编译原理与技术

wenshli@bupt.edu.cn

李文生

wenshli@

北京邮电大学计算机学院(国家示范性软件学院)
SCHOOL OF COMPUTER SCIENCE(NATIONAL PILOT SOFTWARE ENGINEERING SCHOOL)BUPT

第4章 语法分析

wenshli@bupt.edu.cn

李文生

2022年8月16日 星期二

北京邮电大学计算机学院(国家示范性软件学院)
SCHOOL OF COMPUTER SCIENCE(NATIONAL PILOT SOFTWARE ENGINEERING SCHOOL)BUPT

教学内容、目标与要求

- 教学内容
  - 语法分析程序的作用
  - 预测分析，LL(1)分析方法
  - 移进-归约分析方法，LR分析方法
- 教学目标与要求
  - 掌握自顶向下分析方法对文法的要求；
  - 理解递归调用预测分析程序的构造方法；
  - 掌握LL(1)分析程序的模型及工作过程；掌握LL(1)分析表的构造方法；
  - 理解“移进-归约”分析方法的工作过程；
  - 掌握LR(1)分析程序的模型及工作过程；
    掌握SLR(1)、LR(1)、LALR(1)分析表的构造方法。
  - 理解并分析语法分析的需求，能够基于自动机设计语法分析程序，并对符号串进行语法分析。

wenshli@bupt.edu.cn

基础知识

- 高级程序设计语言的语法结构，如：Pascal语言、C语言
- 上下文无关文法
- 最左推导、最右推导
- 语言、句子、句型、短语、句柄
- 分析树、子树、子树与短语之间的关系
- 状态转换图

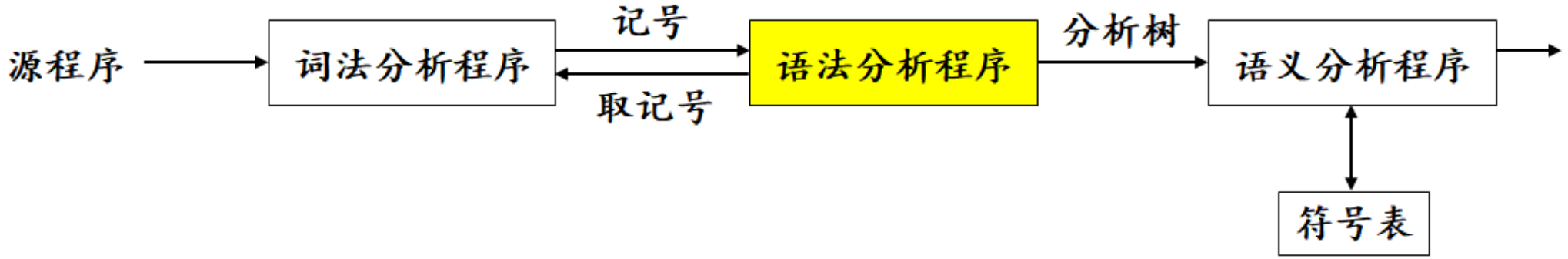
wenshli@bupt.edu.cn

内容目录

wenshli@bupt.edu.cn

4.1 语法分析简介

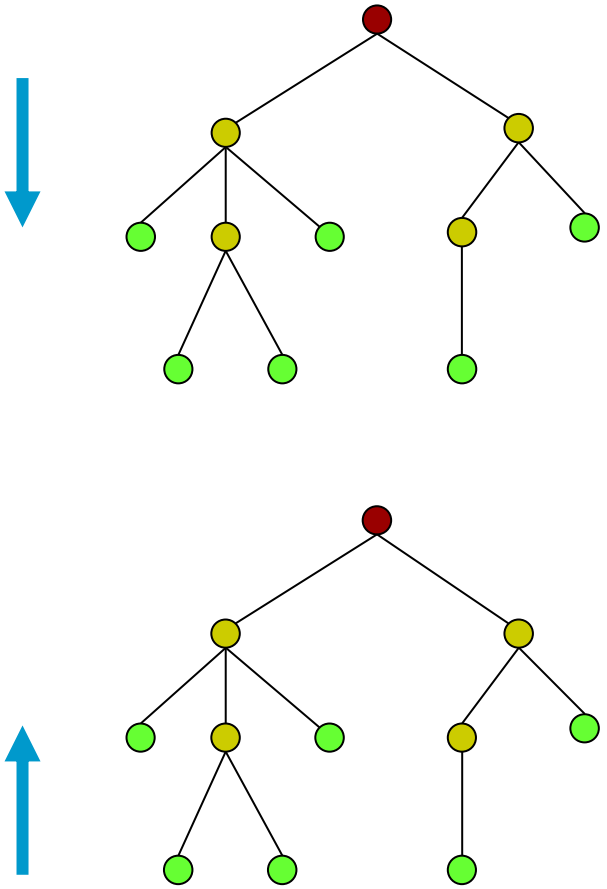
- 核心工作
- 由语法分析程序完成
- 工作依据：源语言的语法规则
- 语法分析程序的任务
  - 从源程序记号序列中识别出各类语法成分
  - 进行语法检查
- 语法分析程序的地位：



wenshli@bupt.edu.cn

语法分析简介

- 语法分析程序
  - 输入：记号流/记号序列
  - 功能：将记号组合成语法成分、进行语法检查
  - 工作依据：语法规则
  - 输出：分析树
  - 错误处理
- 常用的分析方法
  - 自顶向下的方法：
    从树根到叶子自顶向下建立分析树
  - 自底向上的方法：
    从树叶到树根自底向上建立分析树
- 对输入符号串的扫描顺序：自左向右



wenshli@bupt.edu.cn

语法错误的处理

- 错误处理目标
  - 清楚而准确地报告发现的错误，如错误的位置和性质。
  - 迅速地从错误中恢复过来。
  - 不应该明显地影响编译程序对正确程序的处理效率。
- 错误恢复策略
  - 紧急恢复
  - 短语级恢复
  - 出错产生式
  - 全局纠正

> - 简单，适用于大多数分析程序。
> - 做法：一旦发现错误，分析程序每次抛弃一个输入记号，直到扫描到的记号属于某个指定的同步记号集合为止。
> - 同步记号通常是定界符，如语句结束符分号、语句起始符、块结束标识END等。

wenshli@bupt.edu.cn

4.2 自顶向下分析方法

1. 递归下降分析
2. 递归调用预测分析
3. 非递归预测分析

wenshli@bupt.edu.cn

1. 递归下降分析

试图建立一个最左推导序列的过程

- 试探性推导过程
- 文法开始符号对应根结点，自顶向下为输入串建立一棵分析树。
- 试探过程，试用不同产生式谋求匹配输入符号串的过程。
- 例: 分析输入符号串 ω=abbcde 是否为如下文法的一个句子。

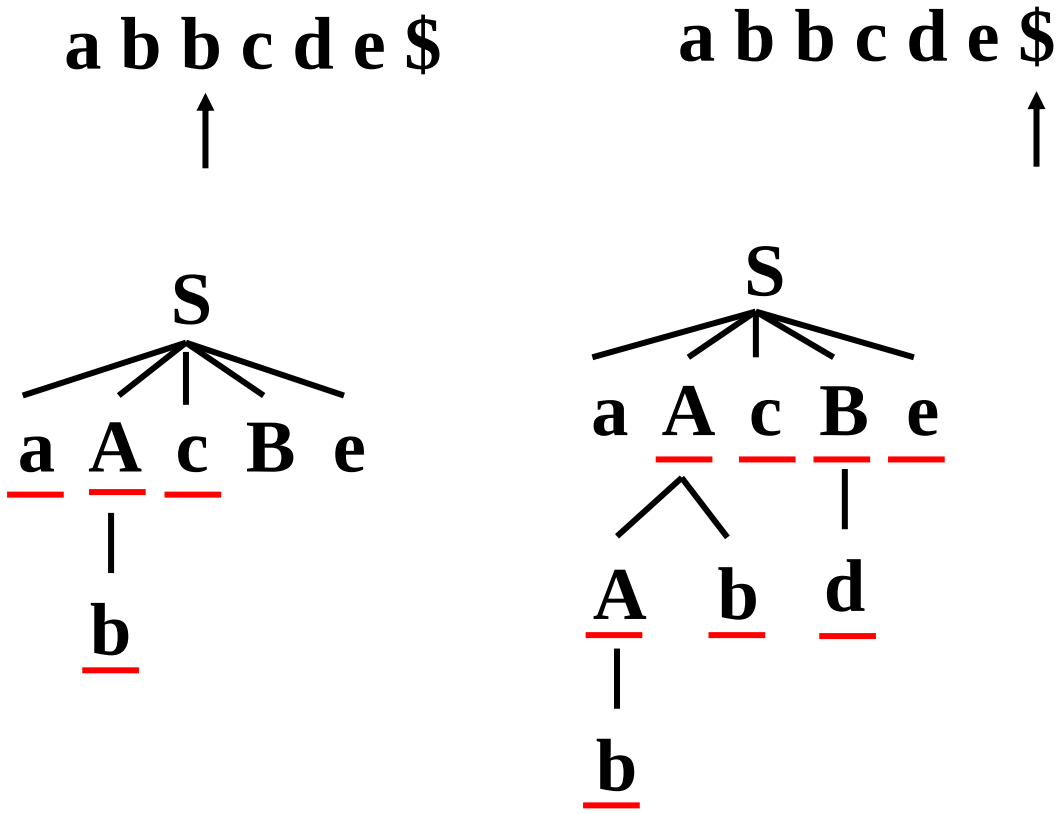
S→aAcBe

A→b | Ab

B→d (文法4.1)

S⇒aAcBe⇒aAbcBe⇒abbcBe⇒abbcde

- abbcde 的递归下降分析



推导？ ==>

wenshli@bupt.edu.cn

递归下降分析

- 为什么采用最左推导？
  - 对输入串的扫描是自左至右进行的。
  - 只有使用最左推导，才能保证按扫描的顺序匹配输入串。
- 递归下降分析方法的实现
  - 每个非终结符号对应一个递归过程，布尔函数。
  - 函数功能：一旦发现它的某个产生式与输入串匹配，则用该产生式展开分析树，并返回true，否则分析树不变，返回false。
- 实践中存在的困难和缺点
  - 左递归的文法，可能导致分析过程陷入死循环。
  - 回溯，工作的重复。
  - 效率低、代价高：穷尽一切可能的试探法。

wenshli@bupt.edu.cn

2. 递归调用预测分析

- 一种确定的、不带回溯的递归下降分析方法

一、如何克服回溯？

二、对文法的要求

三、预测分析程序的构造

wenshli@bupt.edu.cn

递归调用预测分析

- 一、如何克服回溯？
  - 能够根据所面临的输入符号准确地指派一个候选式去执行任务。
  - 该选择的工作结果是确信无疑的。
  - 例：

$$A\rightarrow\alpha_1\,|\,\alpha_2\,|\ldots|\,\alpha_I\,|\ldots|\,\alpha_n$$

当前输入符号：a

指派α_i去匹配输入符号串

- 二、对文法的要求
  1. 不含左递归？
  2. $FIRST(\alpha_i)\cap FIRST(\alpha_j)=\phi \quad (i\neq j)$

如：$A\rightarrow\alpha_1\,|\,\alpha_2\,|\ldots|\,\alpha_n$

$$FIRST(\alpha_i)=\{\,a\,|\,\alpha_i \overset{*}{\Rightarrow} a\beta,\ a\in V_T,\ \alpha_i、\beta\in(V_T\cup V_N)^*\,\}$$

如果 $\alpha_i \overset{*}{\Rightarrow} \varepsilon$，则规定 $\varepsilon\in FIRST(\alpha_i)$。

非终结符号A的所有候选式的开头终结符号集两两互不相交

wenshli@bupt.edu.cn

示例:

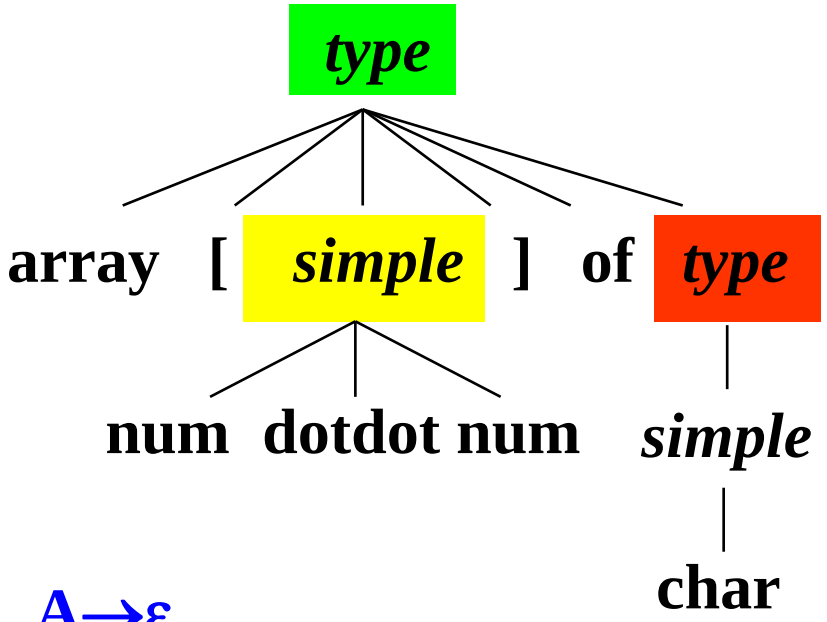
- 有如下产生PASCAL类型子集的文法:
  type→*simple* | ↑id | array[*simple*] of *type*
  simple→integer | char | num dotdot num

  分析输入串:
  array [num dotdot num] of char

- *type*的三个候选式，有:
  FIRST(*simple*)={ integer, char, num }
  FIRST(↑id)={ ↑ }
  FIRST(array[simple] of type)={ array }
- *simple*的三个候选式，有:
  FIRST(integer)={ integer }
  FIRST(char)={ char }
  FIRST(num dotdot num)={ num }

分析：array [num dotdot num] of char

树:



- 产生式：A→ε

当没有适当候选式时，可以用一个ε-候选式，表示其它候选式可以缺席。

wenshli@bupt.edu.cn

三、预测分析程序的构造

- 预测分析程序的转换图
- 转换图的化简
- 转换图的工作过程
- 预测分析程序的实现

wenshli@bupt.edu.cn

预测分析程序的转换图

- 预测分析程序的转换图
  - 每一个非终结符号有一张图。
  - 边的标记是文法符号。
- 边的含义
  - 若标记是非终结符号A，意味着调用相应A的过程。
  - 若标记是终结符号a，意味着下一个输入符号若为a，则做此转移。

- 为文法构造转换图
  - 改写文法
    - 重写文法
    - 消除左递归
    - 提取左公因子
  - 对每一个非终结符号A：
    - 创建一个初态和一个终态。
    - 对每个产生式 $A \rightarrow X_1X_2 \ldots X_n$ 创建一条从初态到终态的路径，有向边的标记依次为 $X_1, X_2, \ldots, X_n$。

改写文法？
==>

wenshli@bupt.edu.cn

示例：为文法构造状态转换图

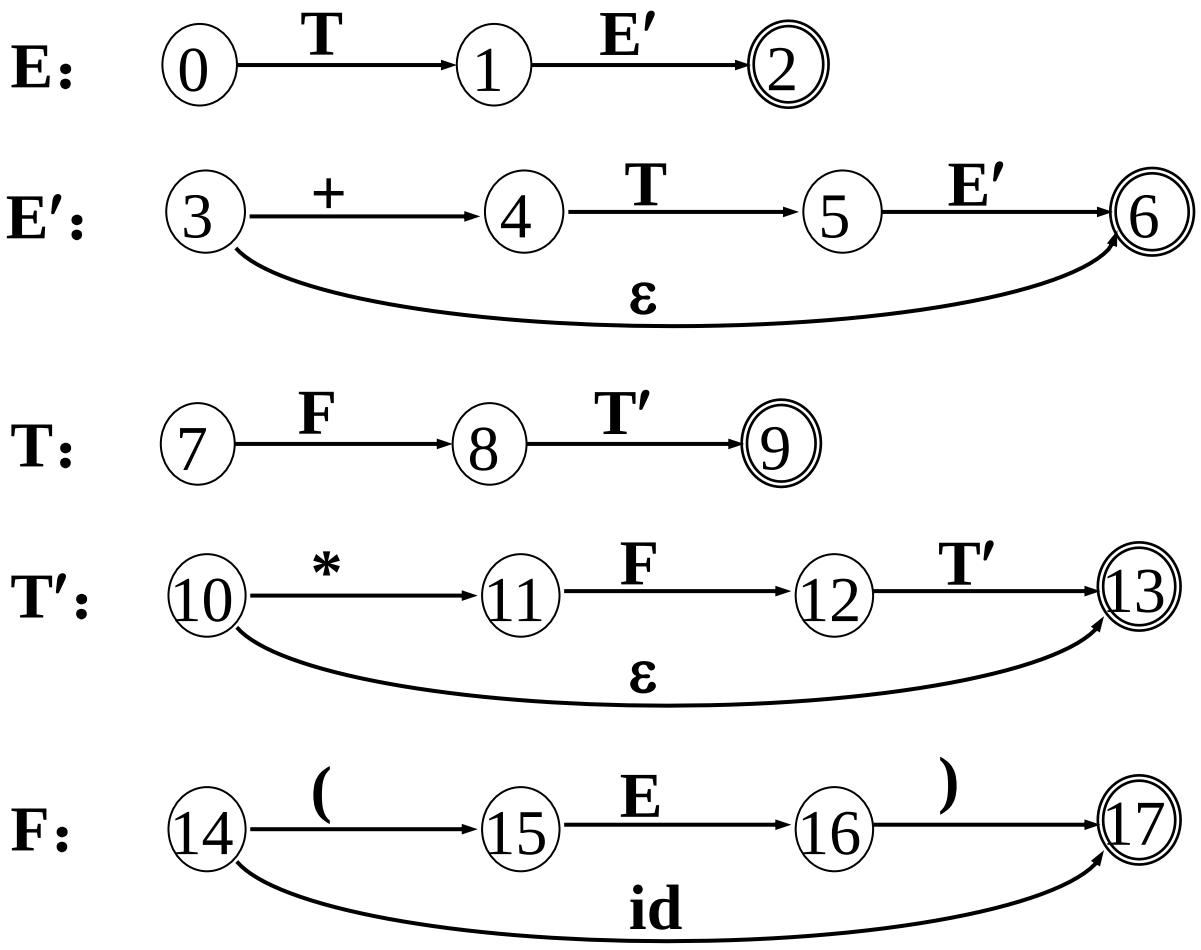
- 为如下文法构造预测分析程序转换图

  E→E+T | T
  T→T*F | F
  F→(E) | id　　(文法4.3)

- 消除文法中存在的左递归，得到

  E→TE′
  E′→+TE′ | ε
  T→FT′
  T′→*FT′ | ε
  F→(E) | id　　(文法4.4)

