



语法分析

——自上向下





第4章 语法分析(Syntax Analysis)

—— 自上而下分析法

4.1 语法分析综述



4.2 不确定的自上而下分析法

4.3 LL(1)分析法与LL(1)分析器

4.4 递归下降分析法与递归下降分析器



4.1 语法分析综述

4.1.1 语法分析程序的功能



4.1.2 语法分析方法

- 一. 自上而下分析
- 二. 自下而上分析



■ 语法分析程序的功能

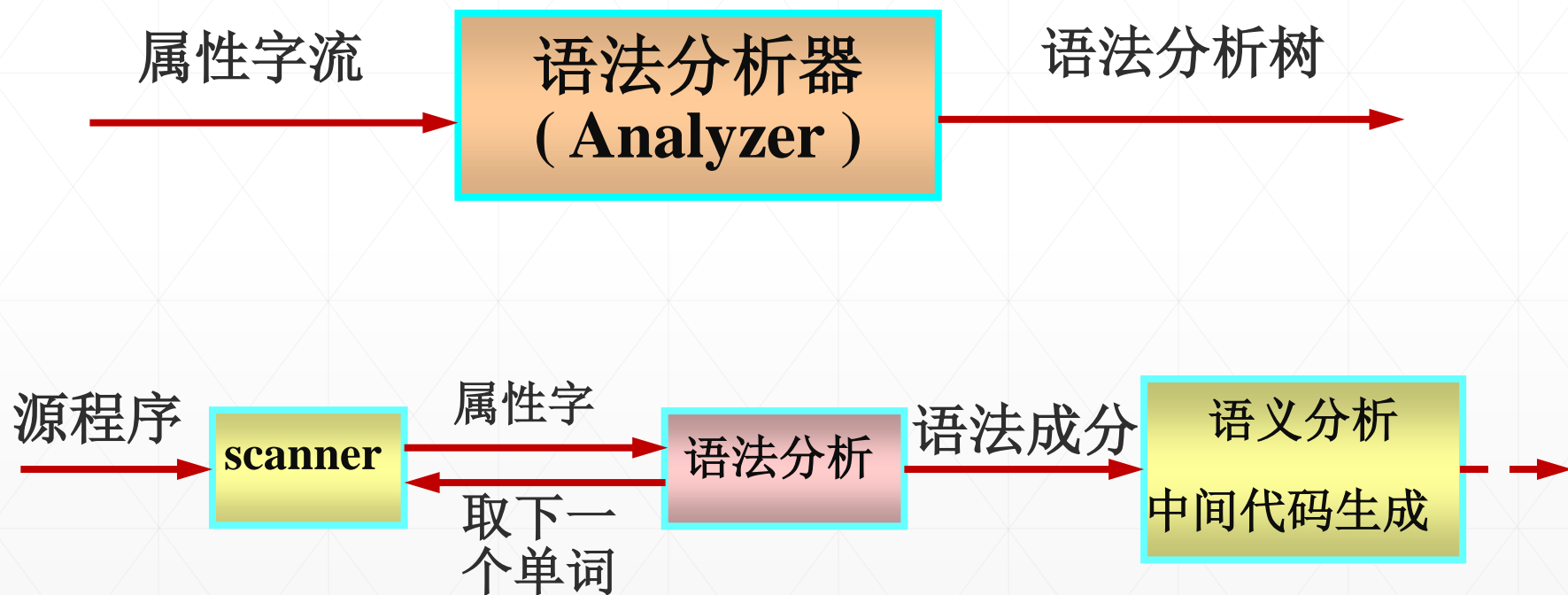
完成语法分析任务的程序称为语法分析器，或语法分析程序。

按照源语言的语法规则，对词法分析的结果(与源程序等价的属性字流) 进行语法检查，并识别出相应的语法成分。

给定文法 G 和字符串 α ($\alpha \in V_T^*$)，检查、判定 $\alpha \in L(G)$? 同时报告和处理语法错误。



■ 语法分析器在编译程序中的位置





■ 语法分析程序的构造要素

源程序串 (属性字流) → 处理对象

源语言的语法规则G → 分析依据

识别出的语法范畴的表示(语法分析树) → 分析结果



4.1 语法分析综述

4.1.1 语法分析程序的功能

4.1.2 语法分析方法

- 一. 自上而下分析
- 二. 自下而上分析





一. 自上而下语法分析方法

给定文法 G 和源程序串 r 。

从 G 的开始符号 S 出发，

反复使用产生式

对句型中的非终结符进行替换(推导)，

逐步推导出 r 。



例: 设有文法 G 和输入串 r

$$G: S \rightarrow aA$$

$$A \rightarrow BaA \mid \varepsilon$$

$$B \rightarrow + \mid - \mid * \mid ,$$

$$r: a*a+a$$

表示方式1(推导):

$$S \Rightarrow \underline{aA} \Rightarrow a\underline{BaA} \Rightarrow a_*aA \Rightarrow a*a\underline{BaA}$$

$$\Rightarrow a*a\underline{+aA} \Rightarrow a*a+a = r \in L(G)$$



分析树的从左到右叶结点= r , 则 $r \in L(G)$ 。

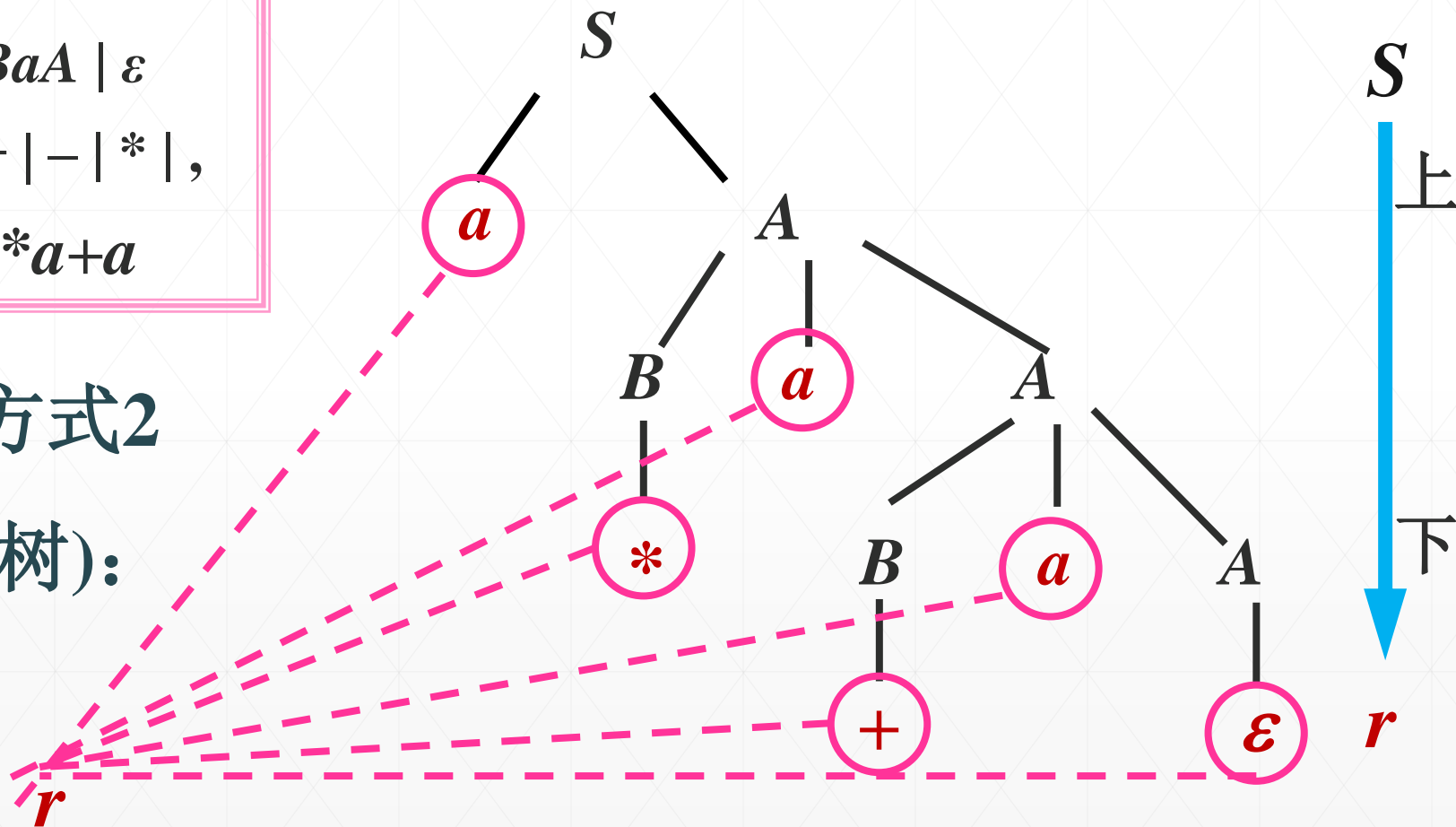
$S \rightarrow aA$

$A \rightarrow BaA \mid \varepsilon$

$B \rightarrow + \mid - \mid * \mid ,$

$r: a*a+a$

表示方式2
(分析树):





自上而下语法分析方法

- * 是一种产生的方法，面向目标的方法。
- * 分析的主旨是选择产生式的合适的候选式进行推导，逐步使推导结果与 r 匹配。



4.1 语法分析综述

4.1.1 语法分析程序的功能

4.1.2 语法分析方法

一. 自上而下分析

二. 自下而上分析





二. 自下而上语法分析方法

从给定的输入串 r 开始,
不断寻找子串
与文法 G 中某个产生式 P 的候选式进行匹配,
并用 P 的左部代替(归约)之,
逐步归约到开始符号 S 。



例:设有文法 G 和输入串 r

$G: S \rightarrow aA$

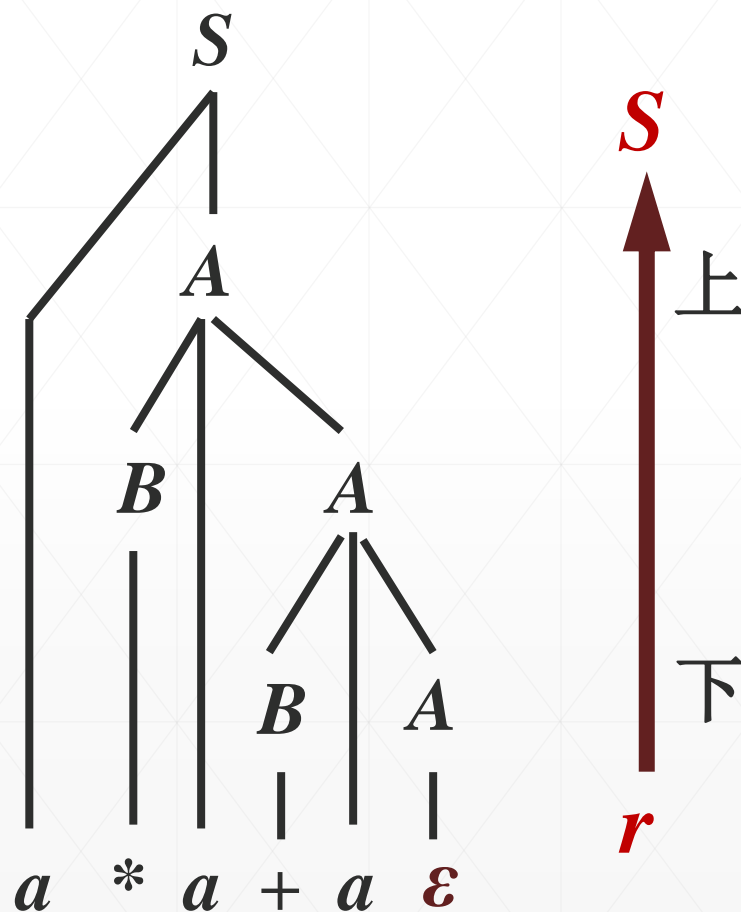
$A \rightarrow BaA \mid \varepsilon$

$B \rightarrow + \mid - \mid * \mid ,$

$r: a*a+a$

表示方式1

(分析树):





