

# 语法分析

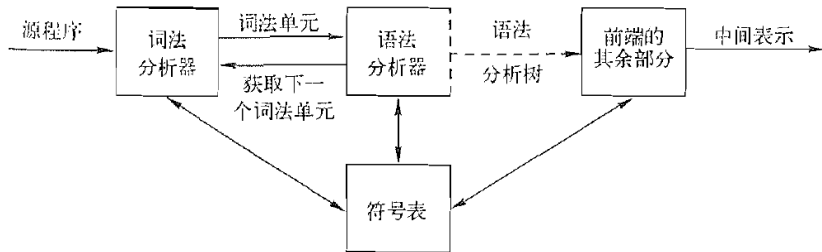
魏恒峰

hfwei@nju.edu.cn

2020 年 11 月 30 日



**输入:** 词法单元流 & 语言的语法规则



**输出:** 语法分析树 (Parse Tree)

## 语法分析举例

$\langle \text{Stmt} \rangle \rightarrow \langle \text{Id} \rangle = \langle \text{Expr} \rangle ;$
$\langle \text{Stmt} \rangle \rightarrow \{ \langle \text{StmtList} \rangle \}$
$\langle \text{Stmt} \rangle \rightarrow \text{if} ( \langle \text{Expr} \rangle ) \langle \text{Stmt} \rangle$
$\langle \text{StmtList} \rangle \rightarrow \langle \text{Stmt} \rangle$
$\langle \text{StmtList} \rangle \rightarrow \langle \text{StmtList} \rangle \langle \text{Stmt} \rangle$
$\langle \text{Expr} \rangle \rightarrow \langle \text{Id} \rangle$
$\langle \text{Expr} \rangle \rightarrow \langle \text{Num} \rangle$
$\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle$
$\langle \text{Id} \rangle \rightarrow x$
$\langle \text{Id} \rangle \rightarrow y$
$\langle \text{Num} \rangle \rightarrow 0$
$\langle \text{Num} \rangle \rightarrow 1$
$\langle \text{Num} \rangle \rightarrow 9$
$\langle \text{Optr} \rangle \rightarrow >$
$\langle \text{Optr} \rangle \rightarrow +$

		$\langle \text{Stmt} \rangle$
if (	$\langle \text{Expr} \rangle$	$\langle \text{Stmt} \rangle$
if (	$\langle \text{Expr} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle$	$\langle \text{Stmt} \rangle$
if (	$\langle \text{Id} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle$	$\langle \text{Stmt} \rangle$
if (	$x \langle \text{Optr} \rangle \langle \text{Expr} \rangle$	$\langle \text{Stmt} \rangle$
if (	$x > \langle \text{Expr} \rangle$	$\langle \text{Stmt} \rangle$
if (	$x > \langle \text{Num} \rangle$	$\langle \text{Stmt} \rangle$
if (	$x > 9$	$\langle \text{Stmt} \rangle$
if (	$x > 9$	{ $\langle \text{StmtList} \rangle$ }
if (	$x > 9$	{ $\langle \text{StmtList} \rangle \langle \text{Stmt} \rangle$ }
if (	$x > 9$	{ $\langle \text{Stmt} \rangle$ }
if (	$x > 9$	{ $\langle \text{Id} \rangle = \langle \text{Expr} \rangle ;$ }
if (	$x > 9$	{ $x = \langle \text{Expr} \rangle ;$ }
if (	$x > 9$	{ $x = \langle \text{Num} \rangle ;$ }
if (	$x > 9$	{ $x = 0 ;$ }
if (	$x > 9$	{ $x = 0 ; \langle \text{Id} \rangle = \langle \text{Expr} \rangle ;$ }
if (	$x > 9$	{ $x = 0 ; y = \langle \text{Expr} \rangle ;$ }
if (	$x > 9$	{ $x = 0 ; y = \langle \text{Expr} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle ;$ }
if (	$x > 9$	{ $x = 0 ; y = \langle \text{Id} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle ;$ }
if (	$x > 9$	{ $x = 0 ; y = y \langle \text{Optr} \rangle \langle \text{Expr} \rangle ;$ }
if (	$x > 9$	{ $x = 0 ; y = y + \langle \text{Expr} \rangle ;$ }
if (	$x > 9$	{ $x = 0 ; y = y + \langle \text{Num} \rangle ;$ }
if (	$x > 9$	{ $x = 0 ; y = y + 1 ;$ }

## 语法分析阶段的主题之一: 上下文无关文法

$\langle \text{Stmt} \rangle \rightarrow \langle \text{Id} \rangle = \langle \text{Expr} \rangle ;$   
 $\langle \text{Stmt} \rangle \rightarrow \{ \langle \text{StmtList} \rangle \}$   
 $\langle \text{Stmt} \rangle \rightarrow \text{if} ( \langle \text{Expr} \rangle ) \langle \text{Stmt} \rangle$   
 $\langle \text{StmtList} \rangle \rightarrow \langle \text{Stmt} \rangle$   
 $\langle \text{StmtList} \rangle \rightarrow \langle \text{StmtList} \rangle \langle \text{Stmt} \rangle$   
 $\langle \text{Expr} \rangle \rightarrow \langle \text{Id} \rangle$   
 $\langle \text{Expr} \rangle \rightarrow \langle \text{Num} \rangle$   
 $\langle \text{Expr} \rangle \rightarrow \langle \text{Expr} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle$   
 $\langle \text{Id} \rangle \rightarrow x$   
 $\langle \text{Id} \rangle \rightarrow y$   
 $\langle \text{Num} \rangle \rightarrow 0$   
 $\langle \text{Num} \rangle \rightarrow 1$   
 $\langle \text{Num} \rangle \rightarrow 9$   
 $\langle \text{Optr} \rangle \rightarrow >$   
 $\langle \text{Optr} \rangle \rightarrow +$

## 语法分析阶段的主题之二: 构建语法分析树

$\langle \text{Stmt} \rangle$			
if (	$\langle \text{Expr} \rangle$	)	$\langle \text{Stmt} \rangle$
if (	$\langle \text{Expr} \rangle$ $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$	)	$\langle \text{Stmt} \rangle$
if (	$\langle \text{Id} \rangle$ $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$	)	$\langle \text{Stmt} \rangle$
if (	x $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$	)	$\langle \text{Stmt} \rangle$
if (	x > $\langle \text{Expr} \rangle$	)	$\langle \text{Stmt} \rangle$
if (	x > $\langle \text{Num} \rangle$	)	$\langle \text{Stmt} \rangle$
if (	x > 9	)	$\langle \text{Stmt} \rangle$
if (	x > 9	) {	$\langle \text{StmtList} \rangle$ }
if (	x > 9	) {	$\langle \text{StmtList} \rangle$ $\langle \text{Stmt} \rangle$ }
if (	x > 9	) {	$\langle \text{Stmt} \rangle$ $\langle \text{Stmt} \rangle$ }
if (	x > 9	) {	$\langle \text{Id} \rangle = \langle \text{Expr} \rangle ;$ $\langle \text{Stmt} \rangle$ }
if (	x > 9	) {	x = $\langle \text{Expr} \rangle ;$ $\langle \text{Stmt} \rangle$ }
if (	x > 9	) {	x = $\langle \text{Num} \rangle ;$ $\langle \text{Stmt} \rangle$ }
if (	x > 9	) {	x = 0 $\langle \text{Stmt} \rangle$ }
if (	x > 9	) {	x = 0 ; $\langle \text{Id} \rangle = \langle \text{Expr} \rangle ;$ }
if (	x > 9	) {	x = 0 ; y = $\langle \text{Expr} \rangle ;$ }
if (	x > 9	) {	x = 0 ; y = $\langle \text{Expr} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle ;$ }
if (	x > 9	) {	x = 0 ; y = $\langle \text{Id} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle ;$ }
if (	x > 9	) {	x = 0 ; y = y $\langle \text{Optr} \rangle \langle \text{Expr} \rangle ;$ }
if (	x > 9	) {	x = 0 ; y = y + $\langle \text{Expr} \rangle ;$ }
if (	x > 9	) {	x = 0 ; y = y + $\langle \text{Num} \rangle ;$ }
if (	x > 9	) {	x = 0 ; y = y + 1 ; }

### 语法分析阶段的主题之三: 错误恢复



报错、恢复、继续分析



## <Context-Free Grammar>

上下文无关文法

## Definition (Context-Free Grammar (CFG); 上下文无关文法)

上下文无关文法  $G$  是一个四元组  $G = (T, N, P, S)$ :

- ▶  $T$  是**终结符号** (Terminal) 集合, 对应于词法分析器产生的词法单元;
- ▶  $N$  是**非终结符号** (Non-terminal) 集合;
- ▶  $P$  是**产生式** (Production) 集合;

$$A \in N \longrightarrow \alpha \in (T \cup N)^*$$

头部/左部 (Head)  $A$ : **单个**非终结符

体部/右部 (Body)  $\alpha$ : 终结符与非终结符构成的串, 也可以是空串  $\epsilon$

- ▶  $S$  为**开始** (Start) 符号。要求  $S \in N$  且唯一。



$$G = (\{a, b\}, \{S\}, P, S)$$

$$S \rightarrow aSb$$

$$S \rightarrow \epsilon$$

$$G = (\{(, )\}, \{S\}, P, S)$$

$$S \rightarrow SS$$

$$S \rightarrow (S)$$

$$S \rightarrow ()$$

$$S \rightarrow \epsilon$$

$stmt \rightarrow$  if  $expr$  then  $stmt$   
| if  $expr$  then  $stmt$  else  $stmt$   
| other

条件语句文法

悬空 (Dangling)-else 文法

$S \rightarrow \text{if } E \text{ then } S \text{ else } S$

$S \rightarrow \text{begin } S L$

$S \rightarrow \text{print } E$

$L \rightarrow \text{end}$

$L \rightarrow ; S L$

$E \rightarrow \text{num} = \text{num}$

**约定:** 如果没有明确指定, 第一个产生式的头部就是开始符号

## 关于终结符号的约定

1) 下述符号是终结符号:

- ① 在字母表里排在前面的小写字母, 比如  $a$ 、 $b$ 、 $c$ 。
- ② 运算符号, 比如  $+$ 、 $*$  等。
- ③ 标点符号, 比如括号、逗号等。
- ④ 数字  $0$ 、 $1$ 、 $\dots$ 、 $9$ 。
- ⑤ **黑体字符串**, 比如 **id** 或 **if**。每个这样的字符串表示一个终结符号。

## 关于非终结符号的约定

2) 下述符号是非终结符号:

- ① 在字母表中排在前面的大写字母, 比如  $A$ 、 $B$ 、 $C$ 。
- ② 字母  $S$ 。它出现时通常表示开始符号。
- ③ 小写、斜体的名字, 比如  $expr$  或  $stmt$ 。



Syntax

Semantics

语义: 上下文无关文法  $G$  定义了一个**语言**  $L(G)$

语言是**串**的集合

串从何来?