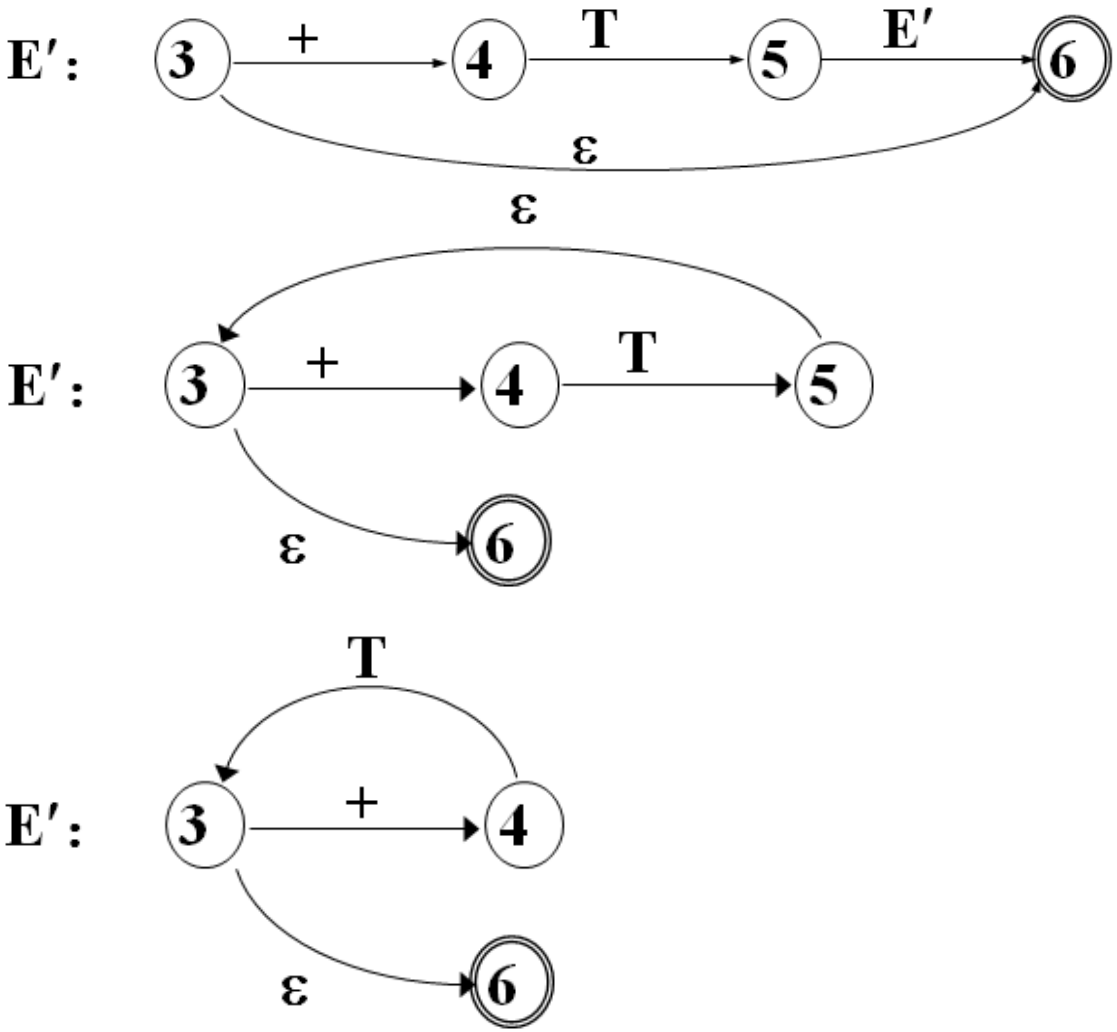
- 为每个非终结符号构造转换图：



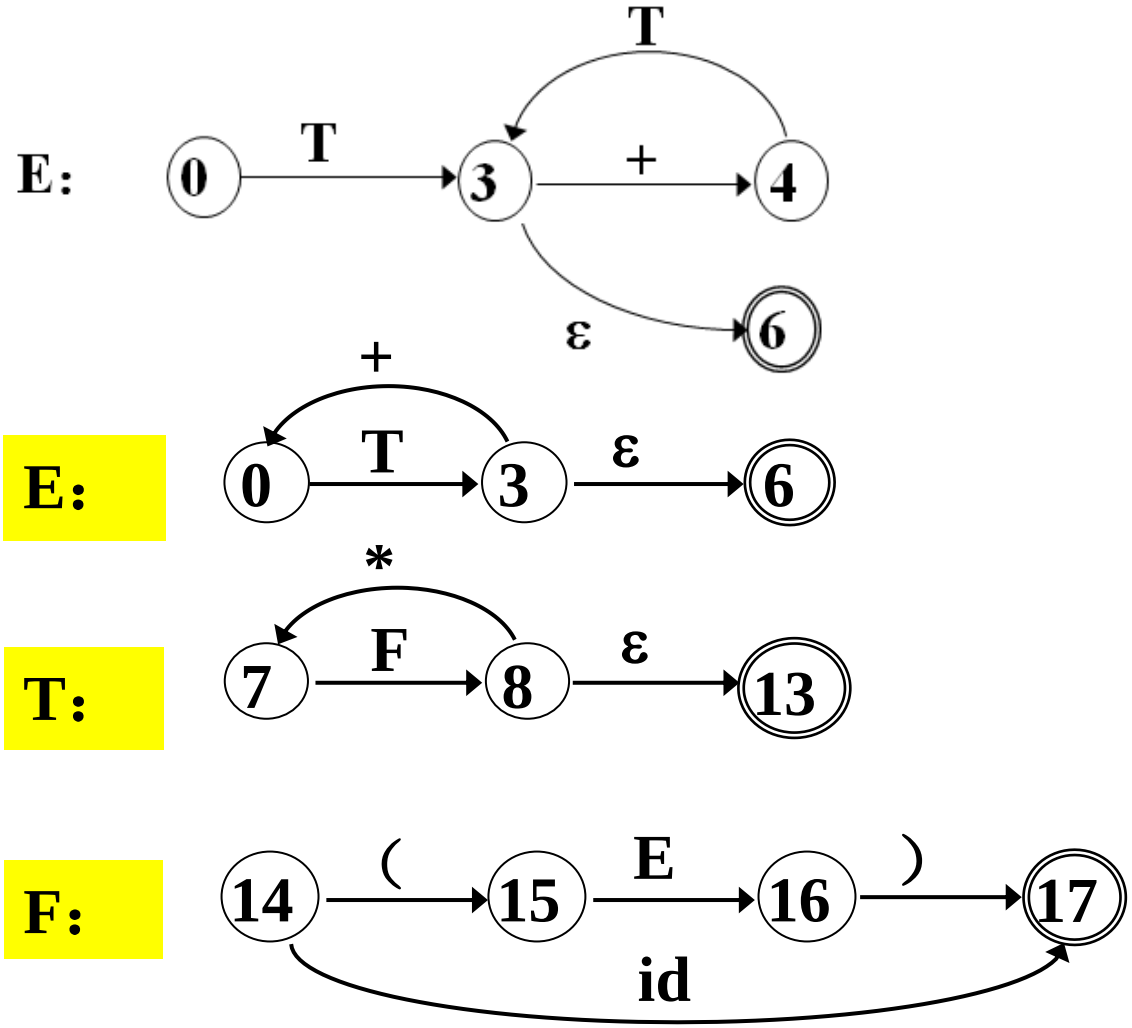
wenshli@bupt.edu.cn

状态转换图的化简

- 用代入的方法进行化简



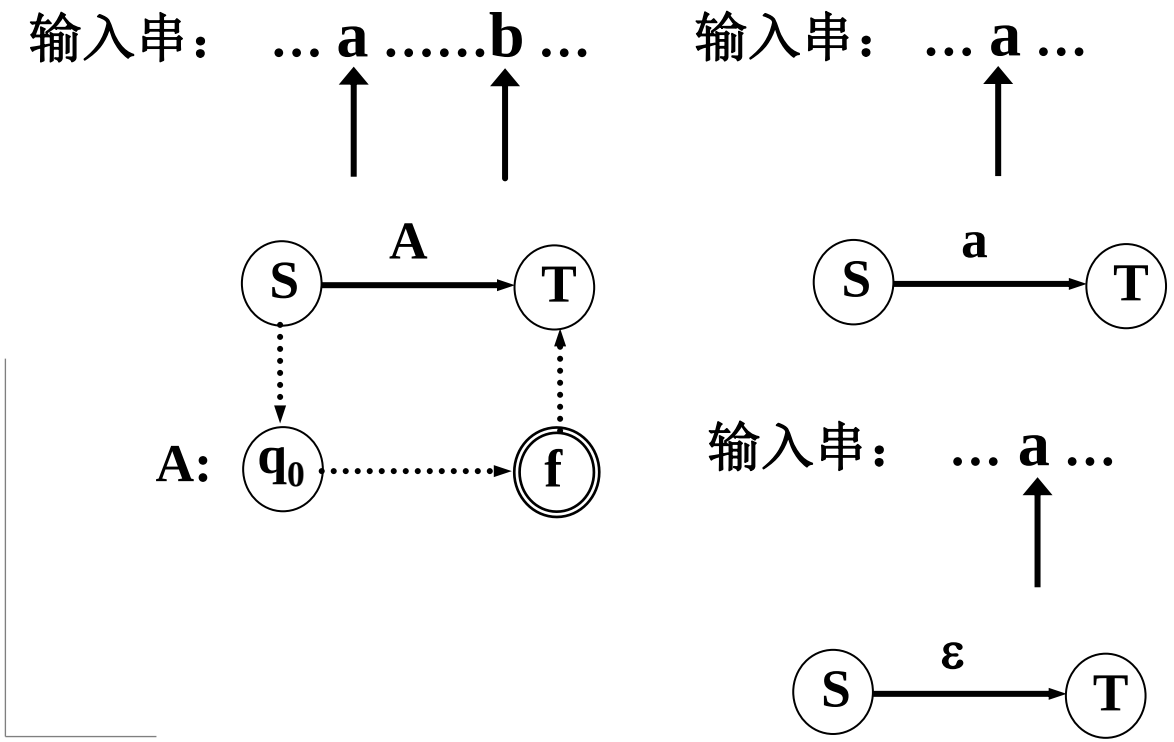
- 把E′的转换图代入E的转换图：



wenshli@bupt.edu.cn

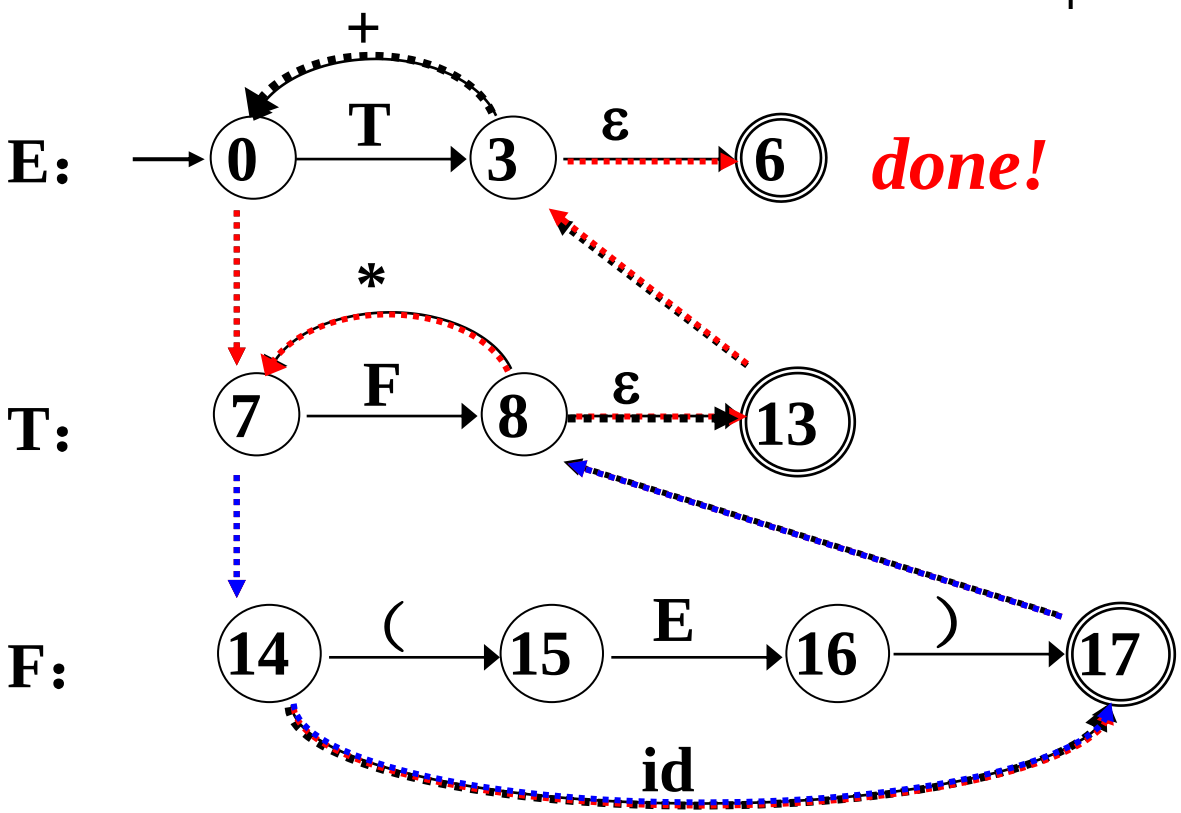
转换图的工作过程

- 起点：文法开始符号所对应的转换图的初态。
- 经过若干动作之后，处于状态S
  扫描指针指向符号a，即当前输入符号



- 分析：id+id*id

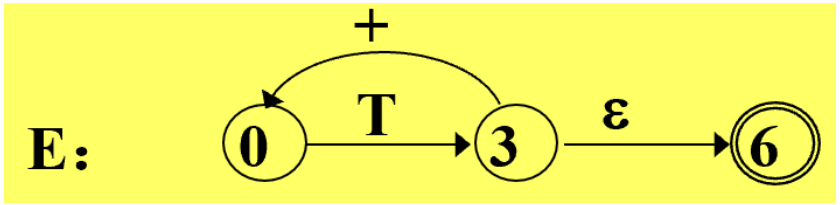
id + id * id $



wenshli@bupt.edu.cn

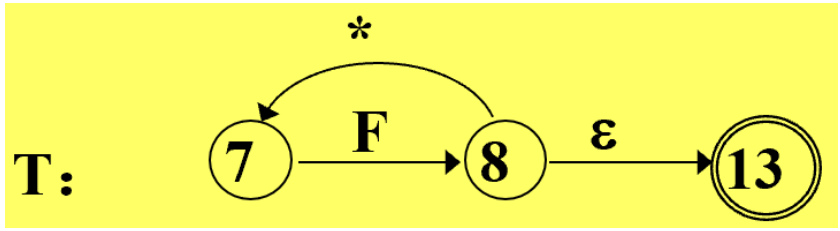
预测分析程序的实现

- 要求实现语言允许递归调用
- E的过程：




```
void procE(void) {
    procT();
    if (char==′+′) {
        forward pointer;
        procE();
    }
}
```

- T的过程：



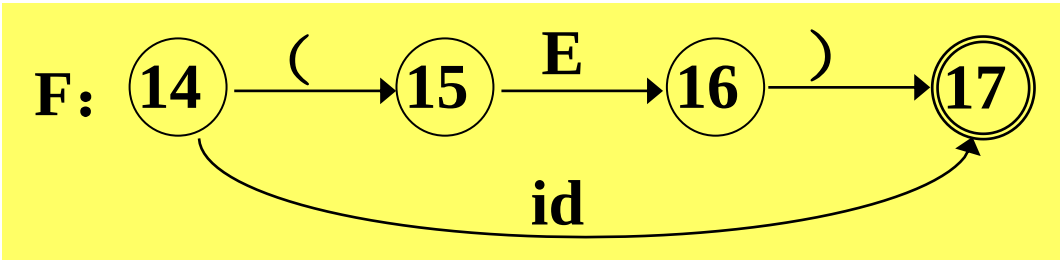

```
void procT(void)  {
    procF();
    if (char==′*′) {
        forward pointer;
        procT();
    }
}
```

wenshli@bupt.edu.cn

预测分析程序的实现

- F的过程：

```
void procF(void) {
    if (char==′(′) {
        forward pointer;
        procE();
        if (char==′)′) {
            forward pointer;
        };
        else  error();
    };
    else  if (char==′id′) {
                forward pointer;
          else  error();
}
```



wenshli@bupt.edu.cn

练习4.1

有如下文法：

bexpr→*bexpr* or *bterm* | *bterm*

bterm→*bterm* and *bfactor* | *bfactor*

bfactor→not *bfactor* | (*bexpr*) | true | false

请构造一个可以用来分析该文法所产生的句子的递归调用分析程序。

wenshli@bupt.edu.cn

Step 1：消除左递归

bexpr→*bterm* *E'*

E' →or *bterm* *E'* | ε

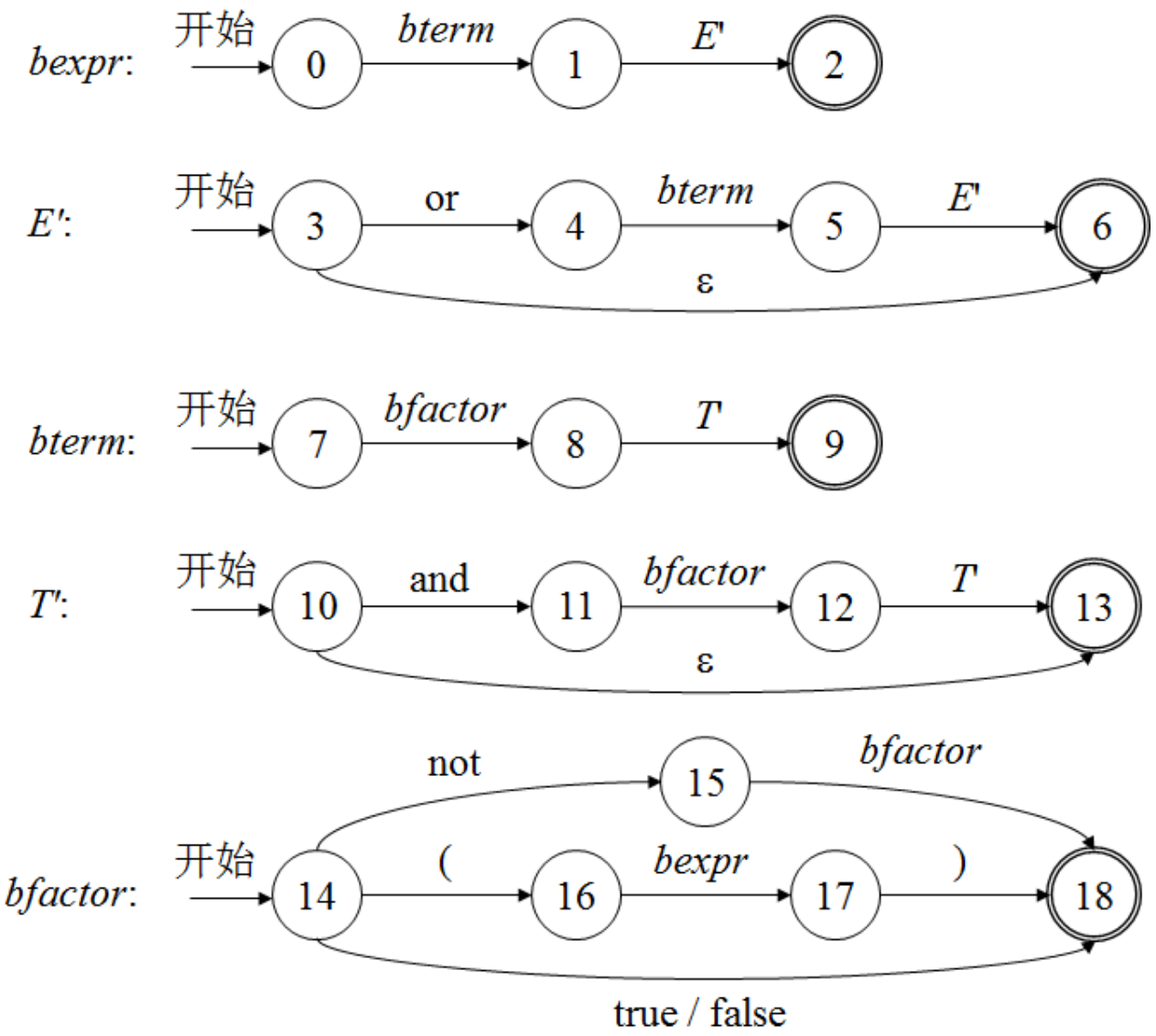
bterm→*bfactor* *T'*

T' →and *bfactor* *T'* | ε

bfactor→not *bfactor* | (*bexpr*) | true | false

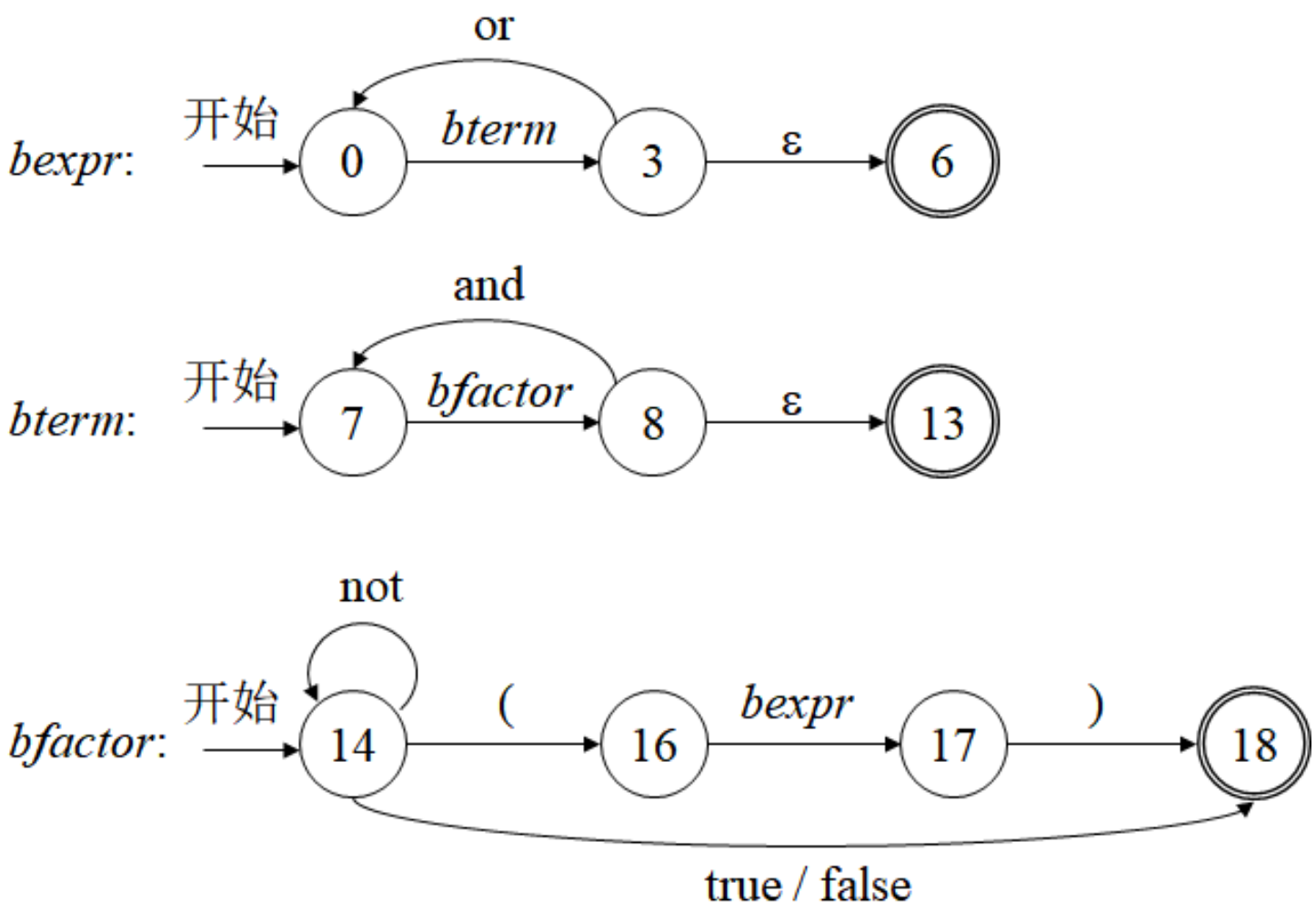
wenshli@bupt.edu.cn

Step 2：文法G'的预测分析程序状态转换图



wenshli@bupt.edu.cn

Step 3：化简后的预测分析程序状态转换图

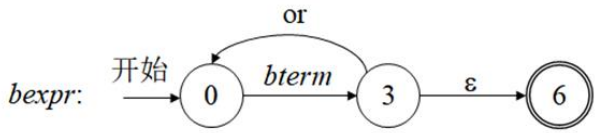


wenshli@bupt.edu.cn

Step 4：根据状态转换图进行程序设计

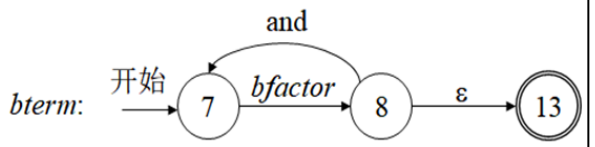
- *bexpr*的函数

```
void proc_expr(void) {
    proc_term();
    if (char==′or′) {
        forward pointer;
        proc_expr();
    }
}
```



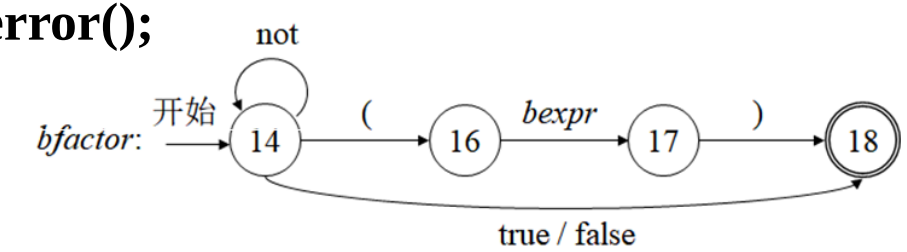
- *bterm*的过程

```
void proc_term(void) {
    proc_factor();
    if (char==′and′) {
        forward pointer;
        proc_term();
    }
}
```



- *bfactor*的过程

```
void proc_factor(void) {
    if (char==′not′){
        forward pointer;
        proc_factor();
    }
    else if (char==′(′) {
            forward pointer;
            proc_expr();
            if (char==′)′)
                forward pointer;
            else  error();
    }
    else if (char==′true′)||(char==′false′)
                forward pointer;
    else error();
}
```



wenshli@bupt.edu.cn

3. 非递归预测分析

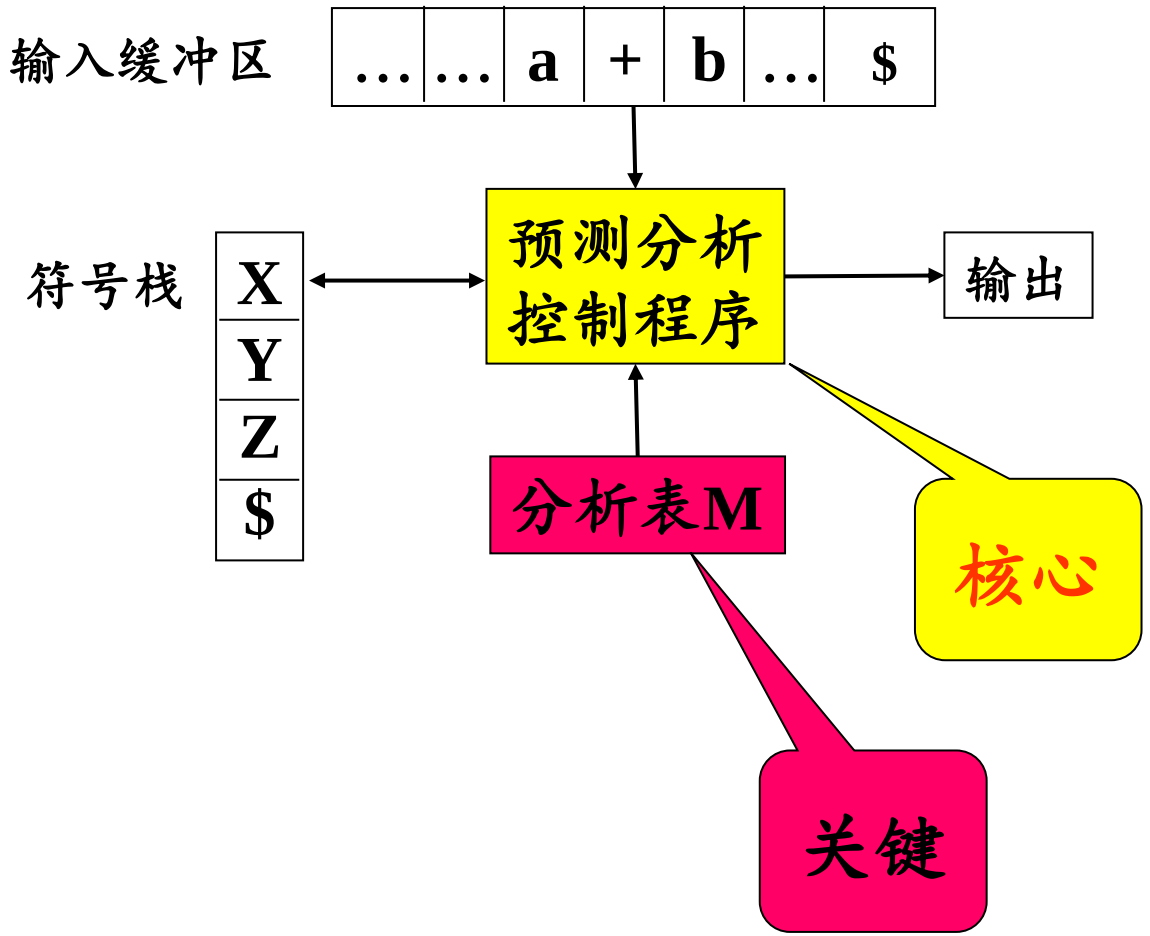
- 分析表和分析栈联合控制

- 预测分析程序的模型及工作过程
- 预测分析表的构造
- LL(1)文法
- 预测分析方法中的错误处理示例

wenshli@bupt.edu.cn

预测分析程序的模型

- 模型



- 输入缓冲区：被分析的输入符号串，串后随右尾标志符$。
- 符号栈：分析开始时，先将$入栈，然后再将文法的开始符号入栈。
- 分析表：

  二维数组M[A, a]，$A \in V_N$，$a \in V_T \cup \{\$\}$。

  根据给定的A和a，在分析表M中找到将被调用的产生式。

  … a …

  A | A→a…

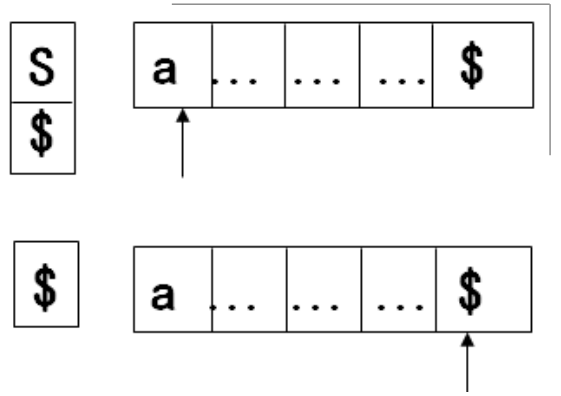
- 输出流：

  分析过程中不断产生的产生式序列。

wenshli@bupt.edu.cn

预测分析控制程序

- 根据栈顶符号X和当前输入符号a，分析动作有4种可能：



(1) X=a=$，宣告分析成功，停止分析；

(2) X=a≠$，从栈顶弹出X，输入指针前移一个位置；

(3) $X \in V_T$，但X≠a，发现错误，调用错误处理程序，报告错误及进行错误恢复；

(4) 若$X \in V_N$，访问分析表M[X, a]

- $M[X, a]=X \rightarrow Y_1 Y_2 \ldots Y_n$

  先将X从栈顶弹出，然后把产生式的右部符号串按反序推入栈中（即按 Y_n、…、Y_{n-1}、Y_2、Y_1 的顺序）；

- $M[X, a]=X \rightarrow \varepsilon$　从栈顶弹出X；

- M[X, a]=error　调用出错处理程序

wenshli@bupt.edu.cn

算法4.1 非递归预测分析方法

输入：输入符号串ω，文法G的预测分析表M。
输出：若ω ∈L(G)，则输出ω的最左推导，否则报告错误。
方法：分析开始时，$在栈底，文法开始符号S在栈顶，ω$在输入缓冲区中。

```
    置ip指向 ω$ 的第一个符号;
    do {                                  // X是栈顶符号，a是ip所指向的符号
        if (X是终结符号或$) {
            if (X==a) { 从栈顶弹出X;    ip前移一个位置; }
            else  error();
        else                              // X是非终结符号
            if (M[X, a]=X→Y1Y2…Yk ) {
                从栈顶弹出X;  把Yk、Yk-1、…、Y2、Y1压入栈，Y1在栈顶;
                输出产生式X→Y1Y2…Yk;
            }
            else  error();
    } while(X!=$)                         // 栈不空，继续
```

wenshli@bupt.edu.cn

示例：文法4.4的预测分析表M，试分析输入串 id+id*id

	id	+	*	(	)	$
E	E→TE′			E→TE′		
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′			T→FT′		
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id			F→(E)		

分析栈

$	E’	T’	id	*	

输入串

id	+	id	*	id	$

done!

