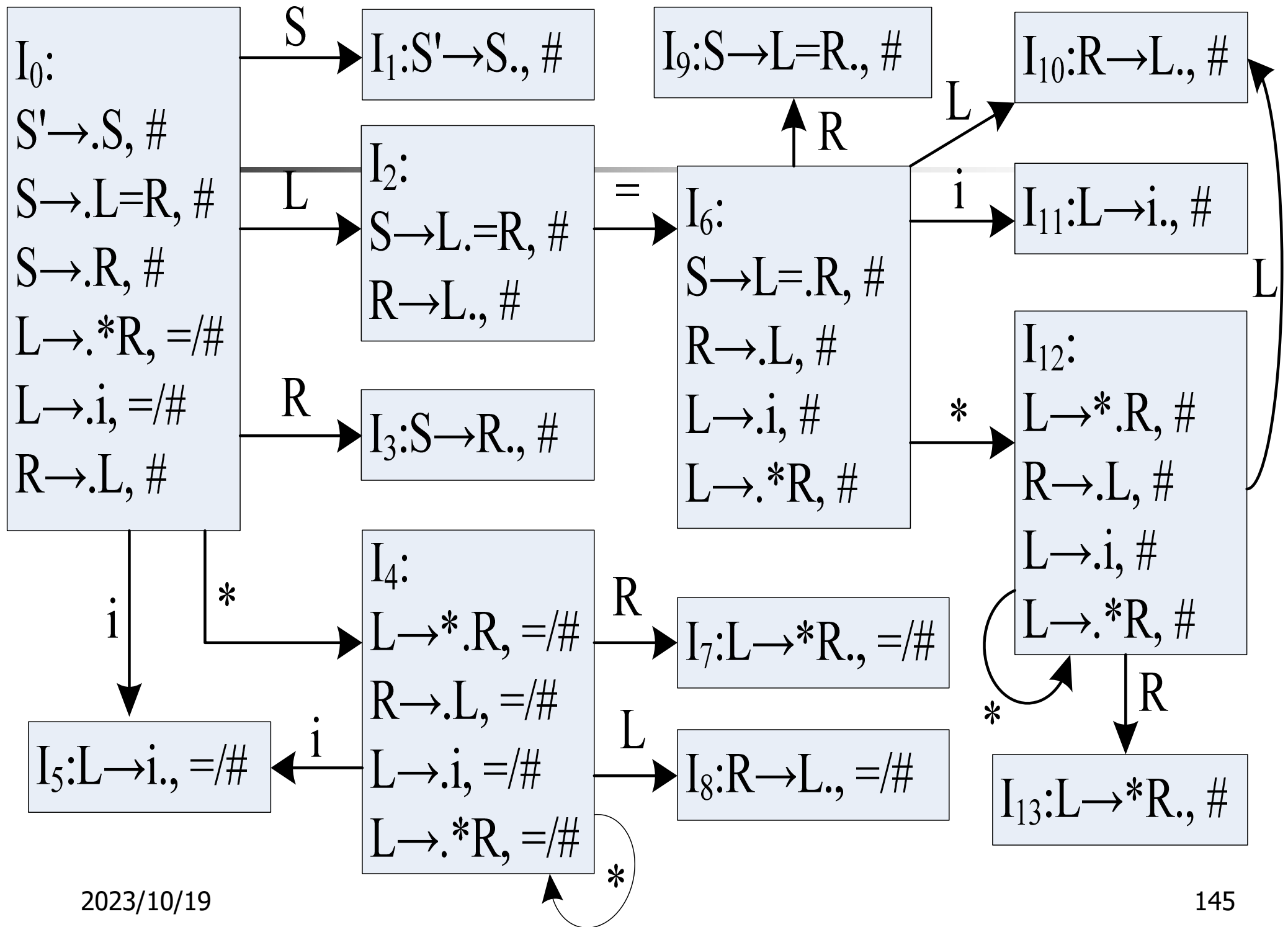
■ 上述(1)到(4)步未填入信息的表项均置为 error。

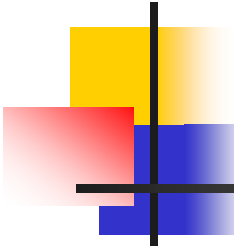


5.3.4 LR(1)分析表的构造

- 不同点主要在归约动作的选择：
 - LR(0)：考虑所有终结符
 - SLR(1)：参考 FOLLOW 集
 - LR(1)：仅考虑 LR(1)项目中的后继符







