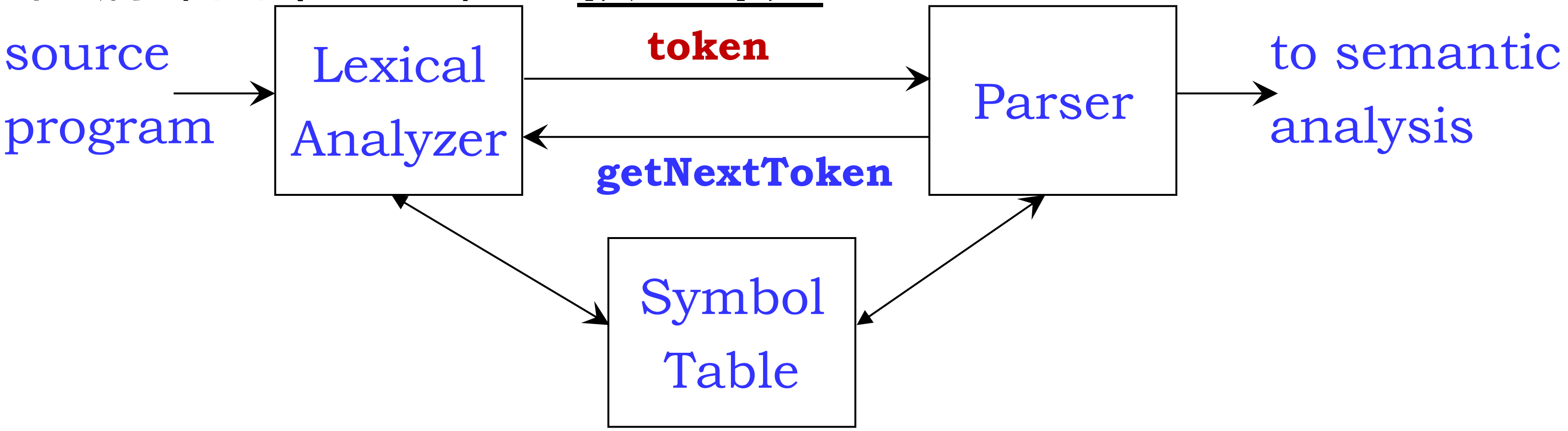
# Chapter 4 语法分析-自上而下分析

# Outlines

- 语法分析器的功能
- 自上而下分析(Top-down parsing)面临的问题
- LL（1）分析法
- 递归下降分析程序构造（ Recursive Descent Parsers ）
- 预测分析程序（ Predict Function ）
- LL（1）分析中的错误处理

# Parser function

- 高级语言的语法结构
  - 适合用**上下文无关文法**描述
- 语法分析器
  - 任务：分析与判定**程序的语法结构是否符合语法规则**
  - 工作本质：**根据产生式识别输入串是否为一个句子**
  - 在编译器中的地位：**<u>核心部分</u>**



# Parser

- 自上而下分析法
  - 从文法的开始符号出发，反复使用文法的产生式，寻找与输入符号串匹配的推导
  - 将文法开始符号做为语法树的根，向下逐步建立语法树，使语法树的结果正好是输入符号串
- 自下而上分析法
  - 从输入符号串开始，逐步进行归约，直至归约到文法的开始符号
  - 从输入符号串开始，以它做为语法树的结果，自底向上地构造语法树