输出：

E→TE′
T→FT′
F→id
T′→ε
E′→+TE′
T→FT′
F→id
T′→*FT′
F→id
T′→ε
E′→ε

wenshli@bupt.edu.cn

id + id * id 分析树的构建过程

- 左句型：
  （1）分析过的符号串+栈中符号串（顶到底）
  （2）分析树的叶子节点（从左至右）
- 输出产生式序列、最左推导

	id	+	*	(	)	$
E	E→TE′			E→TE′		
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′			T→FT′		
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id			F→(E)		

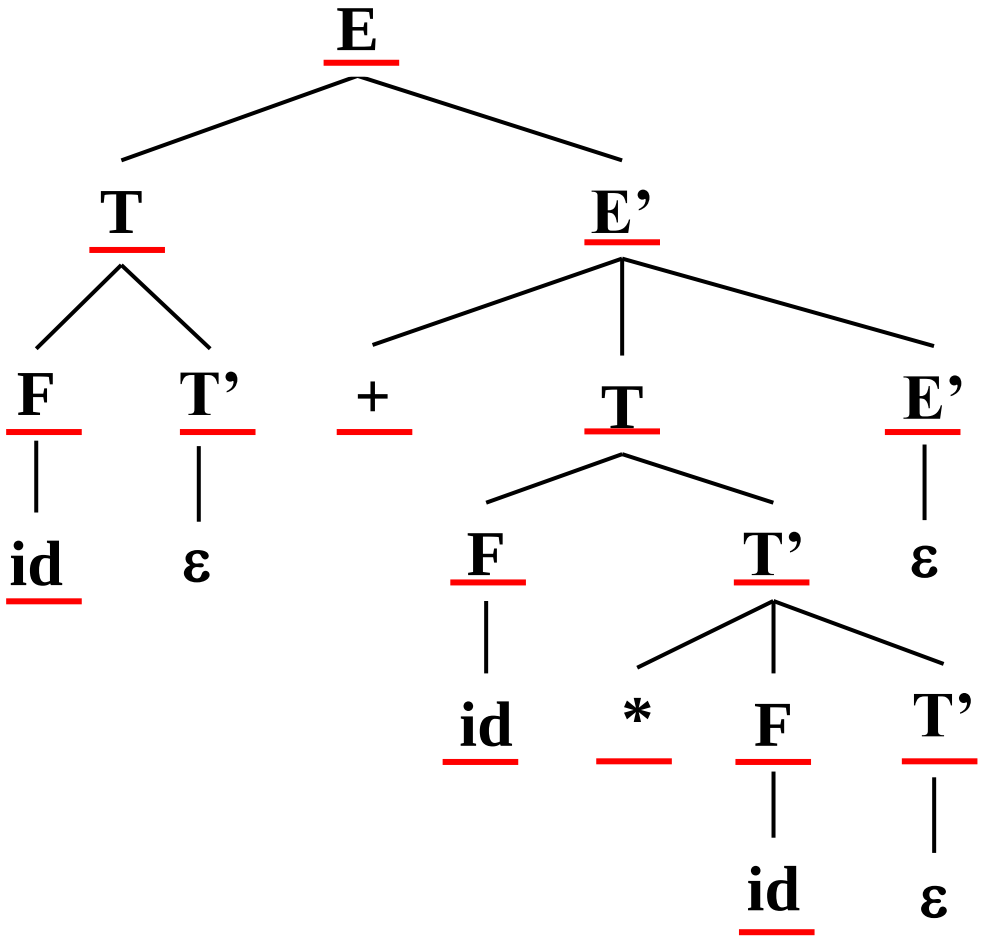
id	+	id	*	id	$

E→TE′
T→FT′
F→id
T′→ε
E′→+TE′

T→FT′
F→id
T′→*FT′
F→id
T′→ε
E′→ε

$	E’	T’	id	*	

done !



wenshli@bupt.edu.cn

预测分析表的构造

- 改写文法
- FIRST集合及其构造
- FOLLOW集合及其构造
- 预测分析表的构造

wenshli@bupt.edu.cn

FIRST集合及其构造

- **FIRST集合**

定义：

对任何文法符号串$\alpha \in (V_T \cup V_N)^*$，

FIRST(α)是α可以推导出的开头终结符号集合。

描述为：

$$FIRST(\alpha) = \{\ a \mid \alpha \overset{*}{\Rightarrow} a\cdots,\ a \in V_T\ \}$$

若$\alpha \overset{*}{\Rightarrow} \varepsilon$，则$\varepsilon \in FIRST(\alpha)$

wenshli@bupt.edu.cn

构造每个文法符号 $X \in V_T \cup V_N$ 的FIRST(X)

- 若$X \in V_T$，则FIRST(X)={X};
- 若$X \in V_N$，且有产生式 $X \to a\ldots$，其中 $a \in V_T$，则把 a 加入到FIRST(X)中;
- 若 $X \to \varepsilon$ 也是产生式，则 ε 也加入到FIRST(X)中。
- 若 $X \to Y\ldots$ 是产生式，且$Y \in V_N$，则把FIRST(Y)中的所有非ε元素加入到FIRST(X)中;

若 $X \to Y_1Y_2\ldots Y_k$ 是产生式，如果对某个 i，$FIRST(Y_1)$、$FIRST(Y_2)$、…、$FIRST(Y_{i-1})$ 都含有ε，即 $Y_1Y_2\ldots Y_{i-1} \overset{*}{\Rightarrow} \varepsilon$，则把$FIRST(Y_i)$中的所有非 ε 元素加入到FIRST(X)中;

若所有$FIRST(Y_i)$均含有ε，其中 i=1、2、…、k，则把 ε 加入到FIRST(X)中。

$E \to TE'$

$E' \to +TE' \mid \varepsilon$

$T \to FT'$

$T' \to *FT' \mid \varepsilon$

$F \to (E) \mid id$

	FIRST
E	(, id
E’	+, ε
T	(, id
T’	*, ε
F	(, id

wenshli@bupt.edu.cn

FOLLOW集合及其构造

- **FOLLOW集合**

定义：

假定S是文法G的开始符号，对于G的任何非终结符号A，集合FOLLOW(A)是在所有句型中，紧跟A之后出现的终结符号或 $ 组成的集合。

描述为：

$$FOLLOW(A)=\{ a \mid S \overset{*}{\Rightarrow} \ldots Aa \ldots，\ a \in V_T \}$$

特别地，若$S \overset{*}{\Rightarrow} \ldots A$，则规定 $\$ \in FOLLOW(A)$

wenshli@bupt.edu.cn

构造每个非终结符号A的集合FOLLOW(A)

- 对文法开始符号S，置$于FOLLOW(S)中，
  $为输入符号串的右尾标志。
- 若A→αBβ是产生式，则把FIRST(β)中的
  所有非ε元素加入到FOLLOW(B)中。
- 若A→αB是产生式，或 A→αBβ 是产生式并且$\beta \overset{*}{\Rightarrow} \varepsilon$，
  则把FOLLOW(A)中的所有元素加入到FOLLOW(B)中。
- 重复此过程，直到所有集合不再变化为止。

E→TE′
E′→+TE′ | ε
T→FT′
T′→*FT′ | ε
F→(E) | id

	FOLLOW
E	$,)
E’	$,)
T	$, + ,)
T’	$, + ,)
F	$, + , * ,)

wenshli@bupt.edu.cn

算法4.2 预测分析表的构造方法

输入：文法G

输出：文法G的预测分析表M

方法：

```
for (文法G的每个产生式 A→α ) {
    for (每个终结符号 a∈FIRST(α))
        把 A→α 放入 M[A, a] 中;
    if (ε∈FIRST(α))
        for (任何 b∈FOLLOW(A))
            把 A→α 放入 M[A, b] 中;
  };
for (所有无定义的M[A, a])
    标上错误标志;
```

E→TE′

E′→+TE′ | ε

T→FT′

T′→*FT′ | ε

F→(E) | id

	FIRST	FOLLOW
E	(, id	$,)
E’	+, ε	$,)
T	(, id	$, + ,)
T’	*, ε	$, + ,)
F	(, id	$, + , * ,)

	id	+	*	(	)	$
E	E→TE′			E→TE′		
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′			T→FT′		
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id			F→(E)		

wenshli@bupt.edu.cn

LL(1)文法

- **LL(1)文法**
  如果一个文法的预测分析表M不含多重定义的表项，则称该文法为LL(1)文法。
- LL(1)的含义：
  - 第一个L表示从左至右扫描输入符号串
  - 第二个L表示生成输入串的一个最左推导
  - 1表示在决定分析程序的每步动作时，向前看一个符号

例：考虑如下映射程序设计语言中if语句的文法

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \varepsilon$

$E \rightarrow b$ （文法4.5）

构造各非终结符号的FIRST和FOLLOW集合：

	S	S′	E
FIRST	i, a	e, ε	b
FOLLOW	$, e	$, e	t

First(iEtSS')∩First(a)=ϕ

First(eS)∩Follow(S')={e}

应用算法4.2，构造该文法的分析表：

	a	b	e	i	t	$
S	S→a			S→iEtSS′		
S′			S′→eS S′→ε			S′→ε
E		E→b				

wenshli@bupt.edu.cn

LL(1)文法的判断

- **根据文法产生式判断：**

  一个文法是LL(1)文法，当且仅当它的每一个产生式A→α|β，满足：

  - FIRST(α)∩FIRST(β)=ϕ　并且
  - 若β推导出 ε，则FIRST(α)∩FOLLOW(A)=ϕ

- **根据分析表判断：**

  如果利用算法4.2构造出的分析表中不含多重定义的表项，则文法是LL(1)文法。

- 文法4.5不是LL(1)文法

	S	S′	E
FIRST	i, a	e, ε	b
FOLLOW	$, e	$, e	t

First(iEtSS') ∩ First(a) = ϕ

First(eS) ∩ Follow(S') = {e}

S→iEtSS′ | a

S′→eS | ε

E→b　　（文法4.5）

	a	b	e	i	t	$
S	S→a			S→iEtSS′		
S′			S′→eS S′→ε			S′→ε
E		E→b				

wenshli@bupt.edu.cn

练习

有文法G[S]: S→(L) | a
L →L , S | S

(1) 判断该文法是否为LL(1)文法？不是，做(2)；是，做(3)。
(2) 改写文法为LL(1)文法，继续做(3)。
(3) 构造文法的FIRST和FOLLOW集合，继续做(4)。
(4) 构造文法的LL(1)分析表。

解答：

(1) 文法含有左递归，故不是LL(1)文法

(2) 改写文法： 消除左递归

S→(L) | a

L →SL′

L′ →,SL′ | ε

判断改写后的文法是LL(1)文法：

FIRST((L)) ∩FIRST(a) = ϕ

FIRST(,SL′) ∩FOLLOW(L′) = ϕ

(3) 构造文法的FIRST和FOLLOW集合：

	FIRST	FOLLOW
S	(a	$,)
L	(a	)
L′	, ε	)

(4) 构造文法的LL(1)分析表：

	a	(	)	,	$
S	S→a	S→(L)			
L	L→SL′	L→SL′			
L′			L′→ε	L′→,SL′	

wenshli@bupt.edu.cn

练习

有文法G[S]:
(1) S→aAcBe (2) A→b (3) A→Ab (4) B→d
请构造其LL(1)分析表，并分析 abbcde。

■ 解答

(1)改写文法:

S→aAcBe

A→bA’

A’ →bA’ | ε

B→d

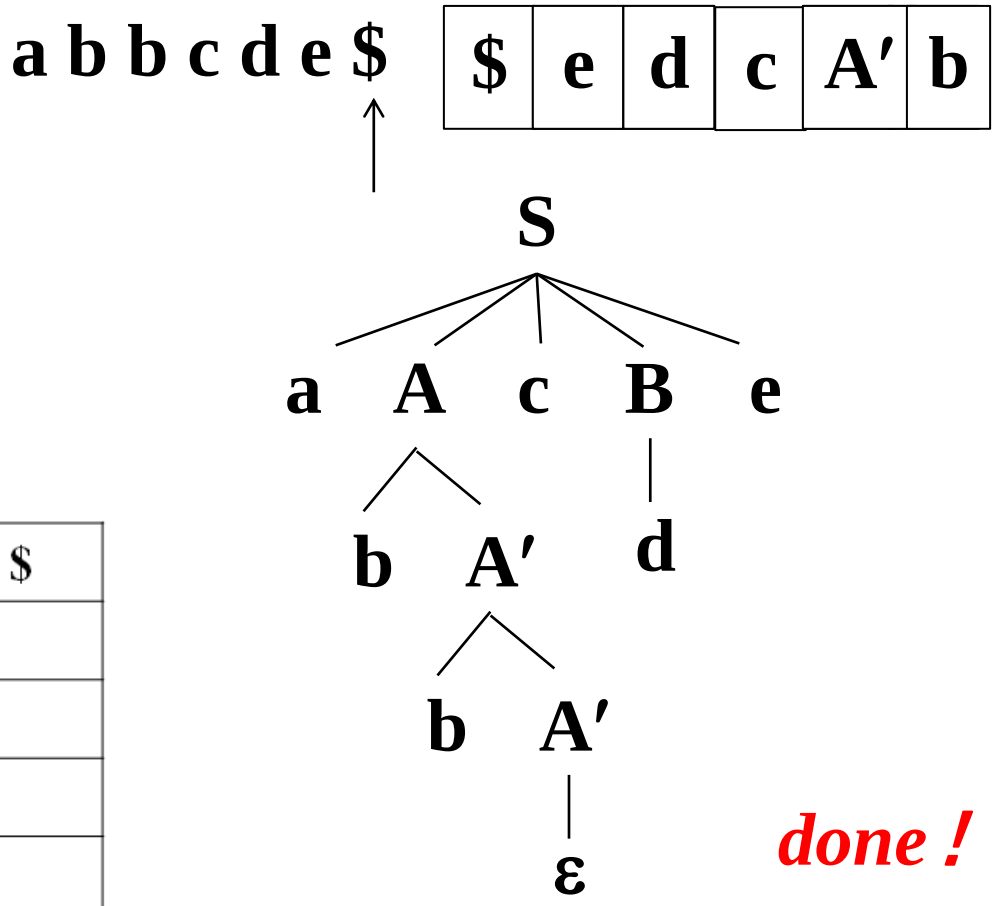
	First	follow
S	a	$
A	b	c
A’	b, ε	c
B	d	e

(2) 构造FIRST和FOLLOW集合:

(3) 构造LL(1)分析表:

	a	b	c	d	e	$
S	S→aAcBe					
A		A→bA’				
A’		A’→bA’	A’→ε			
B				B→d		

■ abbcde 的分析过程



done !

wenshli@bupt.edu.cn

预测分析方法中的错误处理示例

- 发现错误：

  (1) $X \in V_T$，但$X \neq a$；

  (2) $X \in V_N$，但M[X, a]为空。

- 处理方法：

  情况(1)：弹出栈顶的终结符号；

  情况(2)：跳过剩余输入串中的若干个符号，直到可以继续进行分析为止。

- 带有同步化信息的分析表的构造
  - 对于$A \in V_N$，$b \in FOLLOW(A)$，若M[A, b]为空，则加入“synch”
- 带有同步化信息的分析表的使用
  - 若M[A, b]为空，则跳过a；
  - 若M[A, b]为synch，则弹出A。

- 构造文法4.4的带有同步化信息的分析表

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id$

	FIRST	FOLLOW
E	(, id	$,)
E’	+, ε	$,)
T	(, id	$, + ,)
T’	*, ε	$, + ,)
F	(, id	$, + , * ,)

	id	+	*	(	)	$
E	E→TE′			E→TE′	synch	synch
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′	synch		T→FT′	synch	synch
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id	synch	synch	F→(E)	synch	synch

wenshli@bupt.edu.cn

示例

栈	输入	输出
$E	*id*+id$	出错，M[E,*]=空白，跳过*
$E	id*+id$	E→TE′
$E′T	id*+id$	T→FT′
$E′T′F	id*+id$	F→id
$E′T′id	id*+id$	
$E′T′	*+id$	T′→*FT′
$E′T′F*	*+id$	
$E′T′F	+id$	出错，M[F,+]=synch，弹出F
$E′T′	+id$	T′→ε
$E′	+id$	E′→+TE′
$E′T+	+id$	
$E′T	id$	T→FT′
$E′T′F	id$	F→id
$E′T′id	id$	
$E′T′	$	T′→ε
$E′	$	E′→ε
$	$	

	id	+	*	(	)	$
E	E→TE′			E→TE′	synch	synch
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′	synch		T→FT′	synch	synch
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id	synch	synch	F→(E)	synch	synch

- 带有同步化信息的分析表的使用
  - 若M[A, b]为空，则跳过a；
  - 若M[A, b]为synch，则弹出A。

课堂练习1

有如下文法：

$E \rightarrow E \vee T \mid T$

$T \rightarrow T \wedge F \mid F$

$F \rightarrow \neg F \mid (E) \mid t \mid f$

(1) 该文法是LL(1)文法吗？说明理由。
若是，继续做(3)，若不是，继续做(2)。

(2) 请改写该文法为LL(1)文法，继续做(3)。

(3) 构造每个非终结符号的FIRST和FOLLOW函数，继续做(4) 。

(4) 构造LL(1)分析表。

wenshli@bupt.edu.cn

参考答案

E→E∨T | T
T→T∧F | F
F→¬F | （E） | t | f

(1) 文法不是LL(1)文法。
理由：
由E→E∨T | T 可知，
文法中存在左递归。

(2) 消除左递归，
得到文法G’：

E→TE’
E’→∨TE’ | ε
T→FT’
T’→∧FT’ | ε
F→¬F | (E) | t | f

(3) 非终结符号的FIRST集合和FOLLOW集合：

	FIRST	FOLLOW
E	¬, (, t, f	$,)
E’	∨, ε	$,)
T	¬, (, t, f	∨, $,)
T’	∧, ε	∨, $,)
F	¬, (, t, f	∧, ∨, $,)

(4) 文法的LL(1)分析表：

	¬	∧	∨	t	f	(	)	$
E	E→TE’			E→TE’	E→TE’	E→TE’		
E’			E’→∨TE’				E’→ε	E’→ε
T	T→FT’			T→FT’	T→FT’	T→FT’		
T’		T’→∧FT’	T’→ε				T’→ε	T’→ε
F	F→¬F			F→t	F→f	F→(E)		

wenshli@bupt.edu.cn

4.3 自底向上分析方法

- 对输入串的扫描：自左向右
- 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

wenshli@bupt.edu.cn

优先分析法简介（*）

- 分为：简单优先分析法和算符优先分析法。
- 简单优先分析法
  - 规范归约。
  - 按照文法符号之间的优先关系确定当前句型的“可归约串”。
  - 分析效率低，且只适用于简单优先文法。
  - 简单优先文法，满足以下两个条件：

    (1) 任何两个文法符号之间最多存在一种优先关系。

    (2) 不存在具有相同右部的产生式。

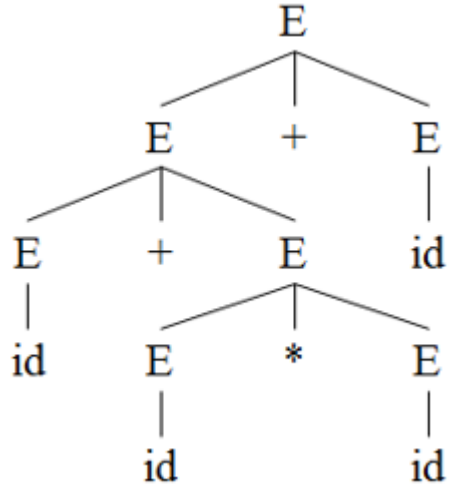
wenshli@bupt.edu.cn

优先分析法简介

- 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。
  - 算符文法：
    没有形如A→…BC…的产生式，其中B、C∈V_N
  - 算符优先文法：
    - 算符文法、且不含有ε-产生式；
    - 任何两个构成序对的终结符号之间最多有 >、= 和 < 三种优先关系中的一种成立。
  - “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。

$E \rightarrow E+E \mid E*E \mid (E) \mid \mathrm{id}$

+<*、*>+、+>+、*>*、
+<(、*<(、)>*、)>+、 (=)、
id>+、+<id、id>*、*<id



“移进-归约”分析方法

- 符号栈：存放文法符号
- 分析过程：

(1) 把输入符号逐个地移进栈中。

(2) 当栈顶的符号串形成某个产生式的一个候选式时，在一定条件下，把该符号串替换/归约为该产生式的左部符号。

(3) 重复(2)，直到栈顶符号串不再是“可归约串”为止。

(4) 重复(1)~(3)，直到最终归约出文法开始符号S。

wenshli@bupt.edu.cn

规范归约

句柄？子树？==>

分析符号串abbcde是否为如下文法的句子。

(1) S→aAcBe

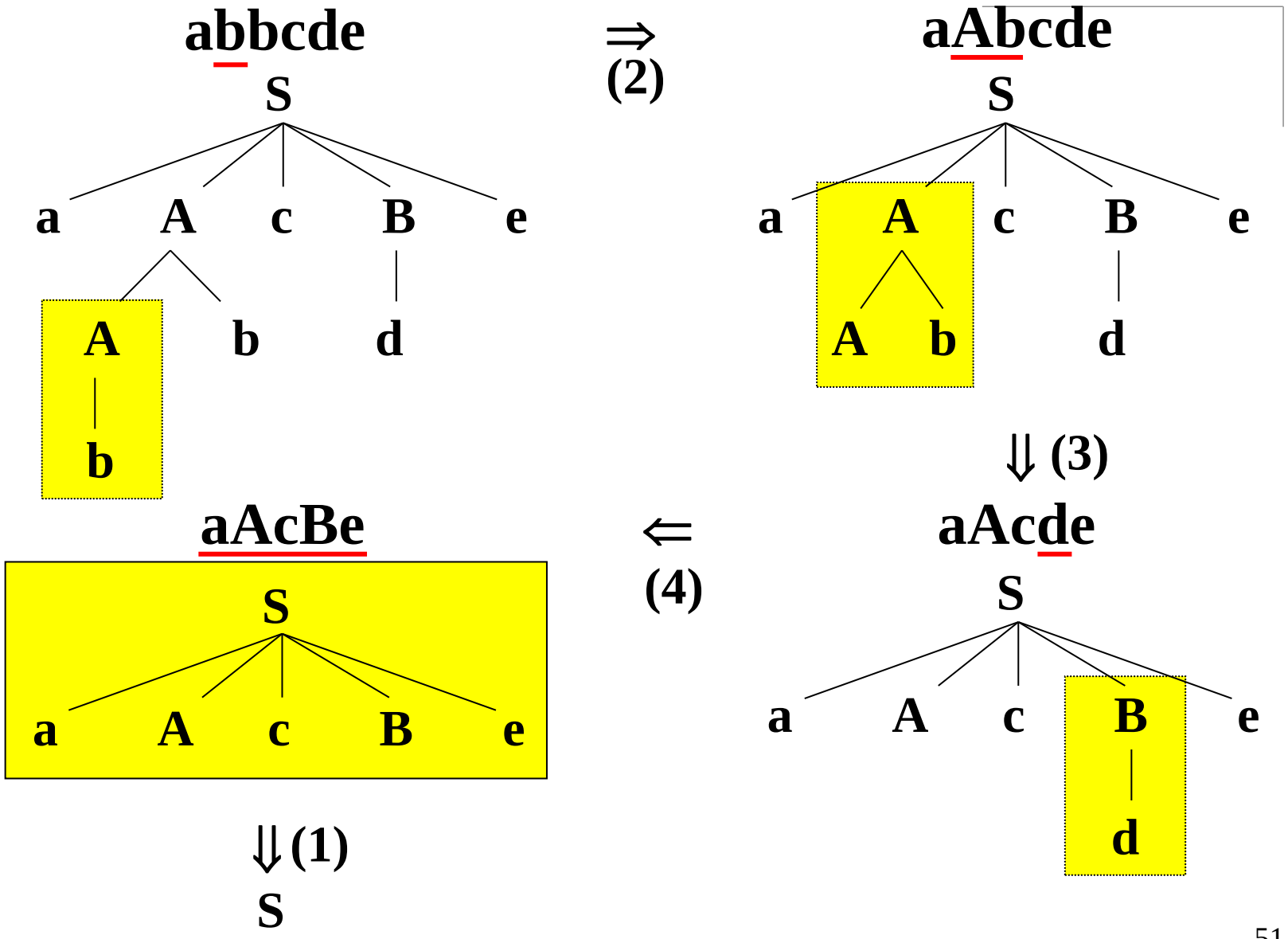
(2) A→b

(3) A→Ab

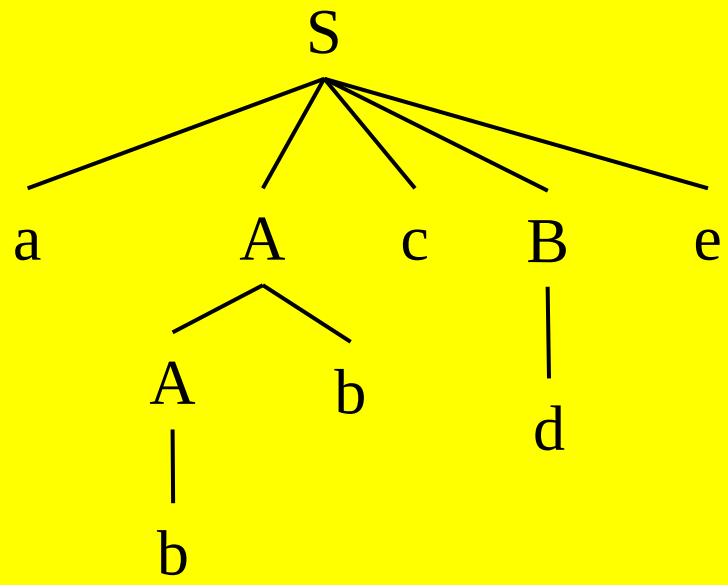
(4) B→d （文法4.1）

- 最右推导：

$$S \underset{rm}{\overset{(1)}{\Rightarrow}} aAcBe \underset{rm}{\overset{(4)}{\Rightarrow}} aAcde \underset{rm}{\overset{(3)}{\Rightarrow}} aAbcde \underset{rm}{\overset{(2)}{\Rightarrow}} abbcde$$