## 推导 (Derivation)

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

推导即是將某个产生式的左边**替换**成它的右边

每一步推导需要选择**替换哪个非终结符号**, 以及**使用哪个产生式**



## 推导 (Derivation)

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

$$E \Longrightarrow -E \Longrightarrow -(E) \Longrightarrow -(E + E) \Longrightarrow -(\mathbf{id} + E) \Longrightarrow -(\mathbf{id} + \mathbf{id})$$

$E \Longrightarrow -E$  : 经过一步推导得出

$E \stackrel{+}{\Longrightarrow} -(\mathbf{id} + E)$  : 经过一步或多步推导得出

$E \stackrel{*}{\Longrightarrow} -(\mathbf{id} + E)$  : 经过零步或多步推导得出

$$E \Longrightarrow -E \Longrightarrow -(E) \Longrightarrow -(E + E) \Longrightarrow -(E + \mathbf{id}) \Longrightarrow -(\mathbf{id} + \mathbf{id})$$

### Definition (Sentential Form; 句型)

如果  $S \xRightarrow{*} \alpha$ , 且  $\alpha \in (T \cup N)^*$ , 则称  $\alpha$  是文法  $G$  的一个**句型**。

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

$$E \Longrightarrow -E \Longrightarrow -(E) \Longrightarrow -(E+E) \Longrightarrow -(\text{id} + E) \Longrightarrow -(\text{id} + \text{id})$$

### Definition (Sentence; 句子)

如果  $S \xRightarrow{*} w$ , 且  $w \in T^*$ , 则称  $w$  是文法  $G$  的一个**句子**。

Definition (文法  $G$  生成的语言  $L(G)$ )

文法  $G$  的**语言**  $L(G)$  是它能推导出的**所有句子**构成的集合。

$$w \in L(G) \iff S \xRightarrow{*} w$$

关于文法  $G$  的**两个基本问题**:

- ▶ **Membership 问题**: 给定字符串  $x \in T^*$ ,  $x \in L(G)$ ?
- ▶  $L(G)$  究竟是什么?

给定字符串  $x \in T^*$ ,  $x \in L(G)$ ?

(即, 检查  $x$  是否符合文法  $G$ )

这就是**语法分析器**的任务:

为输入的词法单元流寻找推导、**构建语法分析树**, 或者报错

## 根节点是文法 $G$ 的起始符号

$\langle \text{Stmt} \rangle$		
if (	$\langle \text{Expr} \rangle$	) $\langle \text{Stmt} \rangle$
if (	$\langle \text{Expr} \rangle$ $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$	) $\langle \text{Stmt} \rangle$
if (	$\langle \text{Id} \rangle$ $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$	) $\langle \text{Stmt} \rangle$
if (	$x$ $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$	) $\langle \text{Stmt} \rangle$
if (	$x$ $>$ $\langle \text{Expr} \rangle$	) $\langle \text{Stmt} \rangle$
if (	$x$ $>$ $\langle \text{Num} \rangle$	) $\langle \text{Stmt} \rangle$
if (	$x$ $>$ $9$	) $\langle \text{Stmt} \rangle$
if (	$x$ $>$ $9$	) { $\langle \text{StmtList} \rangle$ }
if (	$x$ $>$ $9$	) { $\langle \text{StmtList} \rangle$ $\langle \text{Stmt} \rangle$ }
if (	$x$ $>$ $9$	) { $\langle \text{Stmt} \rangle$ $\langle \text{Stmt} \rangle$ }
if (	$x$ $>$ $9$	) { $\langle \text{Id} \rangle = \langle \text{Expr} \rangle ;$ $\langle \text{Stmt} \rangle$ }
if (	$x$ $>$ $9$	) { $x = \langle \text{Expr} \rangle ;$ $\langle \text{Stmt} \rangle$ }
if (	$x$ $>$ $9$	) { $x = \langle \text{Num} \rangle ;$ $\langle \text{Stmt} \rangle$ }
if (	$x$ $>$ $9$	) { $x = 0 ;$ $\langle \text{Stmt} \rangle$ }
if (	$x$ $>$ $9$	) { $x = 0 ;$ $\langle \text{Id} \rangle = \langle \text{Expr} \rangle ;$ }
if (	$x$ $>$ $9$	) { $x = 0 ;$ $y = \langle \text{Expr} \rangle ;$ }
if (	$x$ $>$ $9$	) { $x = 0 ;$ $y = \langle \text{Expr} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle ;$ }
if (	$x$ $>$ $9$	) { $x = 0 ;$ $y = \langle \text{Id} \rangle \langle \text{Optr} \rangle \langle \text{Expr} \rangle ;$ }
if (	$x$ $>$ $9$	) { $x = 0 ;$ $y = y \langle \text{Optr} \rangle \langle \text{Expr} \rangle ;$ }
if (	$x$ $>$ $9$	) { $x = 0 ;$ $y = y + \langle \text{Expr} \rangle ;$ }
if (	$x$ $>$ $9$	) { $x = 0 ;$ $y = y + \langle \text{Num} \rangle ;$ }
if (	$x$ $>$ $9$	) { $x = 0 ;$ $y = y + 1 ;$ }

## 叶子节点是输入的词法单元流

常用的语法分析器以**自顶向下**或**自底向上**的方式构建中间部分

$L(G)$  是什么?

这是**程序设计语言设计者**需要考虑的问题

$$S \rightarrow SS$$

$$S \rightarrow (S)$$

$$S \rightarrow ()$$

$$S \rightarrow \epsilon$$

$$S \rightarrow aSb$$

$$S \rightarrow \epsilon$$

$$L(G) = \{a^n b^n \mid n \geq 0\}$$

$$L(G) = \{\text{良匹配括号串}\}$$



字母表  $\Sigma = \{a, b\}$  上的所有回文串 (Palindrome) 构成的语言

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

$$S \rightarrow a$$

$$S \rightarrow b$$

$$S \rightarrow \epsilon$$

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$$

$$\{b^n a^m b^{2n} \mid n \geq 0, m \geq 0\}$$

$$S \rightarrow bSbb \mid A$$

$$A \rightarrow aA \mid \epsilon$$

$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数相同}\}$

$$V \rightarrow aVbV \mid bVaV \mid \epsilon$$

$\{x \in \{a, b\}^* \mid x \text{ 中 } a, b \text{ 个数不同}\}$

$$S \rightarrow T \mid U$$
$$T \rightarrow VaT \mid VaV$$
$$U \rightarrow VbU \mid VbV$$
$$V \rightarrow aVbV \mid bVaV \mid \epsilon$$


练习 (非作业): 证明之

$S \rightarrow \text{if } E \text{ then } S \text{ else } S$

$S \rightarrow \text{begin } S L$

$S \rightarrow \text{print } E$

$L \rightarrow \text{end}$

$L \rightarrow ; S L$

$E \rightarrow \text{num} = \text{num}$

**顺序语句**、条件语句、打印语句



## *L-System*

(注: 这不是上下文无关文法, 但精神上高度一致, 并且更有趣)

**variables** : A B

**constants** : + -

**start** : A

**rules** :  $(A \rightarrow B-A-B)$ ,  $(B \rightarrow A+B+A)$

**angle** :  $60^\circ$

$A, B$  : 向右移动并画线

+: 左转

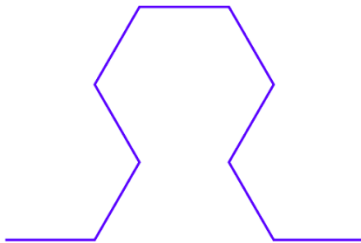
-: 右转

每一步都**并行地**应用**所有**规则

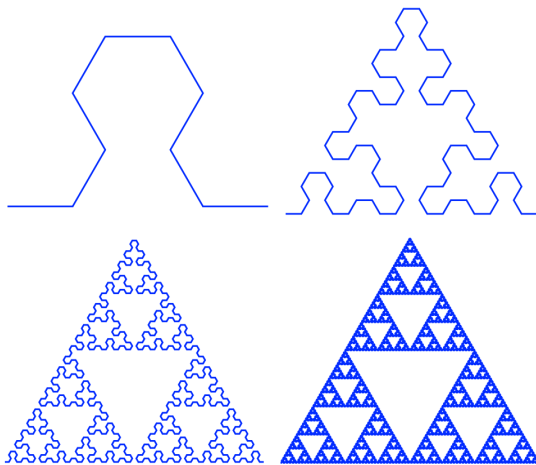
$A$

$B - A - B$

$A + B + A - B - A - B - A + B + A$







Sierpinski arrowhead curve ( $n = 2, 4, 6, 8$ )