# Example

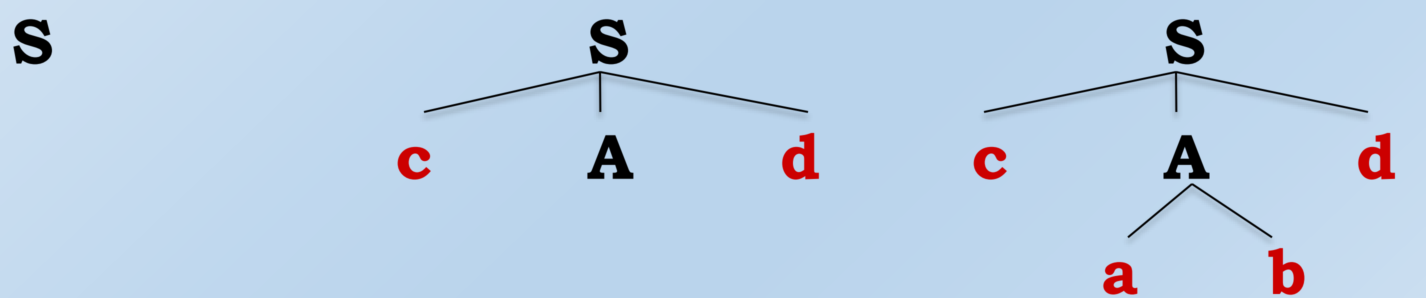
- 例1：文法G：　$S \rightarrow cAd$

  $A \rightarrow ab$

  $A \rightarrow a$

  识别输入串w=cabd是否为该文法的句子

自上而下分析



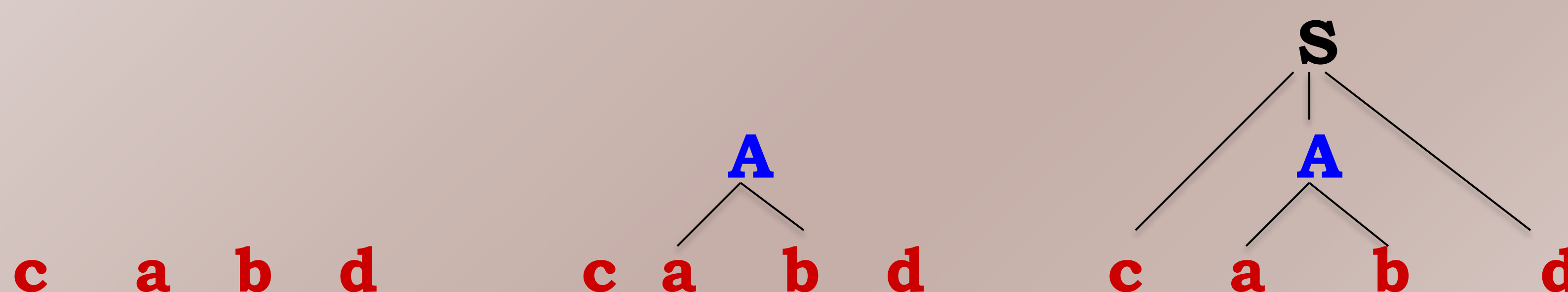

**推导过程：$S \Rightarrow cAd$　　　$cAd \Rightarrow \underline{cab}d$**

# Example

- 例2：文法G：  $S \rightarrow cAd$
  $A \rightarrow ab$
  $A \rightarrow a$

  识别输入串w=cabd是否为该文法的句子

自下而上分析




**规约过程构造的推导：** $cAd \Rightarrow c\underline{ab}d$  $S \Rightarrow cAd$

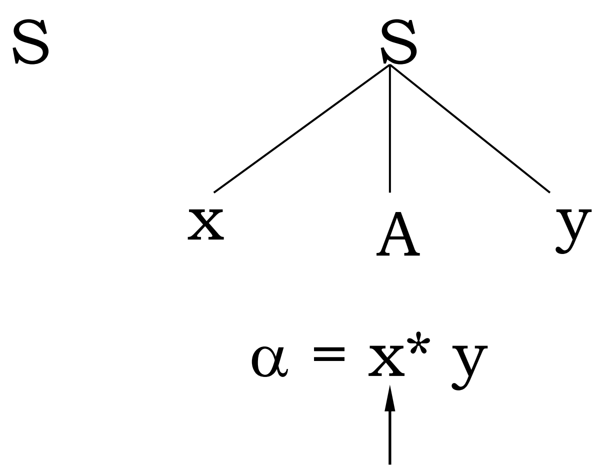
# Backtracking

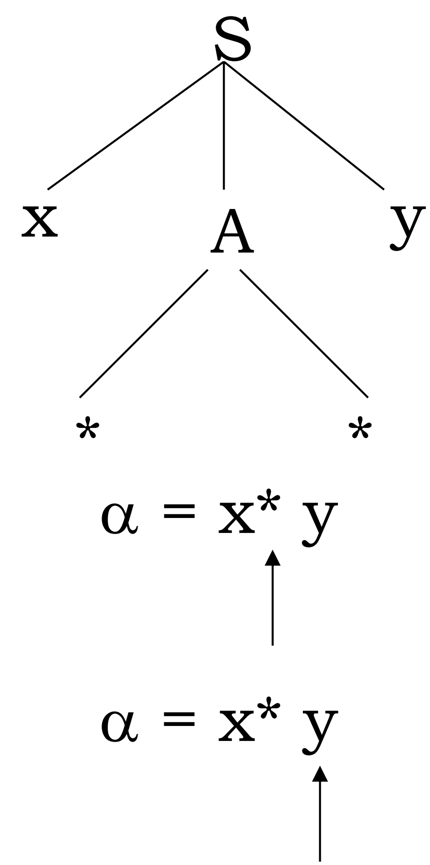
- 例3　假定有文法

  (1) $S \rightarrow xAy$

  (2) $A \rightarrow ** \mid *$

  分析输入串x*y(记为α)。



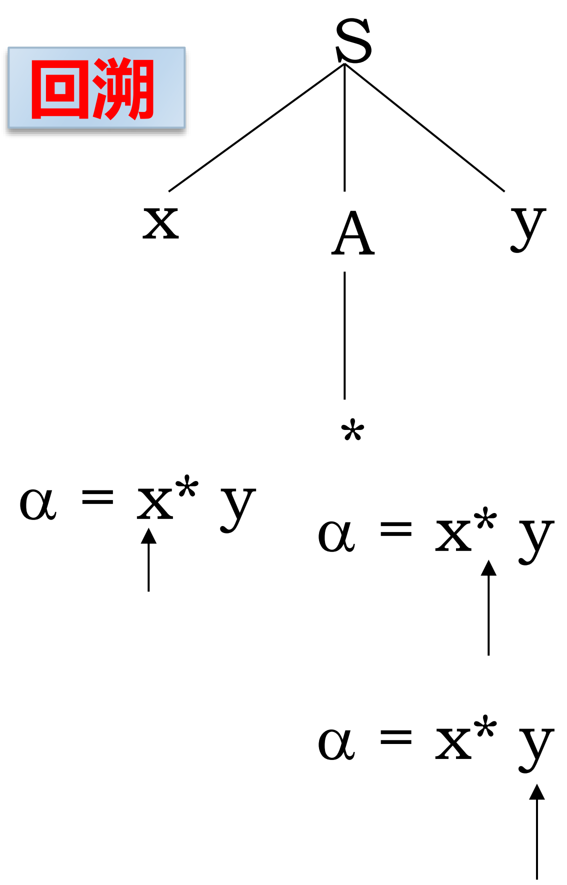
# Backtracking

- 例3 假定有文法

  (1) $S \rightarrow xAy$

  (2) $A \rightarrow ** \mid *$

  分析输入串x*y(记为α)。



# Top-down parsing, problems

- 自上而下
  - **为输入串寻找一个最左推导**
  - **对任何输入串，试图用一切可能的办法，从文法开始符号（根结点）出发，自上而下地为输入串建立一棵语法树**

主要问题：

$$A \to \alpha_1|\alpha_2|\ldots|\alpha_n$$

一个非终结符有多个产生式，如何选择？

最理想情况：

根据当前的第一个字符就可以判断

# LL(1)

- 符号说明：
  - A 大写文字，非终结符
  - a 小写字母，终结符
  - $\alpha$ 希腊字母，由非终结符和终结符组成的符号串

# LL(1)

例：$A \to Aa|Ba|a$，$B \to Ab|Bb|b$

直接左递归

$A \Rightarrow Aa$

$A \Rightarrow Ba \Rightarrow Aba$

间接左递归

左递归，没办法根据当前的一个字符来判断是否使用这条推导

⇩

消去左递归

# LL(1)

例：$A \to aa|aB|Ba$，$B \to b|c$

$A \Rightarrow aa$

$A \Rightarrow aB$

首字母都是a

只根据一个字符无法判断应该用哪一个

定义：$First(\alpha) = \{a|\alpha \overset{*}{\Rightarrow} a\ldots, a \in V^T\}$

若$\alpha \overset{*}{\Rightarrow} \varepsilon$则认为$\varepsilon \in First(\alpha)$

First(aa)={a}

First(aB)={a}

First(Ba)={b,c}

$\Rightarrow$ 提取公共左因子

$\Downarrow$

$First(\alpha_i) \cap First(\alpha_j) = \emptyset$

# LL(1)

例：$A \to ba|\varepsilon$，$B \to b$

$AB \Rightarrow baB$

$AB \Rightarrow \varepsilon B \Rightarrow b$

首字母都是b
只根据一个字符无法判断应该用哪一个

$$Follow(A) = \{a|S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V^T\}$$

若$S \overset{*}{\Rightarrow} \ldots A$，则认为$\# \in Follow(A)$
结束字符

若$\varepsilon \in First(A)$
则$First(A) \cap Follow(A) = \emptyset$

# LL(1)

- LL(1)文法： ⇦ 只用当前一个字符就能唯一确定推导
  - 文法不含左递归　　消除左递归算法
  - 对于文法中每一个非终结符A的各个产生式的候选首符集两两不相交　　公共左因子提取算法
    - 若$A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$，则$FIRST(\alpha_i) \cap FIRST(\alpha_j) = \phi$ $(i \neq j)$
  - 对文法中的每个非终结符A，若它能推导出空串，则首符集与后继符集不重合
    - 若$\varepsilon \in FIRST(A)$，则$FIRST(A) \cap FOLLOW(A) = \emptyset$