例:设有文法 G 和输入串 r

$G: S \rightarrow aA$

$A \rightarrow BaA \mid \varepsilon$

$B \rightarrow + \mid - \mid * \mid ,$

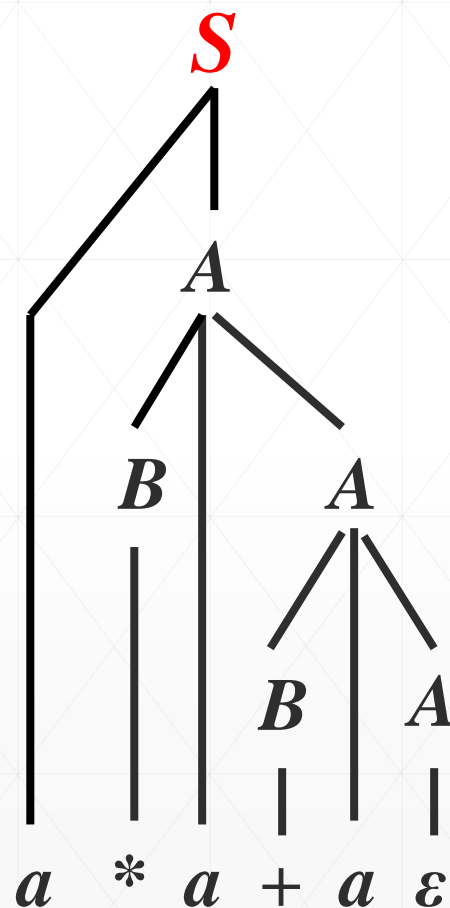
$r: a*a+a$

表示方式2(归约):

$r = a*a+a \leq aBa+a$

$\leq aBaBa\varepsilon \leq aBaBaA$

$\leq aBaA \leq aA \leq S$





自下而上语法分析方法

- * 是一种辨认的方法，基于目标的方法。
- * 分析的主旨是寻找合适的子串与 P 的候选式进行匹配，直到归约到 G 的 S 为止。

自上而下语法分析方法

- * 是一种产生的方法，面向目标的方法。
- * 分析的主旨是选择产生式的合适的候选式进行推导，逐步使推导结果与 r 匹配。



说明：

1. 语法分析两大类方法：

自上而下分析法： $S \longrightarrow r$ (一般、递归下降、LL(1))

自下而上分析法： $r \xrightarrow{\text{Reduce}} S$ (一般S-R、OPG、LR)

2. 自上而下分析法的核心是不断寻找合适候选式对句型中最左的非终结符进行替换的过程；

3. 自下而上分析法的核心是不断寻找可归约串与 P 的候选式匹配，并用 P 的左部的非终结符代替之。



第 4 章 语法分析(Syntax Analysis)

—— 自上而下分析法

4.1 语法分析综述

4.2 不确定的自上而下分析法



4.3 LL(1)分析法与LL(1)分析器

4.4 递归下降分析法与递归下降分析器



4.2 不确定的自上而下分析法

4.2.1 一般自上而下分析



4.2.2 不确定性的原因及解决方法





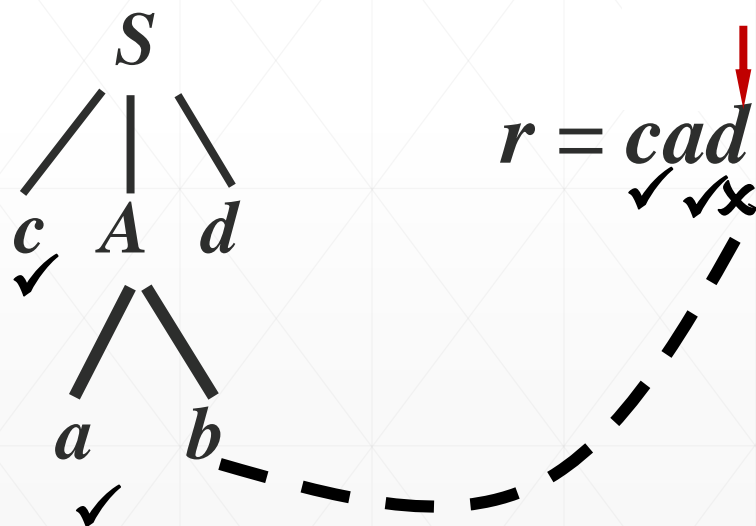
例：设有如下文法 G 和字符串 $r = cad$

$$(1) S \rightarrow cAd$$

$$(2) A \rightarrow ab \mid a$$

Step1:

Step2:



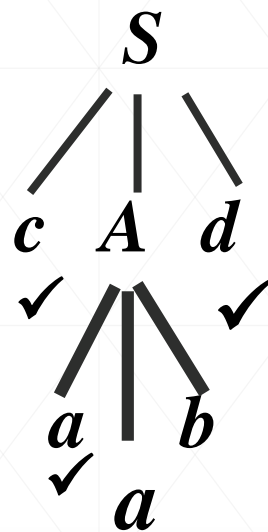


(1) $S \rightarrow cAd$

(2) $A \rightarrow ab \mid a$

Step3:

Step4:



$r = cad$

分析的本质是一种带回溯的自上而下分析，
是一试探推导的过程，
反复使用不同的产生式谋求匹配输入串，
算法效率低开销大。

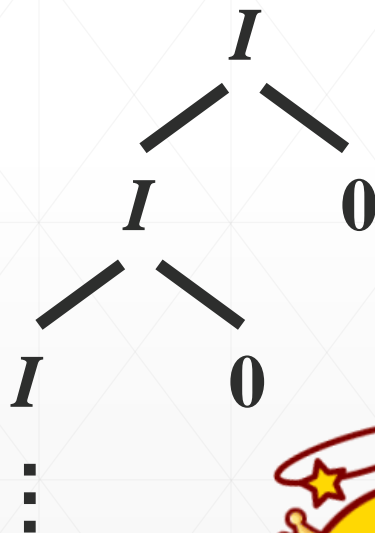


例：设有文法 G 和输入字符串 r

$G: I \rightarrow I0 \mid Ia \mid a$

$r: a00$

$L(G) = a(a|0)^*$



按照自上而下分析法对输入串 r 产生分析树，
对非终结符的替换使分析树无休止的延伸，
自上而下分析陷入死循环





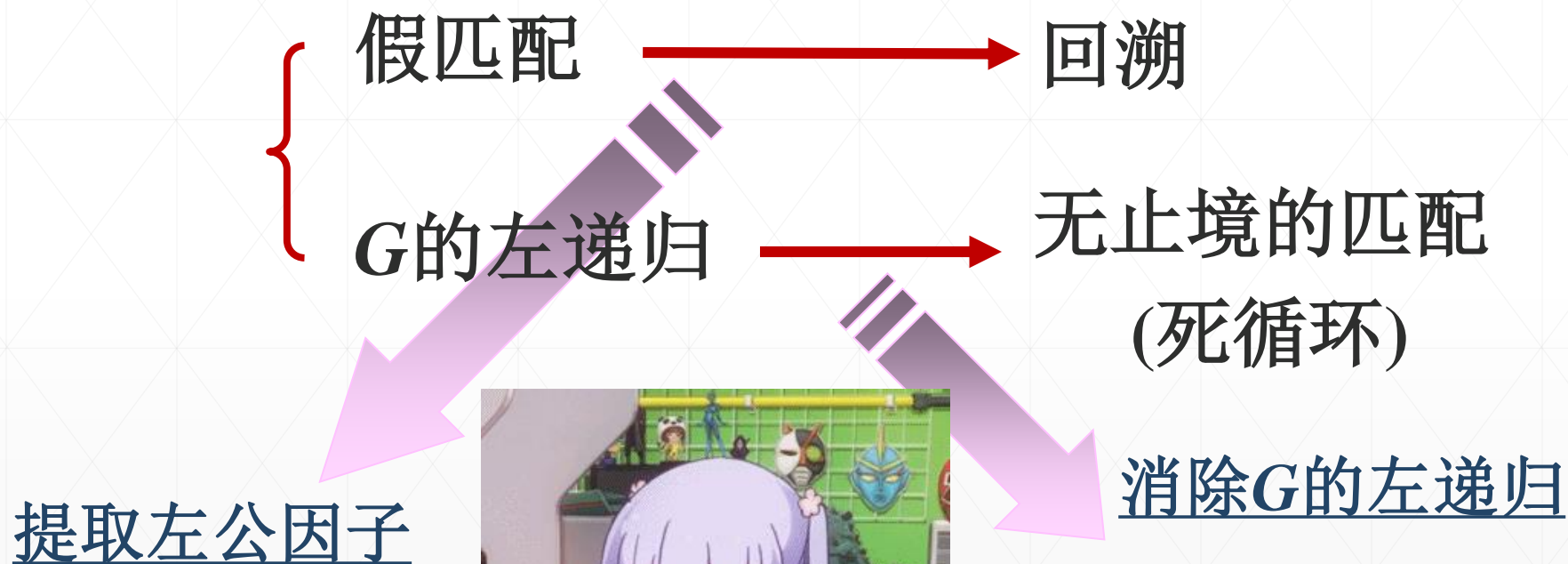
4.2 不确定的自上而下分析法

4.2.1 一般自上而下分析

4.2.2 不确定性的原因及解决方法



不确定性的原因





一. 消除文法的左递归

直接左递归

$$A \rightarrow A\alpha \quad (\alpha \in (V_T \cup V_N)^*)$$

在语法分析的最左推导中会呈现

$A \Rightarrow A \dots$ 的形式,

间接左递归文法会呈现

$A \stackrel{+}{\Rightarrow} A \dots$ 的形式。



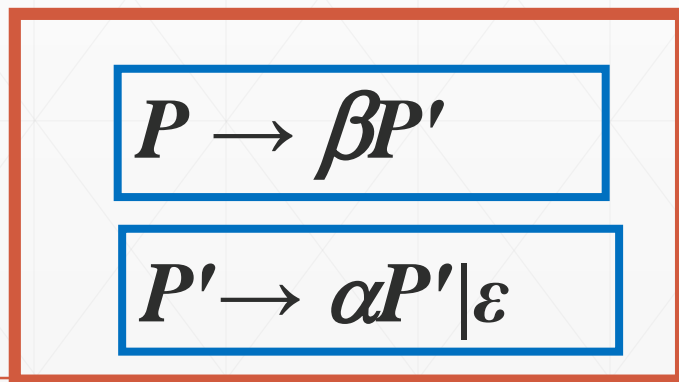
(1) 直接左递归的消除:

假定关于非终结符 P 的规则为

$$P \rightarrow P\alpha \mid \beta \quad \alpha, \beta \in (V_T \cup V_N)^*$$

其中, α 不等于 ε , β 不以 P 开头。

分析: $P \rightarrow P\alpha \mid \beta \quad \longrightarrow \quad L(G(P)) = \{\beta \alpha^*\}$



不变

β

个数可以无穷多个,
只能用 递归表示且
不能 用左递归。

P'



(1) 直接左递归的消除:

假定关于非终结符 P 的规则为

$$P \rightarrow P\alpha \mid \beta \quad \alpha, \beta \in (V_T \cup V_N)^*$$

其中, α 不等于 ε , β 不以 P 开头。

把 P 的规则改写成如下 等价的非直接左递归形式

$$P \rightarrow \beta P'$$

$$P' \rightarrow \alpha P' \mid \varepsilon$$

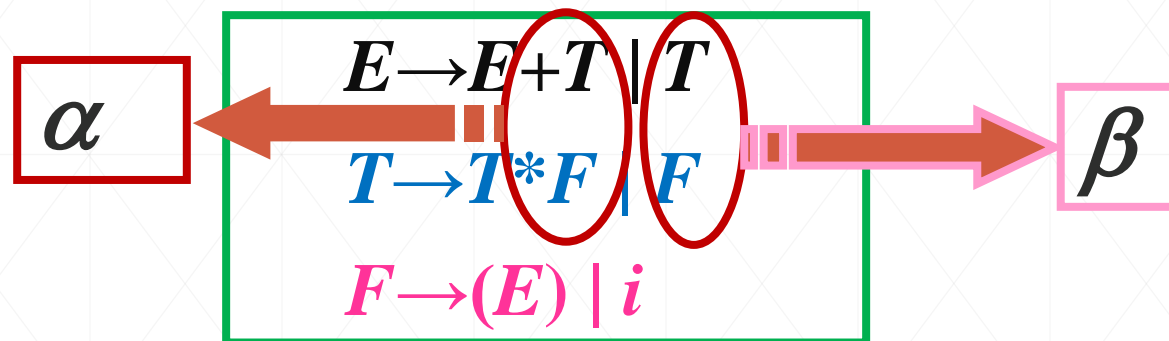




例：设有简单表达式文法 $G(E)$ ：

$$E \rightarrow E + E \mid E * E \mid (E) \mid i$$

对 $G(E)$ 消除二义性后，得到文法 $G'(E)$ ：



继续消除文法 $G'(E)$ 的左递归，得到文法 $G''(E)$

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +TE' \mid \varepsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow *FT' \mid \varepsilon \\
 F &\rightarrow (E) \mid i
 \end{aligned}$$



假定关于非终结符 P 的规则为

$$P \rightarrow P\alpha_1 | P\alpha_2 | \dots | P\alpha_n | \beta_1 | \beta_2 | \dots | \beta_m$$

其中:每个 $\alpha_i (i=1, \dots, n)$ 不等于 ε , $\beta_j (j=1, \dots, m)$ 不以 P 开头。

P 的规则可改写成如下等价的非直接左递归形式

$$P \rightarrow \beta_1 P' | \beta_2 P' | \dots | \beta_m P'$$

$$P' \rightarrow \alpha_1 P' | \alpha_2 P' | \dots | \alpha_n P' | \varepsilon$$



例:设有文法 G :

$$I \rightarrow I0 \mid Ia \mid Ib \mid a \mid b$$

消除 G 的左递归, 得到的等价文法 G' 为

$$I \rightarrow aI' \mid bI'$$

$$I' \rightarrow 0I' \mid aI' \mid bI' \mid \varepsilon$$



(2) 间接左递归的消除:

有些文法表面上不具有左递归性，却隐含着左递归。例如设有文法 $G(A)$:

$$A \rightarrow Ba \mid a$$

$$B \rightarrow Cb \mid b$$

$$C \rightarrow Ac \mid c$$

经若干步推导替换，有:

$$A \Rightarrow Ba \Rightarrow Cba \Rightarrow Acba$$

$$B \Rightarrow Cb \Rightarrow Acb \Rightarrow Bacb$$

$$C \Rightarrow Ac \Rightarrow Bac \Rightarrow Cbac$$



■ 消除间接左递归的方法:

- (1) 把间接左递归文法改写为直接左递归文法;
- (2) 用消除直接左递归的方法改写文法。

后面给出一个消除文法所有左递归性的算法，
算法对文法的要求：

1. 文法不含回路（形如 $P \xRightarrow{+} P$ 的推导）；
2. 不含以 ε 为右部的产生式。



思考题：

任何一文法都等价于一个不含回路且不含以 ε 为右部的产生式的文法。



假设同学小明所在的班级为07111603，学号为1120161684，则小明的吉祥数就是包含子串吉祥种子3684的数字串，即班级的最后一位数字3连接上学号的最后三位数字684为他的吉祥种子，小明的吉祥数的正规式表示为

$(0|1|2|3|4|5|6|7|8|9)^*3684(0|1|2|3|4|5|6|7|8|9)^*$ ，而不包含吉祥种子子串的数字串叫非吉祥数。

- (1) 给出你自己的吉祥数正规式表示；（3分）
- (2) 给出识别非吉祥数的DFA。（5分）
- (3) 写出非吉祥数的3型文法描述或正规式描述（要求给出求解过程）。（7分）



算法：(消除文法左递归)

给定文法 G

①对文法 G 的所有非终结符按任一种顺序排列，

例如 A_1, A_2, \dots, A_n 。

消除 A_1 中的直接左递归。

②for (i=2; i=n; i++)

{for (j=1; j = i-1; j++)

$\gamma \in (V_N \cup V_T)^*$

{

把形如 $A_i \rightarrow A_j \gamma$ 的产生式改写成

$$A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$$

其中 $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$ 是关于 A_j 的全部规则；

}

消除 A_i 规则中的直接左递归； }

③简化由②所得的文法，即去掉多余的规则,还原没做变化的产生式。





令文法 $G(A)$ 的非终结符排序为 C, B, A 。

对于 C ，不存在直接左递归，

对于 B ，产生式变换为

$$B \rightarrow Acb \mid dAb \mid c$$

不含直接左递归，

对于 A ，产生式变换为：

$$A \rightarrow Acba \mid dAba \mid ca \mid bB$$

A 存在直接左递归，消除 A 的直接左递归

$$A \rightarrow dAbaA' \mid caA' \mid bBA'$$

$$A' \rightarrow cbaA' \mid \varepsilon$$

文法 $G(A)$ 改写为：

$$A \rightarrow dAbaA' \mid caA' \mid bBA'$$

$$A' \rightarrow cbaA' \mid \varepsilon$$

$$B \rightarrow Cb \mid c$$

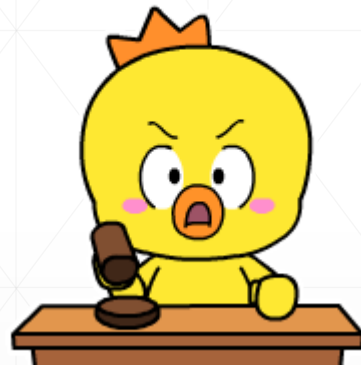
$$C \rightarrow Ac \mid dA$$

$$G(A): A \rightarrow Ba \mid bB$$

$$B \rightarrow Cb \mid c$$

$$C \rightarrow Ac \mid dA$$

注意啦！



温馨提示
开始符号的
规则位置



令文法 $G(A)$ 的非终结符排序为 A, B, C 。

对于 A ，不存在直接左递归，

对于 B ，不存在直接左递归。

对于 C

将 A 带入 C ，代换后的 C 的规则为：

$$C \rightarrow Bac \mid bBc \mid dA$$

将 B 带入 C ，代换后的 C 的规则为：

$$C \rightarrow Cbac \mid cac \mid bBc \mid dA$$

C 存在直接左递归，

消除 C 的直接左递归

$$C \rightarrow cacC' \mid bBcC' \mid dAC'$$

$$C' \rightarrow bacC' \mid \varepsilon$$

$$G(A): A \rightarrow Ba \mid bB$$

$$B \rightarrow Cb \mid c$$

$$C \rightarrow Ac \mid dA$$

文法 $G(A)$ 改写为

$$A \rightarrow Ba \mid bB$$

$$B \rightarrow Cb \mid c$$

$$C \rightarrow cacC' \mid bBcC' \mid dAC'$$

$$C' \rightarrow bacC' \mid \varepsilon$$



二. 消除回溯

$$A \rightarrow \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_n \quad \text{当前 } r: \dots a_i \dots$$

在一般自上而下分析中，
对于一个 V_N 进行替换并试图去匹配句子剩余符号时，
若 V_N 含有两个或两个以上的候选式，
是依次一个一个地去试探，
试图找出一个合乎要求的 γ_i 。
先选 γ_1 ，与当前输入 a_i 匹配成功则替换，
否则选 γ_2 ，依此类推。



■ 定义

设 G 是二型文法，则 G 中的任意 $\gamma \in V^*$ 的
终结首符集**FIRST**(γ)为

$$\text{FIRST}(\gamma) = \{ a \mid \gamma \xRightarrow{*} a \dots, a \in V_T \}$$

若 $\gamma \xRightarrow{*} \varepsilon$ ，则 $\varepsilon \in \text{FIRST}(\gamma)$ 。

不带回溯的条件：(充分非必要)

任意的含多个候选式的非终结符 A 的产生式设为：

$$A \rightarrow \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_n$$

每个候选式 γ_i 均不存在 $\gamma_i \xRightarrow{*} \varepsilon$ ，

且**FIRST**(γ_i)**两两**彼此互不相交。



对文法 G 的任意非终结符 A 的产生式设为:

$$A \rightarrow \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_n$$

满足不带回溯条件。

据当前扫描的单词 a , 若 $a \in \text{FIRST}(\gamma_i)$,

其中 γ_i 是 $\gamma_1 \dots \gamma_n$ 中之一,

选取 $A \rightarrow \gamma_i$ 进行推导是唯一的替换方式。



计算FIRST(X)($X \in V$)的算法描述:

为构造FIRST(X), 可反复应用如下规则:

- 1、若 X 是终结符,则FIRST(X)= $\{X\}$;
- 2、若 X 是非终结符, X 的FIRST为其所有候选式的FIRST集合的并集。



计算 $\text{FIRST}(\alpha)$ ($\alpha \in V^*$)的算法描述:

设 $\alpha = X_1 X_2 \dots X_k$ ($X_i \in V$)

为构造 $\text{FIRST}(\alpha)$, 可反复应用如下规则:

- 1、 $\text{FIRST}(X_1)$ 中的所有**终结符号**加到 $\text{FIRST}(\alpha)$ 中;
- 2、若 $X_1 X_2 \dots X_{i-1} \xRightarrow{*} \varepsilon$, 则将 $\text{FIRST}(X_i)$ 中的所有**终结符号**加到 $\text{FIRST}(\alpha)$ 中;
- 3、若 $X_1 X_2 \dots X_k \xRightarrow{*} \varepsilon$, 则将 ε 加到 $\text{FIRST}(\alpha)$ 中。

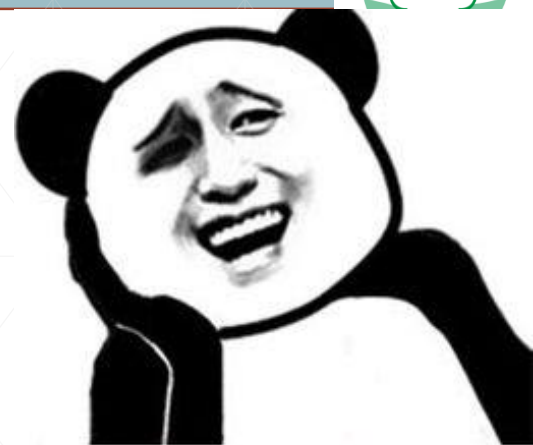
不包括 ε

例：文法 $G(S)$: $S \rightarrow aABbcd \mid \varepsilon$

$A \rightarrow Asd \mid \varepsilon$

$B \rightarrow eC \mid Sah \mid \varepsilon$

$C \rightarrow Sf \mid Cg \mid \varepsilon$



有耐心！很可爱！

计算上述文法中的每个**非终结符**的FIRST集合。

$\text{FIRST}(S) = \text{FIRST}(aABbcd) \cup \text{FIRST}(\varepsilon) = \{a, \varepsilon\}$

$\text{FIRST}(aABbcd) = \text{FIRST}\{a\} = \{a\}$

$\text{FIRST}(A) = \text{FIRST}(Asd) \cup \text{FIRST}(\varepsilon) = \{s, \varepsilon\}$

$\text{FIRST}(Asd) = (\text{FIRST}\{A\} - \{\varepsilon\}) \cup \text{FIRST}(s) = \{s\}$