**variables** :  $X Y$

**constants** :  $F + -$

**start** :  $FX$

**rules** :  $(X \rightarrow X+YF+), (Y \rightarrow -FX-Y)$

**angle** :  $90^\circ$

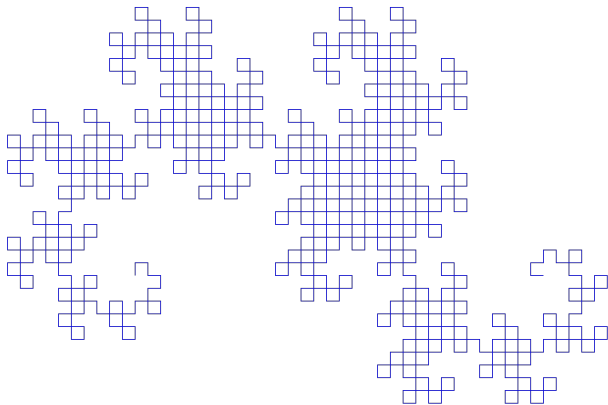
$F$  : 向上移动并画线

$+$  : 右转

$-$  : 左转

$X$  : 仅用于展开, 在作画时被忽略

每一步都**并行地**应用**所有**规则



Dragon Curve ( $n = 10$ )

## 最左 (leftmost) 推导与最右 (rightmost) 推导

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

$$E \xRightarrow{\text{lm}} -E \xRightarrow{\text{lm}} -(E) \xRightarrow{\text{lm}} -(E + E) \xRightarrow{\text{lm}} -(\mathbf{id} + E) \xRightarrow{\text{lm}} -(\mathbf{id} + \mathbf{id})$$

$E \xRightarrow{\text{lm}} -E$  : 经过一步最左推导得出

$E \xRightarrow[\text{lm}]{+} -(\mathbf{id} + E)$  : 经过一步或多步最左推导得出

$E \xRightarrow[\text{lm}]{*} -(\mathbf{id} + E)$  : 经过零步或多步最左推导得出

$$E \xRightarrow{\text{rm}} -E \xRightarrow{\text{rm}} -(E) \xRightarrow{\text{rm}} -(E + E) \xRightarrow{\text{rm}} -(E + \mathbf{id}) \xRightarrow{\text{rm}} -(\mathbf{id} + \mathbf{id})$$

### Definition (Left-sentential Form; 最左句型)

如果  $S \xRightarrow[\text{lm}]{*} \alpha$ , 且  $\alpha \in (T \cup N)^*$ , 则称  $\alpha$  是文法  $G$  的一个**最左句型**。

$$E \xRightarrow{\text{lm}} -E \xRightarrow{\text{lm}} -(E) \xRightarrow{\text{lm}} -(E + E) \xRightarrow{\text{lm}} -(\text{id} + E) \xRightarrow{\text{lm}} -(\text{id} + \text{id})$$

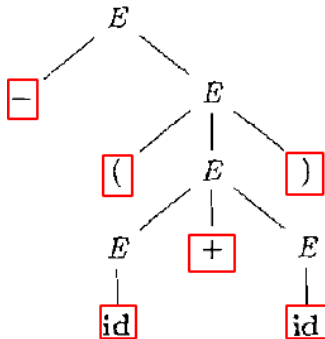
### Definition (Right-sentential Form; 最右句型)

如果  $S \xRightarrow[\text{rm}]{*} \alpha$ , 且  $\alpha \in (T \cup N)^*$ , 则称  $\alpha$  是文法  $G$  的一个**最右句型**。

$$E \xRightarrow{\text{rm}} -E \xRightarrow{\text{rm}} -(E) \xRightarrow{\text{rm}} -(E + E) \xRightarrow{\text{rm}} -(E + \text{id}) \xRightarrow{\text{rm}} -(\text{id} + \text{id})$$

## 语法分析树

语法分析树是静态的, 它不关心动态的推导顺序

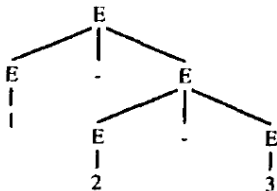
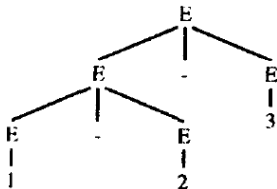


一棵语法分析树对应多个推导

但是, 一棵语法分析树与**最左 (最右) 推导**一一对应

$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \text{id} \mid \text{num}$$

1 - 2 - 3 的语法树?

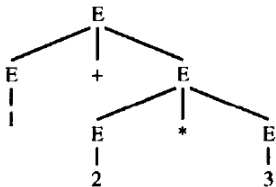
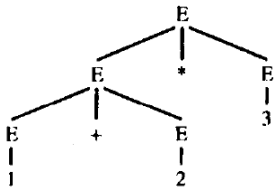


Definition (二义性(Ambiguous) 文法)

如果  $L(G)$  中的某个句子有一个以上语法树/最左推导/最右推导, 则文法  $G$  是二义性的。

$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \text{id} \mid \text{num}$$

1 + 2 \* 3 的语法树?

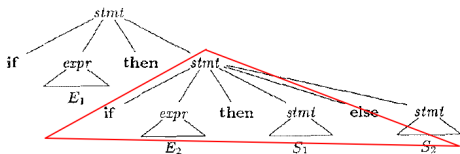




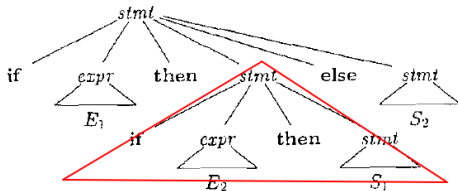
$stmt \rightarrow$  if  $expr$  then  $stmt$   
           | if  $expr$  then  $stmt$  else  $stmt$   
           | other

“悬空-else” 文法

if  $E_1$  then if  $E_2$  then  $S_1$  else  $S_2$



if  $E_1$  then (if  $E_2$  then  $S_1$  else  $S_2$ )

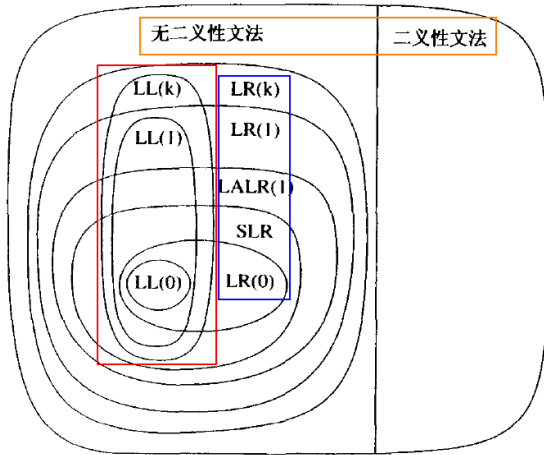


if  $E_1$  then (if  $E_2$  then  $S_1$ ) else  $S_2$

## 二义性文法

不同的语法分析树产生不同的语义





所有语法分析器都要求文法是**无二义性**的

## 二义性文法

Q : 如何**识别**二义性文法?



Q : 如何**消除**文法的二义性?

LEARN BY EXAMPLES

这是**不可判定**的问题

$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \text{id} \mid \text{num}$$

四则运算均是**左结合**的

**优先级**: 括号最先, 先乘除后加减

二义性表达式文法以**相同的方式**处理所有的算术运算符

要消除二义性, 需要**区别对待**不同的运算符

将运算的“先后”顺序信息编码到语法树的“层次”结构中

$$E \rightarrow E + E \mid \text{id}$$

$$E \rightarrow E + T$$

$$T \rightarrow \text{id}$$

左结合文法

$$E \rightarrow T + E$$

$$T \rightarrow \text{id}$$

右结合文法

使用左 (右) 递归实现左 (右) 结合

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

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid \mathbf{id}$$

括号最先, 先乘后加文法

$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \mathbf{id} \mid \mathbf{num}$$

$$E \rightarrow E + T \mid E - T \mid T$$

$$T \rightarrow T * F \mid T / F \mid F$$

$$F \rightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$$

无二义性的表达式文法

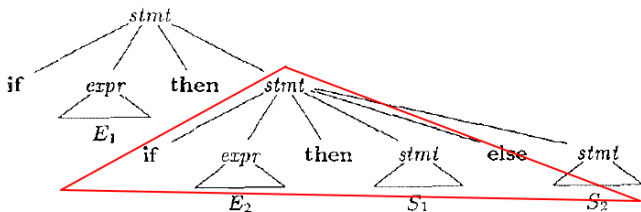
$E$  : 表达式(*expression*);     $T$  : 项(*term*)     $F$  : 因子(*factor*)

将运算的“先后”顺序信息编码到语法树的“层次”结构中



$stmt \rightarrow$  if  $expr$  then  $stmt$   
 $\quad \mid$  if  $expr$  then  $stmt$  else  $stmt$   
 $\quad \mid$  other

if  $E_1$  then if  $E_2$  then  $S_1$  else  $S_2$



“每个else与最近的尚未匹配的then匹配”

$$\begin{aligned}
 stmt &\rightarrow \text{if } expr \text{ then } stmt \\
 &\quad | \text{ if } expr \text{ then } stmt \text{ else } stmt \\
 &\quad | \text{ other}
 \end{aligned}$$