规范归约：最右推导的逆过程



a b b c d e

a A b c d e

a A c d e

a A c B e

S

S

A→b

A→Ab

B→d

S→aAcBe

最右推导：

S⇒aAcBe

⇒aAcde

⇒aAbcde

⇒abbcde

规范归约定义：

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≤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。

wenshli@bupt.edu.cn

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

wenshli@bupt.edu.cn

“移进-归约”方法的实现

abbcde 的归约过程

- 使用一个栈和一个输入缓冲区
  - 分析开始时，将符号 $ 入栈，以示栈底；
  - 在输入符号串之后置$，以示符号串的结束。

栈	输入
$	ω$

开始

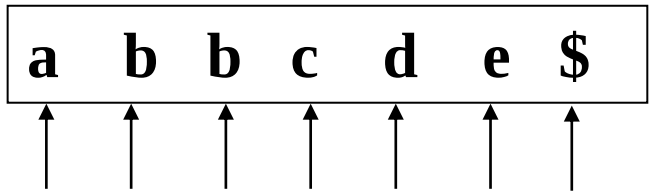
栈	输入
$S	$

结束

	栈	输入	分析动作
(1)	$	abbcde$	shift
(2)	$a	bbcde$	shift
(3)	$ab	bcde$	reduce by A→b
(4)	$aA	bcde$	shift
(5)	$aAb	cde$	reduce by A→Ab
(6)	$aA	cde$	shift
(7)	$aAc	de$	shift
(8)	$aAcd	e$	reduce by B→d
(9)	$aAcB	e$	shift
(10)	$aAcBe	$	reduce by S→aAcBe
(11)	$S	$	accept

wenshli@bupt.edu.cn

句子 abbcde 的规范归约过程

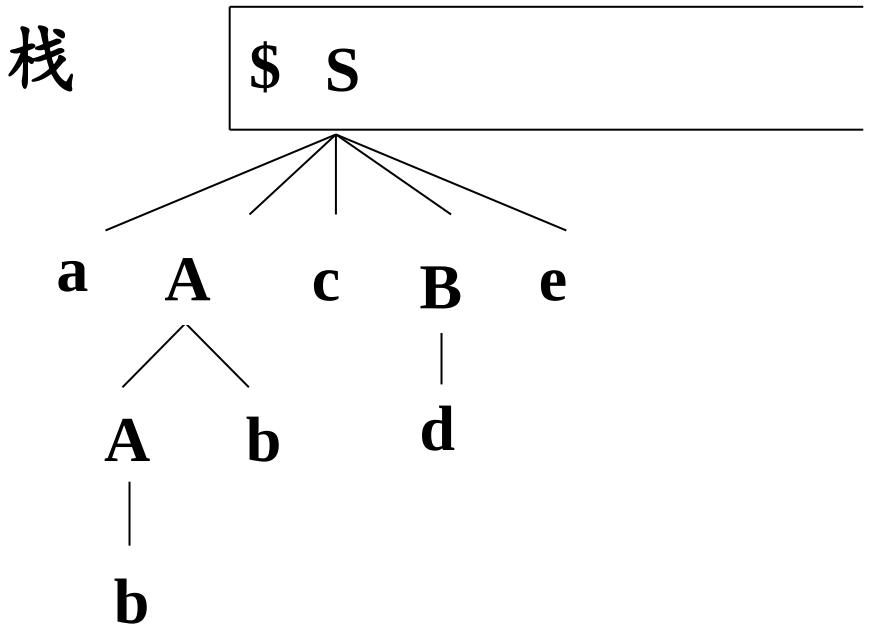


栈	输入
$	ω$

开始

栈	输入
$S	$

结束



规范归约

a b b c d e

a A b c d e

a A c d e

a A c B e

S

A→b

A→Ab

B→d

S→aAcBe

最右推导

wenshli@bupt.edu.cn

“移进-归约”分析方法的分析动作

移进：把下一个输入符号移进到栈顶。

归约：用适当的归约符号替换栈顶可归约串。

接受：宣布分析成功，停止分析。

错误处理：调用错误处理程序进行诊断和恢复。

分析过程中的动作冲突：

“移进-归约”冲突

“归约-归约”冲突

wenshli@bupt.edu.cn

4.4 LR分析方法

LR分析技术概述

1. LR分析程序的模型及工作过程
2. SLR(1)分析表的构造
3. LR(1)分析表的构造
4. LALR(1)分析表的构造
5. LR分析的错误处理与恢复

wenshli@bupt.edu.cn

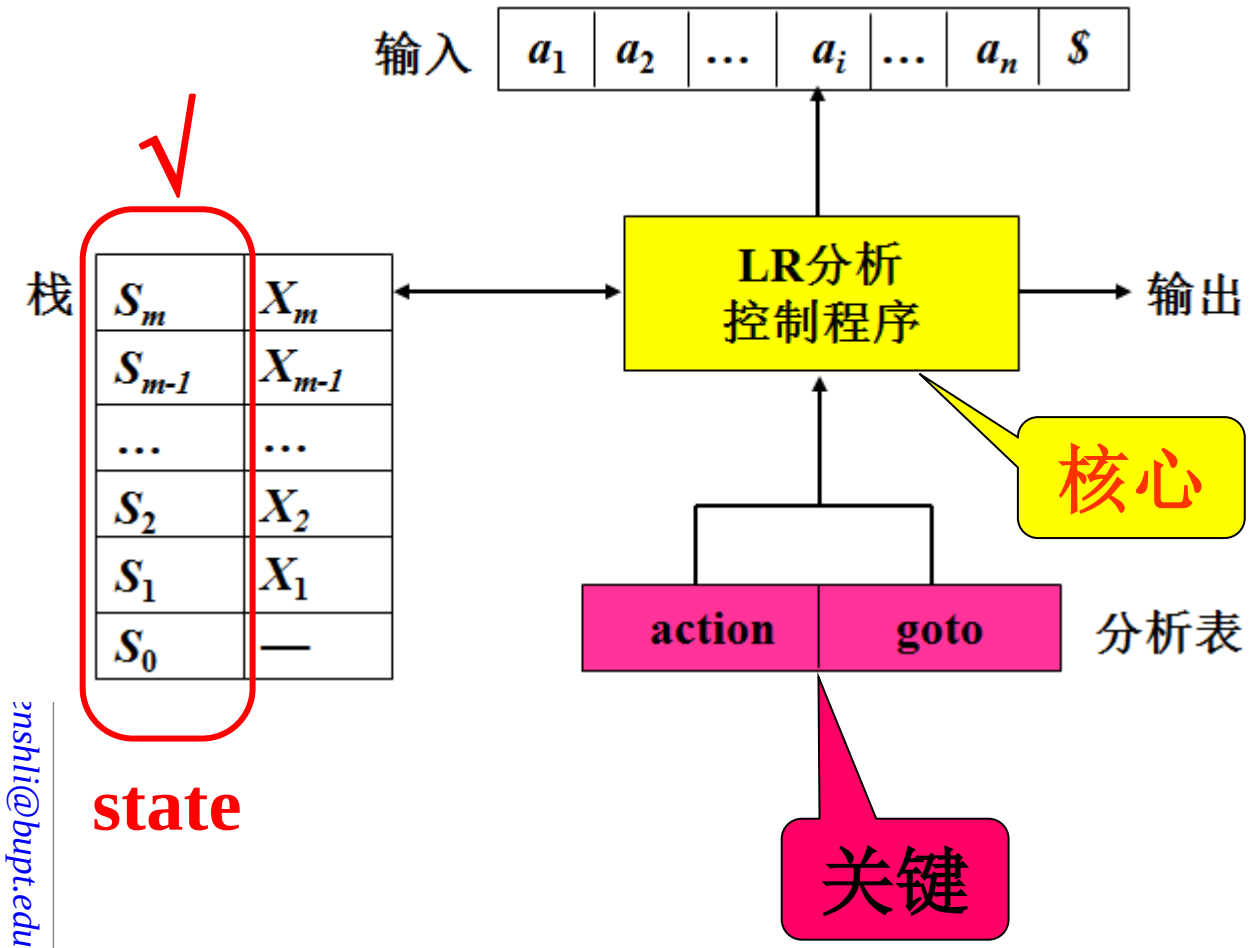
LR分析技术概述

- LR(k)的含义：
  - L：自左至右扫描输入符号串
  - R：为输入符号串构造一个最右推导的逆过程
  - k：为作出分析决定而向前看的输入符号的个数
- 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”：根据所用产生式推测未来可能遇到的输入符号；
  - “历史”+“预测”+“现实”：栈顶的符号串是否构成相对于某一产生式的句柄。
- 比较完备的技术
  - 能分析所有能用上下文无关文法书写的程序设计语言结构；
  - “移进-归约“方法；
  - 无回溯；
  - 能及时发现错误。
  - 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。
- 不足之处
  - 手工编写LR分析程序的工作量太大
  - 专门的工具，LR分析程序生成器（如YACC）

wenshli@bupt.edu.cn

1. LR分析程序的模型及工作过程

- LR分析程序的模型



- LR分析控制程序工作的依据
- goto[S_m, X]：S_m 经 X 的后继状态
- action[S_m, a_i]：
  - shift S：把当前输入符号 a_i 及状态S推进栈顶，向前扫描指针前移。
    这里，S=goto[S_m, a_i] 。
  - reduce by A→β：若 |β|= r，则从栈顶弹出 r 项，使 S_{m-r} 成为栈顶状态，然后把 A及状态S推进栈顶。
    这里， S=goto[S_{m-r}, A] 。
  - accept：宣布分析成功，停止分析。
  - error：调用出错处理程序，进行错误恢复。

wanshli@bupt.edu.cn

LR分析控制程序的工作过程

- 分析开始时，初始的二元式为：$(S_0，a_1 a_2 \ldots a_n \$)$
- 分析过程中每步的结果，均可表示为如下的二元式：
  $(S_0 S_1 S_2 \ldots S_m，a_i a_{i+1} \ldots a_n \$)$
- 若$action[S_m, a_i]=shift\ S$，$S=goto[S_m, a_i]$，
  则二元式变为：$(S_0 S_1 \ldots S_m S，a_{i+1} \ldots a_n \$)$
- 若$action[S_m, a_i]=reduce\ by\ A\rightarrow\beta$，
  则二元式变为：$(S_0 S_1 \ldots S_{m-r} S，a_i a_{i+1} \ldots a_n \$)$
- 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），
  则发现错误，调用错误处理程序。

wenshli@bupt.edu.cn

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 指向第一个符号；

```
do { // 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 A→β;
        };
        else if (action[S, a]==accept)  return;
            else  error();
} while(1).
```

wenshli@bupt.edu.cn

对输入符号串 id+id*id 的分析过程

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			

(0) E'→E
(1) E →E+T
(2) E →T
(3) T →T*F
(4) T →F
(5) F →（E）
(6) F →id

输入：id + id * id $

分析栈：

0	1
-	E

Accept!

wenshli@bupt.edu.cn

“活前缀”的概念

前缀？==>

- 定义：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。
- 例：右句型 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

活前缀与句柄之间的关系？

- 称之为活前缀，是因为在其右边增加某些终结符号之后，就可以构成一个规范句型。
- 分析过程中 $(S_0X_1S_1X_2\ldots X_mS_m,\ a_ia_{i+1}\ldots a_n\$)$

  $X_1X_2\ldots X_m$ 是一个活前缀；

  $X_1X_2\ldots X_m a_i a_{i+1}\ldots a_n$ 是一个右句型。

句柄与栈顶之间的关系？

2. SLR(1)分析表的构造

- 基本思想：
  - 为给定的文法构造一个识别它所有活前缀的DFA
  - 根据该DFA构造文法的分析表

- 内容
  - 识别给定文法的所有活前缀的DFA的构造方法
  - SLR(1)分析表的构造方法

wenshli@bupt.edu.cn

识别给定文法所有活前缀的DFA的构造方法

- 活前缀与句柄之间的关系
  - ① 活前缀不含有句柄的任何符号
  - ② 活前缀只含有句柄的部分符号
  - ③ 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①种情况，期望从剩余输入串中能够看到由某产生式 $A\rightarrow\alpha$ 的右部 α 所推导出的终结符号串；
  - 第②种情况，某产生式 $A\rightarrow\alpha_1\alpha_2$ 的右部子串 α_1 已经出现在栈顶，期待从剩余的输入串中能够看到 α_2 推导出的符号串；
  - 第③种情况，某一产生式 $A\rightarrow\alpha$ 的右部符号串 α 已经出现在栈顶，用该产生式进行归约。

wenshli@bupt.edu.cn

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
} 移进-待约项目

A→XYZ•　　归约项目

  - 移进项目：圆点后第一个符号为终结符号的LR(0)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
  - 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

产生式A→ε，只有一个LR(0)归约项目 A→•

wenshli@bupt.edu.cn

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$S' \rightarrow S\bullet$)

wenshli@bupt.edu.cn

定义：LR(0)有效项目、有效项目集、项目集规范族

- 项目$A\to\beta_1\bullet\beta_2$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$A\to\alpha\bullet B\beta$对活前缀$\gamma=\delta\alpha$是有效的，并且有产生式$B\to\eta$，则项目$B\to\bullet\eta$对活前缀$\gamma=\delta\alpha$也是有效的。

$$S \overset{*}{\Rightarrow} \delta A\omega \Rightarrow \delta\alpha B\beta\omega \overset{*}{\Rightarrow} \delta\alpha B\lambda \Rightarrow \delta\alpha\eta\lambda$$

- LR(0)项目$S'\to\bullet S$是活前缀ε的有效项目

（这里取$\gamma=\varepsilon$，$\beta_1=\varepsilon$，$\beta_2=S$，$A=S'$）

$S' \Rightarrow S$

- LR(0)有效项目集：

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。

- LR(0)项目集规范族：

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。

wenshli@bupt.edu.cn

定义：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G有产生式 B→η，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。

- 算法4.4 closure(I)的构造过程

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).
```

wenshli@bupt.edu.cn

定义：转移函数go

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。

wenshli@bupt.edu.cn

算法4.5 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);
```

- 文法 4.6

  S→aA | bB　　A→cA | d　　B→cB | d

- 拓广文法G′：

  (0) S′→S

  (1) S→aA　(2) S→bB

  (3) A→cA　(4) A→d

  (5) B→cB　(6) B→d

- 活前缀 ε 的有效项目集

  I_0= closure({S′→•S})

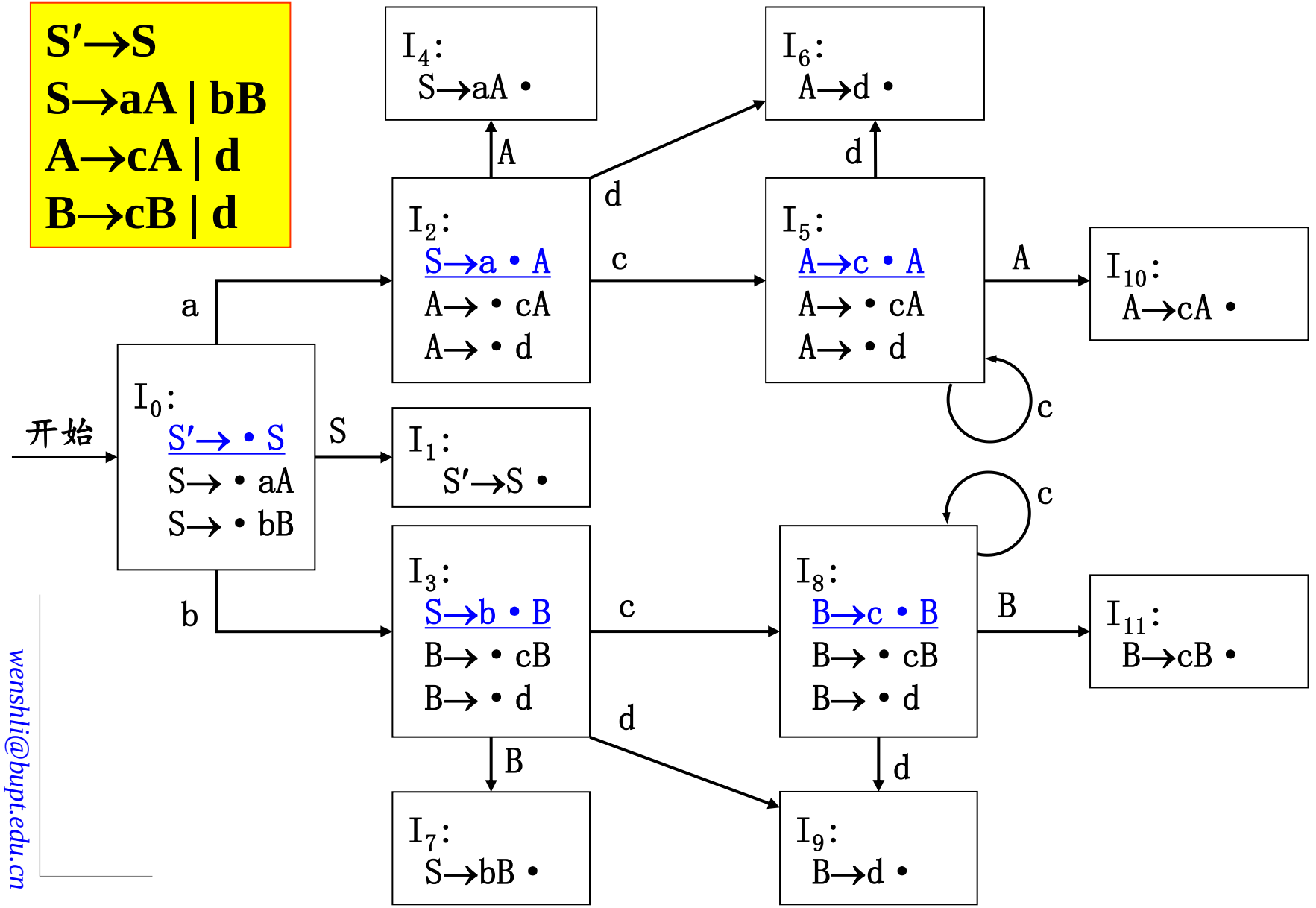
  ={ S′→•S，S→•aA，S→•bB }

wenshli@bupt.edu.cn

识别文法G′的所有活前缀的DFA

练习：分析 accd

S′→S
S→aA | bB
A→cA | d
B→cB | d



a c c d $

0	1	4	10	10
-	S	A	A	A

Accept!

wenshli@bupt.edu.cn

LR(0)项目集中的冲突及解决

- 如项目集：I={X→α•bβ，A→α•，B→β•}
  - 存在移进-归约冲突
  - 存在归约-归约冲突
- 冲突的解决：查看 FOLLOW(A) 和 FOLLOW(B)
  - FOLLOW(A)∩FOLLOW(B)=Φ
  - b∉FOLLOW(A) 并且 b∉FOLLOW(B)
  - 决策：
    - 当 a=b 时，把 b 移进栈顶；
    - 当 a∈FOLLOW(A) 时，用 A→α 进行归约；
    - 当 a∈FOLLOW(B) 时，用 B→β 进行归约。

wenshli@bupt.edu.cn

算法4.6 构造SLR(1)分析表

- SLR(1)分析表
  分析表不含有冲突
- SLR(1)文法
  具有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={$I_0, I_1, \ldots, I_n$}。
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$A\rightarrow\alpha\bullet a\beta\in I_i$，且$go(I_i, a)=I_j$，则置 action[i, a]=Sj
   b) 若$A\rightarrow\alpha\bullet\in I_i$，则对所有$a\in FOLLOW(A)$，置 action[i, a]=R $A\rightarrow\alpha$
   c) 若$S'\rightarrow S\bullet\in I_i$，则置 action[i, $]=ACC，表示分析成功
3. 若$go(I_i, A)=I_j$，A为非终结符号，则置 goto[i, A]=j
4. 分析表中凡不能用规则2、3填入信息的空白表项，均置出错标志error。
5. 分析程序的初态是包含项目$S'\rightarrow S$的有效项目集所对应的状态。

wenshli@bupt.edu.cn

文法4.6的SLR(1)分析表

(0) S′→S
(1) S→aA (2) S→bB (3) A→cA
(4) A→d (5) B→cB (6) B→d

	FOLLOW
S’	$
S	$
A	$
B	$

状态	action					goto		
	a	b	c	d	$	S	A	B
0	S2	S3				1		
1					acc			
2			S5	S6			4	
3			S8	S9				7
4					R1			
5			S5	S6			10	
6					R4			
7					R2			
8			S8	S9				11
9					R6			
10					R3			
11					R5			

I_0: S′→ • S　S→ • aA
S→ • bB

I_1: S′→S •

I_2: S→a • A　A→ • cA
A→ • d

I_3: S→b • B　B→ • cB
B→ • d

I_4: S→aA •

I_5: A→c • A　A→ • cA
A→ • d

I_6: A→d •

I_7: S→bB •

I_8: B→c • B　B→ • cB
B→ • d

I_9: B→d •

I_{10}: A→cA •

I_{11}: B→cB •

wenshli@bupt.edu.cn

课堂练习2　　解答：证明文法是LL(1)文法

- 证明文法G[S]是LL(1)文法，但不是SLR(1)文法。

  S→(X | E] | F)

  X→E) | F]

  E→A

  F→A

  A→ε

- 方法一：

  构造非终结符号的FIRST集：

	S	X	E	F	A
FIRST	(,],)	],)	ε	ε	ε

对于 S→(X | E] | F)
任何两个候选式的FIRST集没有交集。

对于X→E) | F]
两个候选式的FIRST集没有交集。

wenshli@bupt.edu.cn

解答：证明该文法是LL(1)文法

S→(X | E] | F)
X→E) | F]
E→A
F→A
A→ε

■ 方法二：

构造每个非终结符号的FIRST集和FOLLOW集合，如下：

	FIRST	FOLLOW
S	(,],)	$
X	],)	$
E	ε	],)
F	ε	],)
A	ε	],)

	(	)	]	$
S	S→(X	S→F)	S→E]	
X		X→E)	X→F]	
E		E→A	E→A	
F		F→A	F→A	
A		A→ ε	A→ ε	

构造文法的LL(1)分析表，如右：

LL(1)分析表中不含有多重定义的入口，所以该文法是LL(1)文法。

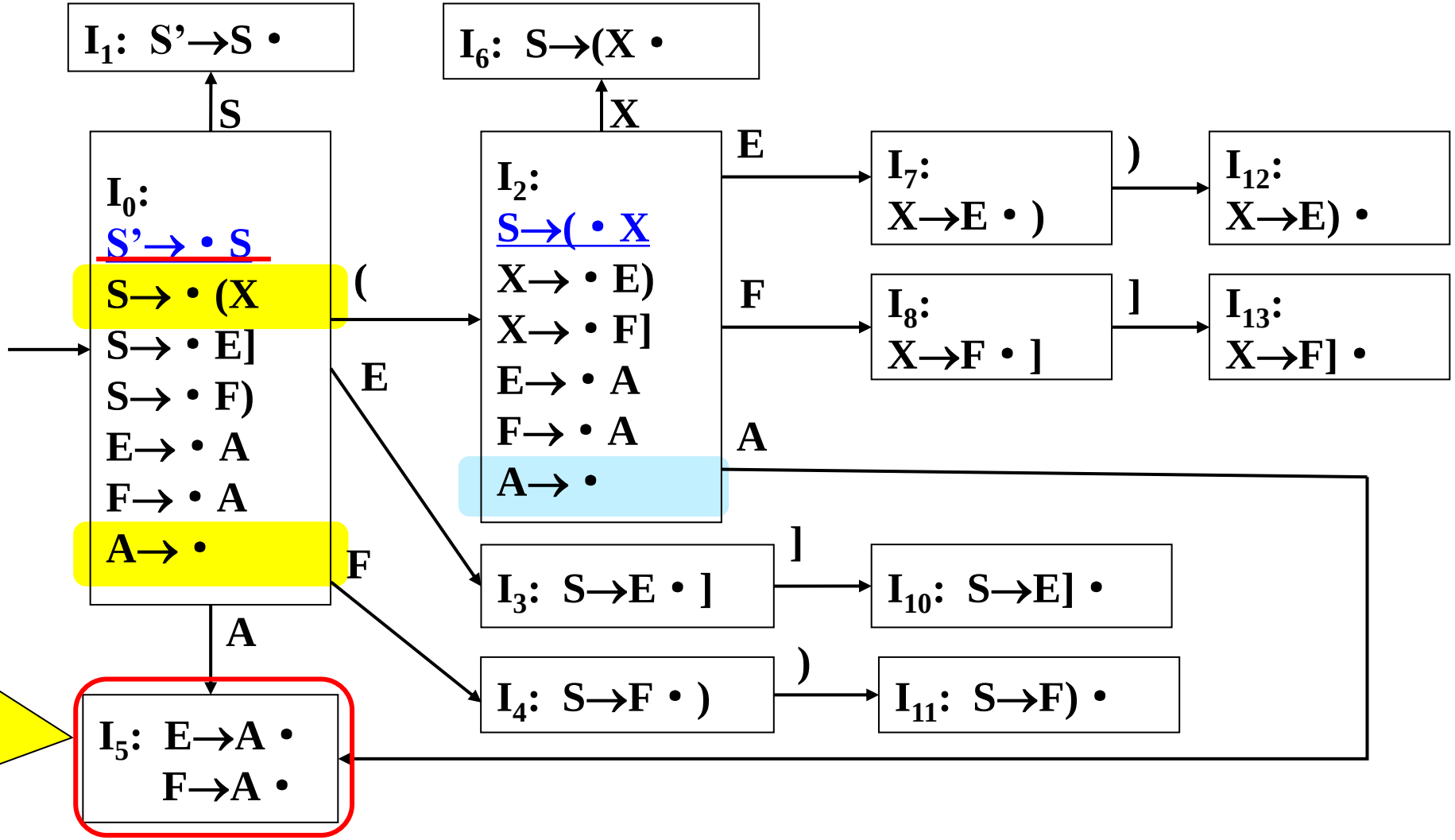
wenshli@bupt.edu.cn

解答：证明该文法不是SLR(1)文法

S→(X | E] | F)　X→E) | F]
E→A　F→A　A→ε

- 拓广文法，构造文法的LR(0)项目集规范族及识别所有活前缀的DFA。

	FOLLOW
S	$
X	$
E	],)
F	],)
A	],)



归约-归约冲突
FOLLOW(E)
=FOLLOW(F)
={],) }
该文法不是
SLR(1)文法。

wenshli@bupt.edu.cn

LR(0)分析表和LR(0)文法

- 一个文法是LR(0)文法，当且仅当该文法的每个活前缀的有效项目集中：
  - 要么所有元素都是移进-待约项目
  - 要么只含有唯一的归约项目
- 具有LR(0)分析表的文法，称为LR(0)文法。
  - 在执行算法4.6的过程中，不需要向前看任何输入符号以解决冲突，则构造的SLR(1)分析表称为LR(0)分析表。
- 例：如下文法G[A]是LR(0)文法。