5.3.5 LALR(1)分析表的构造

- LR(1)对应的分析表规模太大，Pascal语言的语法分析器，其SLR(1)分析表只有几百个状态，而LR(1)分析表则有几千个状态

5.3.5 LALR(1)分析表的构造

- **问题：是否可以将某些闭包/状态合并？**
 - 不同的LR(1)项目闭包可能有相同的LR(0)项目，但后继符可能不同——同心闭包
 - 合并后可能带来归约归约冲突
 - 合并那些不会带来冲突的同心的LR(1)闭包

I6: $A \rightarrow c., d$

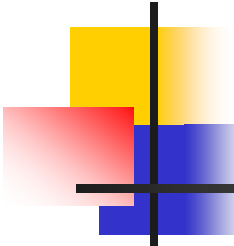
$B \rightarrow c., e$

I9: $A \rightarrow c., e$

$B \rightarrow c., d$

I69: $A \rightarrow c., d/e$

$B \rightarrow c., d/e$



5.3.5 LALR(1)分析表的构造

- *LookAhead-LR*

- 在不带来移进归约冲突的条件下，合并状态，重构分析表



LALR(1) 的分析能力

- **强于 SLR(1)**
 - **合并的后继符仍为 FOLLOW 集的子集**
- **局限性**
 - **合并后会出现归约-归约冲突**
 - **如果CFG G的LALR(1)分析表无冲突，则称G为LALR(1)文法**



5.3.6 二义性文法的应用

1. 任何二义性文法都不是LR文法
2. 某些二义性文法对语言的说明和实现非常有用，只要**指明消除二义性的规则**
3. 还是尽量少用二义性文法

- 采用二义性文法，可以减少结果分析器的状态数，并能减少对单非终结符（ $E \rightarrow T$ ）的归约。
- 在构造分析表时采用消除二义性的规则(按优先级)