$\text{FIRST}(B) = \text{FIRST}(eC) \cup \text{FIRST}(Sah) \cup \text{FIRST}(\varepsilon) = \{e, a, \varepsilon\}$

$\text{FIRST}(C) = \text{FIRST}(Sf) \cup \text{FIRST}(Cg) \cup \text{FIRST}(\varepsilon) = \{a, f, g, \varepsilon\}$



例:设有文法 G :

$$S \rightarrow Ap \mid Bq$$

$$A \rightarrow a \mid cA$$

$$B \rightarrow b \mid dB$$

对 S 的候选式:

$$\text{FIRST}(Ap) = \{a, c\} \quad \text{FIRST}(Bq) = \{b, d\}$$

$$\text{FIRST}(Ap) \cap \text{FIRST}(Bq) = \Phi$$

对 A 的候选式:

$$\text{FIRST}(a) = \{a\} \quad \text{FIRST}(cA) = \{c\}$$

$$\text{FIRST}(a) \cap \text{FIRST}(cA) = \Phi$$

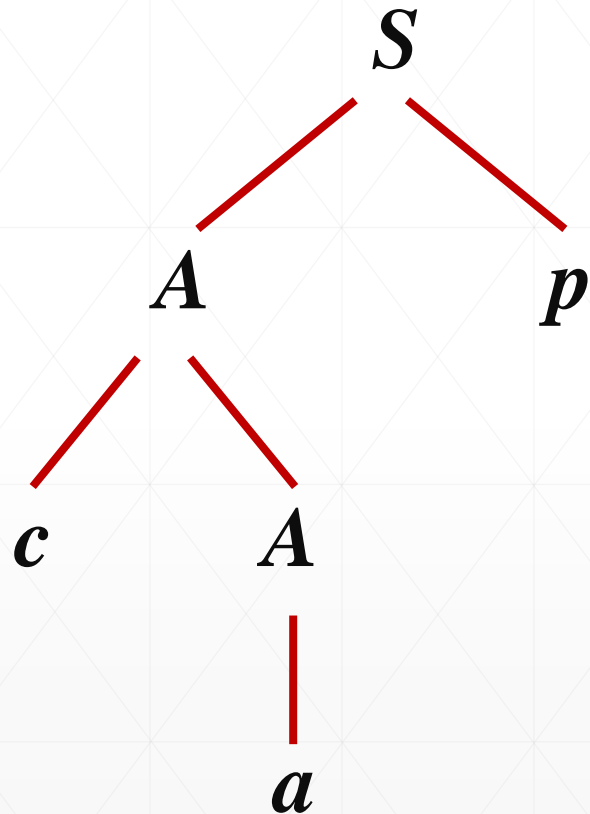
对 B 的候选式:

$$\text{FIRST}(b) = \{b\} \quad \text{FIRST}(dB) = \{d\}$$

$$\text{FIRST}(b) \cap \text{FIRST}(dB) = \Phi$$



若给出 $r=cap$,则有:



$r: \downarrow cap$

$r: \downarrow \text{c}ap$

$r: \text{c} \downarrow ap$

$$S \rightarrow Ap \mid Bq$$
$$A \rightarrow a \mid cA$$
$$B \rightarrow b \mid dB$$
$$\text{FIRST}(Ap) = \{a, c\}$$
$$\text{FIRST}(Bq) = \{b, d\}$$
$$\text{FIRST}(a) = \{a\}$$
$$\text{FIRST}(cA) = \{c\}$$
$$\text{FIRST}(b) = \{b\}$$
$$\text{FIRST}(dB) = \{d\}$$



对非终结符A的多个候选式的 $\text{FIRST}(\alpha_i)$ 的相互两个彼此交集 $\neq \Phi$ ，一般是因为 α_i 中有公共左因子，可以通过提取左公因子来改造文法。(由BNF范式改EBNF范式)

若有文法G:

$$A \rightarrow \delta\beta_1 \mid \delta\beta_2 \mid \dots \mid \delta\beta_n$$
$$\delta(\beta_1 \mid \beta_2 \mid \dots \mid \beta_n)$$

等价改写文法G为G':

$$A \rightarrow \delta A'$$

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



若文法 G 为:

$$A \rightarrow \underbrace{\delta_1 \beta_1 \mid \delta_1 \beta_2 \mid \dots \mid \delta_1 \beta_n}_{\delta_1(\beta_1 \mid \beta_2 \mid \dots \mid \beta_n)} \underbrace{\delta_2 \alpha_1 \mid \delta_2 \alpha_2 \mid \dots \mid \delta_2 \alpha_m}_{\delta_2(\alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m)}$$

$$\qquad \qquad \qquad A' \qquad \qquad \qquad A''$$

改写文法 G 后, 得到的文法 G' 为:

$$A \rightarrow \delta_1 A' \mid \delta_2 A''$$

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

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



第4章 语法分析(Syntax Analysis)

—— 自上而下分析法

4.1 语法分析综述

4.2 不确定的自上而下分析法

 4.3 LL(1)分析法与LL(1)分析器

4.4 递归下降分析法与递归下降分析器



4.3 LL(1)分析法与LL(1)分析器



4.3.1 LL(1)分析器的逻辑结构

4.3.2 LL(1)分析器的构造

4.3.3 关于LL(1)文法



预测分析器

LL(1)

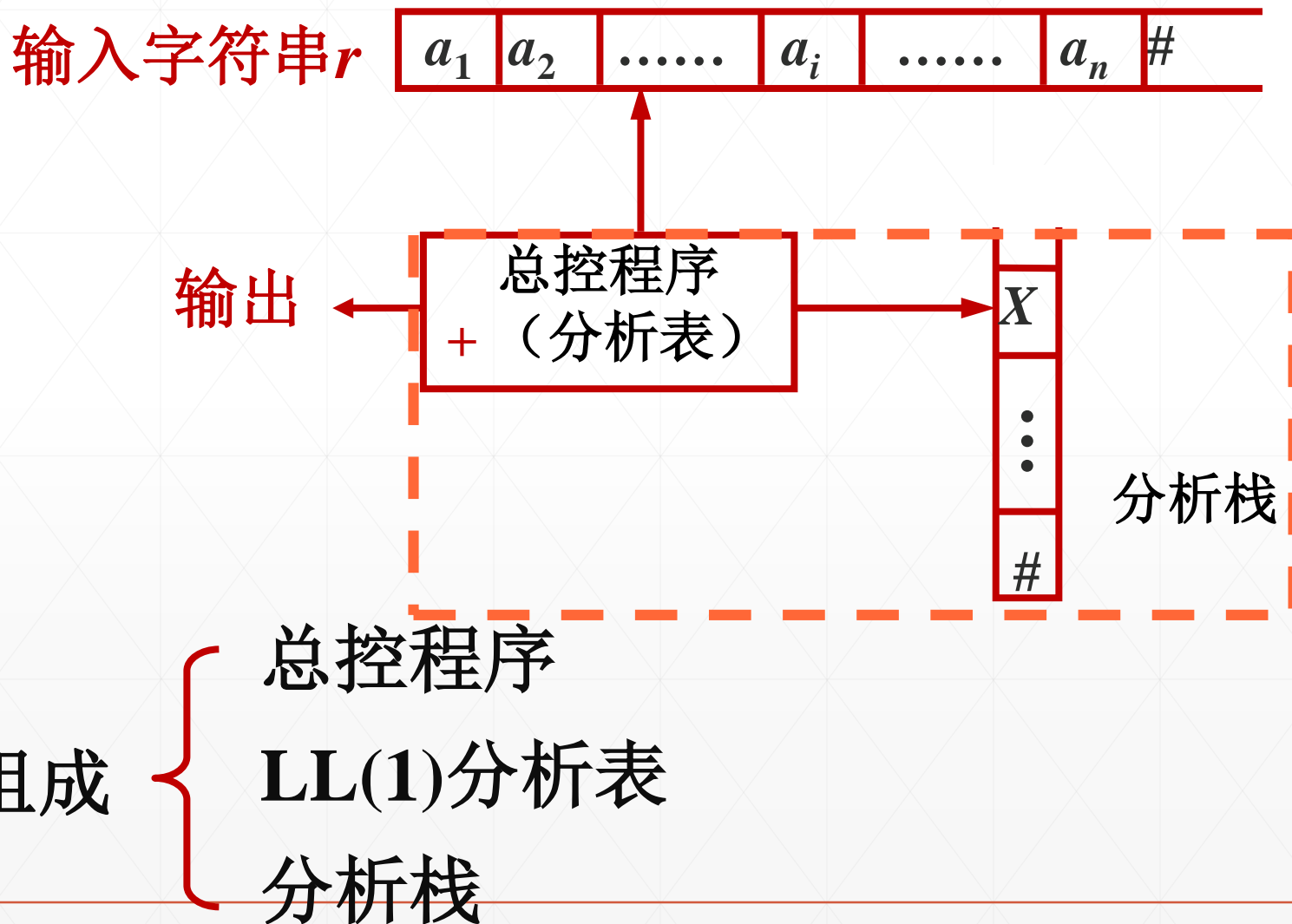
在分析中最多向前看1个输入字符

分析模式：最左推导

扫描模式：自左向右



LL(1)分析器的逻辑结构

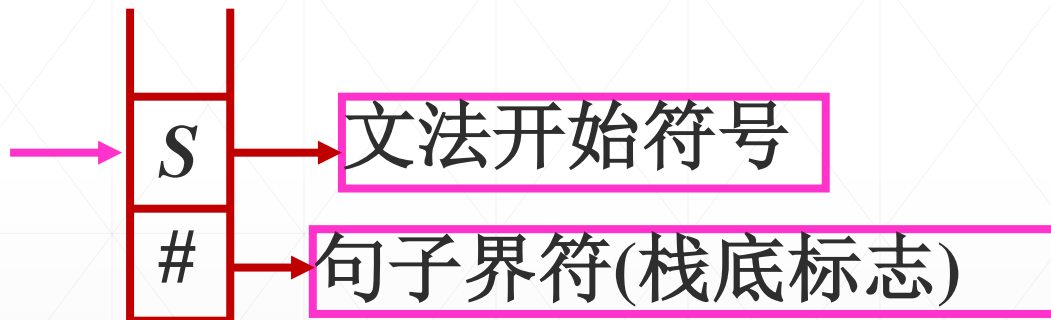




(1) 分析栈

存放分析过程中的文法符号（已经推导出的待处理的串）。

初始为：





(2) 分析表

		文法的 V_T			
		a_1	a_2	...	a_n #
文法的 V_N	A_1	$M(A_1, a_1)$	$M(A_1, a_2)$...	$M(A_1, a_n)$
	A_2				
	...				
	A_n				

预测分析函数

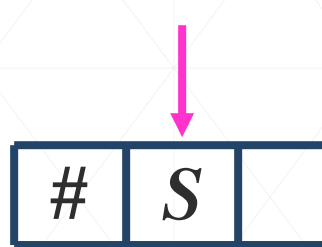
$M(A_i, a_i) \begin{cases} A_i \text{ 的一条产生式规则 (候选式唯一)} \\ \text{出错 (空白)} \end{cases}$



(3) 总控程序 (LL(1)分析)

■ 算法

(1) 初始化工作:



p
 $a_1 a_2 \dots a_n \#$

为描述方便, 设栈顶符号为 X , p 指向的符号为 a_i ,

(2) 若 X 是文法的终结符号, 则

对于:

- ① $X = a_i = \text{"\#"}$, 表示分析成功, 停止分析过程;
- ② $X = a_i \neq \text{"\#"}$, 则将 X 从分析栈顶退掉,
 p 指向下一个输入字符;
- ③ $X \neq a_i$, 表示不匹配的出错情况。



(3) 若 $X \in V_N$ ，则查分析表中的项 $M(X, a_i)$ ：

① 若 $M(X, a_i)$ 中为一个产生式规则，
则将 X 从栈中弹出，
并将此规则右部的符号序列按倒序推进栈
(若产生式规则为 $X \rightarrow \varepsilon$ ，则仅将 X 从栈中弹出)。