# LL(1) conditions

- LL(1)的含义
  - 第一个L
    - 从左至右扫描输入串 Left-to-right
  - 第二个L
    - 最左推导 Leftmost derivation
  - 1
    - 分析时每一步只需向前查看一个符号
- 对于一个LL(1)文法
  - 可以对其输入串进行有效的无回溯的自上而下分析

# Elimination of left recursion

- 直接消除产生式中的左递归
  - 假定关于非终结符P的规则为

$$P \to P\alpha \mid \beta$$

其中β不以P开头

  - 那么，我们可以把P的规则等价地改写为如下的非直接左递归形式：

$\beta\alpha^*$

$$P \to \beta P'$$

$$P' \to \alpha P' \mid \varepsilon$$

# Elimination of left recursion

- 一般而言，假定关于P的全部产生式是

$$P \rightarrow P\alpha_1 \mid P\alpha_2 \mid \ldots \mid P\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

**其中，每个α都不等于ε，而每个β都不以P开头**

**那么，消除P的直接左递归性就是改写这些规则：**

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

$$P' \rightarrow \alpha_1 P' \mid \alpha_2 P' \mid \ldots \mid \alpha_m P' \mid \varepsilon$$

# Elimination of left recursion

- 例4 文法
  - $E \rightarrow E + T \mid T$
  - $T \rightarrow T * F \mid F$
  - $F \rightarrow (E) \mid i$

$$P \rightarrow P\alpha \mid \beta$$

$$\Downarrow$$

$$P \rightarrow \beta P'$$

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

- 消去左递归
  - $E \rightarrow E+T \mid T$
  - $P=E,\ \alpha=+T,\ \beta=T$
    - $E \rightarrow TE'$
    - $E' \rightarrow +TE' \mid \varepsilon$

# Elimination of left recursion

- 例4 文法
  - $E \rightarrow E + T \mid T$
  - $T \rightarrow T*F \mid F$
  - $F \rightarrow (E) \mid i$

- 经消去直接左递归后变成：
  - $E \rightarrow TE'$
  - $E' \rightarrow +TE' \mid \varepsilon$
  - $T \rightarrow FT'$
  - $T' \rightarrow *FT' \mid \varepsilon$
  - $F \rightarrow (E) \mid i$

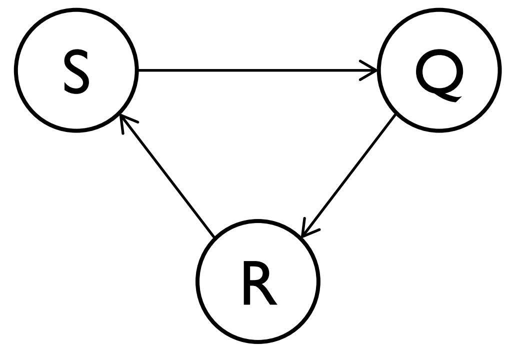
# Elimination of left recursion

- **间接左递归**
  - **S→Qc | c**
  - **Q→Rb | b**
  - **R→Sa | a**
- 多步推导： **S⇒Qc⇒Rbc⇒Sabc**
- 一个文法消除左递归的条件
  - **不含以ε为右部的产生式（空产生式）**
  - **不含回路，形如**

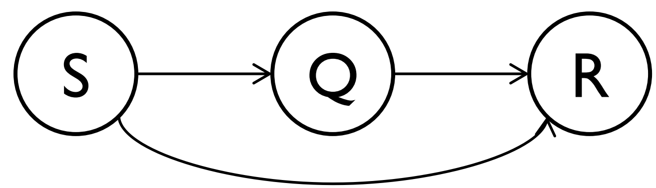
$$\mathbf{P} \overset{+}{\Rightarrow} \mathbf{P}$$

# Elimination of left recursion

- **间接左递归**
  - **S→Qc | c**
  - **Q→Rb | b**
  - **R→Sa | a**




间接左递归对应于回路




全序关系没有回路

规定一个生成顺序
可以消去间接左递归

# Elimination of left recursion

- **消除左递归的算法**
- 把文法G的所有非终结符按任一种顺序排列成$P_1$，$P_2$，…，$P_n$；按此顺序执行；
- FOR i:=1 TO n DO

  BEGIN

  FOR j:=1 TO i-1 DO

  **把形如$P_i \to P_j\gamma$的规则改写成**

  **$P_i \to \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$；**

  **(其中$P_j \to \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$是关于$P_j$ 的所有规则)**

  **消除关于$P_i$ 规则的直接左递归性**

  END
- 化简由第二步所得的文法
  - 即去除那些从开始符号出发永远无法到达的非终结符的产生规则

# Example

- 例5 考虑文法G(S)

$$S \to Qc \mid c$$
$$Q \to Rb \mid b$$
$$R \to Sa \mid a$$

  - 令它的非终结符的排序为**R、Q、S**
  - 对于R，不存在直接左递归
  - 把R代入到Q的有关候选后，把Q的规则变为

$$Q \to Sab \mid ab \mid b$$

    现在的Q不含直接左递归
  - 把Q代入到S的有关候选后，S变成

$$S \to Sabc \mid abc \mid bc \mid c$$

# Example

- **$S \rightarrow Sabc \mid abc \mid bc \mid c$** 存在直接左递归
  - 消除S的直接左递归后

    $S \rightarrow abcS' \mid bcS' \mid cS'$

    $S' \rightarrow abcS' \mid \varepsilon$

    $Q \rightarrow Sab \mid ab \mid b$

    $R \rightarrow Sa \mid a$
  - 关于Q和R的规则已是多余的，化简为

    $S \rightarrow abcS' \mid bcS' \mid cS'$

    $S' \rightarrow abcS' \mid \varepsilon$
  - 由于对非终结符排序的不同，最后所得的文法在形式上可能不一样。但它们都是**等价**的

# Example

- 例6 考虑文法G(S)

$$S \rightarrow Qc|c$$
$$Q \rightarrow Rb|b$$
$$R \rightarrow Sa|a$$

- 非终结符排序选为**S、Q、R**，那么，

$$R \rightarrow Qca \mid ca \mid a$$
$$R \rightarrow Rbca \mid bca \mid ca \mid a$$

- 最后所得的无左递归文法是：

$$S \rightarrow Qc \mid c$$
$$Q \rightarrow Rb \mid b$$
$$R \rightarrow bcaR' \mid caR' \mid aR'$$
$$R' \rightarrow bcaR' \mid \varepsilon$$

- 不同排序所得的文法的等价性是显然的

# Left Factoring

- 令G是一个不含左递归的文法
  - 对G的所有非终结符的每个候选，定义它的终结首符集 FIRST(α)为

$$FIRST(\alpha)=\{ a \mid \alpha \overset{*}{\Rightarrow} a\beta,\ a \in V_T,\ \alpha、\beta \in V^* \}$$