用优先级和结合性来解决冲突

$$(1) E \rightarrow E + E$$

$$(2) E \rightarrow E * E$$

$$(3) E \rightarrow (E)$$

$$(4) E \rightarrow \text{id}$$

$$(1) E \rightarrow E + T$$

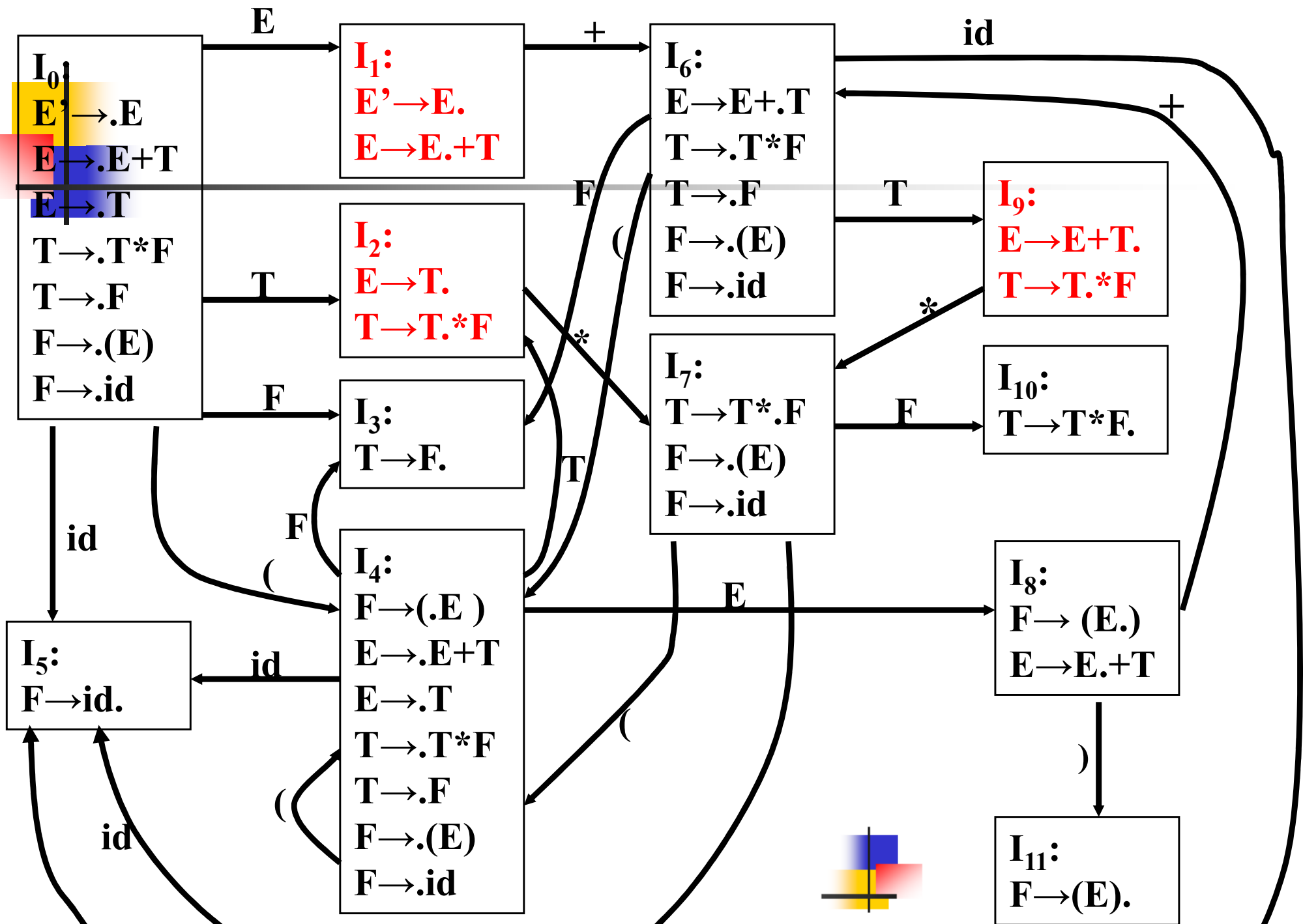
$$(2) E \rightarrow T (\text{单变量的归约})$$

$$(3) T \rightarrow T * F$$

$$(4) T \rightarrow F (\text{单变量的归约})$$

$$(5) F \rightarrow (E)$$

$$(6) F \rightarrow \text{id}$$



I₀:
E' → .E
E → .E + E
E → .E * E
E → .(E)
E → .id

I₁:
E' → E.
E → E. + E
E → E. * E

I₂:
E → (.E)
E → .E + E
E → .E * E
E → .(E)
E → .id

I₃:
E → id.

I₄:
E → E + .E
E → .E + E
E → .E * E
E → .(E)
E → .id

I₅:
E → E * .E
E → .E + E
E → .E * E
E → .(E)
E → .id

I₆:
E → (E.)
E → E .+ E
E → E. * E

I₇:
E → E + E.
E → E .+ E
E → E .* E

I₈:
E → E * E.
E → E. + E
E → E. * E

I₉:
E → (E).