② 若 $M(X, a_i)$ 中为空白，
表示出错，可调用语法出错处理子程序。



例： 设有文法 $G(E)$:

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

$$F \rightarrow (E) \mid id$$

和文法的LL(1)分析表，对输入串 *id+id*id*
使用LL(1)分析器的分析过程。



文法G(E)的LL(1)分析表

	+	*	()	<i>id</i>	#
<i>E</i>			$E \rightarrow TE'$		$E \rightarrow TE'$	
<i>E'</i>	$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$		$E' \rightarrow \varepsilon$
<i>T</i>			$T \rightarrow FT'$		$T \rightarrow FT'$	
<i>T'</i>	$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$		$T' \rightarrow \varepsilon$
<i>F</i>			$F \rightarrow (E)$		$F \rightarrow id$	



步 骤	分析栈(底→顶)	待匹配串	分析动作
1	# <i>E</i>	<i>id</i> + <i>id</i> * <i>id</i> #	$E \rightarrow TE'$
2	# <i>E</i> ' <i>T</i>	<i>id</i> + <i>id</i> * <i>id</i> #	$T \rightarrow FT'$
3	# <i>E</i> ' <i>T</i> ' <i>F</i>	<i>id</i> + <i>id</i> * <i>id</i> #	$F \rightarrow id$
4	# <i>E</i> ' <i>T</i> ' <i>id</i>	<i>id</i> + <i>id</i> * <i>id</i> #	p++
5	# <i>E</i> ' <i>T</i> '	+ <i>id</i> * <i>id</i> #	$T' \rightarrow \varepsilon$
6	# <i>E</i> '	+ <i>id</i> * <i>id</i> #	$E' \rightarrow +TE'$
7	# <i>E</i> ' <i>T</i> +	+ <i>id</i> * <i>id</i> #	p++
8	# <i>E</i> ' <i>T</i>	<i>id</i> * <i>id</i> #	$T \rightarrow FT'$
9	# <i>E</i> ' <i>T</i> ' <i>F</i>	<i>id</i> * <i>id</i> #	$F \rightarrow id$
10	# <i>E</i> ' <i>T</i> ' <i>id</i>	<i>id</i> * <i>id</i> #	p++
11	# <i>E</i> ' <i>T</i> '	* <i>id</i> #	$T' \rightarrow *FT'$



表

步骤分析	栈(底→顶)	待匹配串	所用产生式
12	# $E' T' F^*$	$*id\#$	$p++$
13	# $E' T' F$	$id\#$	$F \rightarrow id$
14	# $E' T' id$	$id\#$	$p++$
15	# $E' T'$	#	$T' \rightarrow \varepsilon$
16	# E'	#	$E' \rightarrow \varepsilon$
17	#	#	分析成功



注意

(1) 整个分析过程是分析栈和待匹配串构成的二元式不断变化的过程。

(2) 不同的源语言仅是分析表不同，分析器结构、总控程序不变。



4.3 LL(1)分析法与LL(1)分析器

4.3.1 LL(1)分析器的逻辑结构



4.3.2 LL(1)分析器的构造

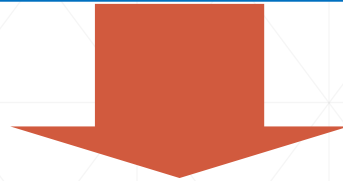
4.3.3 关于LL(1)文法



LL(1)分析器构造关键 —— 分析表的构造



分析表的构造关键 —— 预测函数



根据LL(1)分析过程，问题的关键：

依据下一步要匹配的终结符，

选择当前非终结符要替换的候选式。



第一种情况(候选式的FIRST集合中无 ε):

对文法 G , 非终结符 A 的产生式设为:

$$A \rightarrow \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_n$$

且没有 $\gamma_i^* \Rightarrow \varepsilon (i=1,2,\dots,n)$ 。

首先求取每个候选式 γ_i 的FIRST集合,
再根据 γ_i 的FIRST集合构造LL(1)分析表, 即

若 $a \in \text{FIRST}(\gamma_i)$,

那么 $M(A, a) = \{ A \rightarrow \gamma_i \}$

$$S \rightarrow Ap \mid Bq$$

$$A \rightarrow a \mid cA$$

$$B \rightarrow b \mid dB$$

对S: $\text{FIRST}(Ap) = \{a, c\}$ 对A: $\text{FIRST}(a) = \{a\}$ 对B: $\text{FIRST}(b) = \{b\}$

$\text{FIRST}(Bq) = \{b, d\}$

$\text{FIRST}(cA) = \{c\}$

$\text{FIRST}(dB) = \{d\}$

文法G的LL(1)分析表

	a	b	c	d	p	q	$\#$
S	$S \rightarrow Ap$	$S \rightarrow Bq$	$S \rightarrow Ap$	$S \rightarrow Bq$			
A	$A \rightarrow a$		$A \rightarrow cA$				
B		$B \rightarrow b$		$B \rightarrow dB$			

若有 $\varepsilon \in \text{FIRST}(\gamma)$ ，当后面匹配符号 $a \notin \text{FIRST}(\gamma)$ 时并不一定出错，怎么处理？



■ 定义

设上下文无关文法 G ， S 是文法的开始符号，对于文法 G 的任何非终结符 A

$$\text{FOLLOW}(A) = \{ a | S \overset{*}{\Rightarrow} \dots Aa \dots, a \in V_T \}$$

若 $S \overset{*}{\Rightarrow} \dots A$ ，则 $\# \in \text{FOLLOW}(A)$ 。

FOLLOW(A)的含义：

在文法 G 的句型中，

能够紧跟在 A 之后的一切终结符或“#”。



■ 构造FOLLOW集方法

■ 文法 G 中的每一个 $A \in V_N$ ，可反复应用如下规则来求FOLLOW(A):

① A 是文法的开始符号， $\# \in \text{FOLLOW}(A)$ ；

② 文法 G 中有形如 $B \rightarrow \alpha A \beta$ 的规则，且 $\beta \neq \varepsilon$ ，
 $\text{FIRST}(\beta) - \{\varepsilon\} \subseteq \text{FOLLOW}(A)$ ；

③ 文法 G 中有形如 $B \rightarrow \alpha A$ 或 $B \rightarrow \alpha A \beta$ ($\varepsilon \in \text{FIRST}(\beta)$) 的规则，

$\text{FOLLOW}(B) \subseteq \text{FOLLOW}(A)$ 。

其中： $\alpha, \beta \in V^*$





例：设有文法 $G[S]$ 为：

$$S \rightarrow AB \mid bC$$


$$A \rightarrow \varepsilon \mid b$$

$$B \rightarrow \varepsilon \mid aD \mid CAc$$

$$C \rightarrow AD \mid b$$

$$D \rightarrow aS \mid c$$

计算文法 $G[S]$ 所有 V_N 的FOLLOW集。



$$\begin{aligned}
 S &\rightarrow AB \mid bC \\
 A &\rightarrow \varepsilon \mid b \\
 B &\rightarrow \varepsilon \mid aD / CAc \\
 C &\rightarrow AD \mid b \\
 D &\rightarrow aS \mid c
 \end{aligned}$$

对S:

S为开始符号; $D \rightarrow aS$

$\# \in \text{FOLLOW}(S)$; $\text{FOLLOW}(D) \subseteq \text{FOLLOW}(S)$;

对A:

$S \rightarrow AB$; $B \rightarrow CAc$; $C \rightarrow AD$

$\text{FIRST}(B) - \{\varepsilon\} \subseteq \text{FOLLOW}(A)$; $\text{FOLLOW}(S) \subseteq \text{FOLLOW}(A)$;

$\text{FIRST}(c) \subseteq \text{FOLLOW}(A)$ $\text{FIRST}(D) \subseteq \text{FOLLOW}(A)$

对B:

$S \rightarrow AB$

$\text{FOLLOW}(S) \subseteq \text{FOLLOW}(B)$

对C:

$S \rightarrow bC$; $B \rightarrow CAc$

$\text{FOLLOW}(S) \subseteq \text{FOLLOW}(C)$; $\text{FIRST}(Ac) \subseteq \text{FOLLOW}(C)$

对D:

$B \rightarrow aD$; $C \rightarrow AD$

$\text{FOLLOW}(B) \subseteq \text{FOLLOW}(D)$; $\text{FOLLOW}(C) \subseteq \text{FOLLOW}(D)$

所以:

$\text{FOLLOW}(S) = \text{FOLLOW}(B) = \text{FOLLOW}(C) = \text{FOLLOW}(D) = \{\#, b, c\}$

$\text{FOLLOW}(A) = \text{FOLLOW}(S) \cup \text{FIRST}(B) \cup \text{FIRST}(c) \cup \text{FIRST}(D) - \{\varepsilon\} = \{\#, b, a, c\}$

$\text{FIRST}(B) = \{\varepsilon, a, b, c\}$

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

$\text{FIRST}(Ac) = \{b, c\}$



2. 构造FOLLOW集方法——关系图法

(1)从开始符号 S 的FOLLOW(S)结点到“#”号的结点连一条箭弧。

(2)对文法的每一条产生式:

a)若形如 $B \rightarrow \alpha A \beta$ 的规则, 且 $\beta \neq \epsilon$,

则从FOLLOW(A)结点到FIRST(β)结点连一条弧;

b)若形如 $B \rightarrow \alpha A$ 或 $B \rightarrow \alpha A \beta$ ($\epsilon \in \text{FIRST}(\beta)$),