  $A \rightarrow (A) \mid a$

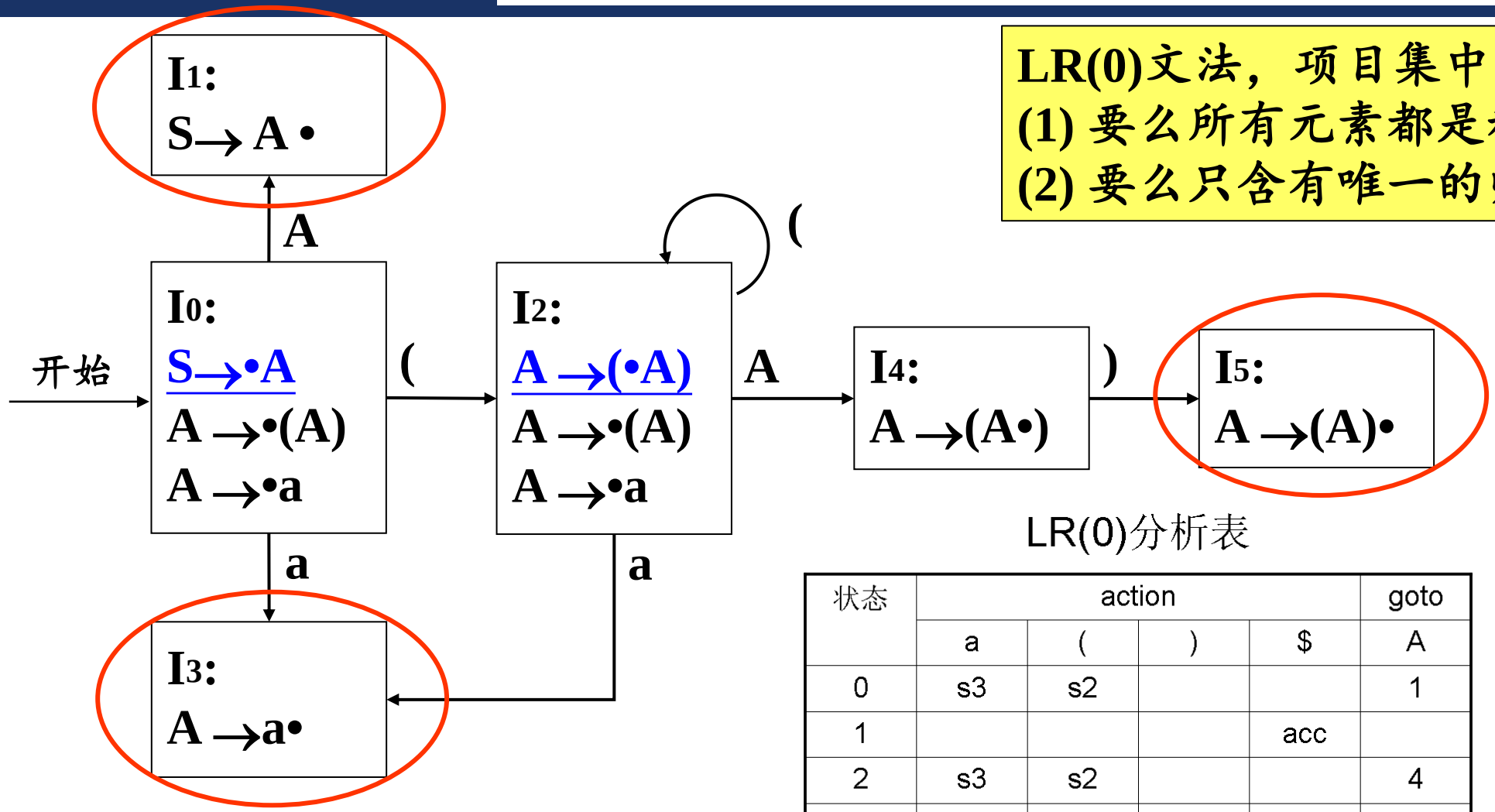
练习：如何判断？

wenshli@bupt.edu.cn

判断过程：

拓广文法：(0) S→A　(1) A→(A)　(2) A→a

LR(0)文法，项目集中：
(1) 要么所有元素都是移进-待约项目
(2) 要么只含有唯一的归约项目



LR(0)分析表

状态	action				goto
	a	(	)	$	A
0	s3	s2			1
1				acc	
2	s3	s2			4
3	r2	r2	r2	r2	
4			s5		
5	r1	r1	r1	r1	

wenshli@bupt.edu.cn

示例

- 判断文法4.3是LR(0)文法，还是SLR(1)文法?

  E→E+T | T

  T→T*F | F

  F→(E) | id

- 解答：
  - 拓广文法4.3为G′：

    (0) E′→E　　(1) E→E+T　　(2) E→T

    (3) T→T*F　　(4) T→F　　(5) F→(E)　　(6) F→id

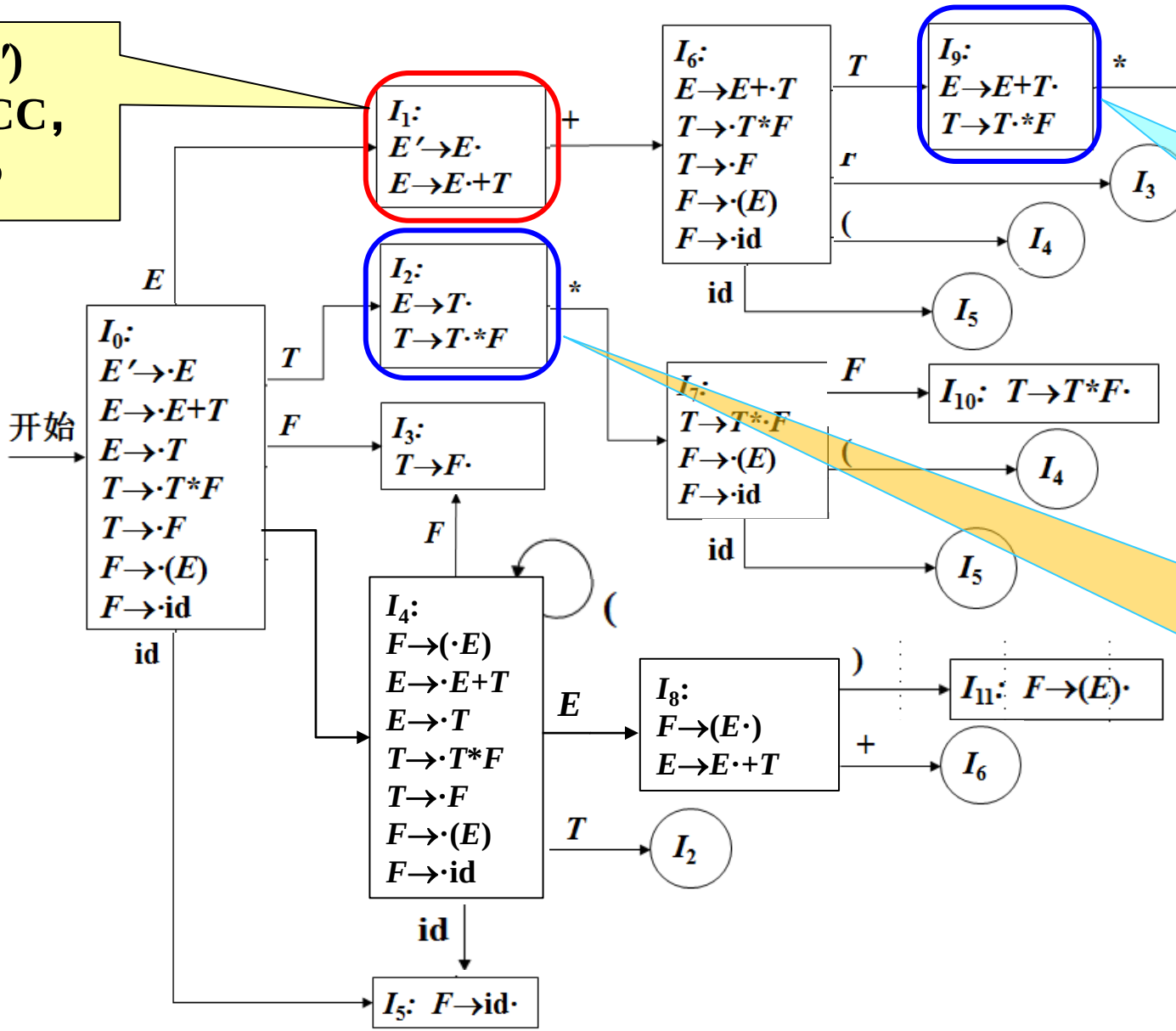
  - 构造G′的LR(0)项目集规范族及识别其所有活前缀的DFA：

I_0:
E′→ • E
E→ • E+T
E→ • T
T→ • T*F
T→ • F
F→ • (E)
F→ • id

wenshli@bupt.edu.cn

G'的 LR(0)项目集规范族及识别它所有活前缀的DFA

+∉FOLLOW(*E'*)
action[1, $] = ACC，
action[1, +] = *S*6



	FOLLOW
E'	$
E	$, +,)
T	$, +, *,)
F	$, +, *,)

*∉FOLLOW(*E*)
action[9, +]
= action[9,)]
= action[9, $] = *R*1
action[2, *] = *S*7

*∉FOLLOW(*E*)
action[2, +]
= action[2,)]
= action[2, $] = *R*2
action[2, *] = *S*7

wenshli@bupt.edu.cn

定理

- 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：如下文法G[S]无二义性，但不是SLR(1)文法。

  S→L=R

  S→R

  L→*R

  L→id

  R→L　　（文法4.7）

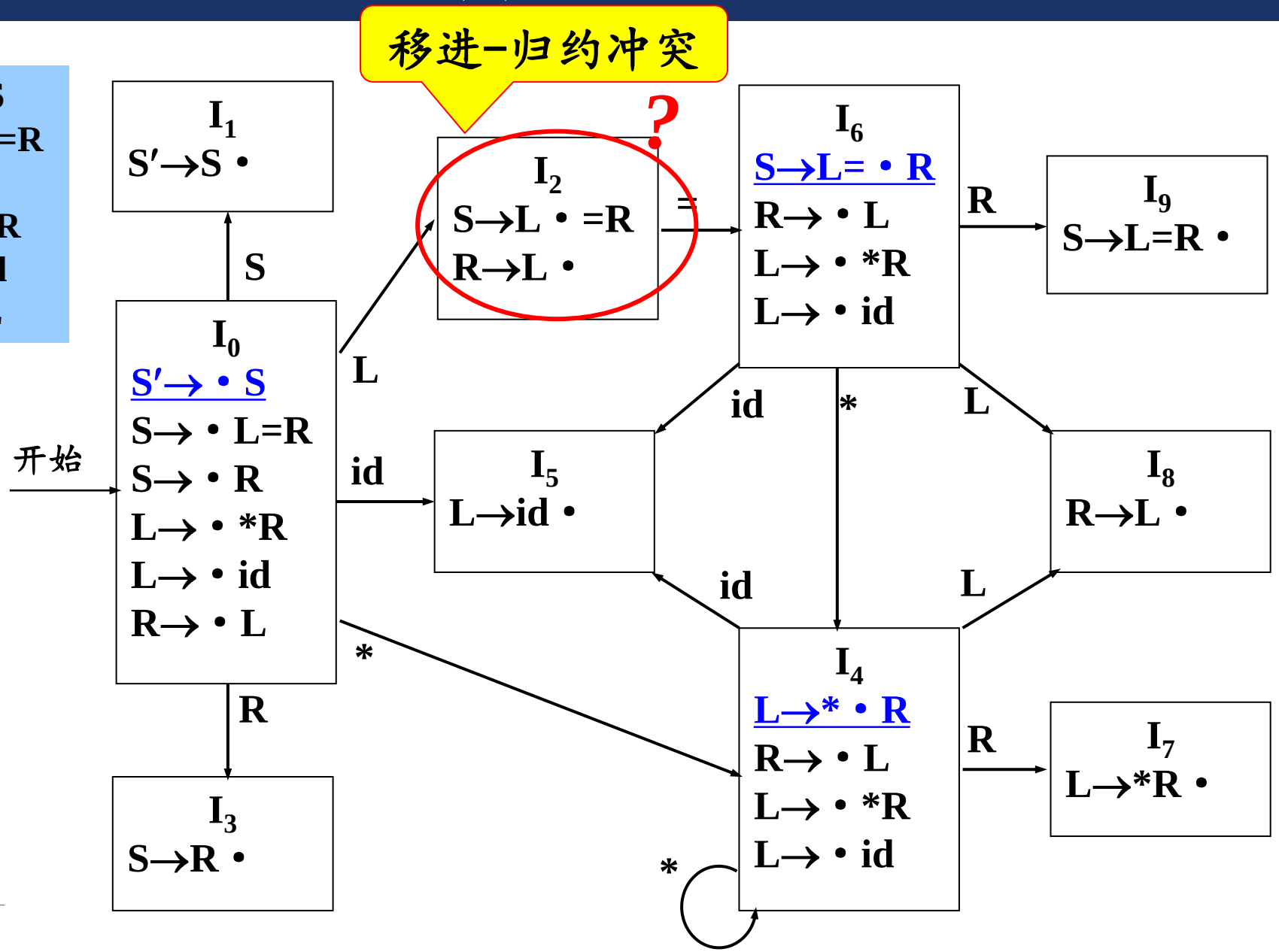
- 拓广文法G′的产生式如下：

  (0) S′→S　　(1) S→L=R　　(2) S→R

  (3) L→*R　　(4) L→id　　(5) R→L

wenshli@bupt.edu.cn

构造文法G′的LR(0)项目集规范族及识别活前缀的DFA



FOLLOW(R) ={ =, $ }

分析 *id=id

* id = id $

0	2	7
-	L	R

wenshli@bupt.edu.cn

3. LR(1)分析表的构造

- LR(k)项目：$[A\rightarrow\alpha\bullet\beta, a_1a_2\ldots a_k]$
  - $A\rightarrow\alpha\bullet\beta$是LR(0)项目
  - a_i (i=1,2,…,k)是终结符号
  - $a_1a_2\ldots a_k$称为该项目的向前看符号串
- 向前看符号串仅对归约项目$[A\rightarrow\alpha\bullet, a_1a_2\ldots a_k]$起作用
- $[A\rightarrow\alpha\bullet, a_1a_2\ldots a_k]$

  当它所属项目集对应的状态在栈顶，且后续的k个输入符号为 $a_1a_2\ldots a_k$ 时，才允许把栈顶的文法符号串α归约为A。

wenshli@bupt.edu.cn

定义：LR(1)有效项目、有效项目集、项目集规范族

- 称一个LR(1)项目 [A→α•β, a] 对活前缀 γ=δα 是有效的，如果存在一个规范推导：

$$S \overset{*}{\Rightarrow} \delta A\omega \Rightarrow \delta\alpha\beta\omega$$

其中ω的第一个符号为 a，或者ω=ε，且a=$。

- 推广：若项目[A→α•Bβ, a]对活前缀 γ=δα 是有效的，且有产生式 B→η，则对任何 b∈FIRST(βa)，项目[B→•η, b] 对活前缀γ=δα 也是有效的。
- b或是从β推出的开头终结符号，若β⇒ε 则 b=a。

$$S \overset{*}{\Rightarrow} \delta A\omega \Rightarrow \delta\alpha B\beta\omega \overset{*}{\Rightarrow} \delta\alpha B\lambda \Rightarrow \delta\alpha\eta\lambda$$

- LR(1)项目[S′→•S, $]对活前缀 ε 是有效的。

- **LR(1)有效项目集**
  活前缀γ的所有LR(1)有效项目组成的集合。
- **LR(1)项目集规范族**
  文法G的所有活前缀的LR(1)有效项目集组成的集合。

wenshli@bupt.edu.cn

定义：闭包（closure）

设I是文法G的一个LR(1)项目集，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, a]属于closure(I)，且G有产生式B→η，则对任何终结符号 b∈FIRST(βa)，若项目[B→•η, b]不属于集合closure(I)，则将它加入closure(I)；