$stmt$	$\rightarrow$	$matched\_stmt$
	$ $	$open\_stmt$
$matched\_stmt$	$\rightarrow$	$\text{if } expr \text{ then } matched\_stmt \text{ else } matched\_stmt$
	$ $	$\text{other}$
$open\_stmt$	$\rightarrow$	$\text{if } expr \text{ then } stmt$
	$ $	$\text{if } expr \text{ then } matched\_stmt \text{ else } open\_stmt$

基本思想: **then** 与 **else** 之间的语句必须是“已匹配的”

我也看不懂啊

~~“我不想去上课啊妈妈”~~

“清醒一点！你是老师啊！”



我们要证明**两**件事情

$$L(G) = L(G')$$

$G'$  是无二义性的

$$\begin{aligned}
 stmt &\rightarrow \text{if } expr \text{ then } stmt \\
 &\quad | \text{ if } expr \text{ then } stmt \text{ else } stmt \\
 &\quad | \text{ other}
 \end{aligned}$$

$stmt$	$\rightarrow$	$matched\_stmt$
	$ $	$open\_stmt$
$matched\_stmt$	$\rightarrow$	$\text{if } expr \text{ then } matched\_stmt \text{ else } matched\_stmt$
	$ $	$\text{other}$
$open\_stmt$	$\rightarrow$	$\text{if } expr \text{ then } stmt$
	$ $	$\text{if } expr \text{ then } matched\_stmt \text{ else } open\_stmt$

$$L(G') \subseteq L(G)$$

$$L(G) \subseteq L(G')$$

对推导步数作数学归纳

$G'$  是无二义性的

$stmt$	$\rightarrow$	$matched\_stmt$
	$ $	$open\_stmt$
$matched\_stmt$	$\rightarrow$	$if\ expr\ then\ matched\_stmt\ else\ matched\_stmt$
	$ $	$other$
$open\_stmt$	$\rightarrow$	$if\ expr\ then\ stmt$
	$ $	$if\ expr\ then\ matched\_stmt\ else\ open\_stmt$

每个句子对应的语法分析树是唯一的

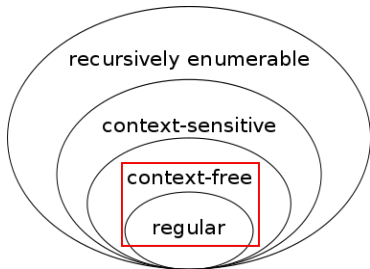
只需证明: 每个非终结符的“展开”方式是唯一的

$$L(matched\_stmt) \cap L(open\_stmt) = \emptyset$$

$$L(matched\_stmt_1) \cap L(matched\_stmt_2) = \emptyset$$

$$L(open\_stmt_1) \cap L(open\_stmt_2) = \emptyset$$

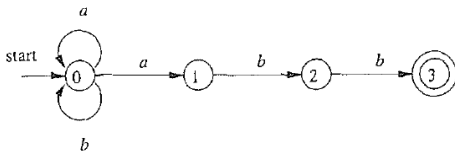
为什么不使用优雅、强大的**正则表达式**描述程序设计语言的语法？



正则表达式的表达能力**严格弱于**上下文无关文法

每个正则表达式  $r$  对应的语言  $L(r)$  都可以使用上下文无关文法来描述

$$r = (a|b)^*abb$$



$$\begin{aligned} A_0 &\rightarrow aA_0 \mid bA_0 \mid aA_1 \\ A_1 &\rightarrow bA_2 \\ A_2 &\rightarrow bA_3 \\ A_3 &\rightarrow \epsilon \end{aligned}$$

此外, 若  $\delta(A_i, \epsilon) = A_j$ , 则添加  $A_i \rightarrow A_j$



$$S \rightarrow aSb$$

$$S \rightarrow \epsilon$$

$$L = \{a^n b^n \mid n \geq 0\}$$

该语言**无法**使用正则表达式来描述

## Theorem

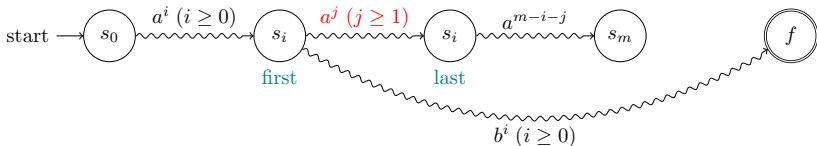
$L = \{a^n b^n \mid n \geq 0\}$  无法使用正则表达式描述。

### 反证法

假设存在正则表达式  $r: L(r) = L$

则存在有限状态自动机  $D(r): L(D(r)) = L$ ; 设其状态数为  $k$

考虑输入  $a^m (m > k)$



$D(r)$  也能接受  $a^{i+j}b^i$ ; 矛盾!

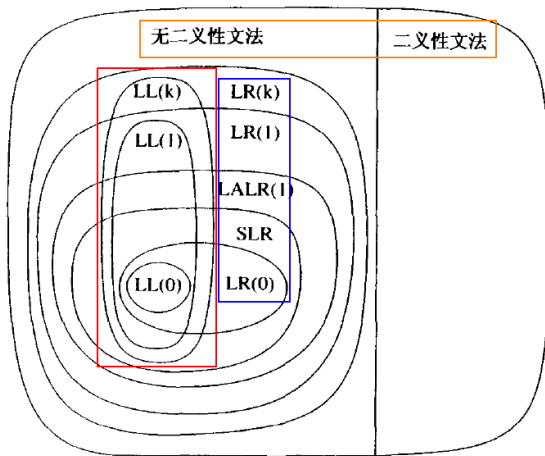
$$L = \{a^n b^n \mid n \geq 0\}$$

Pumping Lemma for Regular Languages

$$L = \{a^n b^n c^n \mid n \geq 0\}$$

Pumping Lemma for Context-free Languages

只考虑**无二义性**的文法  
这意味着, 每个句子对应唯一的一棵语法分析树



今日份主题: **LL(1) 语法分析器**

自顶向下的、  
递归下降的、  
预测分析的、  
适用于 $LL(1)$  文法的、  
 $LL(1)$  语法分析器

## 自顶向下构建语法分析树

根节点是文法的起始符号  $S$

每个中间节点表示对某个非终结符应用某个产生式进行推导  
( $Q$ : 选择哪个非终结符, 以及选择哪个产生式)

叶节点是词法单元流  $w\$$

仅包含终结符号与特殊的文件结束符  $\$$

## 递归下降的实现框架

```
void A() {  
    1) 选择一个 A 产生式,  $A \rightarrow X_1 X_2 \cdots X_k$ ;  
    2)  for (  $i = 1$  to  $k$  ) {  
    3)      if (  $X_i$  是一个非终结符号 )  
    4)          递归下降调用其它非终结符对应的递归函数  
    5)          调用过程  $X_i()$ ;  
    6)      else if (  $X_i$  等于当前的输入符号  $a$  )  
    7)          读入下一个输入符号;  
    8)      else /* 发生了一个错误 */;  
    9)  }  
}
```

不需要任何参数

先不考虑这里是如何选择产生式的

匹配当前词法单元

出现了不期望出现的词法单元

为每个非终结符写一个递归函数

内部按需调用其它非终结符对应的递归函数

$$S \rightarrow F$$

$$S \rightarrow (S + F)$$

$$F \rightarrow a$$

$$w = ((a + a) + a)$$

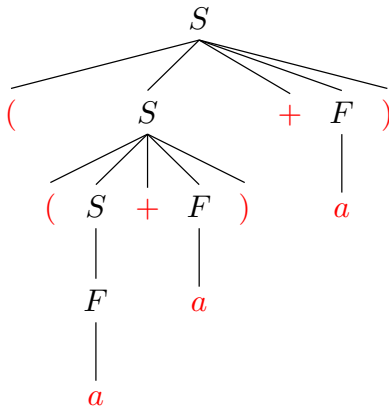


## 演示递归下降过程

$$S \rightarrow F$$

$$S \rightarrow (S + F)$$

$$F \rightarrow a$$

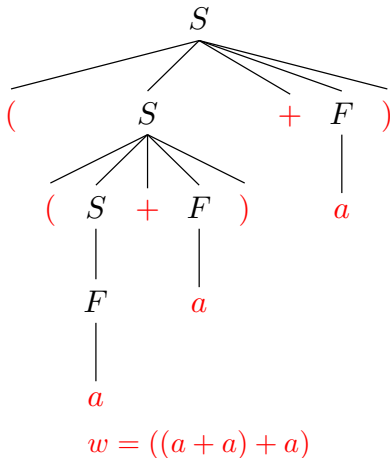


$$w = ((a + a) + a)$$

每次都选择语法分析树**最左边**的非终结符进行展开

同样是展开非终结符  $S$ ,  
为什么前两次选择了  $S \rightarrow (S + F)$ , 而第三次选择了  $S \rightarrow F$ ?