- 规则右部α的开始符号集包括所有终结符 a，使得规则右部α经过若干推导后得到的字符串以a为起始。
- 若$\alpha \overset{*}{\Rightarrow} \varepsilon$，则规定$\varepsilon \in FIRST(\alpha)$。

# Left Factoring

- 如果非终结符A的所有候选首符集两两不相交
  - 即A的任何两个不同候选 $\alpha_i$ 和 $\alpha_j$

$$FIRST(\alpha_i) \cap FIRST(\alpha_j) = \phi$$

- 当要求A匹配输入串时
  - A就能根据它所面临的第一个输入符号a，准确地指派某一个候选前去执行任务
  - 这个候选就是那个终结首符集含a的α

# Left Factoring

- 提取公共左因子
  - 假定关于A的规则是

    $A \to \delta\beta_1 \mid \delta\beta_2 \mid \ldots \mid \delta\beta_n \mid \gamma_1 \mid \gamma_2 \mid \ldots\ \gamma_m$

    (其中，每个$\gamma$不以$\delta$开头)
  - 那么，可以把这些规则改写成

    $A \to \delta A' \mid \gamma_1 \mid \gamma_2 \mid \ldots \mid \gamma_m$

    $A' \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$
  - 经过反复提取左因子，就能够把每个非终结符（包括新引进者）的所有候选首符集变成为**两两不相交**

# Example

- 例8 考察文法G：
  - **S → iCtS | iCtSeS | a**
  - **C → b**

- 解：由于S的前两个候选项中含有左因子iCtS，提取左因子之后，等价文法G'如下：
  - **S → iCtSS' | a**
  - **S' →eS |ε**
  - **C → b**

# LL(1) conditions

- 构造不带回溯的自上而下分析的文法条件
  - 文法不含左递归，
  - 对于文法中每一个非终结符A的各个产生式的候选首符集两两不相交
    - 若$A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$，则$FIRST(\alpha_i) \cap FIRST(\alpha_j) = \phi$ $(i \neq j)$
  - 对文法中的每个非终结符A，若它存在某个候选首符集包含ε，则

$$\mathbf{FIRST(\alpha_i) \cap FOLLOW(A) = \phi}$$

$$\mathbf{i = 1, 2, \ldots, n}$$

- 若一个文法G满足以上条件，则称G为LL(1)文法

# LL(1)

- LL(1)分析过程
  - 假设要用非终结符A进行匹配，面临的输入符号为a，A的所有产生式为$A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$
  - 若$a \in FIRST(\alpha_i)$，则
    - 指派$\alpha_i$执行匹配任务
  - 若a不属于任何一个候选首符集，则
    - 若ε属于某个$FIRST(\alpha_i)$且$a \in FOLLOW(A)$，则让A与ε自动匹配
    - 否则，a的出现是一种语法错误

# Recursive Descent Parser

- 递归下降分析器
  - 在不含左递归和每个非终结符的所有候选式的终结首符集都两两不相交条件下，构造一个不带回溯的自上而下分析程序
  - 该分析程序由一组递归过程组成，每个过程对应文法的一个非终结符

# Recursive Descent Parser

例9：**考虑文法：**

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

# Recursive Descent Parser

**E —> TE'**

- PROCEDURE E;
- BEGIN
- T;E'
- *END;*

**T —> FT'**

- PROCEDURE T;
- BEGIN
- F;T'
- *END;*

# Recursive Descent Parser

**E′—> +TE′ | ε**

- PROCEDURE E′;
- IF SYM=‘+’ THEN BEGIN
- ADVANCE;
- T;E′
- *END;*

**F —> (E) | i**

- PROCEDURE F;
- IF SYM=‘i’ THEN ADVANCE
- ELSE
- IF SYM=‘(’THEN
- BEGIN
- ADVANCE;
- E;
- IF SYM=‘)’ THEN ADVANCE
- ELSE ERROR
- END
- ELSE ERROR;

# Recursive Descent Parser

- 巴科斯范式
  - 元语言符号“→”和“|”
- 扩充的巴科斯范式（扩充几个元语言符号）
  - 用花括号{α}表示闭包运算$\alpha^*$
  - 用表示$\{\alpha\}_0^n$可任意重复0次至n次
  - 用方括号[α]表示$\{\alpha\}_0^1$
    - α的出现可有可无
    - 等价于α | ε