(3) 重复规则(2)，直到closure(I)不再增大为止。

■ 算法4.7 closure(I)的构造过程

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中的每个[A→α•Bβ, a] 和
         文法G的每个产生式B→η )
     for (FIRST(βa)中的每个终结符号b)
        if  ([B→•η，b]∉J)
            把[B→•η，b]加入J;
} while (J_new!=J);
```

wenshli@bupt.edu.cn

定义：转移函数go

若I是文法G的一个LR(1)项目集，X是一个文法符号，定义：

go(I, X)=closure(J)

其中：J={ [A→αX•β, a] | 当[A→α•Xβ, a]属于I时 }

项目[A→αX•β, a]称为[A→α•Xβ, a]的后继。

直观含义：

若I是某个活前缀γ的有效项目集，

则 go(I, X) 便是对活前缀 γX 的有效项目集。

wenshli@bupt.edu.cn

算法4.8 构造文法G的LR(1)项目集规范族

输入：拓广文法G′

输出：G′的LR(1)项目集规范族

方法：

C={closure({ [S′→•S, $] }) };
do
　for (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).

例：构造文法G[S]的LR(1)项目集规范族：

(1) S→CC　(2) C→cC　(3) C→d　（文法4.8）

拓广文法：

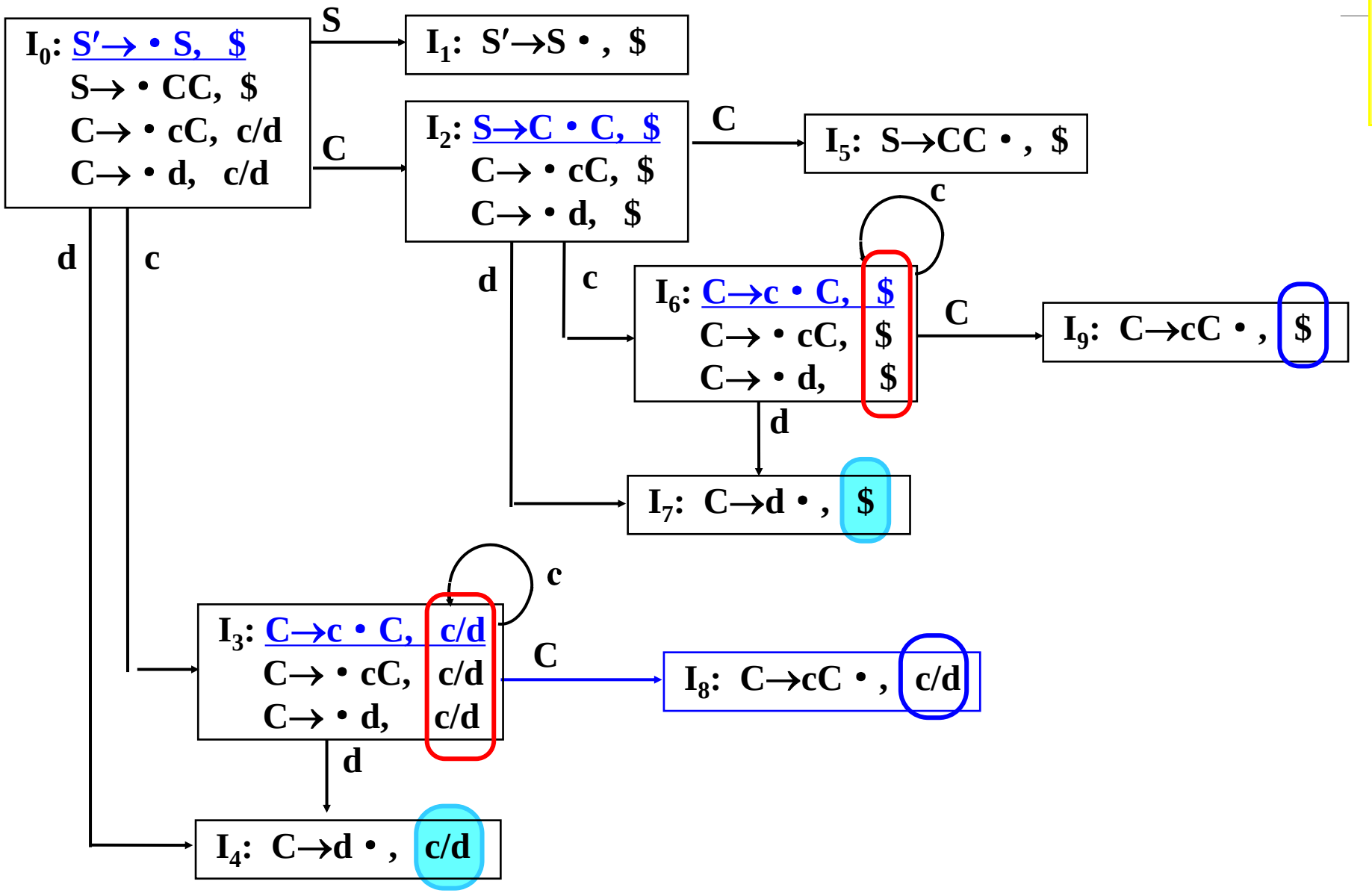
(0) S′→S
(1) S→CC　(2) C→cC　(3) C→d

I_0= closure({[S′→•S, $]})
= { [S′→•S, $]
　　[S→•CC, $]
　　[C→•cC, c/d]
　　[C→•d, c/d] }

wenshli@bupt.edu.cn

识别文法4.8所有活前缀的DFA

(0) S′→S
(1) S→CC
(2) C→cC
(3) C→d



算法4.9 构造LR(1)分析表

输入：拓广文法G′　　输出：文法G′的分析表

方法如下：

1. 构造文法G′的LR(1)项目集规范族C={$I_0, I_1, \ldots, I_n$}
2. 对于状态i（代表项目集I_i），分析动作如下：
   a) 若[A→α•aβ, b]∈I_i，且go(I_i, a)=I_j，则置 action[i, a]=Sj
   b) 若[A→α•, a]∈I_i，且A≠S′，则置 action[i, a]=Rj
   c) 若[S′→S•, $]∈$I_i$，则置 action[i, $]=ACC
3. 若对非终结符号A，有go(I_i, A)=I_j，则置 goto[i, A]=j
4. 凡是不能用上述规则填入信息的空白表项，均置上出错标志error。
5. 分析程序的初态是包括[S′→•S, $]的有效项目集所对应的状态。

wenshli@bupt.edu.cn

例：构造文法4.8的LR(1)分析表

(0) S′→S
(1) S→CC
(2) C→cC
(3) C→d

I_0: S′→ • S, $
S→ • CC, $
C→ • cC, c/d
C→ • d, c/d

I_1: S′→S • , $

I_2: S→C • C, $
C→ • cC, $
C→ • d, $

I_3: C→c • C, c/d
C→ • cC, c/d
C→ • d, c/d

I_4: C→d • , c/d

I_5: S→CC • , $

I_6: C→c • C, $
C→ • cC, $
C→ • d, $

I_7: C→d • , $

I_8: C→cC • , c/d

I_9: C→cC • , $

状态	action			goto	
	c	d	$	S	C
0	S3	S4		1	2
1			ACC		
2	S6	S7			5
3	S3	S4			8
4	R3	R3			
5			R1		
6	S6	S7			9
7			R3		
8	R2	R2			
9			R2		

wenshli@bupt.edu.cn

示例

- 试构造文法G[S]的LR(1)分析表。

  (1) S→L=R　(2) S→R

  (3) L→*R　(4) L→id

  (5) R→L　　(文法4.7)

- 拓广文法G′

  (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(1)项目集规范族及识别所有活前缀的DFA

I_0:

S′→ • S,　$

S→ • L=R,　$

S→ • R,　$

L→ • *R,　=/$

L→ • id,　=/$

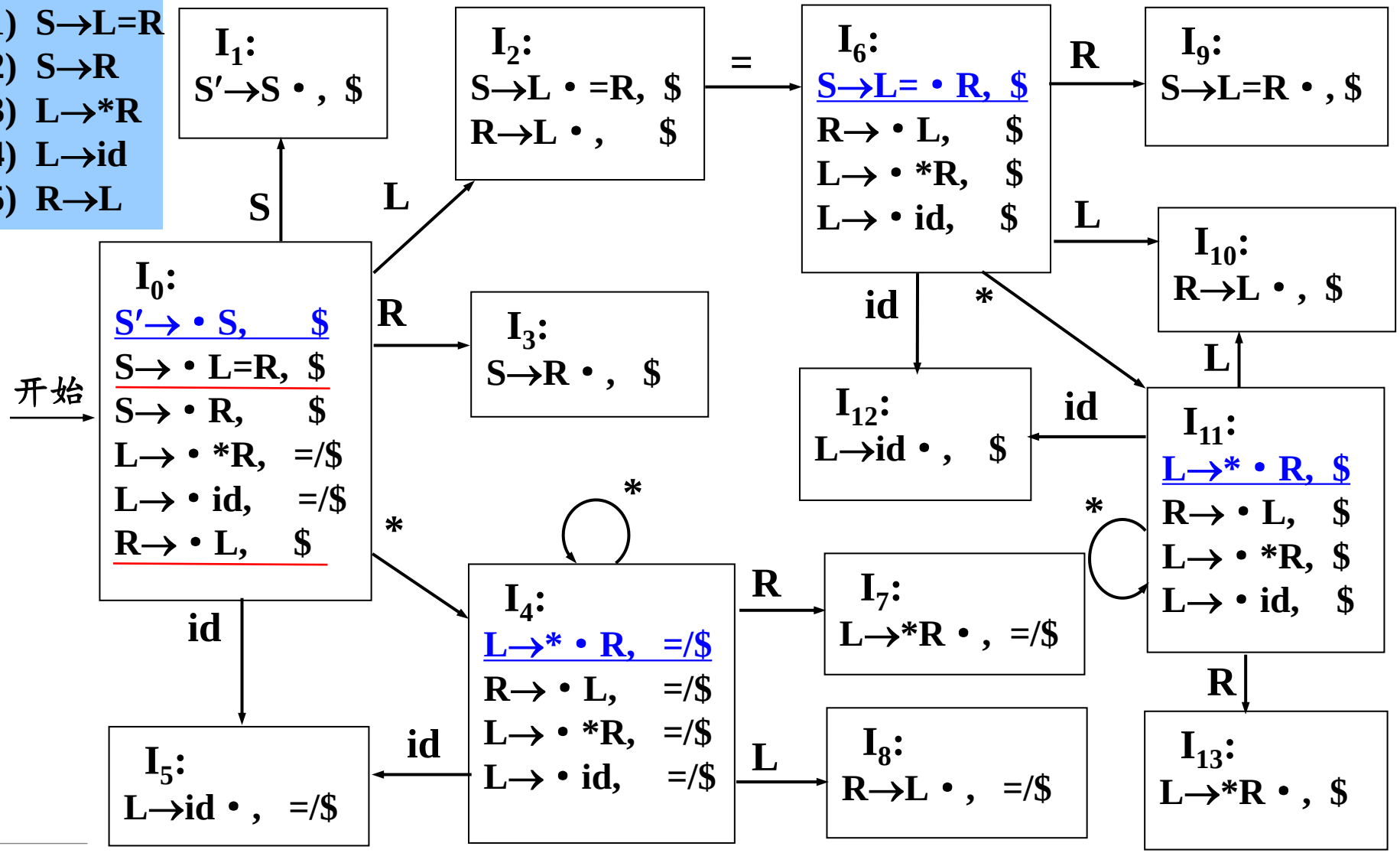
R→ • L,　$

wenshli@bupt.edu.cn

示例

练习：
分析 *id=id

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



* id = id $

0	1	6	9
-	S	=	R

Accept!

wenshli@bupt.edu.cn

文法4.7的LR(1)分析表

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

I_0:
S′→ • S, $
S→ • L=R, $
S→ • R, $
L→ • *R, =/$
L→ • id, =/$
R→ • L, $

I_2:
S→L • =R, $
R→L • , $

I_4:
L→* • R, =/$
R→ • L, =/$
L→ • *R, =/$
L→ • id, =/$

状态	action				goto		
	=	*	id	$	S	L	R
0		S4	S5		1	2	3
1				ACC			
2	S6			R5			
3				R2			
4		S4	S5			8	7
5	R4			R4			
6		S11	S12			10	9
7	R3			R3			
8	R5			R5			
9				R1			
10				R5			
11		S11	S12			10	13
12				R4			
13				R3			

I_1: S′→S • , $

I_3: S→R • , $

I_5: L→id • , =/$

I_6:
S→L= • R, $
R→ • L, $
L→ • *R, $
L→ • id, $

wenshli@bupt.edu.cn

课堂练习3

说明文法G[X]是LR(1)文法，但不是SLR(1)文法。

X→Ma | bMc | dc | bda

M→d

解答：

- 首先，拓广文法

(0) S→X

(1) X→Ma (2) X→ bMc (3) X→ dc (4) X→ bda

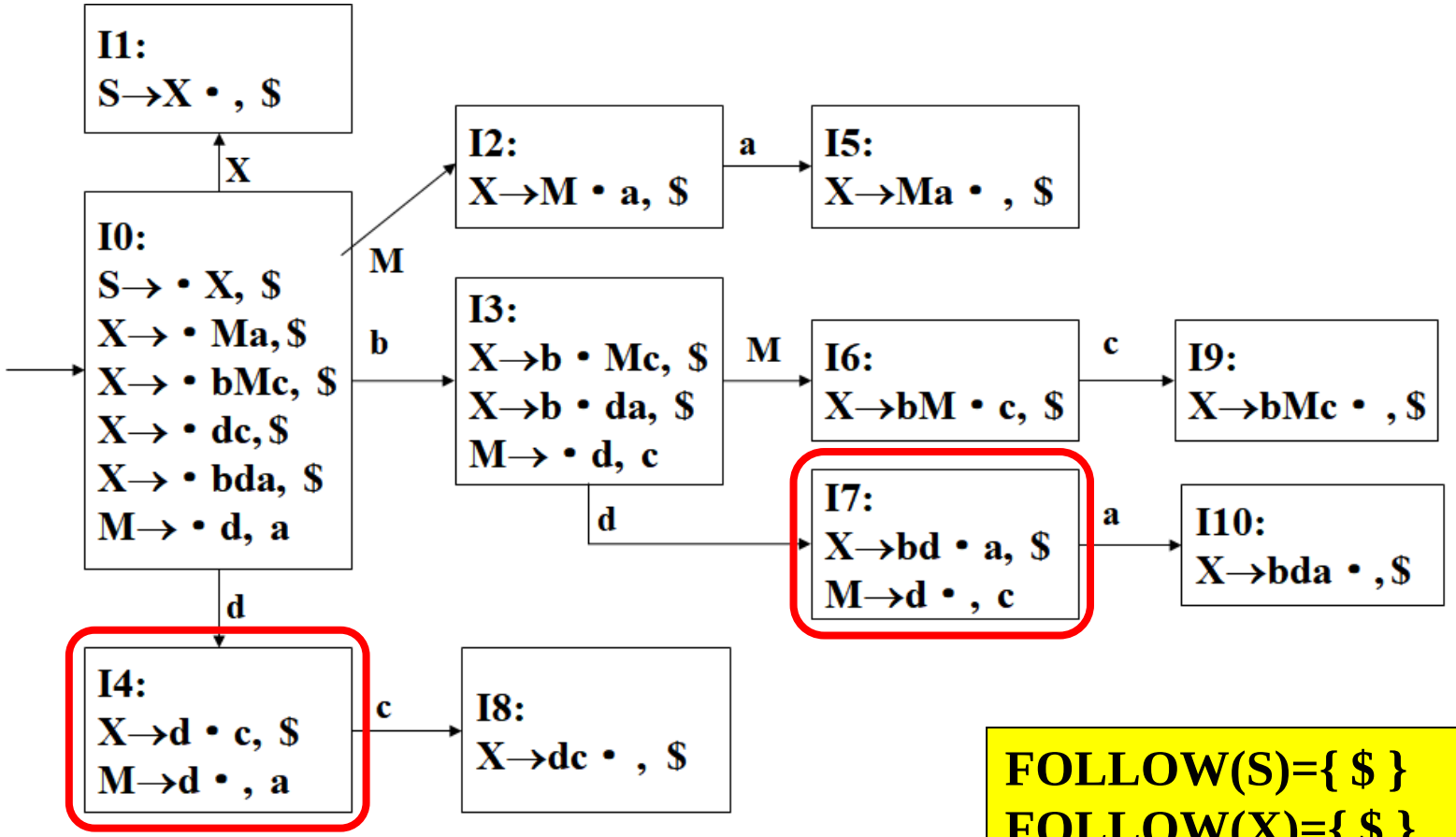
(5) M→d

wenshli@bupt.edu.cn

解答：

(0) S→X
(1) X→Ma
(2) X→ bMc
(3) X→ dc
(4) X→ bda
(5) M→d

其次，构造文法的LR(1)项目集规范族及识别其所有活前缀的DFA。



① 集合I0、I3中没有归约项目，所以，不存在冲突；
② 集合I1、I2、I5、I6、I8、I9、I10 各只有一个归约项目，所以这些集合中没有冲突；
③ 集合I4和I7中既有移进项目又有归约项目，但是归约符号和移进符号不同，所以也没有冲突。

结论：是LR(1)。

FOLLOW(S)={ $ }
FOLLOW(X)={ $ }
FOLLOW(M)={ a, c }

I_4、I_7：存在移进-归约冲突。
FOLLOW(M)={ a, c }
用SLR(1)方法无法解决，
所以该文法不是SLR(1)文法。

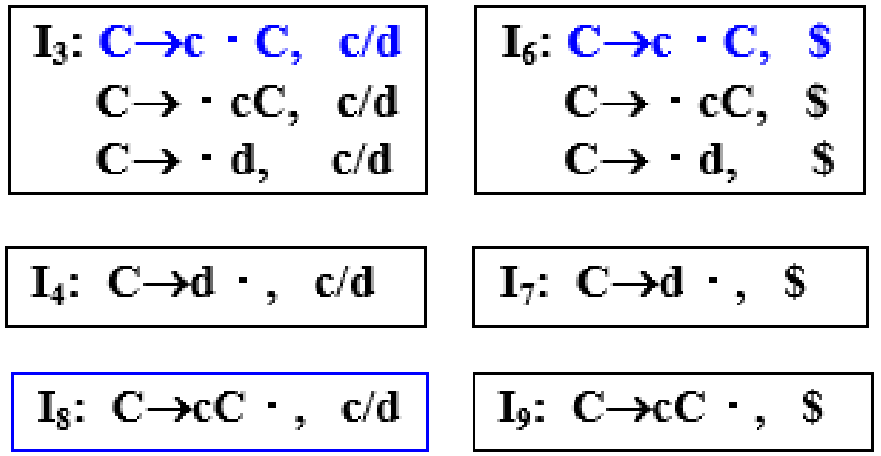
判断该文法是LR(1)文法：

然后，构造文法的LR(0)项目集规范族及识别其所有活前缀的DFA。

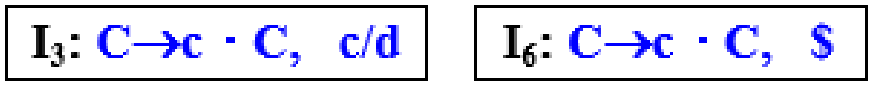
wenshli@bupt.edu.cn

4. LALR(1)分析表的构造

- 定义：同心集
  如果两个LR(1)项目集去掉搜索符号之后是相同的，则称这两个项目集具有相同的心（core），即这两个项目集是同心集。



- 定义：项目集的核
  除去初态项目集外，一个项目集的核（kernel）是由该项目集中那些圆点不在最左边的项目组成。
  LR(1)初态项目集的核中有且只有项目[S′→•S, $]。



wenshli@bupt.edu.cn

构造LALR(1)分析表的基本思想

- 基本思想：
  - 合并LR(1)项目集规范族中的同心集，以减少分析表的状态数。
  - 用核代替项目集，以减少项目集所需的存储空间
- go(I, X)仅仅依赖于I的心，因此LR(1)项目集合并后的转移函数可以通过go(I, X)自身的合并得到。
- 同心集的合并，可能导致归约-归约的冲突，但不会产生新的移进-归约冲突。
- 如果合并后的项目集中存在移进-归约冲突，则意味着：
  - 项目[A→α•, a]和[B→β•aγ, b] 处于合并后的同一项目集中。
  - 合并前必存在某个c，使得[A→α•, a]和[B→β•aγ, c]同处于某个项目集中。
  - 说明，原来的LR(1)项目集中已经存在移进-归约冲突。

wenshli@bupt.edu.cn

示例：同心集的合并可能导致归约-归约冲突

例：有文法G[S]：

S→aAd | bBd | aBe | bAe

A→c

B→c　　（文法4.9）

- 拓广文法G′

(0) S′→S

(1) S→aAd　(2) S→bBd　(3) S→aBe　(4) S→bAe

(5) A→c　(6) B→c

- 构造文法G′的LR(1)项目集规范族及识别其所有活前缀的DFA

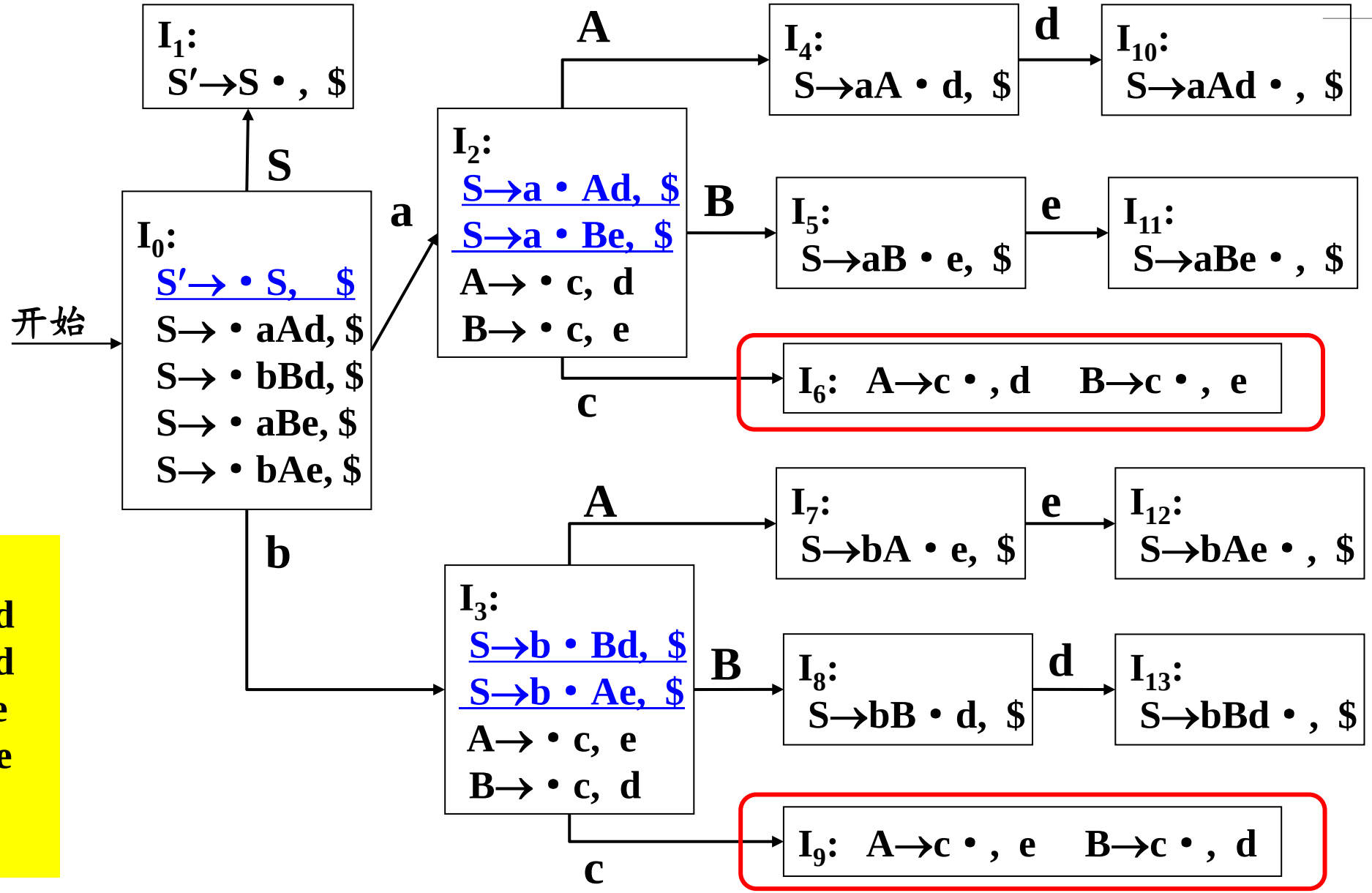
I_0={ [S′→•S, $]

[S→•aAd, $] [S→•bBd, $] [S→•aBe, $] [S→•bAe, $] }

wenshli@bupt.edu.cn

示例

I_{69}: A→c•, d/e　B→c•, d/e　归约-归约冲突！



0) S′→S
1) S→aAd
2) S→bBd
3) S→aBe
4) S→bAe
5) A→c
6) B→c

LALR(1)分析表的构造方法

- 首先构造LR(1)项目集规范族
- 检查LR(1)项目集规范族
  - 含有冲突，则文法不是LR(1)文法。
  - 不存在冲突，文法是LR(1)文法，检查是否存在同心集。
    - 不存在，则LR(1)项目集规范族即LALR(1)项目集规范族；
    - 存在，合并同心集；
    - 检查合并后的项目集是否存在冲突
      - ✓有冲突，则文法不是LALR文法。
      - ✓无冲突，则文法是LALR(1)文法。
        合并同心集得到的项目集规范族为LALR(1)项目集规范族。
- 根据LALR(1)项目集规范族构造分析表。

wenshli@bupt.edu.cn

算法4.10 构造LALR(1)分析表

输入：一个拓广文法G′　　输出：文法G′的LALR(1)分析表

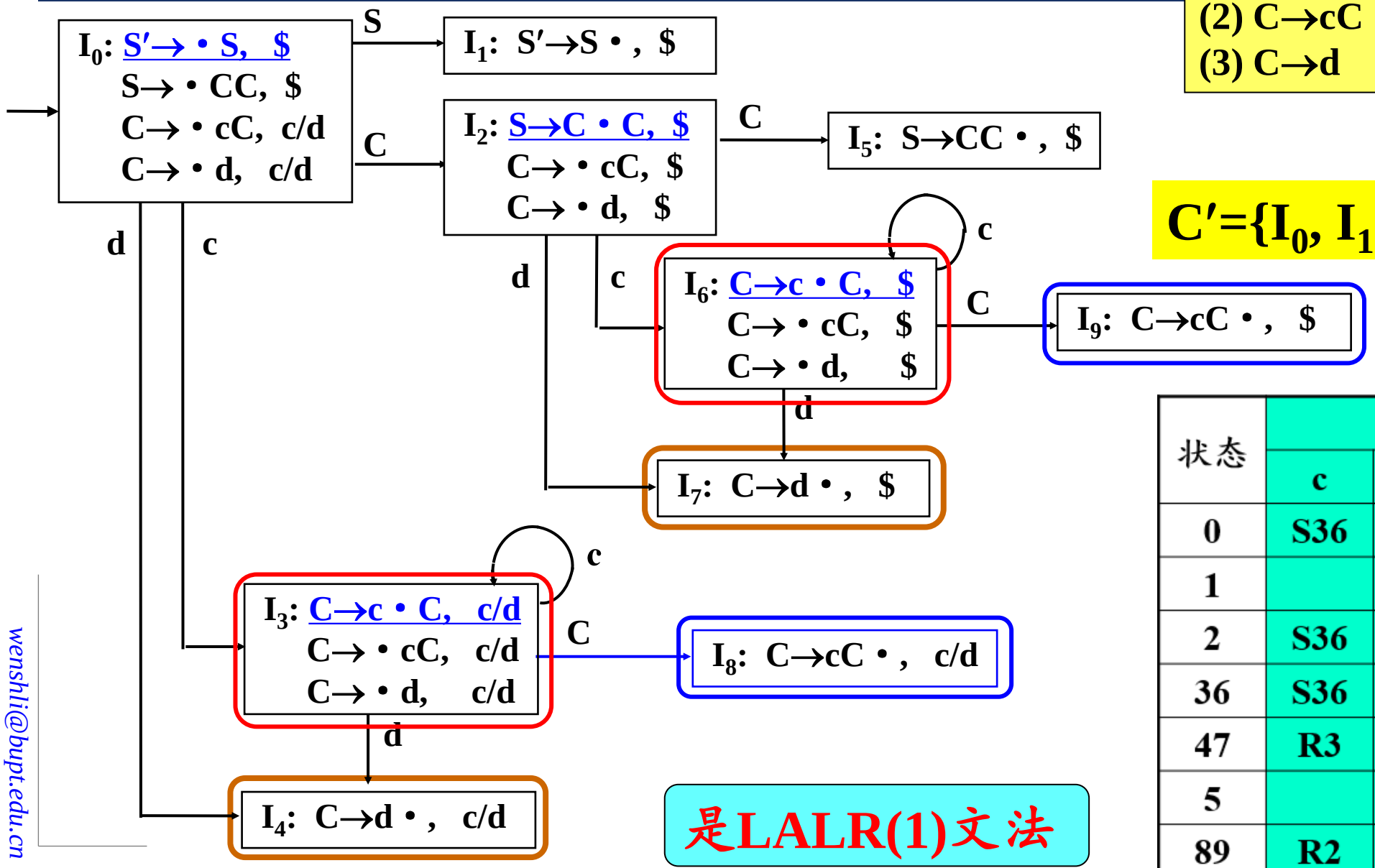
方法：

1. 构造文法G′的LR(1)项目集规范族 $C=\{I_0, I_1, \ldots, I_n\}$。
2. 合并C中的同心集，得到一个新的项目集规范族$C'=\{J_0, J_1, \ldots, J_m\}$，
   其中含有项目$[S'\rightarrow\bullet S, \$]$的$J_k$为分析表的初态。
3. 从C′出发，构造action子表
   (a) 若$[A\rightarrow\alpha\bullet a\beta, b]\in J_i$，且$go(J_i, a)=J_j$，则置 $action[i, a]=Sj$
   (b) 若$[A\rightarrow\alpha\bullet, a]\in J_i$，则置 $action[i, a]=R\ A\rightarrow\alpha$
   (c) 若$[S'\rightarrow S\bullet, \$]\in J_i$，则置 $action[i, \$]=ACC$
4. 构造 goto 子表
   设$J_k=\{I_{i1}, I_{i2}, \ldots, I_{it}\}$，由于这些$I_i$是同心集，因此，$go(I_{i1}, X)$、$go(I_{i2}, X)$、…、$go(I_{it}, X)$也是同心集。把所有这些项目集合并后得到的集合记作$J_i$，则有：$go(J_k, X)=J_i$
   于是，若 $go(J_k, A)=J_i$，则置 $goto[k, A]=i$
5. 分析表中凡不能用上述规则填入信息的空表项，均置上出错标志。

wenshli@bupt.edu.cn

示例: 构造文法4.8的LALR(1)分析表

(0) S′→S
(1) S→CC
(2) C→cC
(3) C→d



I36: C→c • C, $/c/d
C→ • cC, $/c/d
C→ • d, $/c/d

I47: C→d • , $/c/d

I89: C→cC • , $/c/d

C′={I0, I1, I2, I36, I47, I5, I89}

状态	action			goto	
	c	d	$	S	C
0	S36	S47		1	2
1			ACC		
2	S36	S47			5
36	S36	S47			89
47	R3	R3	R3		
5			R1		
89	R2	R2	R2		

是LALR(1)文法

wenshli@bupt.edu.cn

LALR(1)分析程序和LR(1)分析程序的比较

- **LR(1)分析程序：符号串 ccd**

步骤	栈	输入	分析动作
0	0	ccd$	shift
1	0c3	cd$	shift
2	0c3c3	d$	shift
3	0c3c3d4	$	error

- **LALR(1)分析程序：符号串 ccd**

步骤	栈	输入	分析动作
0	0	ccd$	shift
1	0c36	cd$	shift
2	0c36c36	d$	shift
3	0c36c36d47	$	reduce by C→d
4	0c36c36C89	$	reduce by C→cC
5	0c36C89	$	reduce by C→cC
6	0C2	$	error

- **cdcd**

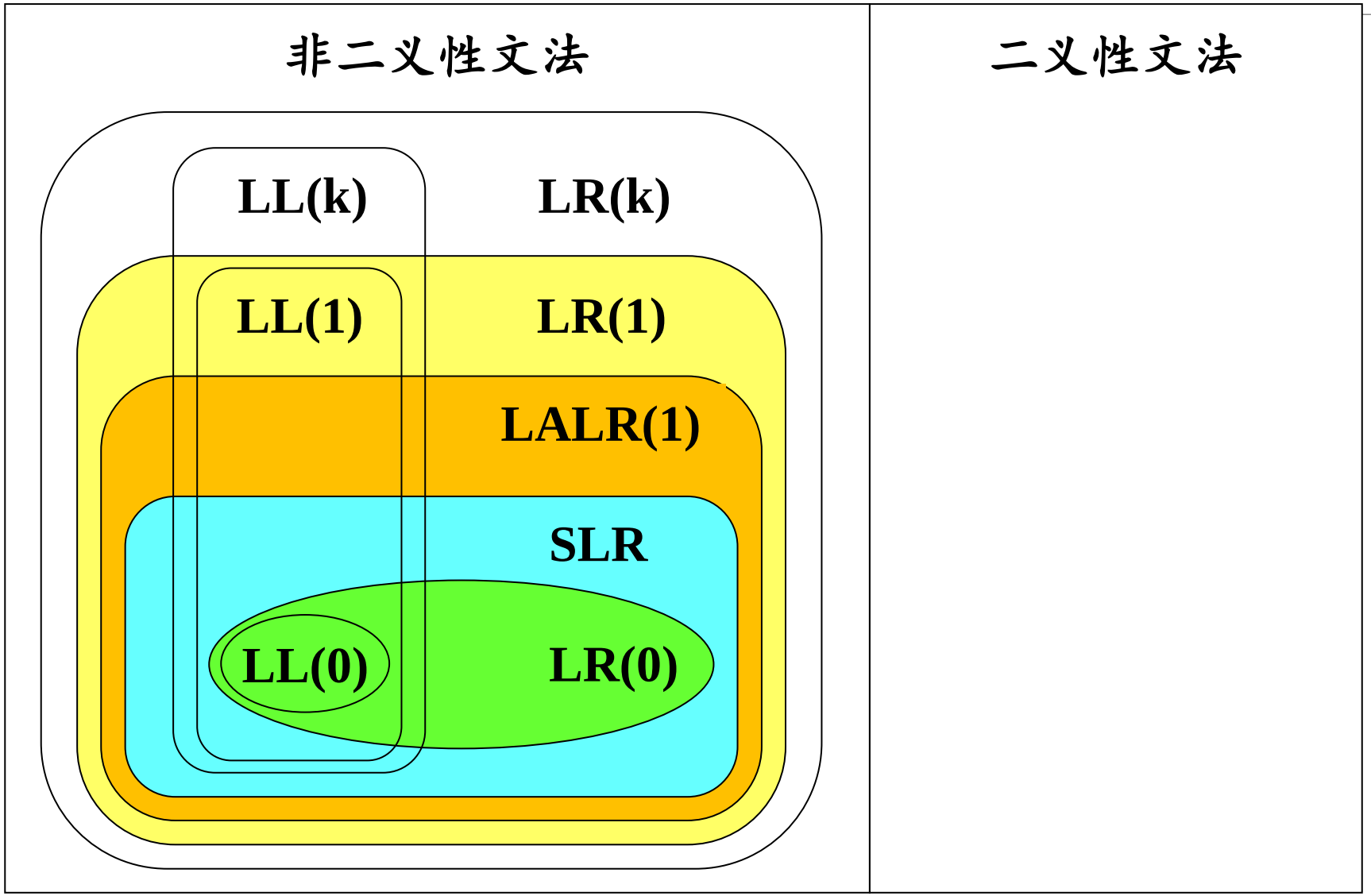
步骤	栈	输入	分析动作
0	0	cdcd$	shift
1	0c3	dcd$	shift
2	0c3d4	cd$	reduce by C→d
3	0c3C8	cd$	reduce by C→cC
4	0C2	cd$	shift
5	0C2c6	d$	shift
6	0C2c6d7	$	reduce by C→d
7	0C2c6C9	$	reduce by C→cC
8	0C2C5	$	reduce by S→CC
9	0S1	$	accept

- **cdcd**

步骤	栈	输入	分析动作
0	0	cdcd$	shift
1	0c36	dcd$	shift
2	0c36d47	cd$	reduce by C→d
3	0c36C89	cd$	reduce by C→cC
4	0C2	cd$	shift
5	0C2c36	d$	shift
6	0C2c36d47	$	reduce by C→d
7	0C2c36C89	$	reduce by C→cC
8	0C2C5	$	reduce by S→CC
9	0S1	$	accept

wenshli@bupt.edu.cn

文法分类



wenshli@bupt.edu.cn

5. LR分析的错误处理与恢复

- LR分析程序可采取以下恢复策略：
  - 首先，从栈顶开始退栈，可能弹出0个或若干个状态，直到出现状态S为止。

    (1) 状态 S 有相对于当前输入符号的转移，或者

    (2) goto 表中有 S 相对于某非终结符号A的后继。
  - 针对(1)：移进，分析继续。
  - 针对(2)：跳过0个或若干个输入符号，直到出现符号a为止，a∈FOLLOW(A)；

    然后，把状态goto[S, A]压入栈顶，继续分析。
- A的选择：通常选择表示主要结构成分的非终结符号。
- 跳过了包含错误的一段终结符号串。

wenshli@bupt.edu.cn