$S \rightarrow F$
$S \rightarrow (S + F)$
$F \rightarrow a$



因为它们面对的**当前词法单元**不同

## 使用预测分析表确定产生式

$S \rightarrow F$
$S \rightarrow (S + F)$
$F \rightarrow a$

		(	)	$a$	+	\$
$S$		2		1		
$F$				3		

指明了每个非终结符在面对不同的词法单元或文件结束符时，  
该选择哪个产生式（按编号进行索引）或者报错

## Definition ( $LL(1)$ 文法)

如果文法  $G$  的**预测分析表**是**无冲突**的, 则  $G$  是  $LL(1)$  文法。

**无冲突:** 每个单元格里只有一个生成式 (编号)

$S \rightarrow F$
$S \rightarrow (S + F)$
$F \rightarrow a$

	(	)	$a$	+	\$
$S$	2		1		
$F$			3		

对于当前选择的**非终结符**,  
仅根据输入中**当前的词法单元**即可确定需要使用哪条**产生式**

## 递归下降的、预测分析实现方法

$$S \rightarrow F$$
$$S \rightarrow (S + F)$$
$$F \rightarrow a$$

	(	)	a	+	\$
S	2		1		
F			3		

---

```
1: procedure MATCH(t)
2:   if token = t then
3:     token ← NEXT-TOKEN()
4:   else
```

---

```
1: procedure S()
2:   if token = '(' then
3:     MATCH('(')
4:     S()
5:     MATCH('+')
6:     F()
7:     MATCH(')')
8:   else if token = 'a' then
9:     F()
10:  else
11:    ERROR(token, {'(', 'a'})
```

---

---

```
1: procedure F()
2:   if token = 'a' then
```

## 如何计算给定文法 $G$ 的预测分析表?

$\text{FIRST}(\alpha)$  是可从  $\alpha$  推导得到的句型的**首终结符号**的集合

Definition ( $\text{FIRST}(\alpha)$  集合)

对于任意的 (产生式的右部)  $\alpha \in (N \cup T)^*$ :

$$\text{FIRST}(\alpha) = \{t \in T \cup \{\epsilon\} \mid \alpha \xRightarrow{*} t\beta \vee \alpha \xRightarrow{*} \epsilon\}.$$

考虑非终结符  $A$  的所有产生式  $A \rightarrow \alpha_1, A \rightarrow \alpha_2, \dots, A \rightarrow \alpha_m$ ,

如果它们对应的  $\text{FIRST}(\alpha_i)$  集合互不相交,

则只需查看当前输入词法单元, 即可确定选择哪个产生式 (或报错)

## 如何计算给定文法 $G$ 的预测分析表?

$\text{FOLLOW}(A)$  是可能在某些句型中**紧跟在  $A$  右边的终结符**的集合

Definition ( $\text{FOLLOW}(A)$  集合)

对于任意的 (产生式的左部) 非终结符  $A \in N$  :

$$\text{FOLLOW}(A) = \{t \in T \cup \{\$\} \mid \exists s. S \xRightarrow{*} s \triangleq \beta A t \gamma\}.$$

考虑产生式  $A \rightarrow \alpha$ ,

如果从  $\alpha$  可能推导出空串 ( $\alpha \xRightarrow{*} \epsilon$ ),

则只有当当前词法单元  $t \in \text{FOLLOW}(A)$ , 才可以选择该产生式

## 先计算每个符号 $X$ 的 $\text{FIRST}(X)$ 集合

---

```
1: procedure  $\text{FIRST}(X)$ 
2:   if  $X \in T$  then                                ▷ 规则 1:  $X$  是终结符
3:      $\text{FIRST}(X) = X$ 
4:   for  $X \rightarrow Y_1 Y_2 \dots Y_k$  do                    ▷ 规则 2:  $X$  是非终结符
5:      $\text{FIRST}(X) \leftarrow \text{FIRST}(X) \cup \{\text{FIRST}(Y_1) \setminus \{\epsilon\}\}$ 
6:     for  $i \leftarrow 2$  to  $k$  do
7:       if  $\epsilon \in L(Y_1 \dots Y_{i-1})$  then
8:          $\text{FIRST}(X) \leftarrow \text{FIRST}(X) \cup \{\text{FIRST}(Y_i) \setminus \{\epsilon\}\}$ 
9:       if  $\epsilon \in L(Y_1 \dots Y_k)$  then                ▷ 规则 3:  $X$  可推导出空串
10:       $\text{FIRST}(X) \leftarrow \text{FIRST}(X) \cup \{\epsilon\}$ 
```

---

不断应用上面的规则, 直到每个  $\text{FIRST}(X)$  都不再变化 (闭包!!!)



再计算每个符号串  $\alpha$  的  $\text{FIRST}(\alpha)$  集合

$$\alpha = X\beta$$

$$\text{FIRST}(\alpha) = \begin{cases} \text{FIRST}(X) & \epsilon \notin L(X) \\ (\text{FIRST}(X) \setminus \{\epsilon\}) \cup \text{FIRST}(\beta) & \epsilon \in L(X) \end{cases}$$

最后, 如果  $\epsilon \in L(\alpha)$ , 则将  $\epsilon$  加入  $\text{FIRST}(\alpha)$ 。

$$(1) X \rightarrow Y$$

$$(2) X \rightarrow a$$

$$(3) Y \rightarrow \epsilon$$

$$(4) Y \rightarrow c$$

$$(5) Z \rightarrow d$$

$$(6) Z \rightarrow XYZ$$

$$\text{FIRST}(X) = \{a, c, \epsilon\}$$

$$\text{FIRST}(Y) = \{c, \epsilon\}$$

$$\text{FIRST}(Z) = \{a, c, d\}$$

$$\text{FIRST}(XYZ) = \{a, c, d\}$$

为每个非终结符  $X$  计算  $\text{FOLLOW}(X)$  集合

---

```
1: procedure FOLLOW( $X$ )
2:   for  $X$  是开始符号 do ▷ 规则 1:  $X$  是开始符号
3:      $\text{FOLLOW}(X) \leftarrow \text{FOLLOW}(X) \cup \{\$ \}$ 

4:   for  $A \rightarrow \alpha X \beta$  do ▷ 规则 2:  $X$  是某产生式右部中间的一个符号
5:      $\text{FOLLOW}(X) \leftarrow \text{FOLLOW}(X) \cup (\text{FIRST}(\beta) \setminus \{\epsilon\})$ 
6:     if  $\epsilon \in \text{FIRST}(\beta)$  then
7:        $\text{FOLLOW}(X) \leftarrow \text{FOLLOW}(X) \cup \text{FOLLOW}(A)$ 

8:   for  $A \rightarrow \alpha X$  do ▷ 规则 3:  $X$  是某产生式右部的最后一个符号
9:      $\text{FOLLOW}(X) \leftarrow \text{FOLLOW}(X) \cup \text{FOLLOW}(A)$ 
```

---

不断应用上面的规则, 直到每个  $\text{FOLLOW}(X)$  都不再变化 (闭包!!!)

$$(1) X \rightarrow Y$$

$$(2) X \rightarrow a$$

$$(3) Y \rightarrow \epsilon$$

$$(4) Y \rightarrow c$$

$$(5) Z \rightarrow d$$

$$(6) Z \rightarrow XYZ$$

$$\text{FOLLOW}(X) = \{a, c, d, \$\}$$

$$\text{FOLLOW}(Y) = \{a, c, d, \$\}$$

$$\text{FOLLOW}(Z) = \emptyset$$

如何根据FIRST 与 FOLLOW 集合计算给定文法  $G$  的预测分析表?

按照以下规则, 在表格  $[A, t]$  中填入生成式  $A \rightarrow \alpha$  (编号):

$$t \in \text{FIRST}(\alpha) \quad (1)$$

$$\alpha \xRightarrow{*} \epsilon \wedge t \in \text{FOLLOW}(A) \quad (2)$$

Definition ( $LL(1)$  文法)

如果文法  $G$  的预测分析表是无冲突的, 则  $G$  是  $LL(1)$  文法。

$$(1) X \rightarrow Y$$

$$(2) X \rightarrow a$$

$$(3) Y \rightarrow \epsilon$$

$$(4) Y \rightarrow c$$

$$(5) Z \rightarrow d$$

$$(6) Z \rightarrow XYZ$$

$$\text{FIRST}(X) = \{a, c, \epsilon\}$$

$$\text{FIRST}(Y) = \{c, \epsilon\}$$

$$\text{FIRST}(Z) = \{a, c, d\}$$

$$\text{FIRST}(XYZ) = \{a, c, d\}$$

$$\text{FOLLOW}(X) = \{a, c, d, \$\}$$

$$\text{FOLLOW}(Y) = \{a, c, d, \$\}$$

$$\text{FOLLOW}(Z) = \emptyset$$

	<i>a</i>	<i>c</i>	<i>d</i>	\$
<i>X</i>	1, 2	1	1	1
<i>Y</i>	3	3, 4	3	3
<i>Z</i>	6	6	5, 6	

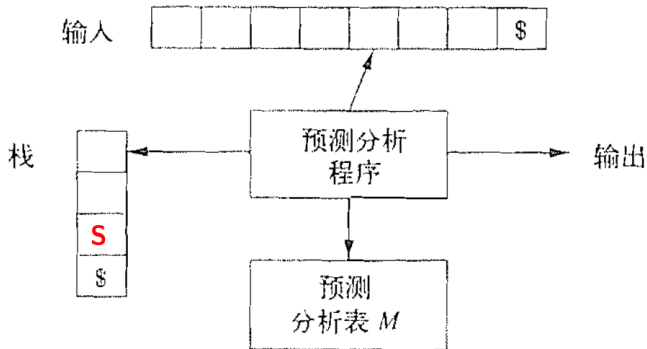
## $LL(1)$ 语法分析器

$L$ : 从左向右 (left-to-right) 扫描输入

$L$ : 构建最左 (leftmost) 推导

1: 只需向前看一个输入符号便可确定使用哪条产生式

## 非递归的预测分析算法





## 非递归的预测分析算法

设置  $ip$  使它指向  $w$  的第一个符号, 其中  $ip$  是输入指针;

令  $X$  = 栈顶符号;

while (  $X \neq \$$  ) { /\* 栈非空 \*/

    if (  $X$  等于  $ip$  所指向的符号  $a$  ) 执行栈的弹出操作, 将  $ip$  向前移动一个位置;

    else if (  $X$  是一个终结符号 )  $error()$ ;

    else if (  $M[X, a]$  是一个报错条目 )  $error()$ ;

    else if (  $M[X, a] = X \rightarrow Y_1 Y_2 \cdots Y_k$  ) {

        输出产生式  $X \rightarrow Y_1 Y_2 \cdots Y_k$ ;

        弹出栈顶符号;

        将  $Y_k, Y_{k-1}, \dots, Y_1$  压入栈中, 其中  $Y_1$  位于栈顶。

    }

    令  $X$  = 栈顶符号;

}

不是  $LL(1)$  文法怎么办?