# Recursive Descent Parser

- 例10，通常的“实数”可定义为：

  **decimal→[sign]integer.{digit}[exponent]**

  **exponent→E[sign]integer**

  **integer→digit{digit}**

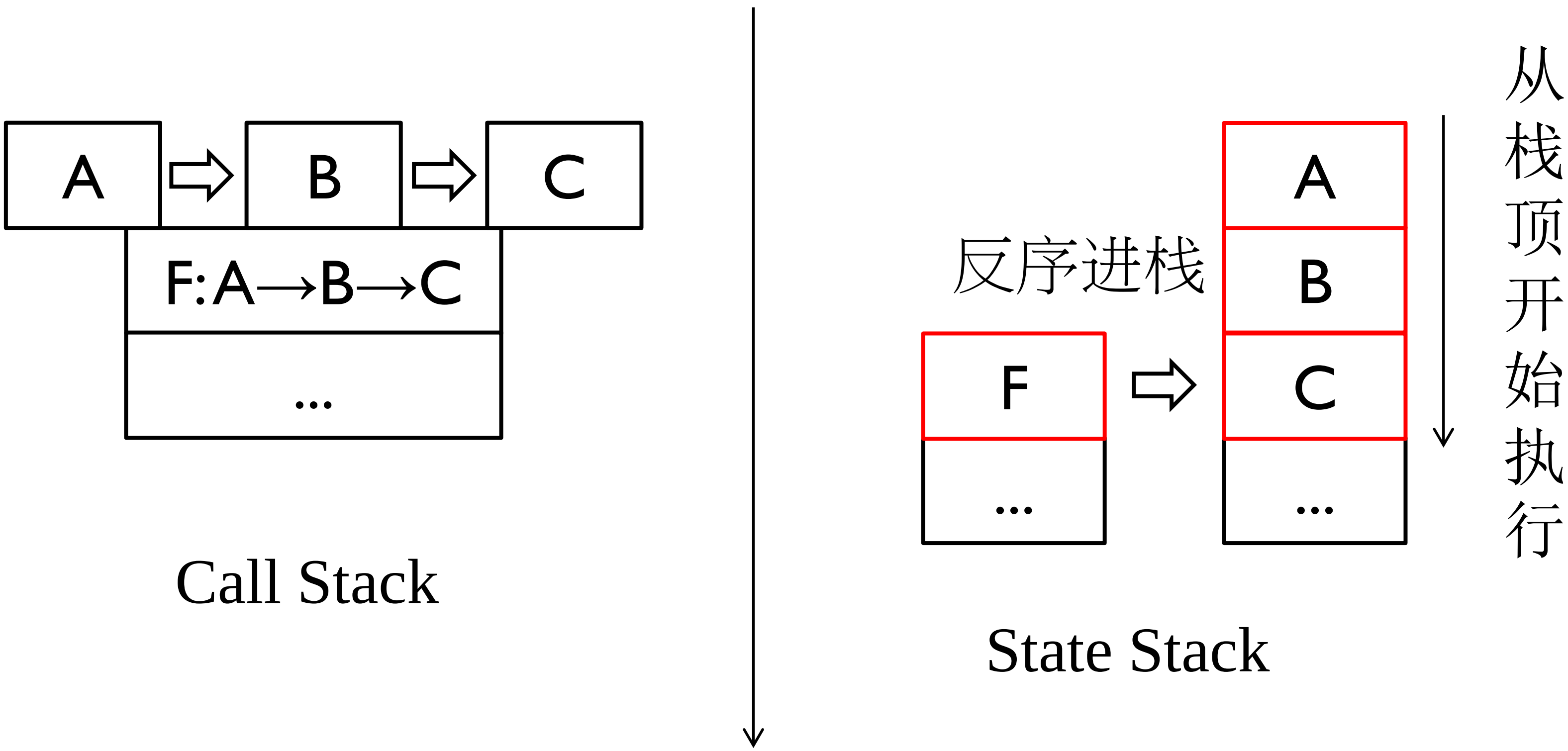
  **sign→ + | -**

- 用扩充的巴科斯范式来描述语法
  - 直观易懂
  - 便于表示左递归消去和因子提取

| | | |
|---|---|---|
| **E→T \| E+T** | | **E→T{+T}** |
| **T→F \| T*F** | ⇒ | **T→F{*F}** |
| **F→i \| (E)** | | **F→i \| (E)** |

# Predictive Parsers

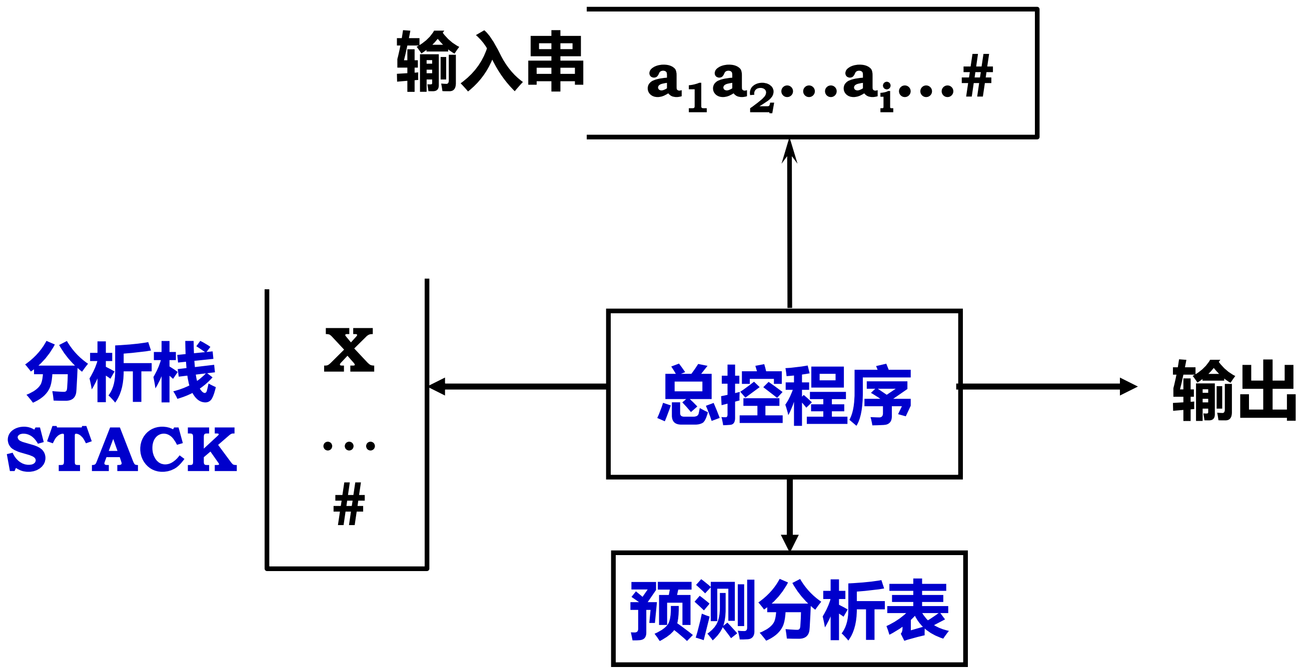
- 递归调用 → 栈



# Predictive Parsers

- 预测分析程序或LL(1)分析法工作原理：
  - 总控程序
  - 分析表 **M[A，*a*]**矩阵，$A \in V_N$，$a \in V_T$是终结符或'#'
  - 分析栈 **STACK** 用于存放文法符号

# Predictive Parsers




分析表：M[A，$a$]矩阵

$A \in V_N$，$a \in V_T$，#是输入串结束符，不是终结符

分析栈：用于存放文法符号

# Predictive Parsers

- 总控程序根据STACK栈顶符号X和当前输入符号$a$，执行下列三种动作之一
  - **若X = a = ‘#’，则宣布分析成功，停止分析**
  - **若X = a ≠‘#’，则把X从STACK栈顶逐出，让a指向下一个输入符号**
  - **若X是一个非终结符，则查看分析表M**
    - 若M[X，a]中存放着关于X的一个产生式，把X逐出STACK栈顶，把产生式的右部符号串按反序一一推进STACK栈(若右部符号为ε，则意味不推什么东西进栈)。在把产生式的右部符号推进栈的同时应做这个产生式相应的语义动作（目前暂且不管）
    - 若M[X，a]中存放着“出错标志”，则调用出错诊察程序ERROR

# Predictive Parsers

```
BEGIN
   首先把’ #‘入栈；然后把文法开始符号推入栈；
   把第一个输入符号读进a；FLAG：=TRUE；
   WHILE FLAG  DO
     BEGIN
        把栈顶符号上托出去并放在X中；
              IF X ∈ Vt  THEN
                 IF X=a  THEN  把下一个输入符号读进a
                 ELSE ERROR
              ELSE  IF X=‘#’ THEN
                 IF X=a  THEN  FLAG:=FALSE   ELSE ERROR
              ELSE IF M[X,a]={X → X1X2..XK} THEN
                 把XK，XK-1,..,X1——推进栈
              ELSE   ERROR
     END OF WHILE;
   STOP/*分析成功，过程完毕* /
END
```

# Predictive Parsers

例12：对于文法G(E)

**E→TE′**

**E′→+TE′ | ε**

**T→FT′**

**T′→*FT′ | ε**

**F→(E) | i**

- 对应的LL(1)分析表（预测分析表）如下

| | i | + | * | ( | ) | # |
|---|---|---|---|---|---|---|
| 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→i | | | F→(E) | | |