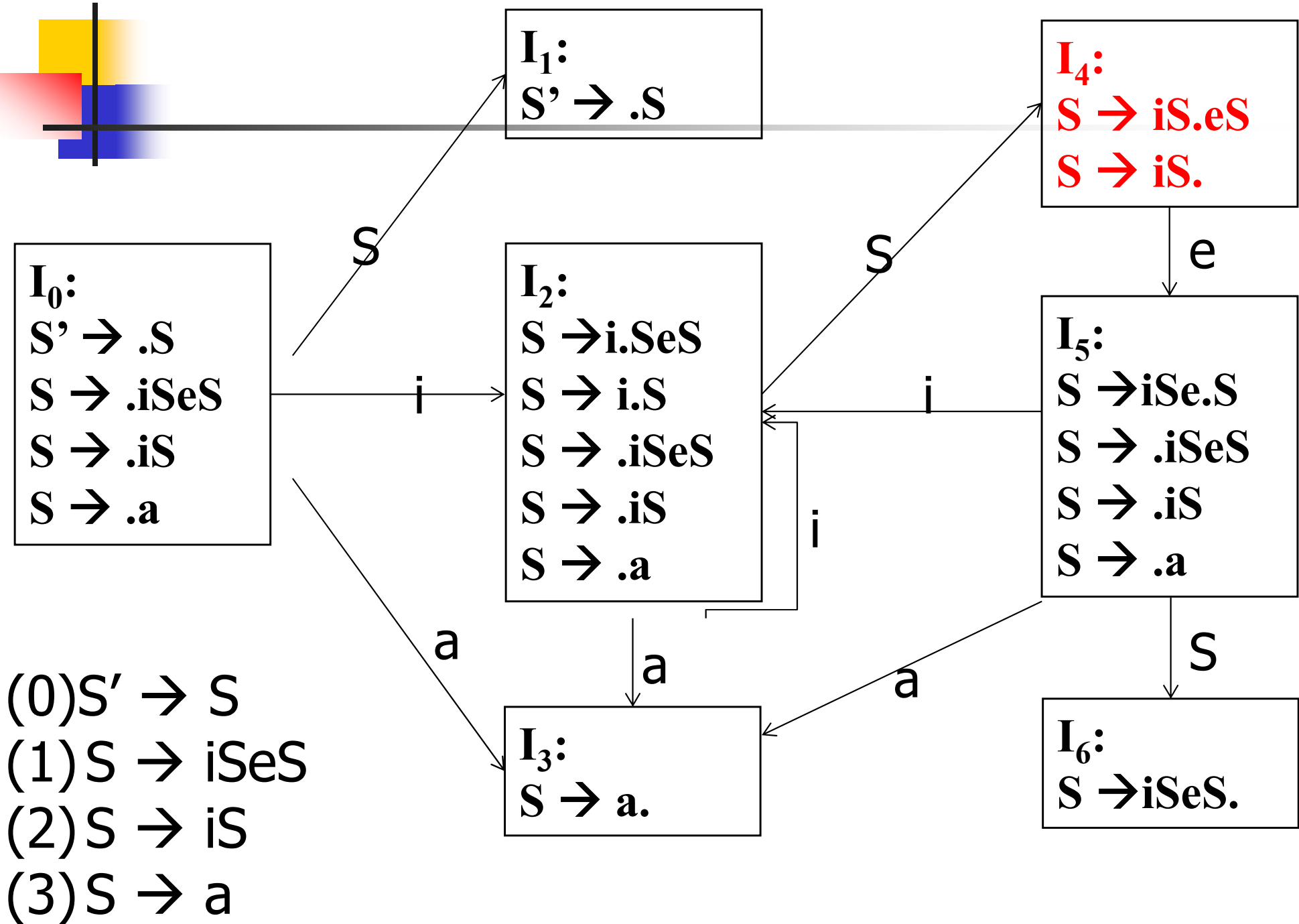
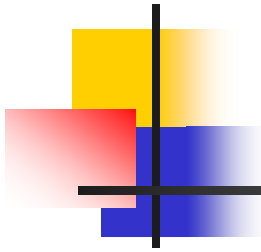


悬空-else的二义性

(1) $S \rightarrow iSeS$

(2) $S \rightarrow iS$

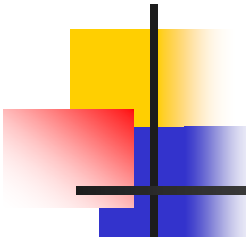
(3) $S \rightarrow a$





5.3.7 LR分析中的出错处理

- 当分析器处于某一状态 S ，且当前输入符号为 a 时，就以符号对 (S, a) 查 LR 分析表，如果分析表元素 $action[S, a]$ 为空(或出错)，则表示检测到了一个语法错误。



5.3.7 LR分析中的出错处理

■ 紧急方式的错误恢复

- 从栈顶开始退栈，直至发现在某个语法变量 A 上具有转移的状态 S 为止， A 通常是主要程序结构的语法变量
- 丢弃零个或多个输入符号，直至找到符号 $a \in \text{FOLLOW}(A)$ 为止
- 分析器把状态 $\text{goto}[S, A]$ 压进栈，并恢复正常分析



LR分析的基本步骤

- 1、编写拓广文法，求Follow集
- 2、求识别所有活前缀的DFA
- 3、构造LR分析表

5.4 语法分析程序的自动生成工具Yacc

YSP(Yacc Specification)

%{变量定义：头文件和全局变量

%开始符号

词汇表： %Token n_1, n_2, \dots (自动定义种别码)

%Token n_1, i_1 (用户指定种别码)

.....