**改造它**

消除左递归

提取左公因子

$E$  在不消耗任何词法单元的情况下, 直接递归调用  $E$ , 造成死循环

$$E \rightarrow E + T \mid E - T \mid T$$

$$T \rightarrow T * F \mid T / F \mid F$$

$$F \rightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$$

$$\text{FIRST}(E + T) \cap \text{FIRST}(T) \neq \emptyset$$

不是  $LL(1)$  文法

## 消除左递归

$$E \rightarrow E + T \mid T$$

$$E \rightarrow TE'$$

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

将左递归转为右递归

(注: 右递归对应右结合; 需要在后续阶段进行额外处理)

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots A\alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \beta_n$$

其中,  $\beta_i$  都不以  $A$  开头

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

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \epsilon$$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid \mathbf{id}$$

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

$$F \rightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$$

## 非直接左递归

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sb \mid \epsilon$$

$$S \Rightarrow Aa \Rightarrow Sda$$

- 1) 按照某个顺序将非终结符号排序为  $A_1, A_2, \dots, A_n$ .
- 2) for ( 从 1 到  $n$  的每个  $i$  ) {
- 3)     for ( 从 1 到  $i-1$  的每个  $j$  ) {
- 4)         将每个形如  $A_i \rightarrow A_j \gamma$  的产生式替换为产生式组  $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$ ,  
       其中  $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$  是所有的  $A_j$  产生式
- 5)     }
- 6)     消除  $A_i$  产生式之间的立即左递归
- 7) }

图 4-11 消除文法中的左递归的算法

$$A_k \rightarrow A_l \alpha \Rightarrow l > k$$

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sb \mid \epsilon$$

$$A \rightarrow Ac \mid Aad \mid bd \mid \epsilon$$

$$S \rightarrow Aa \mid b$$

$$A \rightarrow bdA' \mid A'$$

$$A' \rightarrow cA' \mid adA' \mid \epsilon$$

$$A_k \rightarrow A_l \alpha \implies l > k$$



	非终结符号	输入符号					
		id	+	*	(	)	\$
$E \rightarrow TE'$	$E$	$E \rightarrow TE'$			$E \rightarrow TE'$		
$E' \rightarrow +TE' \mid \epsilon$	$E'$		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
$T \rightarrow FT'$	$T$	$T \rightarrow FT'$			$T \rightarrow FT'$		
	$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
$T' \rightarrow *FT' \mid \epsilon$	$F$	$F \rightarrow \text{id}$			$F \rightarrow (E)$		
$F \rightarrow (E) \mid \text{id} \mid \text{num}$							

$$\text{FIRST}(F) = \{ (, \text{id} \}$$

$$\text{FIRST}(T) = \{ (, \text{id} \}$$

$$\text{FIRST}(E) = \{ (, \text{id} \}$$

$$\text{FIRST}(E') = \{ +, \epsilon \}$$

$$\text{FIRST}(T') = \{ *, \epsilon \}$$

$$\text{FOLLOW}(E) = \text{FOLLOW}(E') = \{ ), \$ \}$$

$$\text{FOLLOW}(T) = \text{FOLLOW}(T') = \{ +, ), \$ \}$$

$$\text{FOLLOW}(F) = \{ +, *, ), \$ \}$$

已匹配	栈	输入	动作
<b>句型</b>	$E\$$	$id + id * id\$$	
	$TE'\$$	$id + id * id\$$	输出 $E \rightarrow TE'$
	$FT'E'\$$	$id + id * id\$$	输出 $T \rightarrow FT'$
	$id T'E'\$$	$id + id * id\$$	输出 $F \rightarrow id$
$id$	$T'E'\$$	$+ id * id\$$	匹配 $id$
$id$	$E'\$$	$+ id * id\$$	输出 $T' \rightarrow \epsilon$
$id$	$+ TE'\$$	$+ id * id\$$	输出 $E' \rightarrow + TE'$
$id +$	$TE'\$$	$id * id\$$	匹配 $+$
$id +$	$FT'E'\$$	$id * id\$$	输出 $T \rightarrow FT'$
$id +$	$id T'E'\$$	$id * id\$$	输出 $F \rightarrow id$
$id + id$	$T'E'\$$	$* id\$$	匹配 $id$
$id + id$	$* FT'E'\$$	$* id\$$	输出 $T' \rightarrow * FT'$
$id + id *$	$FT'E'\$$	$id\$$	匹配 $*$
$id + id *$	$id T'E'\$$	$id\$$	输出 $F \rightarrow id$
$id + id * id$	$T'E'\$$	$\$$	匹配 $id$
$id + id * id$	$E'\$$	$\$$	输出 $T' \rightarrow \epsilon$
$id + id * id$	$\$$	$\$$	输出 $E' \rightarrow \epsilon$

图 4-21 对输入  $id + id * id$  进行预测分析时执行的步骤

$$S \rightarrow i E t S \mid i E t S e S \mid a$$

$$E \rightarrow b$$

提取左公因子

$$S \rightarrow i E t S S' \mid a$$

$$S' \rightarrow e S \mid \epsilon$$

$$E \rightarrow b$$

$$S \rightarrow iEtS \mid iEtSeS \mid a$$

$$E \rightarrow b$$

非终结符号	输入符号					
	$a$	$b$	$e$	$i$	$t$	$\$$
$S$	$S \rightarrow a$			$S \rightarrow iEtSS'$		
$S'$			$S' \rightarrow \epsilon$ $S' \rightarrow eS$			$S' \rightarrow \epsilon$
$E$		$E \rightarrow b$				

**解决二义性:** 选择  $S' \rightarrow eS$ , 将 **else** 与前面最近的 **then** 关联起来

### 语法分析阶段的主题之三: 错误恢复



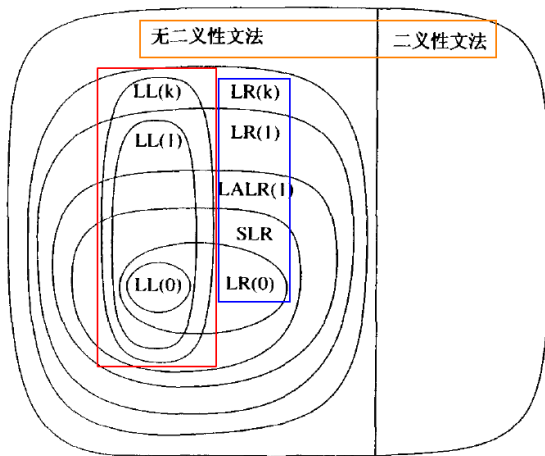
报错、恢复、继续分析





只考虑**无二义性**的文法

这意味着, 每个句子对应唯一的一棵语法分析树



今日份主题:  **$LR(1)$  ( $LR(0)$ ) 语法分析器**



自顶向下的、  
不断规约的、  
基于句柄查找自动机的、  
适用于 $LR(*)$  文法的、  
 $LR(*)$  语法分析器

自底向上构建语法分析树

根节点是文法的起始符号  $S$

每个中间非终结符节点表示使用它的某条产生式进行归约

叶节点是词法单元流  $w\$$

仅包含终结符号与特殊的文件结束符  $\$$

## 自顶向下的“推导”与 自底向上的“归约”

$$E \xRightarrow{\text{rm}} T \xRightarrow{\text{rm}} T * F \xRightarrow{\text{rm}} T * \mathbf{id} \xRightarrow{\text{rm}} F * \mathbf{id} \xRightarrow{\text{rm}} \mathbf{id} * \mathbf{id}$$

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

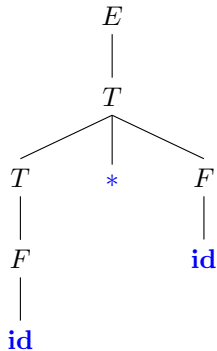
$$(2) E \rightarrow T$$

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

$$(4) T \rightarrow F$$

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

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



$$w = \mathbf{id} * \mathbf{id}$$

$$E \longleftarrow T \longleftarrow T * F \longleftarrow T * \mathbf{id} \longleftarrow F * \mathbf{id} \longleftarrow \mathbf{id} * \mathbf{id}$$

“推导” ( $A \rightarrow \alpha$ ) 与 “归约” ( $A \leftarrow \alpha$ )

$$S \triangleq \gamma_0 \Rightarrow \dots \gamma_{i-1} \Rightarrow \gamma_i \Rightarrow \gamma_{r+1} \Rightarrow \dots \Rightarrow r_n = w$$

$$S \triangleq \gamma_0 \Leftarrow \dots \gamma_{i-1} \Leftarrow \gamma_i \Leftarrow \gamma_{r+1} \Leftarrow \dots \Leftarrow r_n = w$$