# Predictive Parsers

- 输入串为i*i+i，利用分析表进行预测分析

| 步骤 | 符号栈 | 输入串 | 所用产生式 |
|---|---|---|---|
| 0 | #E | i*i+i# | |
| 1 | #E'T | i*i+i# | E→TE' |
| 2 | #E'T'F | i*i+i# | T→FT' |
| 3 | #E'T'i | i*i+i# | F→i |
| 4 | #E'T' | *i+i# | |
| 5 | #E'T'F* | *i+i# | T'→*FT' |
| 6 | #E'T'F | i+i# | |
| 7 | #E'T'i | i+i# | F→i |
| 8 | #E'T' | +i# | |
| 9 | #E' | +i# | T'→ε |
| 10 | #E'T+ | +i# | E'→+TE' |
| 11 | #E'T | i# | |
| 12 | #E'T'F | i# | T→FT' |
| 13 | #E'T'i | i# | F→i |
| 14 | #E'T' | # | |
| 15 | #E' | # | T'→ε |
| 16 | # | # | E→ε |

# LL(1)

- 如果文法G是左递归或二义的，那么分析表M至少含有一个多重定义入口
- 因此，消除左递归和提取左因子将有助于获得无多重定义的分析表M
- 一个文法,若它的分析表M不含多重定义入口,则称它是一个LL(1)文法
- 一个文法G的预测分析表M不含多重定义入口，当且仅当该文法为LL(1)的

# LL(1)

- 一个文法G为LL(1)的，当且仅当对于文法G中每一个非终结符A的任何两个产生式$A \rightarrow \alpha \mid \beta$，下面的条件成立：
  - 1. 文法不含左递归
  - 2. $FIRST(\alpha) \cap FIRST(\beta) = \phi$
  - 3. 若$\beta \overset{*}{\Rightarrow} \varepsilon$ (即$\varepsilon \in FIRST(\beta)$)，则

    $FIRST(\alpha) \cap FOLLOW(A) = \phi$

# Proof

- 对 A-> α | β，若条件（1）不成立，
- 则 FIRST(α)∩ FIRST(β) ≠ ϕ，
- 假设，FIRST(α)∩ FIRST(β) = { a }
- 那么，当A面临输入符号a，而a同时属于FIRST(α)和 FIRST(β)，则分析无法继续进行下去，因为不能确定用哪一个候选式可以保证一定能够得到匹配而不进行回溯。
- 实质就是分析表的M[A, a]中包含两条候选式

  **A -> α**　　　　**A -> β**
- 反之，分析表的M[A, a]中只包含一条候选式则意味着可以进行确定性的无回溯的分析。

# Proof

- 对 A-> α | β，若$\beta \stackrel{*}{\Rightarrow} \varepsilon$，且条件（2）不成立，
- 则 FIRST(α )∩FOLLOW(A) ≠ φ
- 那么，当A面临输入符号a时，
- 若选候选式A -> α，则由于a∈ FIRST(α)可以使a一定得到匹配；
- 同时，若选候选式A -> β也可以满足要求，
- 这是由 $\beta \stackrel{*}{\Rightarrow} \varepsilon$，而a∈ FOLLOW(A)。
- 因此，不能确定用哪一个候选式可以保证一定能够得到a的后续匹配而不进行回溯。

# Proof

- 实质同样是由于分析表的M[A, a]中包含两条候选式:
- $A \rightarrow \alpha \qquad A \rightarrow \beta$

- 而这与LL(1)文法的定义互相矛盾。

- 综上所述，若某文法为LL(1)文法，则该文法一定满足这两个条件，它意味着进行自上而下分析时可以对候选式进行不带回溯的确定性的选择。

# Predict table

- 构造FIRST集
  - 计算单个文法符号的FIRST集
  - 计算一组文法符号串的FIRST集
- 构造FOLLOW集
- 构造预测分析表M[A,a]

# FIRST & FOLLOW

- 对于单个文法符合X，应用下列规则，直到没有终结符或ε可加到FIRST(X)集为止
  - **如果X是终结符，则FIRST(X)是 {X}**
  - **如果X是非终结符，且X→ε是一个产生式，则将ε加到FIRST(X)中；若有X→a…，则将a加入到FIRST(X)**
  - **如果X是非终结符，且X→Y…， 则将First(Y)\{ε}放入FIRST(X)**
  - **如果X是非终结符，且$X \rightarrow Y_1Y_2 \ldots Y_k$（k≥1）是一个产生式，则**
    *
    - 若$\varepsilon \in FIRST(Y_j)$（j=1,2,…,k），则将ε加入FIRST(X)
    - 当某终结符$a \in FIRST(Y_i)$ $(1<i\le k)$ 且 $Y_1Y_2\ldots Y_{i-1} \Rightarrow \varepsilon$时，就把a 加入到FIRST(X)中去

# FIRST & FOLLOW

- 假设一组文法符号串由$X_1X_2\ldots X_n$构成
  - **将集合FIRST($X_1$)中的除ε的终结符号加入FIRST($X_1X_2\ldots X_n$)**
  - **如果ε∈FIRST($X_1$)，那么将集合FIRST($X_2$)中除ε的终结符号也加入FIRST($X_1X_2\ldots X_n$)**
  - **依次类推**
  - **如果$X_1,X_2,\ldots,X_n$中每一个文法符号的FIRST集中都有ε,那么把ε也加入到FIRST($X_1X_2\ldots X_n$)中**

# FIRST & FOLLOW

- 计算文法中每个非终结符A的FOLLOW(A)，应用如下的三条规则，直到没有任何一个终结符能被添加到任何非终结符的FOLLOW集当中为止
  - **如果S是文法的开始符号，那么把#添加进FOLLOW(S)（ #是输入串的结束符)**
  - **如果有一个产生式A →αBβ，那么将集合FIRST(β)中除ε外的所有元素加入到FOLLOW(B)中**
  - **如果有一个产生式 A →αB，或有一个产生式 A →αBβ且ε∈FIRST(β), 那么将集合FOLLOW(A)中的所有元素加入到集合FOLLOW(B)中**

# Follow

- 1. S是开始符号，#在Follow(S)中
- 2. $A \rightarrow \alpha B\gamma$，得到$First(\gamma)/\{\varepsilon\} \subset Follow(B)$
  - $c \in First(\gamma)$，得到$\gamma \overset{*}{\Rightarrow} c\ldots$
  - $S \overset{*}{\Rightarrow} \ldots A \ldots \overset{*}{\Rightarrow} \ldots \alpha B\gamma \ldots \overset{*}{\Rightarrow} \ldots \alpha Bc \ldots$，得到$c \in Follow(B)$
- 3. $A \rightarrow \alpha B$或$A \rightarrow \alpha B\gamma$且$\varepsilon \in First(\gamma)$，得到$Follow(A) \subset Follow(B)$
  - $c \in Follow(A)$，得到$S \overset{*}{\Rightarrow} \ldots Ac \ldots$
  - $S \overset{*}{\Rightarrow} \ldots Ac \ldots \overset{*}{\Rightarrow} \ldots \alpha B\gamma c \ldots \overset{*}{\Rightarrow} \ldots \alpha B\varepsilon c \ldots \overset{*}{\Rightarrow} \ldots \alpha Bc \ldots$，得到$c \in Follow(B)$