则从FOLLOW(A)结点到FOLLOW(B)结点连一条弧。

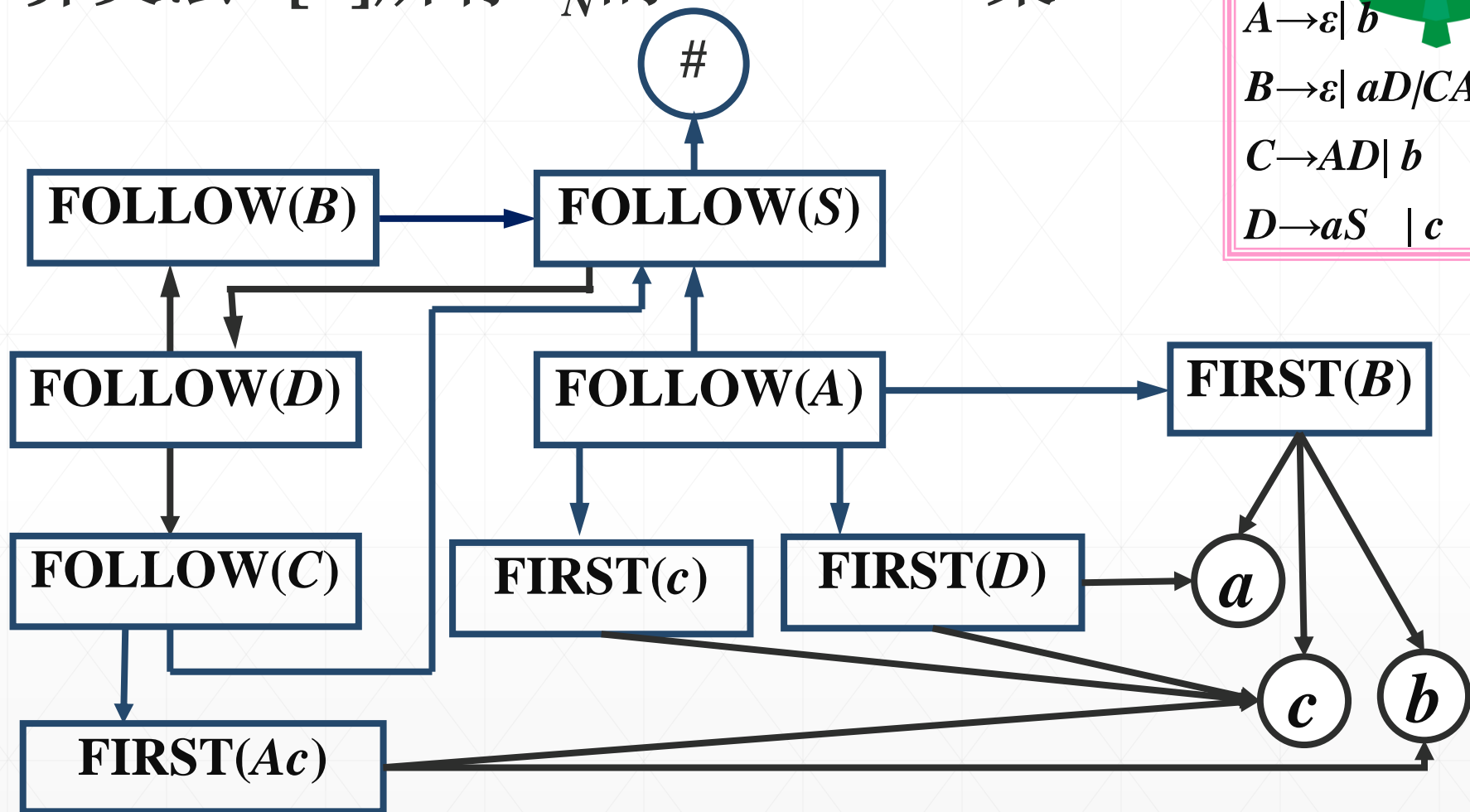
(3)FIRST(β)结点到FIRST(β)集合中的每个终结符号结点连一条弧。

(4)从FOLLOW(A)结点有路径可以到达的终结符号或“#”, 就是FOLLOW(A)的成员。

其中: $\alpha, \beta \in V^*$

$S \rightarrow AB \mid bC$
 $A \rightarrow \varepsilon \mid b$
 $B \rightarrow \varepsilon \mid aD/CAc$
 $C \rightarrow AD \mid b$
 $D \rightarrow aS \mid c$

计算文法 $G[S]$ 所有 V_N 的FOLLOW集。



$\text{FOLLOW}(S) = \text{FOLLOW}(B) = \text{FOLLOW}(D) = \text{FOLLOW}(C) = \{\#, b, c\};$

$\text{FOLLOW}(A) = \{\#, a, b, c\}$



第二种情况(候选式的FIRST集合有 ε):

对文法 G , 非终结符 A 的产生式设为:

$$A \rightarrow \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_n$$

且存在 i , 有 $\gamma_i \overset{*}{\Rightarrow} \varepsilon$ ($i=1,2,\dots,n$)。

求取每个候选式 γ_i 的FIRST集合、 **A 的FOLLOW集合**

根据 γ_i 的FIRST集合和 A 的FOLLOW集合构造LL(1)分析表, 即

若 $\varepsilon \in \text{FIRST}(\gamma_i)$

对 $\forall a \in (\text{FIRST}(\gamma_i) \cup \text{FOLLOW}(A))$, $M(A, a) = \{A \rightarrow \gamma_i\}$

若 $\varepsilon \notin \text{FIRST}(\gamma_i)$,

对 $\forall a \in \text{FIRST}(\gamma_i)$, $M(A, a) = \{A \rightarrow \gamma_i\}$



分析

$\text{FOLLOW}(A) = \{a \mid S \xRightarrow{*} \cdots A a \cdots, a \in V_T\}$

若 $a \in \text{FOLLOW}(A)$

则必有 $\cdots A a \cdots (a \in V_T)$ 这样的句型。

设 $S \xRightarrow{*} \alpha A a \cdots$

存在 $A \rightarrow \gamma$ 且 $\varepsilon \in \text{FIRST}(\gamma)$, 当 $\gamma \xRightarrow{*} \varepsilon$, 则 $A \xRightarrow{*} \varepsilon$ 。

故有 $S \xRightarrow{*} \alpha A a \cdots \Rightarrow \alpha \gamma a \cdots \xRightarrow{*} \alpha \underline{a} \cdots$

获得匹配

$\therefore M(A, b) = A \rightarrow \gamma$ (这样在自上而下分析的推导

中就可以从栈中退掉 A 。)



■ 算法:LL(1)分析表构造

输入: 文法 G ; G 候选式的FIRST、候选式FIRST有 ε 的左侧非终结符号的FOLLOW集合

输出: 文法 G 的LL(1)分析表

方法: for 文法 G 的每个产生式 $A \rightarrow \gamma_1 | \gamma_2 | \cdots | \gamma_m$

{ if $a \in \text{FIRST}(\gamma_i)$ 置 $\mathbf{M(A, a)} = A \rightarrow \gamma_i$;

if $\varepsilon \in \text{FIRST}(\gamma_i)$

for 任何 $b \in \text{FOLLOW}(A)$

{ 置 $\mathbf{M(A, b)} = A \rightarrow \gamma_i$ }

else 置所有无定义的 $\mathbf{M(A, a)}$ 为出错

}



前例： 设有文法 $G(E)$:

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

$$F \rightarrow (E) \mid i$$

构造该文法的LL(1)分析表

对E: $\text{FIRST}(TE') = \{ (, i \}$

对E' : $\text{FIRST}(+ TE') = \{ + \}$

$\text{FIRST}(\varepsilon) = \{ \varepsilon \}$

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

对T: $\text{FIRST}(FT') = \{ (, i \}$

对T' : $\text{FIRST}(*FT') = \{ * \}$

$\text{FIRST}(\varepsilon) = \{ \varepsilon \}$

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

对F: $\text{FIRST}((E)) = \{ (\}$ $\text{FIRST}(i) = \{ i \}$

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) | i$



$E \rightarrow TE'$: $\text{FIRST}(TE') = \{ (, i \}$

$E' \rightarrow +TE' | \varepsilon$: $\text{FIRST}(+TE') = \{ + \}$ $\text{FIRST}(\varepsilon) = \{ \varepsilon \}$ $\text{FOLLOW}(E') = \{ \#,) \}$

$T \rightarrow FT'$: $\text{FIRST}(FT') = \{ (, i \}$

$T' \rightarrow *FT' | \varepsilon$: $\text{FIRST}(*FT') = \{ * \}$ $\text{FIRST}(\varepsilon) = \{ \varepsilon \}$ $\text{FOLLOW}(T') = \{ \#,), + \}$

$F \rightarrow (E) | i$: $\text{FIRST}((E)) = \{ (\}$ $\text{FIRST}(i) = \{ i \}$

文法 $G(E)$ 的LL(1)分析表

	+	*	()	i	#
E			$E \rightarrow TE'$		$E \rightarrow TE'$	
E'	$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$		$E' \rightarrow \varepsilon$
T			$T \rightarrow FT'$		$T \rightarrow FT'$	
T'	$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$		$T' \rightarrow \varepsilon$
F			$F \rightarrow (E)$		$F \rightarrow i$	



4.3 LL(1)分析法与LL(1)分析器

4.3.1 LL(1)分析器的逻辑结构

4.3.2 LL(1)分析器的构造

4.3.3 关于LL(1)文法





■ 定义

一部文法 G ，若它的LL(1)分析表 M 不含多重定义入口，则称它是一个LL(1)文法。由LL(1)文法产生的语言称为LL(1)语言。

LL(k)

在分析中最多向前看 k 个输入字符
分析模式：最左推导
扫描模式：自左向右



关于LL(1)文法及LL(1)语言重要的性质

- (1)任何LL(1)文法是无二义性的。
- (2)若一文法为左递归文法，则它必然是非LL(1)文法。
- (3)非LL(1)语言是存在的。
- (4)存在一种**算法**，它能判定任一文法是否为LL(1)文法。
- (5)不存在这样的算法，它能判定上下文无关语言能否由LL(1)文法产生。



非左递归文法 G 为LL(1)文法 $\Leftrightarrow G$ 的任何一个非终结符 A ，设关于 A 的产生式为

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

满足下面的条件:

- 1) 每个候选式 α_i , $\text{FIRST}(\alpha_i)$ 两两彼此互不相交。
- 2) 若有 $\alpha_i \Rightarrow \varepsilon$, 则 $\text{FOLLOW}(A)$ 与其他候选式的 FIRST 亦两两互不相交。



例：判断下面文法是否为LL(1)文法

$$S \rightarrow MH/a$$
$$H \rightarrow LSo|\varepsilon$$
$$K \rightarrow dML|\varepsilon$$
$$L \rightarrow eHf$$
$$M \rightarrow K|bLM$$

方法：

首先观察：

左递归 \Rightarrow 不是LL(1)文法

左公因子 \Rightarrow 不是LL(1)文法

二义性 \Rightarrow 不是LL(1)文法

其次验证

有2个或2个以上的候选式



解:

对S:

$$\text{FIRST}(MH) = \{d, b, e, \varepsilon\}$$

$$\text{FOLLOW}(S) = \{\#, o\}$$

$$\text{FIRST}(a) = \{a\}$$

$$(\text{FIRST}(MH) \cup \text{FOLLOW}(S)) \cap \text{FIRST}(a) = \emptyset$$

对H:

$$\text{FIRST}(LSO) = \{e\}$$

$$\text{FIRST}(\varepsilon) = \{\varepsilon\}$$

$$\text{FOLLOW}(H) = \{\#, o, f\}$$

$$\text{FIRST}(LSO) \cap (\text{FIRST}(\varepsilon) \cup \text{FOLLOW}(H)) = \emptyset$$

对K:

$$\text{FIRST}(dML) = \{d\}$$

$$\text{FIRST}(\varepsilon) = \{\varepsilon\}$$

$$\text{FOLLOW}(K) = \{e, \#, o\}$$

$$\text{FIRST}(dML) \cap (\text{FIRST}(\varepsilon) \cup \text{FOLLOW}(K)) = \emptyset$$

对M:

$$\text{FIRST}(K) = \{d, \varepsilon\}$$

$$\text{FOLLOW}(M) = \{e, \#, o\}$$

$$\text{FIRST}(bLM) = \{b\}$$

$$(\text{FIRST}(K) \cup \text{FOLLOW}(M)) \cap \text{FIRST}(bLM) = \emptyset$$

$$S \rightarrow MH/a$$

$$H \rightarrow LSO|\varepsilon$$

$$K \rightarrow dML|\varepsilon$$

$$L \rightarrow eHf$$

$$M \rightarrow K|bLM$$

该文法为
LL(1)文法。



例：判断下面文法是否为LL(1)文法？如果不是，改写为LL(1)文法。

$$(2) \quad S \rightarrow AB$$

$$A \rightarrow Ba | \varepsilon$$

$$B \rightarrow Db | D$$

$$D \rightarrow d | \varepsilon$$

答：非终结符 B 的两个候选式有左公因子，则其FIRST集合有交集。该文法不是LL(1)文法。

$$B \rightarrow Db | D$$

提取左公因子改写为：

$$B \rightarrow DB' \quad B' \rightarrow b | \varepsilon$$

已有的修改方法：

1.消除左递归

2.提取左公因子



新方法



文法改写为:

$$S \rightarrow AB$$

$$A \rightarrow Ba | \varepsilon$$

$$B \rightarrow DB'$$

$$B' \rightarrow b | \varepsilon$$

$$D \rightarrow d | \varepsilon$$

对产生式 $A \rightarrow Ba | \varepsilon$

$$\text{First}(Ba) = \{d, b, a\},$$

$$\text{Follow}(A) = (\text{First}(B) - \{\varepsilon\}) \cup \text{Follow}(S) \\ = \{d, b\} \cup \{\#\} = \{d, b, \#\}$$

$$\text{First}(Ba) \cap \text{Follow}(A) \neq \emptyset$$

修改后的文法还不是LL(1)文法。

继续修改, 把A的产生式代入S的候选式, 得

$$S \rightarrow BaB \mid B$$

提取S产生式的公因子:

$$S \rightarrow BS'$$

$$S' \rightarrow aB \mid \varepsilon$$

修改方法:

1. 消除左递归

2. 提取左公因子

3. 产生式代入



得文法

$$S \rightarrow BS'$$

$$S' \rightarrow aB \mid \varepsilon$$

$$B \rightarrow DB'$$

$$B' \rightarrow b \mid \varepsilon$$

$$D \rightarrow d \mid \varepsilon$$

修改后的文法为LL(1)文法。

验证:

对 S' :

$$\text{First}(aB) = \{a\}; \text{Follow}(S') = \text{Follow}(S) = \{\#\}$$

$$\text{First}(aB) \cap \text{Follow}(S') = \emptyset$$

对 B' :

$$\text{First}(b) = \{b\}; \text{Follow}(B') = \text{Follow}(B)$$

$$= (\text{First}(S') - \{\varepsilon\}) \cup \text{Follow}(S) \cup \text{Follow}(S')$$

$$= \{a\} \cup \{\#\} \cup \{\#\} = \{a, \#\}$$

$$\text{First}(b) \cap \text{Follow}(B') = \emptyset$$

对 D :

$$\text{First}(d) = \{d\}; \text{Follow}(D) =$$

$$(\text{First}(B') - \{\varepsilon\}) \cup \text{Follow}(B) = \{b\} \cup \{a, \#\} = \{a, b, \#\}$$

$$\text{First}(d) \cap \text{Follow}(D) = \emptyset$$



一、在原文法结构上的修改方法

1. 消除左递归
2. 提取左公因子
3. 产生式代入

二、改变文法结构的修改

1. 写出文法描述的语言
2. 给出语言相对应的LL(1)文法



例:设有文法 $G(S)$

$$S \rightarrow iCtSS' \mid a$$

$$S' \rightarrow eS \mid \varepsilon$$

$$C \rightarrow b$$

构造该文法的LL(1)分析表

对 S : $\text{FIRST}(iCtSS') = \{ i \}$

$\text{FIRST}(a) = \{ a \}$

对 S' : $\text{FIRST}(eS) = \{ e \}$

$\text{FIRST}(\varepsilon) = \{ \varepsilon \}$ $\text{FOLLOW}(S') = \{ \#, e \}$

对 C : $\text{FIRST}(b) = \{ b \}$



$S \rightarrow iCtSS' \mid a: \text{FIRST}(iCtSS') = \{ i \} \quad \text{FIRST}(a) = \{ a \}$
 $S' \rightarrow eS \mid \varepsilon: \text{FIRST}(eS) = \{ e \} \quad \text{FIRST}(\varepsilon) = \{ \varepsilon \} \quad \text{FOLLOW}(S') = \{ \#, e \}$
 $C \rightarrow b: \text{FIRST}(b) = \{ b \}$

该文法非LL(1)文法

文法G(S)的LL(1)分析表

	a	b	e	i	t	$\#$
S	$S \rightarrow a$			$S \rightarrow iCtSS'$		
S'			$S' \rightarrow eS$ $S' \rightarrow \varepsilon$			$S' \rightarrow \varepsilon$
C		$C \rightarrow b$				

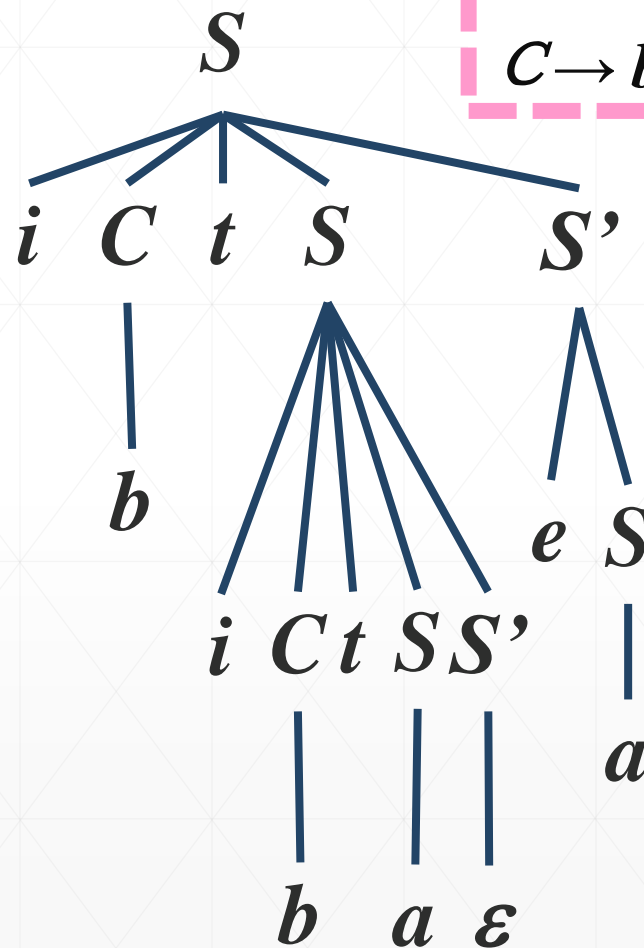
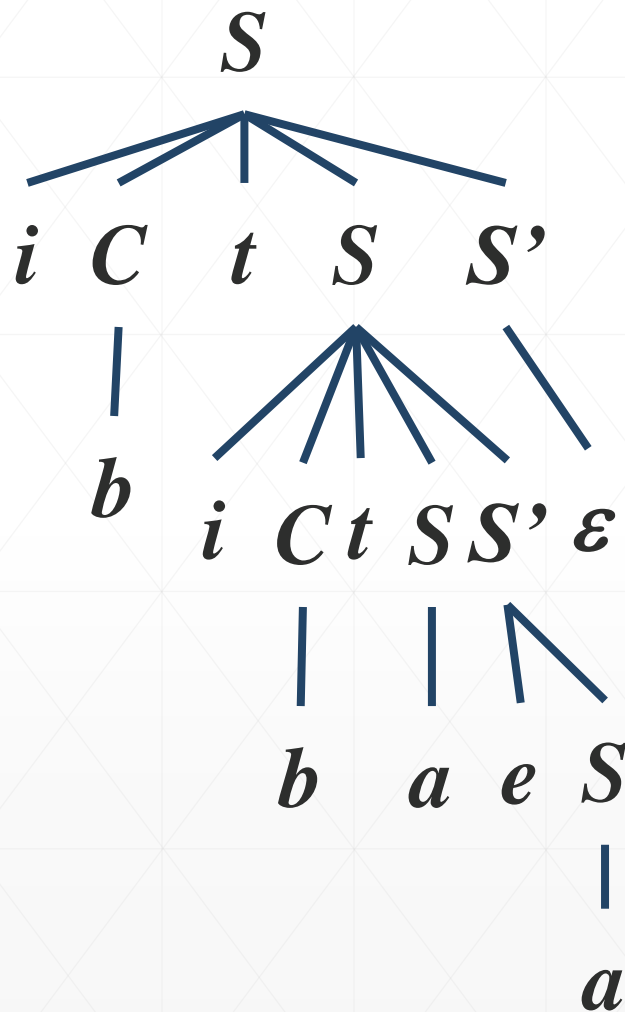
多重定义



句子 *ibtibtaea* 有两棵不同的语法分析树，

$$S \rightarrow iCtSS' \mid a$$

$$S' \rightarrow eS \mid \varepsilon$$

$$C \rightarrow b$$


文法为二义性文法。



文法G(S)的LL(1)分析表

$$S \rightarrow iCtSS' \mid a$$

$$S' \rightarrow eS \mid \varepsilon$$

$$C \rightarrow b$$

	a	b	e	i	t	$\#$
S	$S \rightarrow a$			$S \rightarrow iCtSS'$		
S'						$S' \rightarrow \varepsilon$
			$S' \rightarrow eS$			
C		$C \rightarrow b$				

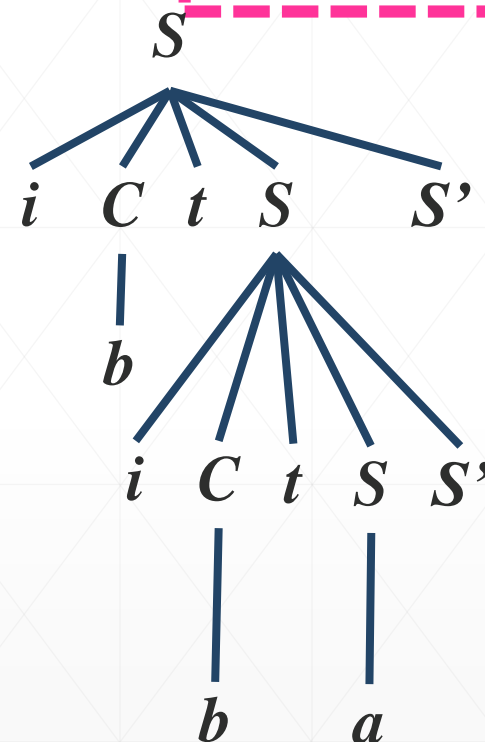


句子 *ibtibtaea* 的 LL(1) 分析过程及分析树的构造

步骤	分析栈	待匹配串	分析动作
1	# <i>S</i>	<i>ibtibtaea</i> #	$S \rightarrow iCtSS'$
2	# <i>S'StCi</i>	<i>ibtibtaea</i> #	$P++$
3	# <i>S'StC</i>	<i>btibtaea</i> #	$C \rightarrow b$
4	# <i>S'Stb</i>	<i>btibtaea</i> #	$P++$
5	# <i>S'St</i>	<i>tibtaea</i> #	$P++$
6	# <i>S'S</i>	<i>ibtaea</i> #	$S \rightarrow iCtSS'$
7	# <i>S'S'StCi</i>	<i>ibtaea</i> #	$P++$
8	# <i>S'S'StC</i>	<i>btaea</i> #	$C \rightarrow b$
9	# <i>S'S'Stb</i>	<i>btaea</i> #	$P++$
10	# <i>S'S'St</i>	<i>taea</i> #	$P++$
11	# <i>S'S'S</i>	<i>aea</i> #	$S \rightarrow a$
12	# <i>S'S'a</i>	<i>aea</i> #	$P++$
13	# <i>S'S'</i>	<i>ea</i> #	$S' \rightarrow eS$

$$S \rightarrow iCtSS' \mid a$$

$$S' \rightarrow eS \mid \varepsilon$$

$$C \rightarrow b$$


分析表


$$S \rightarrow iCtSS' \mid a$$
$$C \rightarrow b$$

```

graph TD
    S1[S] --- i1[i]
    S1 --- C1[C]
    S1 --- t1[t]
    S1 --- S2[S]
    S1 --- Sp1[S']
    S2 --- b1[b]
    S2 --- S3[S]
    S3 --- i2[i]
    S3 --- C2[C]
    S3 --- t2[t]
    S3 --- S4[S]
    S3 --- Sp2[S', ε]
    S4 --- b2[b]
    S4 --- a1[a]
    Sp2 --- e[e]
    Sp2 --- S5[S]
    S5 --- a2[a]
  
```



第4章 语法分析(Syntax Analysis)

—— 自上而下分析法

4.1 语法分析综述

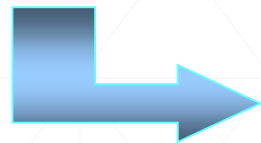
4.2 不确定的自上而下分析法

4.3 LL(1)分析法与LL(1)分析器

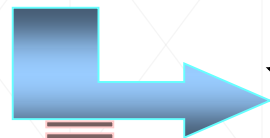
➡ 4.4 递归下降分析法与递归下降分析器



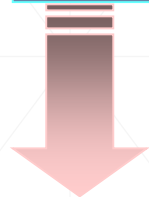
据语言语法规则



LL(1)分析表



可行的自上而下语法分析器



存储和激活问题

数据中心法

—— LL(1)语法分析器

程序中心法

—— 递归下降分析器



- 递归下降分析器(**Recursive-Descent Parser**)