自底向上语法分析器为输入构造**反向推导**

## LR 语法分析器

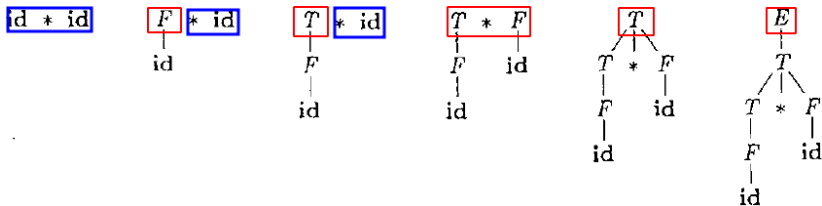
$L$ : 从左向右 (Left-to-right) 扫描输入

$R$ : 构建反向 (Reverse) 最右推导

“反向最右推导”与“从左到右扫描”相一致

## LR 语法分析器的状态

在任意时刻, 语法分析树的**上边缘**与**剩余的输入**构成当前句型



$$E \Leftarrow T \Leftarrow T * F \Leftarrow T * id \Leftarrow F * id \Leftarrow id * id$$

LR 语法分析器使用**栈**存储语法分析树的**上边缘**

它包含了语法分析器目前所知的所有信息

## 板书演示“栈”上操作

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

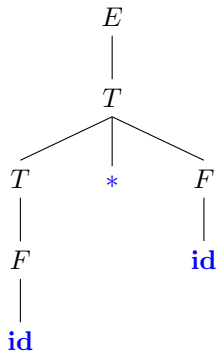
$$(2) E \rightarrow T$$

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

$$(4) T \rightarrow F$$

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

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



$w = \mathbf{id} * \mathbf{id}$

两大操作: **移入输入符号** 与 **按产生式归约**

直到栈中仅剩开始符号  $S$ , 且输入已结束, 则成功停止

## 基于栈的 $LR$ 语法分析器

$Q_1$  : 何时归约? (何时移入?)

$Q_2$  : 按哪条产生式进行规约?



## 基于栈的 LR 语法分析器

(1)  $E \rightarrow E + T$

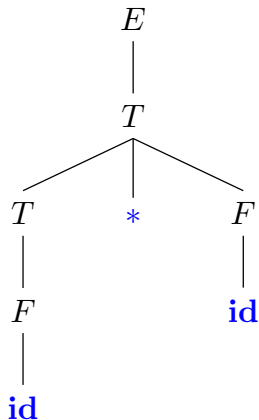
(2)  $E \rightarrow T$

(3)  $T \rightarrow T * F$

(4)  $T \rightarrow F$

(5)  $F \rightarrow (E)$

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



为什么第二个  $F$  以  $T * F$  整体被归约为  $T$ ?

这与栈的当前状态 “ $T * F$ ” 相关

## LR(0) 分析表指导 LR(0) 语法分析器

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			

在当前状态 (编号)下, 面对当前文法符号时, 该采取什么动作

ACTION 表指明动作, GOTO 表仅用于归约时的状态转换

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			

<i>sn</i>	移入输入符号, 并进入状态 <i>n</i>
<i>rk</i>	使用 <i>k</i> 号产生式进行归约
<i>gn</i>	转换到状态 <i>n</i>
<i>acc</i>	成功接受, 结束
空白	错误

## Definition ( $LR(0)$ 文法)

如果文法  $G$  的  $LR(0)$  分析表是无冲突的, 则  $G$  是  $LR(0)$  文法。

**无冲突:** ACTION 表中每个单元格最多只有一种动作

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			

**两类可能的冲突:** “移入/归约”冲突、“归约/归约”冲突

## 再次板书演示“栈”上操作: 移人与归约

(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			

$w = \text{id} * \text{id}\$$

栈中存储语法分析器的状态(编号), “编码”了语法分析树的上边缘

---

```

1: procedure LR()
2:   PUSH( $S, s_0$ )                                ▷ 或 PUSH( $S, \$_{s_0}$ )
3:   token  $\leftarrow$  NEXT-TOKEN()
4:   while (1) do
5:      $s \leftarrow$  TOP( $S$ )
6:     if ACTION[ $s, \text{token}$ ] =  $s_i$  then                ▷ 移入
7:       PUSH( $S, i$ )                                ▷ 或 PUSH( $S, \text{token}_{s_i}$ )
8:       token  $\leftarrow$  NEXT-TOKEN()
9:     else if ACTION[ $s, \text{token}$ ] =  $r_j$  then            ▷ 规约;  $j : A \rightarrow \alpha$ 
10:       $|\alpha|$  次 POP( $S$ )
11:       $s \leftarrow$  TOP( $S$ )
12:      PUSH( $S, \text{GOTO}[s, A]$ ) ▷ 转换状态; 或 PUSH( $S, A_{\text{GOTO}[s, A]}$ )
13:    else if ACTION[ $s, \text{token}$ ] =  $acc$  then            ▷ 接受
14:      break
15:    else
16:      ERROR(...)

```

---

行号	栈	=	符号	输入	动作
(1)	0	\$		id * id \$	移入到 5
(2)	0 5	\$ id		* id \$	按照 $F \rightarrow id$ 归约
(3)	0 3	\$ F		* id \$	按照 $T \rightarrow F$ 归约
(4)	0 2	\$ T		* id \$	移入到 7
(5)	0 2 7	\$ T *		id \$	移入到 5
(6)	0 2 7 5	\$ T * id		\$	按照 $F \rightarrow id$ 归约
(7)	0 2 7 10	\$ T * F		\$	按照 $T \rightarrow T * F$ 归约
(8)	0 2	\$ T		\$	按照 $E \rightarrow T$ 归约
(9)	0 1	\$ E		\$	接受

$w = id * id\$$  的分析过程

## 如何构造 $LR(0)$ 分析表?

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			

在当前状态 (编号)下, 面对当前文法符号时, 该采取什么动作



## 状态是什么？如何跟踪状态？

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			

状态是语法分析树的上边缘, 存储在栈中

可以用**自动机**跟踪状态变化 (自动机中的路径  $\Leftrightarrow$  栈中符号/状态编号)

## 何时归约？使用哪条产生式进行归约？

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			

**必要条件:** 当前状态中, 已观察到某个产生式的完整右部

对于  $LR(0)$  文法, 这是当前**唯一**的选择

## 何时归约？使用哪条产生式进行归约？

### Definition (句柄 (Handle))

在输入串的 (唯一) 反向最右推导中, **如果** 下一步是逆用产生式  $A \rightarrow \alpha$  将  $\alpha$  规约为  $A$ , 则称  $\alpha$  是**当前句型的句柄**。

最右句型	句柄	归约用的产生式
$\boxed{id_1} * id_2$	$id_1$	$F \rightarrow id$
$\boxed{F} * id_2$	$F$	$T \rightarrow F$
$T * \boxed{id_2}$	$id_2$	$F \rightarrow id$
$\boxed{T * F}$	$T * F$	$T \rightarrow T * F$
$\boxed{T}$	$T$	$E \rightarrow T$

$LR$  语法分析器的关键就是高效**寻找每个归约步骤所使用的句柄**。

## 句柄可能在哪里？

### Theorem

存在一种 LR 语法分析方法，保证句柄总是出现在栈顶。

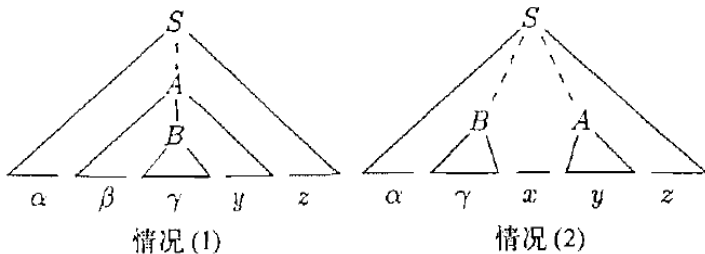


图 4-29 一个最右推导中两个连续步骤的两种情况

$$S \xrightarrow{*}_{\text{rm}} \alpha Az \xrightarrow{*}_{\text{rm}} \alpha \beta B y z \xrightarrow{*}_{\text{rm}} \alpha \beta \gamma y z \quad S \xrightarrow{*}_{\text{rm}} \alpha B x A z \xrightarrow{*}_{\text{rm}} \alpha B x y z \xrightarrow{*}_{\text{rm}} \alpha \gamma x y z$$

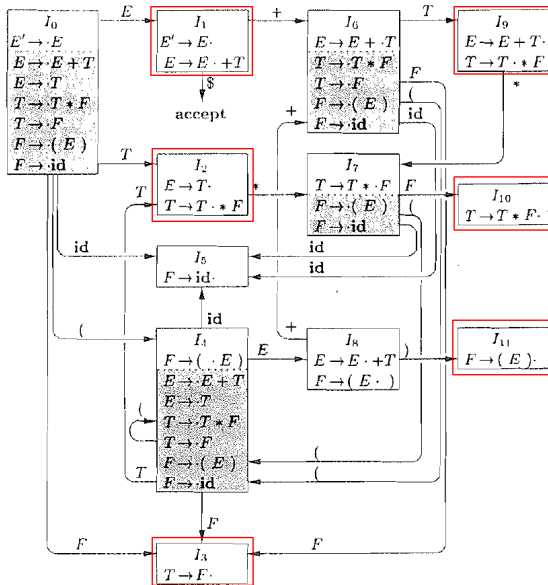
可以用**自动机**跟踪状态变化  
(自动机中的路径  $\Leftrightarrow$  栈中符号/状态编号)

### Theorem

**存在**一种  $LR$  语法分析方法, 保证**句柄总是出现在栈顶**。

在自动机的当前状态识别可能的句柄 (观察到的**完整右部**)  
(自动机的当前状态  $\Leftrightarrow$  栈顶)