# Predict table

- 在对文法G的每个非终结符A及其任意候选α都构造出FIRST(α)和FOLLOW(A)之后，用它们来构造G的分析表M[A,a]
  - **对文法G的每个产生式A→α执行第2步和第3步**
  - **对每个终结符a ∈FIRST(α)，把A→α加至M[A,a]中**
  - **若ε∈FIRST(α)，则对任何b∈FOLLOW(A)把A→α加至M[A,b]中**
  - **把所有无定义的M[A,a]标上“出错标志”**

# Example

例13：对于文法G(E)，计算FIRST和FOLLOW集

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

First集

**E→TE′** ⟶ {(,i)

**E′→+TE′ | ε** → {+,ε}

**T→FT′** ⟶ {(,i)

**T′→*FT′ | ε** → {*,ε}

**F→(E) | i** → {(,i}

| | Fi | Fo |
|---|---|---|
| E | {(,i} | |
| E’ | {+,ε} | |
| T | {(,i} | |
| T’ | {*,ε} | |
| F | {(,i} | |

# Example

例13：对于文法G(E)，计算FIRST和FOLLOW集

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

Follow集

计算Fo(E)

E起始 ⇨ #

F→(E) | i ⇨ )

{#,)}

| | Fi | Fo |
|---|---|---|
| E | {(,i} | |
| E’ | {+,ε} | |
| T | {(,i} | |
| T’ | {*,ε} | |
| F | {(,i} | |

# Example

例13：对于文法G(E)，计算FIRST和FOLLOW集

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

Follow集

计算Fo(F)

$$\left.\begin{array}{l} \varepsilon \in Fi(T') \quad \Rightarrow \quad Fi(T') \\ T \rightarrow FT' \qquad\;\; Fo(T) \\ \\ \varepsilon \in Fi(T') \quad \Rightarrow \quad Fi(T') \\ T' \rightarrow {*}FT' \qquad Fo(T') \end{array}\right\} \{*, Fo(T), Fo(T')\}$$

| | Fi | Fo |
|---|---|---|
| E | {(,i} | {#,)} |
| E’ | {+,ε} | |
| T | {(,i} | |
| T’ | {*,ε} | |
| F | {(,i} | |

# Example

例13：对于文法G(E)，计算FIRST和FOLLOW集

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

Follow集

计算Fo(T')

$$\left.\begin{array}{l} T' \rightarrow *FT' \Rightarrow Fo(T') \\ \\ T \rightarrow FT' \Rightarrow Fo(T) \end{array}\right\} \{Fo(T)\}$$

| | Fi | Fo |
|---|---|---|
| E | {(,i} | {#,)} |
| E' | {+,ε} | |
| T | {(,i} | |
| T' | {*,ε} | |
| F | {(,i} | {*,Fo(T),Fo(T')} |

# Example

例13：对于文法G(E)，计算FIRST和FOLLOW集

$E \to TE', E' \to +TE' \mid \varepsilon, T \to FT',$
$T' \to *FT' \mid \varepsilon, F \to (E) \mid i$

Follow集

计算Fo(T)

E'→+TE'
ε∈Fi(E')
⇨ Fi(E') Fo(E')

E→TE'
ε∈Fi(E')
⇨ Fi(E') Fo(E)

{+,#,),Fo(E')}

| | Fi | Fo |
|---|---|---|
| E | {(,i} | {#,)} |
| E' | {+,ε} | |
| T | {(,i} | |
| T' | {*,ε} | {Fo(T)} |
| F | {(,i} | {*,Fo(T),Fo(T')} |

# Example

例13：对于文法G(E)，计算FIRST和FOLLOW集

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

Follow集

计算Fo(E’)

E’→+TE’ ⇨ Fo(E’)

E→TE’ ⇨ Fo(E)

{#,)}

| | Fi | Fo |
|---|---|---|
| E | {(,i} | {#,)} |
| E’ | {+,ε} | |
| T | {(,i} | {+,#,),Fo(E’)} |
| T’ | {*,ε} | {Fo(T)} |
| F | {(,i} | {*,Fo(T),Fo(T’)} |

# Example

例：对于文法G(E)

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

构造每个非终结符的FIRST和FOLLOW集：

FIRST(E) ={ (, i }

FIRST(E′)={ +, ε }

FIRST(T) ={ (, i }

FIRST(T′)={ *, ε }

FIRST(F) ={ (, i }

FOLLOW(E) ={ ), # }

FOLLOW(E′)={ ), # }

FOLLOW(T) ={ +, ), # }

FOLLOW(T′)={ +, ), # }

FOLLOW(F) ={ *, +, ), #}

# Example

例：对于文法G(E)

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

构造每个非终结符的FIRST和FOLLOW集：

FIRST(E) ={ (, i }　　FOLLOW(E) ={ ), # }

FIRST(E′)={ +, ε }　　FOLLOW(E′)={ ), # }

FIRST(T) ={ (, i }　　FOLLOW(T) ={ +, ), # }

FIRST(T′)={ *, ε }　　FOLLOW(T′)={ +, ), # }

FIRST(F) ={ (, i }　　FOLLOW(F) ={ *, +, ), #}

| | i | + | * | ( | ) | # |
|---|---|---|---|---|---|---|
| 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→i | | | F→(E) | | |

# Example

- 例14 考察文法G：

$$S \rightarrow iCtS \mid iCtSeS \mid a$$

$$C \rightarrow b$$

提取公因式以后

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

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

$$C \rightarrow b$$

计算FIRST，FOLLOW集

# Example

- 例14 考察文法G：

$$S \rightarrow iCtS \mid iCtSeS \mid a$$

$$C \rightarrow b$$

提取公因式以后

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

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

$$C \rightarrow b$$

- FIRST( S ) = { i， a }　　FOLLOW( S ) = { #， e }
- FIRST( S’)= {e， ε}　　FOLLOW( S’)={ #， e }
- FIRST( C ) = { b }　　FOLLOW( C ) = { t }

# Example

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

- FIRST( S ) = { i, a }　　FOLLOW( S ) = { #, e }
- FIRST( S')= {e, ε}　　FOLLOW( S')={ #, e }
- FIRST( C ) = { b }　　FOLLOW( C ) = { t }