%Token n_h, i_h (用户指定种别码)

类型说明 %type

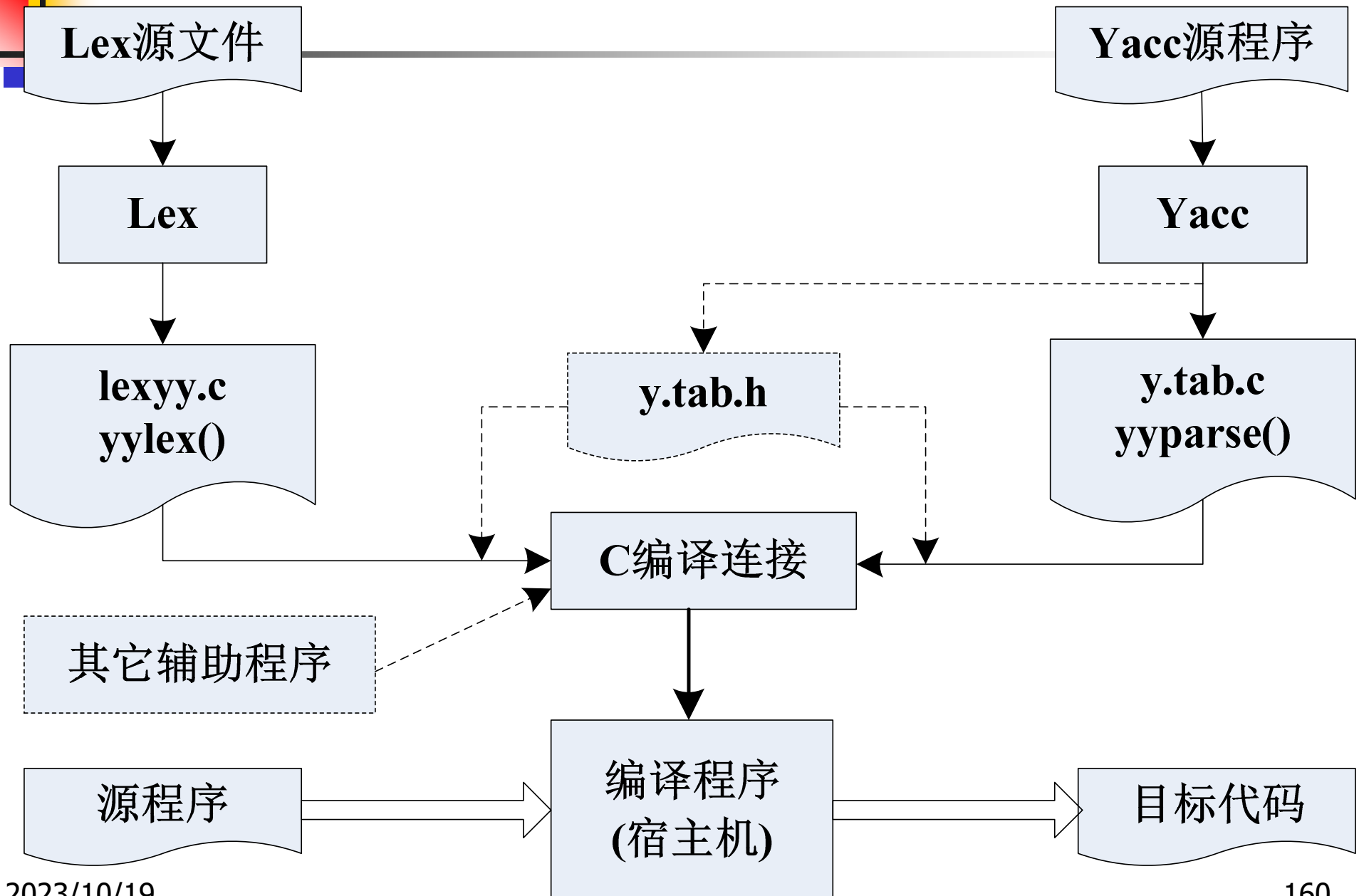
其它说明 %}

%%规则部分 给出文法规则的描述

%%程序部分 扫描器和语义动作程序

■ 输出： LALR(1)分析器

用Yacc和Lex合建编译程序





本章小结

- **自底向上的语法分析从给定的输入符号串 w 出发，自底向上地为其建立一棵语法分析树。**
- **移进-归约分析是最基本的分析方式，分为优先法和状态法。**
- **算符优先分析法是一种有效的方法，通过定义终结符号之间的优先关系来确定移进和归约。**



本章小结

- ***LR*分析法有着更宽的适应性。该方法通过构建识别规范句型活前缀的DFA来设计分析过程中的状态。可以将*LR*分析法分成*LR*(0)、*SLR*(1)、*LR*(1)、*LALR*(1)。**
- **通过增加的附加信息可以解决一些二义性问题。**
- ***Yacc*是*LALR*(1)语法分析器的自动生成工具。**