例：表达式文法4.11，“*优先于+，左结合”

期待：id或(
出现：+、*或$。
诊断：“缺少运算对象”
恢复策略：把一个假想的id压入栈，并将状态3推入栈顶。

期待：id、（、+、*
遇到：右括号
诊断：“括号不匹配”
恢复策略：删掉输入的右括号

状态	action						goto
	id	+	*	(	)	$	E
0	S3	e1	e1	S2	e2	e1	1
1	e3	S4	S5	e3	e2	ACC	
2	S3	e1	e1	S2	e2	e1	6
3	R4	R4	R4	R4	R4	R4	
4	S3	e1	e1	S2	e2	e1	7
5	S3	e1	e1	S2	e2	e1	8
6	e3	S4	S5	e3	S9	e4	
7	R1	R1	S5	R1	R1	R1	
8	R2	R2	R2	R2	R2	R2	
9	R3	R3	R3	R3	R3	R3	

期待：+、*、或）
出现：id或（
诊断：“缺少运算符号”
恢复策略：把运算符号‘+’压入栈，转移到状态4。

期待：+、*、或）
遇到：$
诊断：“缺少右括号”
恢复策略：把右括号压入栈，转移到状态9。

wenshli@bupt.edu.cn

示例：分析符号串 id+)

状态	action						goto
	id	+	*	(	)	$	E
0	S3	e1	e1	S2	e2	e1	1
1	e3	S4	S5	e3	e2	ACC	
2	S3	e1	e1	S2	e2	e1	6
3	R4	R4	R4	R4	R4	R4	
4	S3	e1	e1	S2	e2	e1	7
5	S3	e1	e1	S2	e2	e1	8
6	e3	S4	S5	e3	S9	e4	
7	R1	R1	S5	R1	R1	R1	
8	R2	R2	R2	R2	R2	R2	
9	R3	R3	R3	R3	R3	R3	

步骤	栈	输入	分析动作
(1)	0 –	id+)$	shift
(2)	0 3 – id	+)$	reduce by E→id
(3)	0 1 – E	+)$	shift
(4)	0 1 4 – E +	)$	CALL e2“括号不匹配”，删掉‘）’
(5)	0 1 4 – E +	$	CALL e1“缺少运算对象”，id压入栈
(6)	0 1 4 3 – E + id	$	reduce by E→id
(7)	0 1 4 7 – E + E	$ $	reduce by E→E+E reduce by E→E+E
(8)	0 1 – E	$	accept

wenshli@bupt.edu.cn

本章小结

一、自顶向下的分析方法

- 递归下降分析方法
  - 试探性、回溯
  - 要求：文法不含左递归
- 递归调用预测分析方法
  - 不带回溯的递归分析方法
  - 要求：
    - 文法不含左递归，并且
    - 对任何产生式：$A\rightarrow\alpha_1|\alpha_2|\ldots|\alpha_n$
      $FIRST(\alpha_i)\cap FIRST(\alpha_j)=\phi$
  - 构造步骤：
    - 描述结构的上下文无关文法
    - 根据文法构造预测分析程序的状态转换图
    - 状态转换图化简
    - 根据状态转换图构造递归过程
- 非递归预测分析方法
  - 不带回溯、不含递归
  - 模型：

    输入缓冲区：存放输入符号串$a_1a_2\ldots a_n\$$

    符号栈：分析过程中存放文法符号

    分析表：二维表，每个A有一行，每个a包括$有一列

    表项内容是产生式（关键）

    控制程序：根据栈顶X和当前输入a决定分析动作

    （永恒的核心）

    X=a=$ 分析成功

    X=a≠$ 弹出X，扫描指针前移

    X是非终结符号，查分析表：M[X, a]

    $M[X, a]=X\rightarrow Y_1Y_2\ldots Y_K$，弹出X，$Y_K$、...、$Y_2$、$Y_1$入栈

    M[X, a]= $X\rightarrow\varepsilon$，弹出X

    M[X, a]=空白，出错处理

    输出：对输入符号串进行最左推导所用的产生式序列

wenshli@bupt.edu.cn

本章小结

- 预测分析表的构造
  - 构造每个文法符号的FIRST集合
  - 构造每个非终结符号的FOLLOW集合
  - 检查每个产生式A→α
    - 对任何a∈FIRST(α)，M[A, a]= A→α
    - 若α⇒ε，对所有b∈FOLLOW(A)，M[A, b]= A→α
- LL(1)文法
  - LL(1)的含义
  - 判断一个文法是否为LL(1) 文法
    - 构造分析表，或者
    - 检查每个产生式：A→α|β

      FIRST(α)∩FIRST(β)=ϕ

      若β ⇒ε，则FIRST(α)∩FOLLOW(A)=ϕ

二、自底向上分析方法

- 移进-归约分析方法
  - 分析栈、输入缓冲区
  - 可归约串
  - 规范归约：最右推导的逆过程
- LR分析方法
  - 模型：

    输入缓冲器：　　输出：分析动作序列

    分析栈：$S_0X_1S_1\ldots XnSn$

    分析表：包括 action 和 goto 两部分（关键）

    控制程序：根据栈顶状态 S_n 和当前输入符号 a 查分析表

  action[S_n, a]，决定分析动作（永恒的核心）

  action[S_n, a]=S i，a入符号栈，i入状态栈

  i=goto(S_n, a)

  action[S_n, a]=R A→β，弹出 |β|个符号，

  A入符号栈，goto(S_{n-r}, A)入状态栈

  action[S_n, a]=ACC，分析成功

  action[S_n, a]=空白，出错处理

wenshli@bupt.edu.cn

本章小结

- SLR(1)分析表的构造
  - LR(0)项目集规范族
  - 识别文法所有活前缀的DFA
  - 构造分析表：检查每个状态集
    - 若$A\rightarrow\alpha\cdot a\beta \in I_i$，且$go(I_i, a)=I_j$，则置action[i, a]=Sj，
    - 若$A\rightarrow\alpha\cdot \in I_i$，则对所有$a\in FOLLOW(A)$，置action[i, a]=R $A\rightarrow\alpha$
    - 若$S'\rightarrow S\cdot \in I_i$，则置action[i, $]=ACC，分析成功。
    - 若$go(I_i, A)=I_j$，A为非终结符号，则置goto[i, A]=j
- LR(1)分析表的构造
  - LR(1)项目集规范族
  - 构造分析表：检查每个状态集
    - 若$[A\rightarrow\alpha\cdot a\beta$ ， $b]\in I_i$，且$go(I_i, a)=I_j$，则置action[i, a]=Sj，
    - 若$[A\rightarrow\alpha\cdot$， $a] \in I_i$，则置action[i, a]=R $A\rightarrow\alpha$，
    - 若$[S'\rightarrow S\cdot$， $] \in I_i$，则置action[i, $]=ACC，分析成功。
    - 若$go(I_i, A)=I_j$，A为非终结符号，则置goto[i, A]=j
- LALR(1)分析表的构造
  - LR(1)项目集规范族，若没有冲突，继续
  - 合并同心集，若没有冲突，则为LALR(1)项目集规范族
  - 构造分析表，方法同LR(1)分析表的构造方法

wenshli@bupt.edu.cn

算法清单

算法4.1 非递归预测分析方法

算法4.2 预测分析表的构造方法

算法4.3 LR分析程序

算法4.4 closure(I)的构造过程（I为LR(0)项目集）

算法4.5 构造文法G的LR(0)项目集规范族

算法4.6 构造文法G的SLR(1)分析表

算法4.7 closure(I)的构造过程（I为LR(1)项目集）

算法4.8 构造文法G的LR(1)项目集规范族

算法4.9 构造文法G的LR(1)分析表

算法4.10 构造LALR(1)分析表

给定文法，构造 候选式/非终结符号 的 FIRST 集合

构造 非终结符号 的 FOLLOW 集合

wenshli@bupt.edu.cn

学习任务

- 作业要求：
  - 改造文法，使之满足预测分析方法的要求；
  - 为给定文法构造递归调用分析程序；
  - 为给定文法构造LL(1)分析表，分析输入符号串；
  - 为给定文法构造SLR(1)、LR(1)、以及LALR(1)分析表，分析输入符号串。
- 研究性学习
  - 总结自顶向下分析方法和自底向上分析方法的区别；
  - 如何判断对于给定文法的句子，可以采用哪种方法进行分析？
  - LR分析方法对二义性文法的应用。

wenshli@bupt.edu.cn

程序设计2

YACC?
==>

题目：语法分析程序的设计与实现

实验内容：编写语法分析程序，实现对算术表达式的语法分析。要求所分析算数表达式由如下的文法产生。

E→ E+T | E–T | T

T→ T*F | T/F | F

F→ (E) | num

实验要求：在对输入的算术表达式进行分析的过程中，依次输出所采用的产生式。

方法1：编写递归调用程序实现自顶向下的分析。

方法2：编写LL(1)语法分析程序，要求如下。（必做）

(1) 编程实现算法4.2，为给定文法自动构造预测分析表。

(2) 编程实现算法4.1，构造LL(1)预测分析程序 。

方法3：编写语法分析程序实现自底向上的分析，要求如下。（必做）

(1) 构造识别该文法所有活前缀的DFA。

(2) 构造该文法的LR分析表。

(3) 编程实现算法4.3，构造LR分析程序。

方法4：利用YACC自动生成语法分析程序，调用LEX自动生成的词法分析程序。

wenshli@bupt.edu.cn

实验报告要求

- 实验题目、要求
- 程序设计说明
- 源程序
- 可执行程序
- 测试报告：包括输入、运行结果、及结果分析

- 提交：
  - 所有资料打包，命名规则：　班级-学号-姓名
- 在线提交。

wenshli@bupt.edu.cn

课堂练习4

已知文法G[A]为：

A→aABe|a

B→Bb|d

(1) 试给出与G[A]等价的LL(1)文法G'[A]

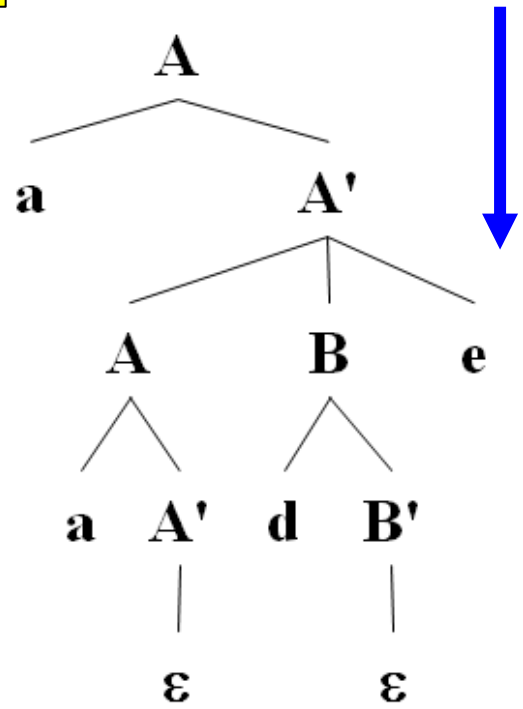
(2) 构造G'[A]的预测分析表

(3) 给出输入串aade的分析过程。

A→aA'
A'→ABe | ε
B→dB'
B'→bB' | ε

	First	Follow
A	a	$, d
A'	a, ε	$, d
B	d	e
B'	b, ε	e

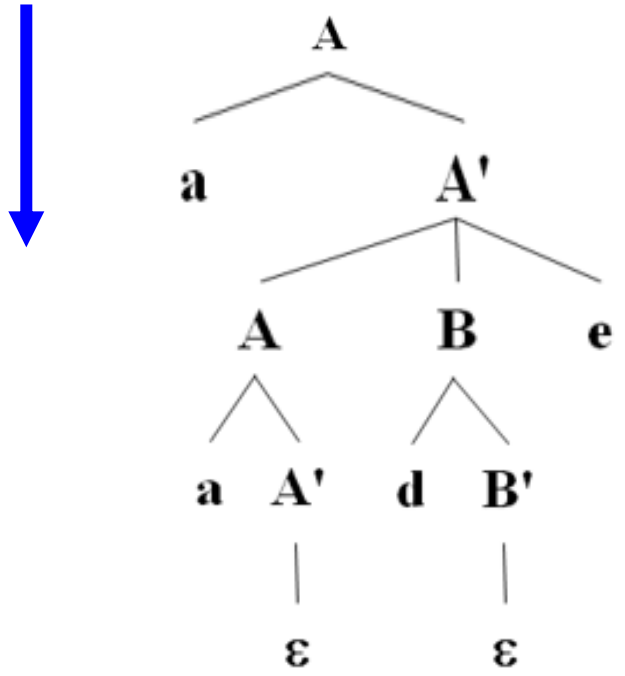
	a	b	d	e	$
A	A→aA'				
A'	A'→ABe		A'→ε		A'→ε
B			B→dB'		
B'		B'→bB'		B'→ε	



wenshli@bupt.edu.cn

aade的分析过程

	a	b	d	e	$
A	A→aA'				
A'	A'→ABe		A'→ε		A'→ε
B			B→dB'		
B'		B'→bB'		B'→ε	



步骤	栈	输入	分析动作
(1)	$A	aade$	A→aA'
(2)	$A'a	aade$	
(3)	$A'	ade$	A'→ABe
(4)	$eBA	ade$	A→aA'
(5)	$eBA'a	ade$	
(6)	$eBA'	de$	A'→ε
(7)	$eB	de$	B→dB'
(8)	$eB'd	de$	
(9)	$eB'	e$	B'→ε
(10)	$e	e$	
(11)	$	$	分析成功

wenshli@bupt.edu.cn

课堂练习5

有如下文法G[A]:

A→BA | a

B→aB | b

(1) 判断该文法是以下哪些类型的文法，要求给出判断过程。

LL(1)、LR(0)、SLR(1)

(2) 构造该文法的LR(1)项目集规范族及识别其所有活前缀的DFA。

(3) 构造该文法的LR(1)分析表

(4) 给出对输入符号串abb的分析过程。

不是LL(1)文法，理由？

（0） S →A
（1）A→BA
（2）A→a
（3）B→aB
（4）B→b

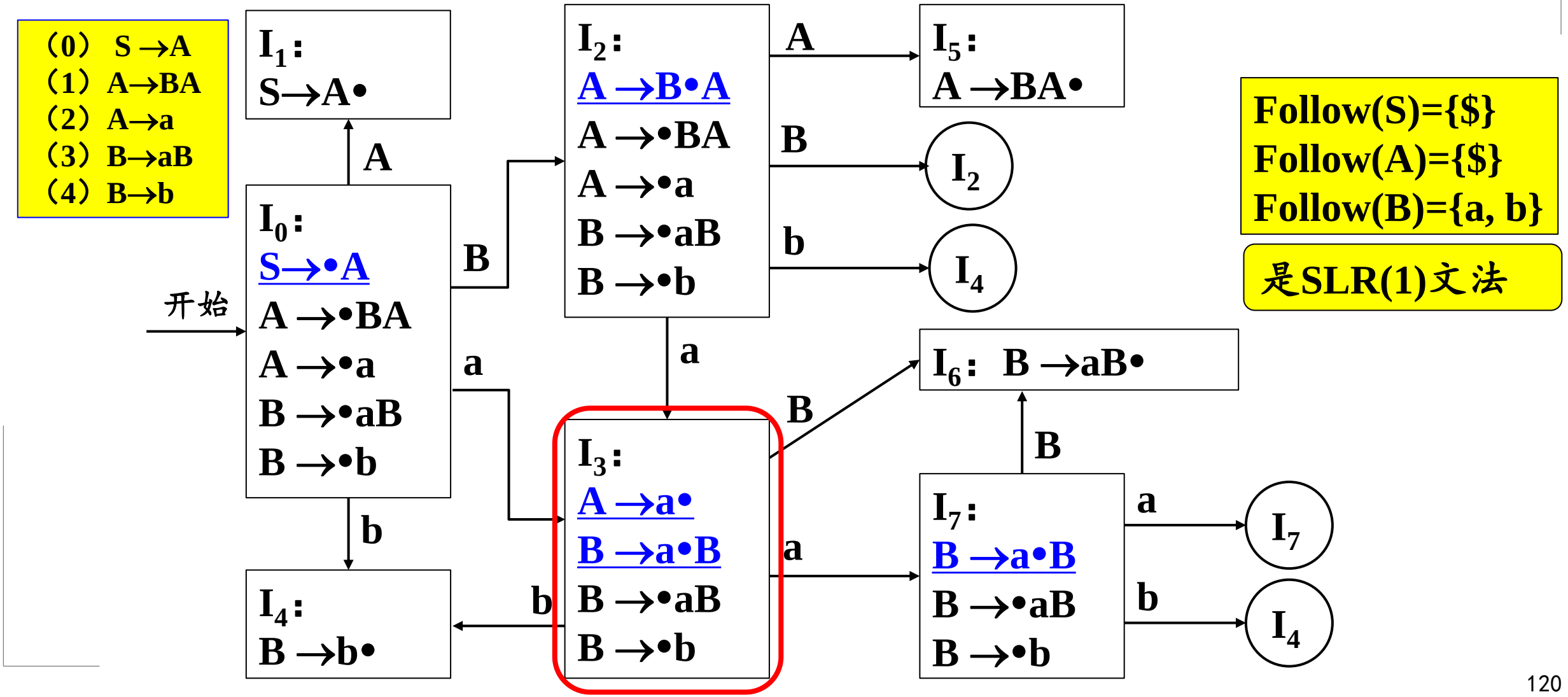
wenshli@bupt.edu.cn

参考答案

LR(0)文法，项目集中：
(1) 要么所有元素都是移进-待约项目
(2) 要么只含有唯一的归约项目

不是LR(0)文法

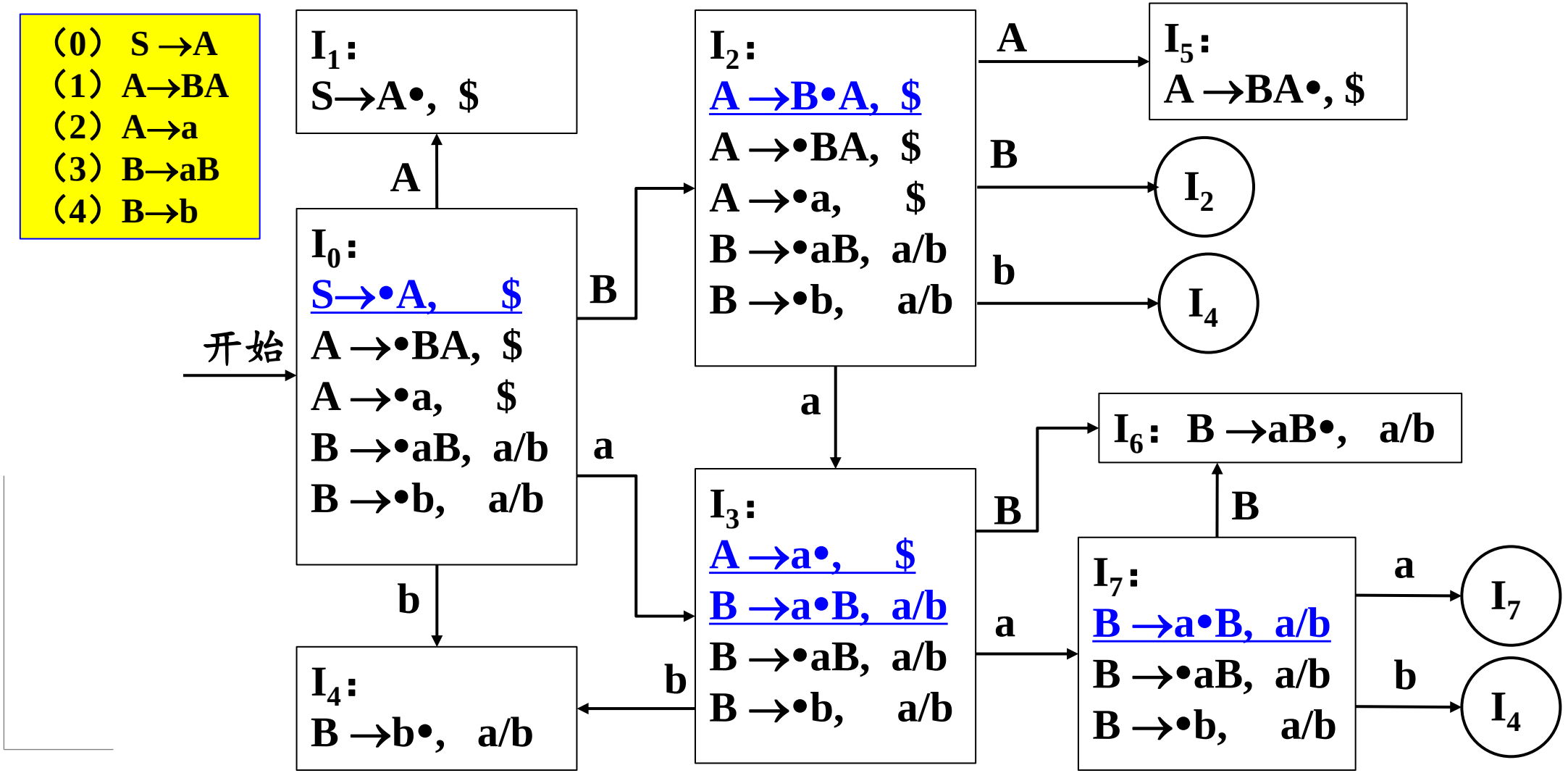
LR(0)项目集规范族及识别其所有活前缀的DFA：



wenshli@bupt.edu.cn

参考答案

LR(1)项目集规范族及识别其所有活前缀的DFA：



文法的LR(1)分析表

状态	action			goto	
	a	b	$	A	B
0	S3	S4		1	2
1			ACC		
2	S3	S4		5	2
3	S7	S4			6
4	R4	R4			
5			R1		
6	R3	R3			
7	S7	S4			6

abb的分析过程

步骤	栈	输入	分析动作
(1)	0	abb$	S3
(2)	0 3 - a	bb$	S4
(3)	0 3 4 - a b	b$	R4 B→b
(4)	0 3 6 - a B	b$	R3 B→aB
(5)	0 2 - B	b$	S4
(6)	0 2 4 - B b	$	error 弹出栈顶状态4
(7)	0 2 - B	$	goto(2, A)=5 将状态5压入栈顶
(8)	0 2 5 - B A	$	R1 A→BA
(9)	0 1 - A	$	accept

wenshli@bupt.edu.cn

* 推导和短语

例：考虑简单算术表达式的文法G:

G=({+，*，（，），i}，{E，T，F}，E，φ)

φ：E→E + T | T

T→T * F | F

F→（E）| i

- 文法所产生的语言
  - 从文法的开始符号出发，反复连续使用产生式对非终结符号进行替换和展开，得到的终结符号串是该文法定义的句子。
  - 文法所有句子构成的集合就是相应文法的语言。

wenshli@bupt.edu.cn

* 推导

假定$A \to \gamma$是一个产生式，α和β是任意的文法符号串，则有：$\alpha A \beta \Rightarrow \alpha\gamma\beta$

- “$\Rightarrow$”表示 “一步推导”

  利用产生式对左边符号串中的一个非终结符号进行替换，得到右边的符号串。

- 称 $\alpha A \beta$ 直接推导出 $\alpha\gamma\beta$
- 也可以说 $\alpha\gamma\beta$ 是 $\alpha A \beta$ 的直接推导
- 或说 $\alpha\gamma\beta$ 直接归约 到 $\alpha A \beta$

如果有直接推导序列：$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$

则说α_1推导出α_n，记作：$\alpha_1 \overset{*}{\Rightarrow} \alpha_n$

从α_1到α_n的长度为n的推导

0步或多步推导

wenshli@bupt.edu.cn

* 推导

从文法开始符号E推导出符号串i+i的详细过程

αAβ	αγβ	α	β	所用产生式	从E到 αγβ 的推导步数
E	E+T	ε	ε	E→E+T	1
E+T	T+T	ε	+T	E→T	2
T+T	F+T	ε	+T	T→F	3
F+T	i+T	ε	+T	F→i	4
i+T	i+F	i+	ε	T→F	5
i+F	i+i	i+	ε	F→i	6

$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow i+T \Rightarrow i+F \Rightarrow i+i$$

wenshli@bupt.edu.cn

* 推导

最左推导

如果 $\alpha \overset{*}{\Rightarrow} \beta$，并且在每“一步推导”中，都替换α中最左边的非终结符号，则称这样的推导为最左推导。记作：$\alpha \overset{*}{\underset{lm}{\Rightarrow}} \beta$

$$E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow i+T \Rightarrow i+F \Rightarrow i+i$$