# LR(0) 句柄识别有穷状态自动机 (Handle-Finding Automaton)

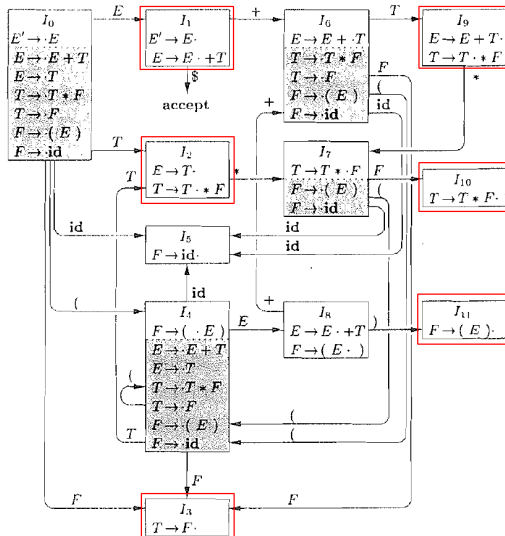


## $LR(0)$ 句柄识别自动机

为给定的文法  $G$  构造相应的句柄识别自动机

该自动机用于识别该文法  $G$  所允许的所有可能的句柄

## LR(0) 句柄识别自动机



状态是什么？状态之间如何转移？



**状态**刻画了“当前观察到的**针对所有产生式的右部的前缀**”

Definition ( $LR(0)$  项 (Item))

文法  $G$  的一个  $LR(0)$  **项**是  $G$  的某个产生式加上一个位于体部的**点**。

$$A \rightarrow XYZ$$

$$A \rightarrow \cdot XYZ$$

$$A \rightarrow X \cdot YZ$$

$$A \rightarrow XY \cdot Z$$

$$A \rightarrow XYZ \cdot$$

(产生式  $A \in$  只有一个项  $A \rightarrow \cdot$ )

**项**指明了语法分析器已经观察到了某个产生式的哪些部分

**点**指示了**栈顶**, 左边 (与路径) 是栈中内容, 右边是期望看到的文法符号

**状态**刻画了“当前观察到的**针对所有产生式的右部的前缀**”

Definition (项集)

**项集**就是若干**项**构成的集合。

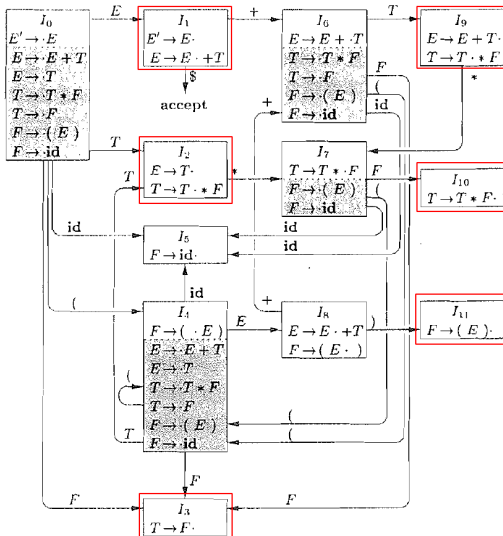
因此, 句柄识别自动机的一个**状态**可以表示为一个**项集**

Definition (项集族)

**项集族**就是若干**项集**构成的集合。

因此, 句柄识别自动机的**状态集**可以表示为一个**项集族**

## LR(0) 句柄识别自动机



## 项、项集、项集族

## Definition (增广文法 (Augmented Grammar))

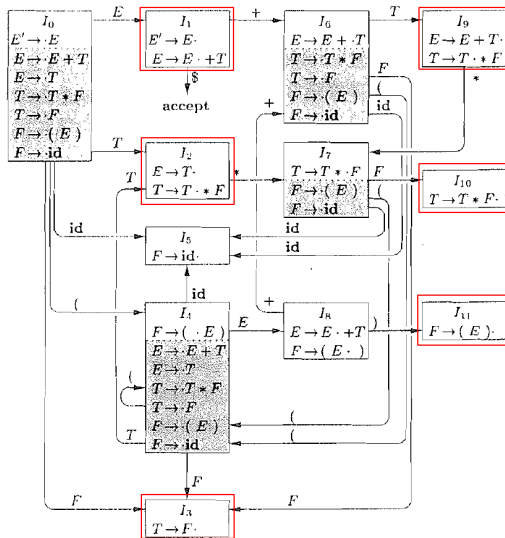
文法  $G$  的**增广文法**是在  $G$  中加入产生式  $S' \rightarrow S$  得到的文法。

**目的:** 告诉语法分析器何时停止分析并接受输入符号串

当语法分析器**面对  $\$$  且要使用  $S' \rightarrow S$  进行归约**时, 输入符号串被接受

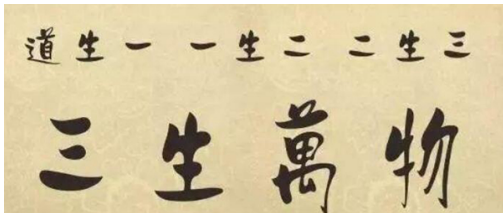
注: 此“接受”(输入串) 非彼“接受”(句柄识别自动机)

## LR(0) 句柄识别自动机



注：此“接受”（输入串）非彼“接受”（句柄识别自动机）

## $LR(0)$ 句柄识别自动机



初始状态是什么？

状态之间如何转移？

点指示了栈顶, 左边 (与路径) 是栈中内容, 右边是期望看到的文法符号

$$(0) E' \rightarrow E$$

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

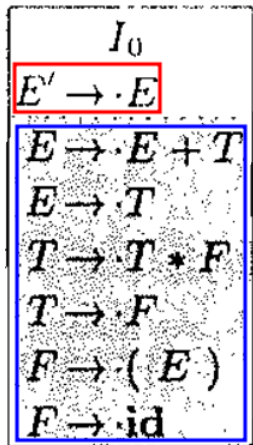
$$(2) E \rightarrow T$$

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

$$(4) T \rightarrow F$$

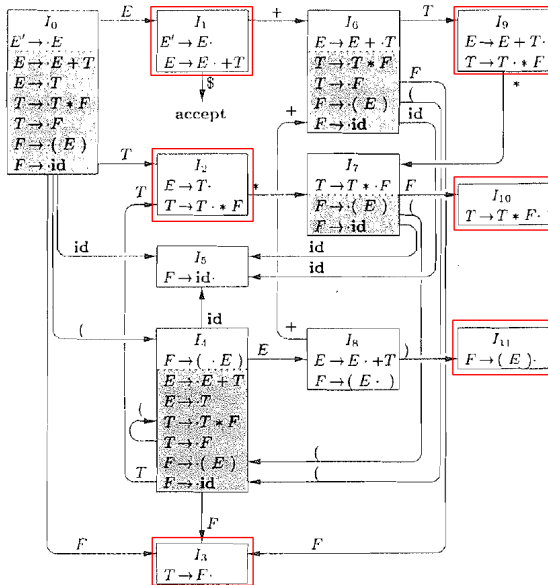
$$(5) F \rightarrow (E)$$

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



$\text{CLOSURE}(\{[E' \rightarrow \cdot E]\})$

## 板书演示 $LR(0)$ 句柄识别自动机的构造过程





```

SetOfItems CLOSURE(I) {
    J = I;
    repeat
        for (J中的每个项  $A \rightarrow \alpha \cdot B \beta$ )
            for (G 的每个产生式  $B \rightarrow \gamma$ )
                if (项  $B \rightarrow \cdot \gamma$  不在 J 中)
                    将  $B \rightarrow \cdot \gamma$  加入 J 中;
    until 在某一轮中没有新的项被加入到 J 中;
    return J;
}

```

$$\text{GOTO}(I, X) = \text{CLOSURE}\left(\left\{ [A \rightarrow \alpha X \cdot \beta] \mid [A \rightarrow \alpha \cdot X \beta] \in I \right\}\right)$$

$$(X \in N \cup T \cup \{\$, \})$$

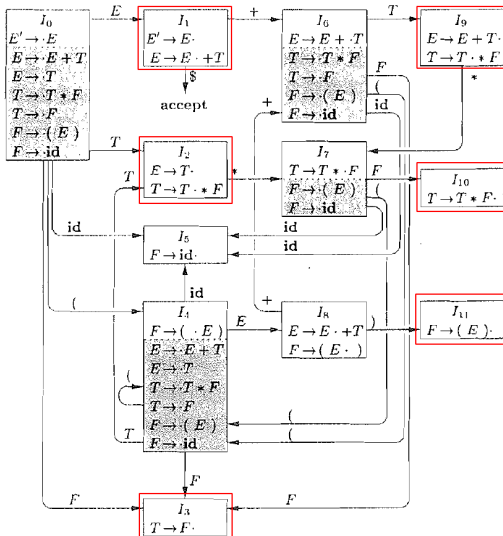
```

void items( $G'$ ) {
     $C = \{ \text{CLOSURE}(\{[S' \rightarrow \cdot S]\}) \}$ ; 初始状态
    repeat
        for (  $C$  中的每个项集  $I$  )
            for ( 每个文法符号  $X$  )
                if (  $\text{GOTO}(I, X)$  非空且不在  $C$  中 )
                    下一个状态 将  $\text{GOTO}(I, X)$  加入  $C$  中;
    until 在某一轮中没有新的项集被加入到  $C$  中;
}

```

图 4-33 **规范 LR(0) 项集族** 的计算

## LR(0) 句柄识别自动机



Q: 为什么是个有穷状态自动机?

Thank  
You!



Office 926

hfwei@nju.edu.cn