一组递归过程组成，每个过程对应文法的一个 V_N 的分析程序(替换过程)。

- 递归下降分析器对文法要求： **LL(1)**文法

- 递归下降分析器的基本构造方法

对文法的每个非终结符号，根据其 **(LL(1)分析表中填写的)** 候选式的结构，为其编写一个对应的子程序(或函数)，该子程序完成相应的语法规成份的识别和分析任务。



例4.9 设有文法 $G(E)$:

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

$$F \rightarrow (E) | i$$

构造文法 $G(E)$ 的递归下降语法分析程序。



文法 $G(E)$ 的LL(1)分析表

	+	*	()	i	#
E			$E \rightarrow TE'$		$E \rightarrow TE'$	
E'	$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$		$E' \rightarrow \varepsilon$
T			$T \rightarrow FT'$		$T \rightarrow FT'$	
T'	$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$		$T' \rightarrow \varepsilon$
F			$F \rightarrow (E)$		$F \rightarrow i$	



+ * () i #

E

$E \rightarrow TE'$

$E \rightarrow TE'$

E' $E' \rightarrow +TE'$

$E' \rightarrow \varepsilon$

$E' \rightarrow \varepsilon$

E()

```
{if(c=='('||c=='i')
  {T(); E'();
  if(c=='#')return;
  else error;
  else error;}
```

非法结束

缺少运算对象

E'()

```
{if (c=='+')
  {n++; T(); E'();}
elseif(c=='('||c=='#')
  return;
else error;}
```

缺少运算符

其中: n — 读单词指针; c — 单词指针指的单词;



$+$ $*$ $($ $)$ i $\#$

T $T \rightarrow FT'$ $T \rightarrow FT'$

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

T (

```

{if(c==( ' || c=='i')
  {F(); T'();}
else error;}
  
```

缺少运算
对象

T' (

```

{if (c=='*')
  {n++; F(); T'();}
elseif(c=='+' || c=='(' || c=='#')
return;
else error;}
  
```

缺少运算
符

其中: **n** — 读单词指针; **c** — 单词指针指的单词;



+

*

()

i

#

 F $F \rightarrow (E)$ $F \rightarrow i$ **F (****{if (c='i') n++;****else if (c='(')****{ n++; E();****if (c=')') n++;****else error; }****else error; }**

缺少")"

缺少运算
对象

其中：*n* — 读单词指针； *c* — 单词指针指的单词；



E() **主程序**
{T(); E'();
 if(c=='#')
 return;
 else error;}

E'()
{if (c=='+')
 {n++; T(); E'();}
}

T ()
 {F(); T'();}

T'()
{if (c=='*')
 {n++; F(); T'();}

F ()
{if (c='i') n++;
 else if (c='(')
 { n++; E();
 if (c=')') n++;
 else error; }
 else error; }

E → **TE'**
E' → **+TE' | ε**
T → **FT'**
T' → ***FT' | ε**
F → **(E) | i**

一般的递归下降语法分析器



句子 $i+i*i \#$ 的递归下降分析过程

E()

```
{T(); E'();
  if(c=='#')
    return;
  else error;}
```

E'()

```
{if (c=='+')
  {n++; T(); E'();}
}
```

T()

```
{F(); T'();}
```

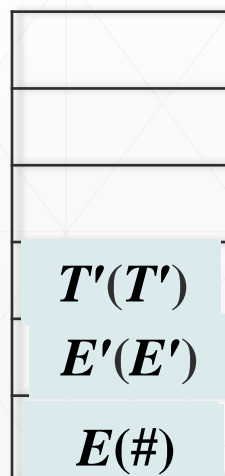
T'()

```
{if (c=='*')
  {n++; F(); T'();}
}
```

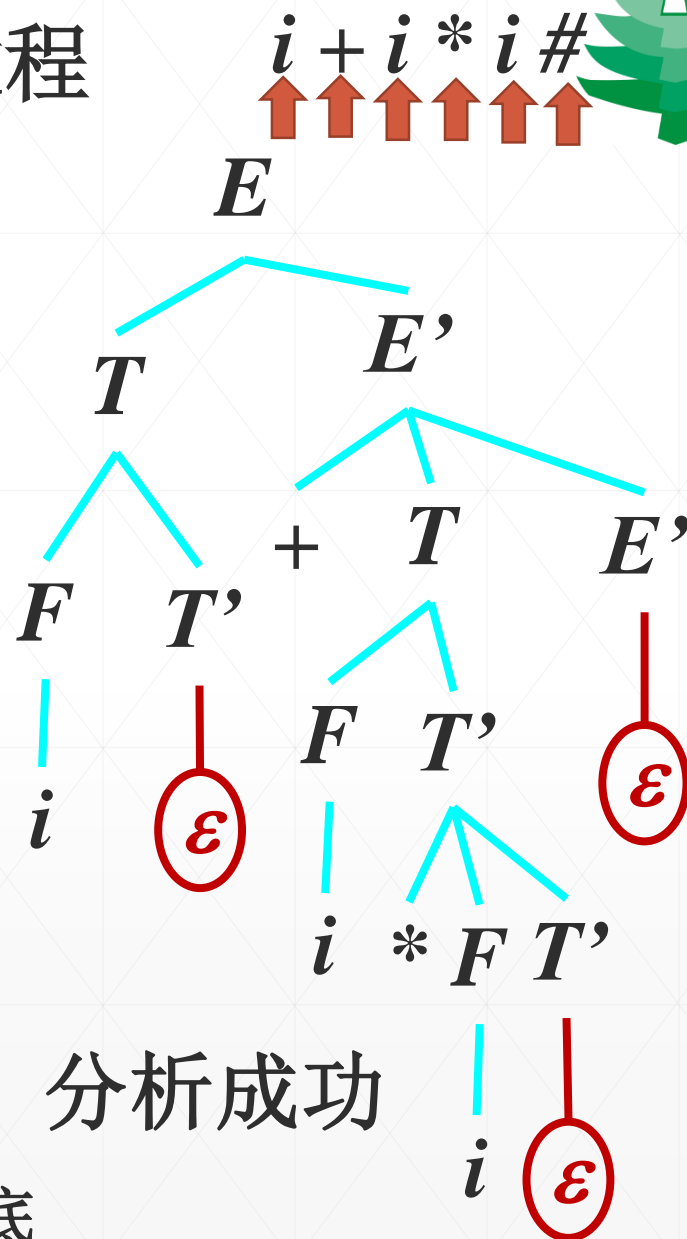
F()

```
{if (c=='i') n++;
  else if (c=='(')
  { n++; E();
    if (c=='') n++;
    else error; }
  else error; }
```

运行栈



栈底





递归下降分析器与LL(1)分析器的区别：

LL(1)分析器：显式地维护一个分析栈，

递归下降分析器：通过隐式的递归调用来使用栈。



(+i #的分析过程

```
E()
{T(); E'();
  if(c=='#')
    return;
  else error;}
```

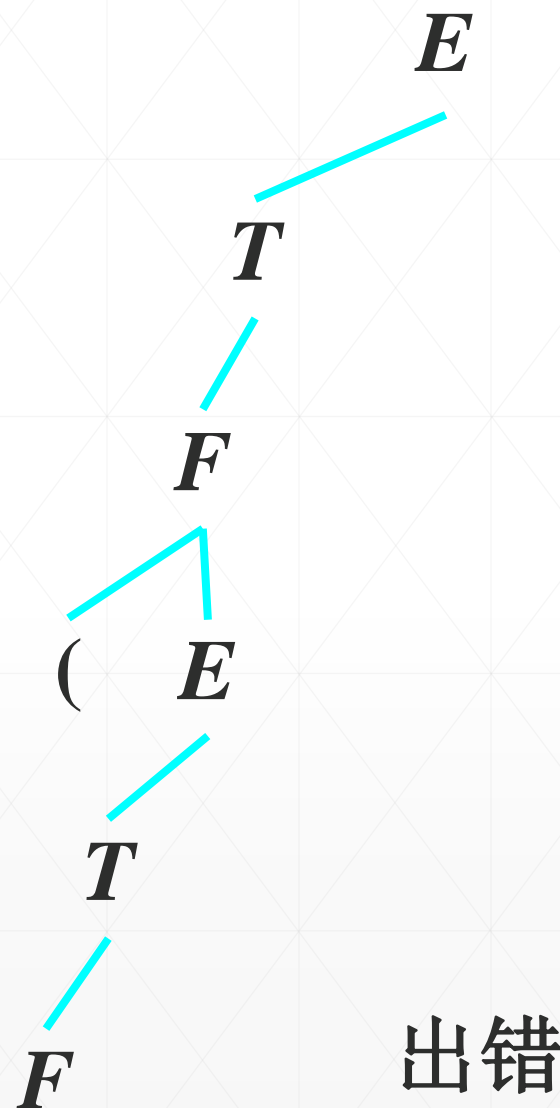
```
E'()
{if (c=='+')
  {n++; T(); E'();}
}
```

```
T()
{F(); T'();}

```

```
T'()
{if (c=='*')
  {n++; F(); T'();}
}
```

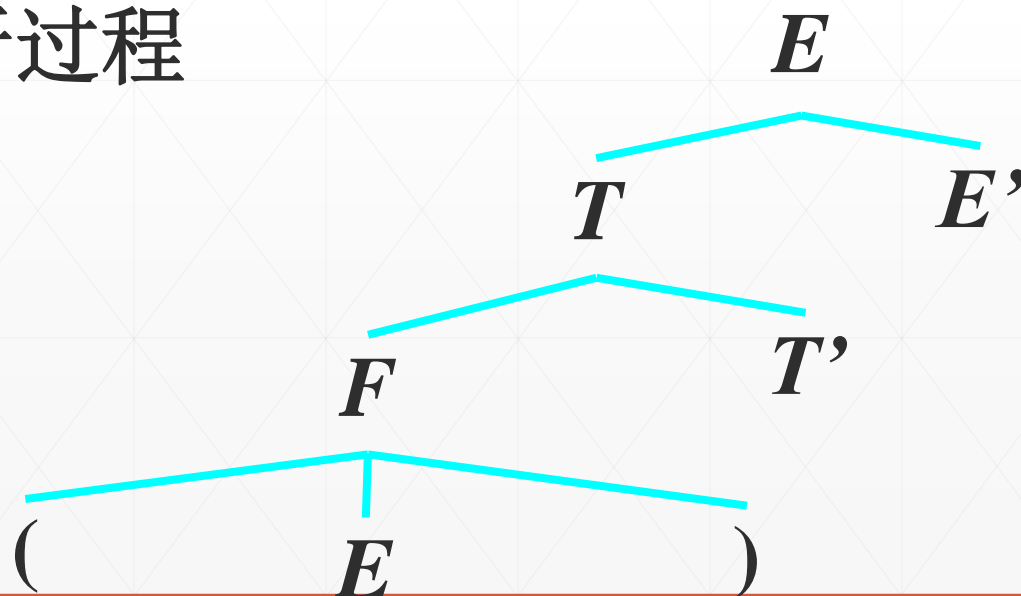
```
F()
{if ( c=='i') n++;
  else if ( c=='(')
  { n++; E();
    if (c=='') n++;
    else error; }
  else error; }
```





	+	*	()	<i>i</i>	#
<i>E</i>			$E \rightarrow TE'$		$E \rightarrow TE'$	
E'	$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$		$E' \rightarrow \varepsilon$
<i>T</i>			$T \rightarrow FT'$		$T \rightarrow FT'$	
T'	$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$		$T' \rightarrow \varepsilon$
<i>F</i>			$F \rightarrow (E)$		$F \rightarrow i$	

(+i #的分析过程



出错



end

Every end is
a new
beginning