最右推导

如果 $\alpha \overset{*}{\Rightarrow} \beta$，并且在每“一步推导”中，都替换α中最右边的非终结符号，则称这样的推导为最右推导。记作：$\alpha \overset{*}{\underset{rm}{\Rightarrow}} \beta$

最右推导也称为规范推导。

$$E \Rightarrow E+T \Rightarrow E+F \Rightarrow E+i \Rightarrow T+i \Rightarrow F+i \Rightarrow i+i$$

wenshli@bupt.edu.cn

* 句型、句子和语言

■ 句型

对于文法$G=(V_T, V_N, S, \varphi)$，如果$S \overset{*}{\Rightarrow} \alpha$，则称 α 是当前文法的一个句型。

若$S \overset{*}{\underset{lm}{\Rightarrow}} \alpha$，则$\alpha$是当前文法的一个左句型，

若$S \overset{*}{\underset{rm}{\Rightarrow}} \alpha$，则$\alpha$是当前文法的一个右句型。

■ 句子

仅含有终结符号的句型是文法的一个句子。

■ 语言

文法G产生的所有句子组成的集合是文法G所定义的语言，记作L(G)。

$$L(G)=\{\alpha \mid S \overset{+}{\Rightarrow} \alpha，并且 \alpha \in V_T^*\}$$

wenshli@bupt.edu.cn

* 短语

对于文法$G=(V_T, V_N, S, \varphi)$，假定αβδ是文法G的一个句型，如果存在：

$$S \overset{*}{\Rightarrow} \alpha A \delta, \text{ 并且 } A \overset{+}{\Rightarrow} \beta$$

则称β是句型αβδ关于非终结符号A的短语。

如果存在：

$$S \overset{*}{\Rightarrow} \alpha A \delta, \text{ 并且 } A \Rightarrow \beta$$

则称β是句型αβδ关于非终结符号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。

例：

$$\underline{E} \Rightarrow \underline{T} \Rightarrow \underline{T*F} \Rightarrow \underline{T*(E)} \Rightarrow \underline{F*(E)} \Rightarrow \underline{i*(E)} \Rightarrow \underline{i*(E+T)} \Rightarrow \underline{i*(T+T)} \Rightarrow \underline{i*(F+T)} \Rightarrow \underline{i*(i+T)}$$

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩

wenshli@bupt.edu.cn

Ret

* 改写文法

- 文法二义性的消除
- 左递归的消除
- 提取左公因子

wenshli@bupt.edu.cn

* 二义性文法

- 如果一个文法的某个句子有不止一棵分析树，则这个句子是二义性的。
- 含有二义性句子的文法是二义性的文法。
- 例：文法 G=({+,*,（,）,i}, {E}, E, φ)

  φ： E→ E+E | E*E | (E) | id
- 句子 id+id*id 存在
  - 两个不同的最左推导：

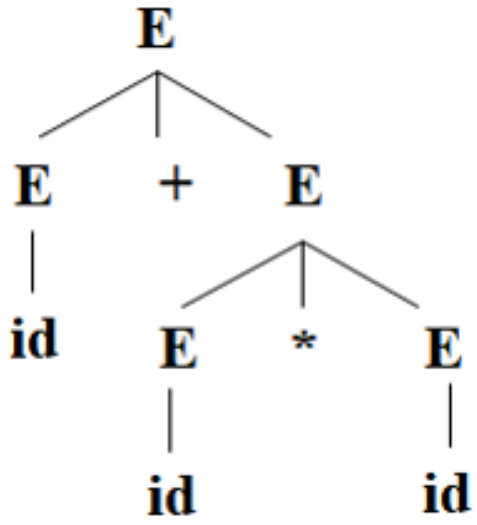
E ⇒ E+E ⇒ id+E ⇒ id+E*E

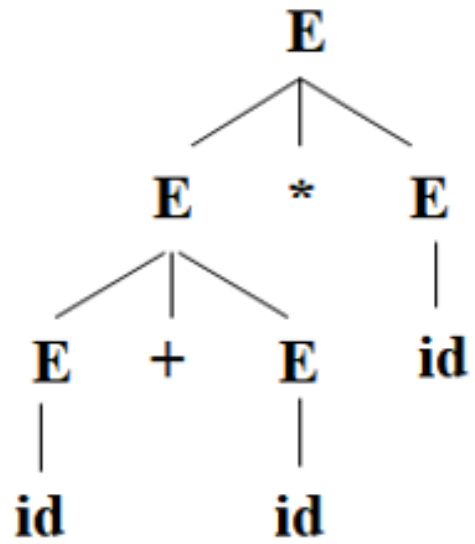
⇒ id+id*E ⇒id+id*id

E ⇒ E*E ⇒ E+E*E ⇒ id+E*E

⇒ id+id*E ⇒id+id*id

  - 两棵不同的分析树：





wenshli@bupt.edu.cn

* 文法的二义性和语言的二义性

- 两个文法是等价的，如果它们产生的语言相同，即L(G)=L(G′)。
- 二义性的语言：语言不存在无二义性的文法。
- 二义性问题是不可判定的
  - 不存在一种算法，能够在有限的步骤内确切地判定出一个文法是否是二义性的。
  - 可以找出一些充分条件（未必是必要条件），当文法满足这些条件时，就可以确信该文法是无二义性的。

wenshli@bupt.edu.cn

* 文法二义性的消除

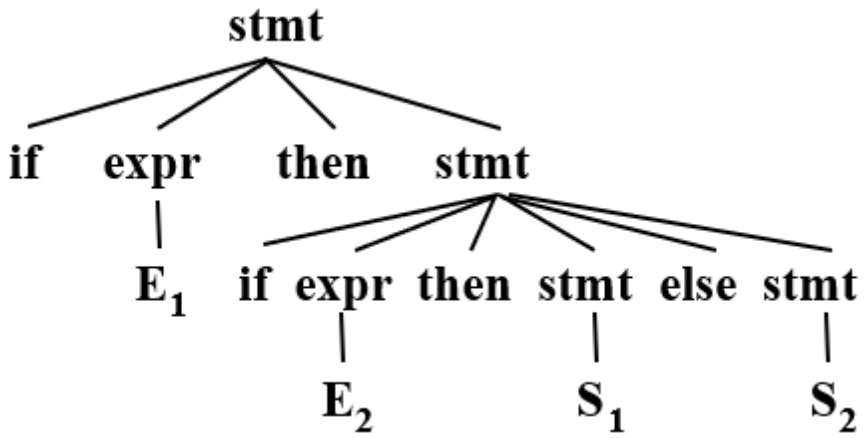
- 映射程序设计语言中IF语句的文法：

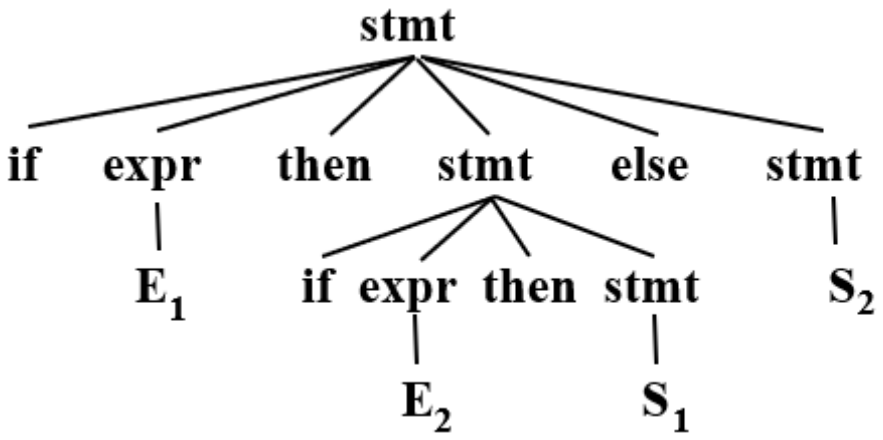
stmt → if ***expr*** then ***stmt***

| if ***expr*** then ***stmt*** else ***stmt***

| other

- 句子if E_1 then if E_2 then S_1 else S_2有两棵不同的分析树：





最近最后匹配原则

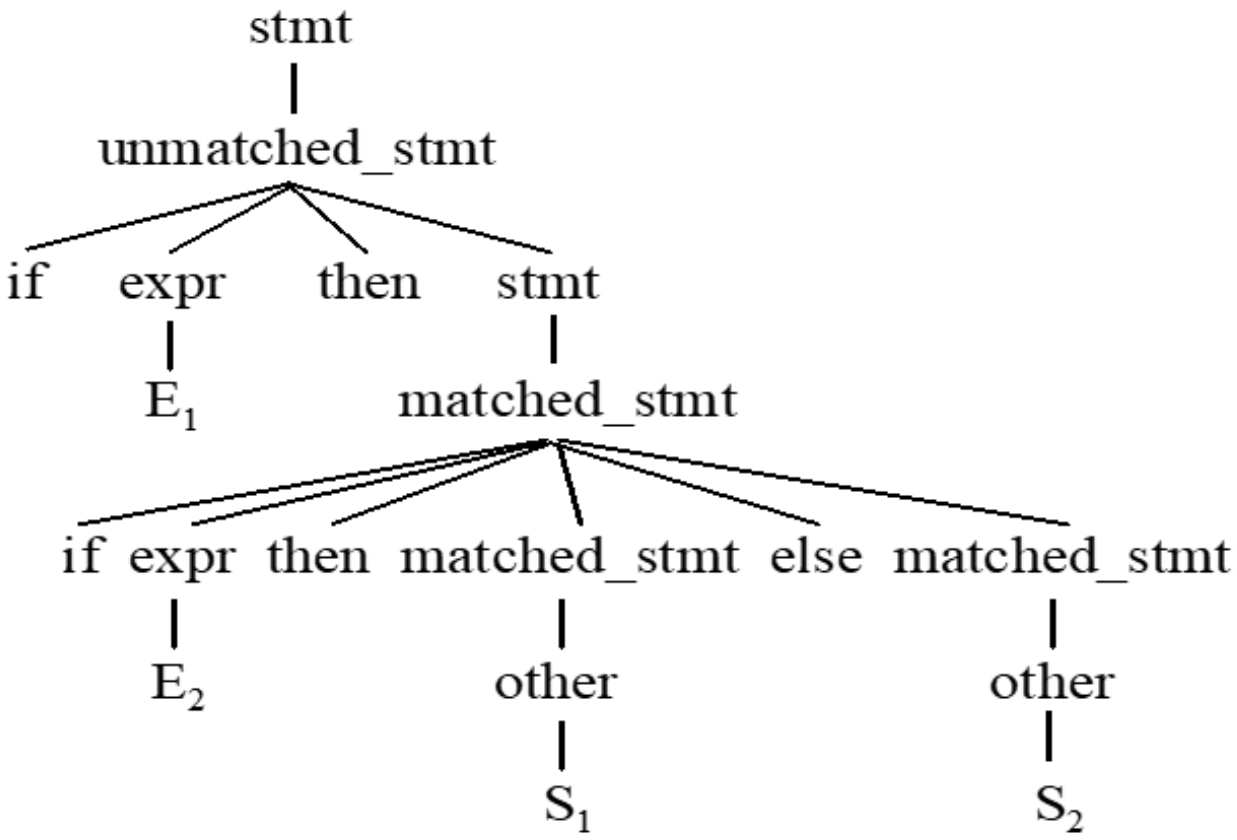
wenshli@bupt.edu.cn

* 最近最后匹配原则

- 出现在then和else之间的语句必须是“匹配的”。
- 改写后的文法：

stmt→ *matched_stmt*
　　| *unmatched_stmt*
matched_stmt→
　　if *expr*
　　then *matched_stmt*
　　else *matched_stmt*
　　| **other**
unmatched_stmt→
　　if *expr* **then** *stmt*
　　| **if** *expr* **then** *matched_stmt*
　　else *unmatched_stmt*

- 句子if E_1 then if E_2 then S_1 else S_2的分析树



wenshli@bupt.edu.cn

* 左递归的消除

- 一个文法是左递归的，如有非终结符号A，对某个文法符号串α，存在推导：$A \overset{+}{\Rightarrow} A\alpha$
  - 若存在某个α=ε，则称该文法是有环路的。
- 消除左递归的方法：
  - 简单情况：如果文法G有产生式：$A \to A\alpha \mid \beta$

    可以把A的这两个产生式改写为：$A \to \beta A'$

    $A' \to \alpha A' \mid \varepsilon$
  - 一般情况：假定关于A的全部产生式是：

    $A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

    产生式可以改写为：$A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$

    $A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$

例：消除文法中的左递归

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to (E) \mid id$

文法改写为：

$E \to TE'$

$E' \to +TE' \mid \varepsilon$

$T \to FT'$

$T' \to *FT' \mid \varepsilon$

$F \to (E) \mid id$

间接左递归文法：

$S \to Aa \mid b$

$A \to Ac \mid Sd \mid \varepsilon$

wenshli@bupt.edu.cn

* 算法：消除左递归

间接左递归文法：
$S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd \mid \varepsilon$

输入：无环路、无ε-产生式的文法G

输出：不带有左递归的、与G等价的文法 G'

方法：

(1) 把文法G的所有非终结符号按某种顺序排列成$A_1, A_2, \ldots, A_n$

(2) for i:=1 to n do

for j:=1 to i-1 do

begin

把每个形如 $A_i \rightarrow A_j\gamma$ 的产生式改写为：$A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$；

其中 $A_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ 是关于当前A_j的所有产生式

消除关于A_i的产生式中的直接左递归

end

(3) 化简第(2)步得到的文法，即去除无用的非终结符号和产生式。

这种方法得到的非递归文法可能含有ε-产生式

wenshli@bupt.edu.cn

* 消除文法G=(V_T, V_N, S, φ)中的ε-产生式

- 若产生式$A\to X_1X_2\ldots X_n\in\varphi$　则把产生式$A\to\alpha_1\alpha_2\ldots\alpha_n$加入φ′

  其中：X_i、α_i为文法符号，即X_i、$\alpha_i\in(V_T\cup V_N)$

  若X_i不能产生ε，则 $\alpha_i=X_i$

  若X_i能产生ε，　则 $\alpha_i=X_i$ 或 $\alpha_i=\varepsilon$

  不能所有的α_i都取 ε

- 文法G′满足：

$$L(G')=\begin{cases}L(G)-\{\varepsilon\} & \text{如果}L(G)\text{中含有}\varepsilon\\ L(G) & \text{如果}L(G)\text{中不含有}\varepsilon\end{cases}$$

间接左递归文法：

S→Aa | b

A→Ac | Sd | ε

无ε-产生式的文法：

S→Aa | a | b

A→Ac | c | Sd

- 一个文法是ε-无关的
  - 如果它没有ε-产生式（即形如A→ε的产生式），或者
  - 只有一个ε-产生式，即S→ε，并且文法的开始符号S不出现在任何产生式的右部

wenshli@bupt.edu.cn

* 例：

消除文法中的左递归： S→Aa | b
A→Ac | Sd | ε

- 首先，必须保证此文法中无环路、无ε-产生式。

无ε-产生式的文法：
S→Aa | a | b
A→Ac | c | Sd

- 消除左递归：

step1: 把文法的非终结符号排列为：S、A；

step2: 由于S不存在直接左递归，所以算法第2步在i=1时不做工作；

在i=2时，把产生式 S→Aa | a | b 代入A的有关产生式中，得到：

A→Ac | c | Aad | ad | bd

消除A产生式中的直接左递归，得到：

A→cA′ | adA′ | bdA′

A′→cA′ | adA′ | ε

- 文法：

S→Aa | a | b
A→cA′ | adA′ | bdA′
A′→cA′ | adA′ | ε

wenshli@bupt.edu.cn

* 提取左公因子

- 如有产生式 $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

  提取左公因子α，改写为：

  $A \rightarrow \alpha A'$

  $A' \rightarrow \beta_1 \mid \beta_2$

- 若有产生式

  $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma$

  可改写为：

  $A \rightarrow \alpha A' \mid \gamma$

  $A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

- 例：映射IF语句的文法

  stmt→ if *expr* then *stmt*

  | if *expr* then *stmt* else *stmt*

  | a

  expr→b

- 提取左公因子，得到：

  stmt→if *expr* then *stmt* S′ | a

  S′→ else *stmt* | ε

  expr→b

wenshli@bupt.edu.cn

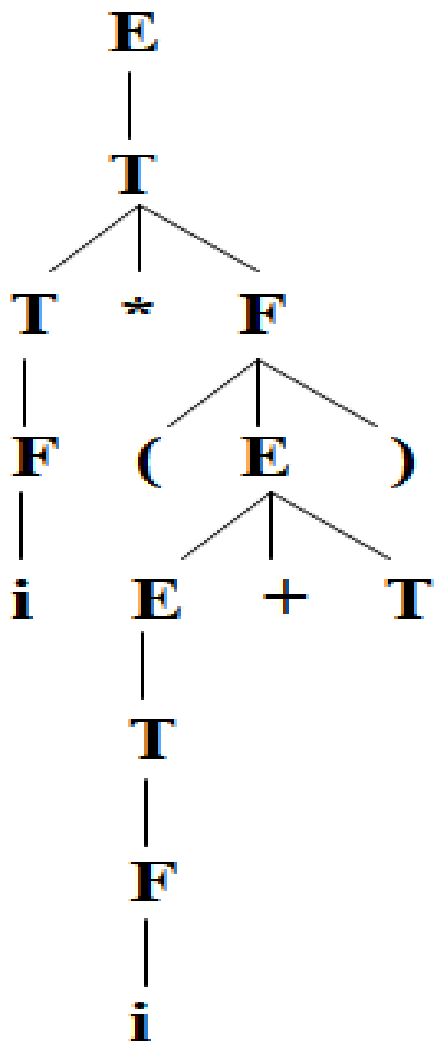
Ret

* 分析树

- 推导的图形表示，又称推导树。
- 一棵有序有向树，具有树的性质；
- 特点：每一个结点都有标记。
  - 根结点由文法的开始符号标记；
  - 内部结点由非终结符号标记，其子结点由这个非终结符号的这次推导所用产生式的右部各符号从左到右依次标记；
  - 叶结点由非终结符号或终结符号标记，它们从左到右排列起来，构成句型。
- $E \Rightarrow T \Rightarrow T*F \Rightarrow F*F \Rightarrow i*F \Rightarrow i*(E)$

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

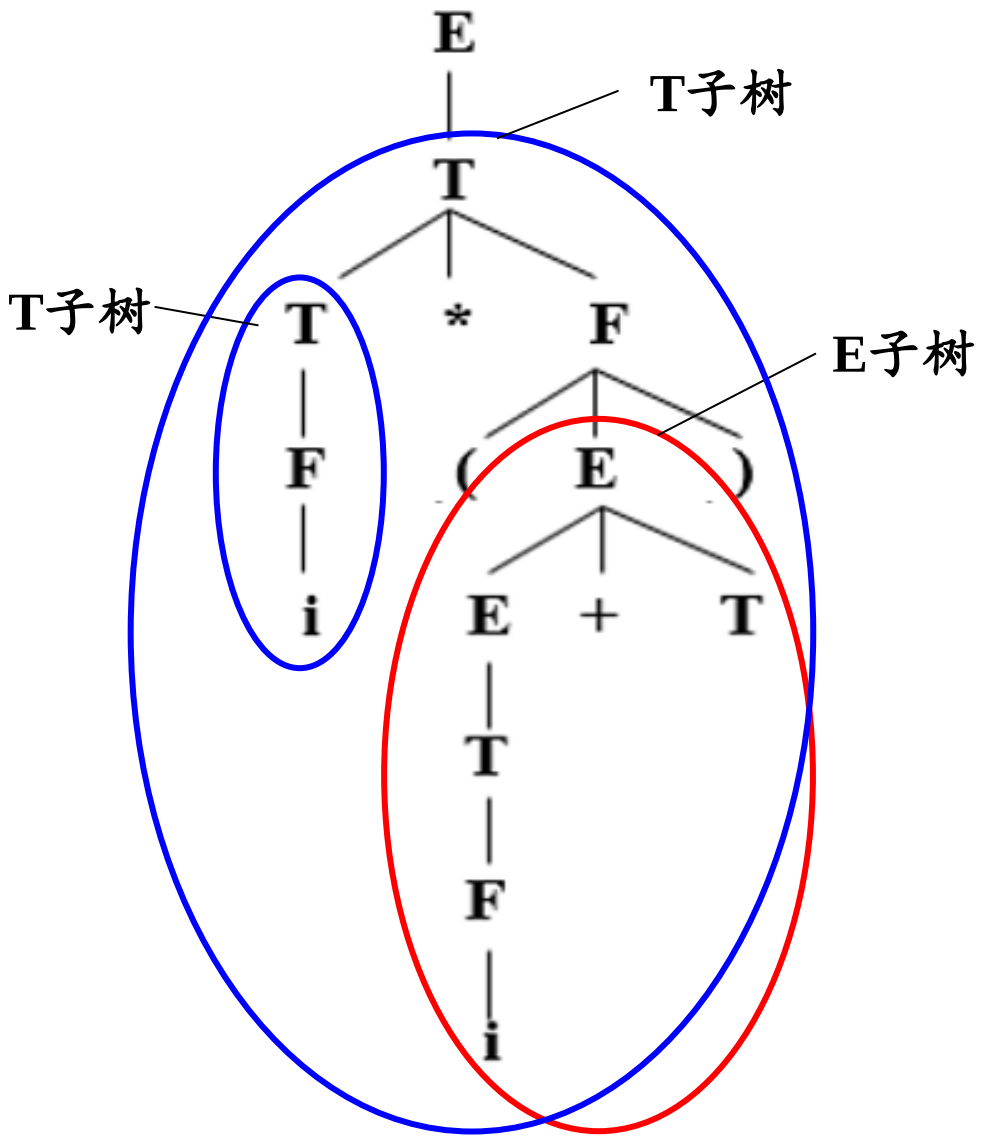
  $\Rightarrow i*(E+T) \Rightarrow i*(T+T) \Rightarrow i*(F+T) \Rightarrow i*(i+T)$



wenshli@bupt.edu.cn

* 子树

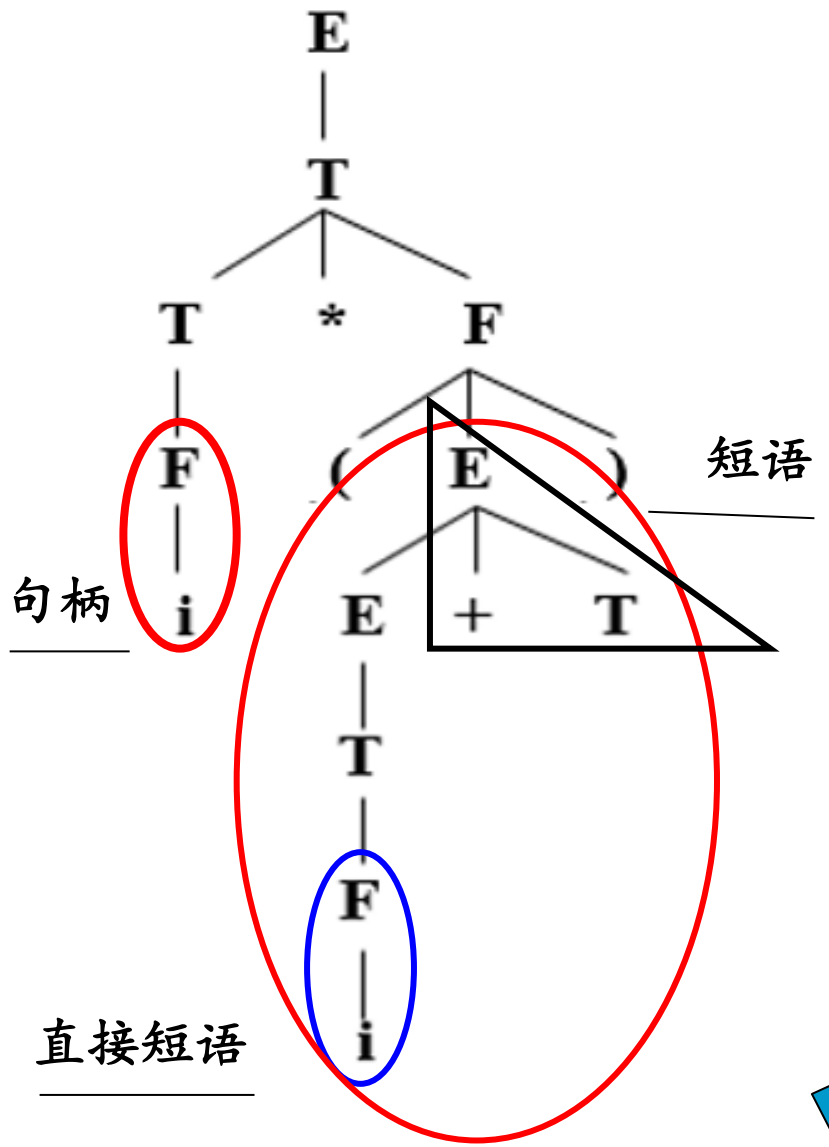
- 分析树中一个特有的结点、连同它的全部后裔结点、连接这些结点的边、以及这些结点的标记。
- 子树的根结点的标记可能不是文法的开始符号。
- 如果子树的根结点标记为非终结符号A，则可称该子树为A-子树。



wenshli@bupt.edu.cn

* 子树与短语的关系

- 一棵子树的所有叶结点自左至右排列起来，形成此句型相对于该子树根的短语；
- 分析树中只有父子两代的子树的所有叶结点自左至右排列起来，形成此句型相对于该子树根的直接短语；
- 分析树中最左边的那棵只有父子两代的子树的所有叶结点自左至右排列起来，就是该句型的句柄。



wenshli@bupt.edu.cn

Ret

* 符号串相关基本概念

- 字母表
  - 符号的非空有限集合
  - 典型的符号是字母、数字、各种标点和运算符等。
- 符号串
  - 定义在某一字母表上
  - 由该字母表中的符号组成的有限符号序列
  - 同义词：句子、字
- 符号串α的长度
  - α中出现的符号的个数，记作|α|。
  - 空串的长度为0，常用ε表示。
- 符号串α的前缀
  - 从α的末尾删除0个或多个符号后得到的符号串。如：univ 是 university 的前缀。
- 符号串α的后缀
  - 从α的开头删除0个或多个符号后得到的符号串。如：sity 是 university 的后缀。
- 符号串α的子串
  - 删除了α的前缀和/或后缀后得到的符号串。如：ver 是 university 的子串。
- 符号串α的真前缀、真后缀、真子串
  - 如果非空符号串β是α的前缀、后缀或子串，并且β≠α，则称β是α的真前缀、真后缀、或真子串。
- 符号串α的子序列
  - 从α中删除0个或多个符号(这些符号可以是不连续的)后得到的符号串。如：nvst

wenshli@bupt.edu.cn

* 符号串相关基本概念

■ 连接

- 符号串α和符号串β的连接αβ是把符号串β加在符号串α之后得到的符号串
- 若α=ab，β=cd，则αβ=abcd，βα=cdba。
- 对任何符号串α来说，都有εα=αε=α

■ 幂

- 若α是符号串，α的n次幂α^n定义为：$\underbrace{\alpha\alpha\ldots\alpha\alpha}_{n个}$
- 当n=0时，α^0是空串，$\alpha^0=\varepsilon$。
- 假如α=ab，则有：

$\alpha^0=\varepsilon$　$\alpha^1=ab$　$\alpha^2=abab$

……

$\alpha^n=\underbrace{abab\ldots\ldots ab}_{n个ab}$

wenshli@bupt.edu.cn

Ret