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

# Example

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

- FIRST( S ) = { i, a }    FOLLOW( S ) = { #, e }
- FIRST( S')= {e, ε}    FOLLOW( S')={ #, e }
- FIRST( C ) = { b }    FOLLOW( C ) = { t }

| | a | b | e | i | t | # |
|---|---|---|---|---|---|---|
| S | $S \rightarrow a$ | | | $S \rightarrow iCtSS'$ | | |
| S' | | | $S' \rightarrow \varepsilon$<br>$S' \rightarrow eS$ | | | $S' \rightarrow \varepsilon$ |
| C | | $C \rightarrow b$ | | | | |

# Error handling

- 发现错误
  - 栈顶的终结符与当前输入符不匹配
  - 非终结符A于栈顶，面临的输入符为a，但分析表M的M[A, a]为空
- “应急”恢复策略
  - 跳过输入串中的一些符号直至遇到“同步符号”为止
- 同步符号的选择
  - “synch”表示由相应非终结符的后继符号集得到的同步符号

# Error handling

- 例15 把FOLLOW(A)中的所有符号作为A的同步符号

FIRST(E) ={ (, i }　　FOLLOW(E) ={ ), # }

FIRST(E′)={ +,  ε }　　FOLLOW(E′)={ ), # }

FIRST(T) ={ (,  i }　　FOLLOW(T) ={ +, ), # }

FIRST(T′)={ *,  ε }　　FOLLOW(T′)={ +, ), # }

FIRST(F) ={ (, i }　　FOLLOW(F) ={ *, +, ), # }

| | i | + | * | ( | ) | # |
|---|---|---|---|---|---|---|
| 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→i | synch | synch | F→(E) | synch | synch |

# Error handling

- 把FOLLOW(A)中的所有符号作为A的同步符号
  - 跳过输入串中的一些符号直至遇到这些“同步符号”，把A从栈中弹出，可使分析继续
- 把FIRST(A)中的符号加到A的同步符号集
  - 当FIRST(A)中的符号在输入中出现时，可根据A恢复语法分析
- 若不能匹配栈顶的终结符号，一种简单的想法是
  - 弹出栈顶的这个终结符号，并发出一条信息，说明已经插入这个终结符，继续进行语法分析

| 步骤 | 符号栈 | 输入串 | 所用产生式 |
|---|---|---|---|
| 0 | #E | )i*+i# | 错，跳过) |
| 1 | #E | i*+i# | i∈FIRST(E) |
| 2 | #E'T | i*+i# | E→TE' |
| 3 | #E'T'F | i*+i# | T→FT' |
| 4 | #E'T'i | i*i+i# | F→i |
| 5 | #E'T' | *+i# | |
| 6 | #E'T'F* | *+i# | T'→*FT' |
| 7 | #E'T'F | +i# | 错，M[F,+]=synch |
| 8 | #E'T' | +i# | F已弹出栈 |
| 9 | #E' | +i# | T'→ε |
| 10 | #E' | +i# | E'→+TE' |
| 11 | #E'T+ | +i# | |
| 12 | #E'T | i# | T→FT' |
| 13 | #E'T'F | i# | F→i |
| 14 | #E'T'i | i# | |
| 15 | #E'T' | # | T'→ε |
| 16 | #E' | # | E'→ε |
| 17 | # | # | |

# 练习

文法：G(A): A→a|Ba, B→b|Ab

a) 尝试转化为LL(1)文法

b) 如果为LL(1)，画出预测分析表

Step:

1. 消去左递归
2. 提取公共左因子
3. 计算First,Follow集
4. 画出预测分析表

# 练习

文法：G(A): A→a|Ba, B→b|Ab

a) 尝试转化为LL(1)文法

b) 如果为LL(1)，画出预测分析表

Step1 消去左递归

A→a|Ba

B→b|(a|Ba)b→b|ab|Bab

A→a|Ba

B→bB’|abB’

B’→abB’|ε

# 练习

文法：G(A): A→a|Ba, B→b|Ab

a) 尝试转化为LL(1)文法

b) 如果为LL(1)，画出预测分析表

Step2 消去公共左因子

A→a | Ba ⇨ Fi(a)={a}

B→bB’| abB’ Fi(Ba)={a,b} ⇨ A→a | bB’a | abB’a

B’→abB’| ε

⇩

A→aA’| bB’a

A’→bB’a | ε

B’→abB’| ε

Fi(B’)={a,ε}

Fi(B)={a,b}

Fi(A)={a,b}

# 练习

文法：G(A): A→a|Ba, B→b|Ab

a) 尝试转化为LL(1)文法

b) 如果为LL(1)，画出预测分析表

Step3 计算First,Follow集

A→aA'|bB'a

A'→bB'a|ε

B'→abB'|ε

| | First | Follow |
|---|---|---|
| A | a,b | # |
| A' | b,ε | # |
| B' | a,ε | a |

B'遇到a时无法判别

# Exercise

- 例16 文法G[V]:

  **V→N | N[E]**

  **E→V | V+E**

  **N→i**

  是否为LL(1)文法？若不是，如何改造成LL(1)文法？

# Exercise

- 例16 文法G[V]：

  **V→N | N[E]**

  **E→V | V+E**

  **N→i**

  是否为LL(1)文法？若不是，如何改造成LL(1)文法？

- 解：LL(1)文法的基本条件是不含左递归和回溯（公共左因子），而G[V]中含有回溯，所以先消除回溯得到文法G'[V]：

  G'[V]：　V→NV'　V'→ε | [E]

  　　　　E→VE'　E'→ε | +E

  　　　　N→i

  - 由LL(1)文法的充要条件可证
    - G'[V]是LL(1)文法

# Exercise

- 例17 文法G[s]:

  **S→BA**

  **A→BS | d**

  **B→aA | bS | c**

  - 证明文法G是LL(1)文法
  - 构造LL(1)分析表
  - 写出句子adccd的分析过程

# Exercise

- 对于文法G[s]:  S→BA  A→BS|d  B→aA|bS|c
- FIRST集
  - FIRST(B)={a, b, c}; FIRST(A)={a, b, c, d};FIRST(S)={a, b, c}
- FOLLOW集
  - FOLLOW(S)={#}
  - 对S→BA有
    - FIRST(A)\{ε}加入FOLLOW(B)，即FOLLOW(B)={a, b, c, d }
  - 对A→BS有
    - FIRST(S)\{ε}加入FOLLOW(B)，即FOLLOW(B)={a, b, c, d }
  - 对B→aA有
    - FOLLOW(B)加入FOLLOW(A)，即FOLLOW(A)={a, b, c, d }
  - 对B→bS有
    - FOLLOW(B)加入FOLLOW(S)，即FOLLOW(S)={#, a, b, c, d }

# Exercise

- 由$A \rightarrow BS \mid d$得
  - **FIRST(BS) ∩FIRST(d) = { a, b, c } ∩{d} = Φ**
- 由$B \rightarrow aA \mid bS \mid c$得
  - **FIRST(aA)∩FIRST(bS)∩FIRST(c)={a}∩{b}∩{c}= Φ**
- 由于文法G[s]不存在形如 $\beta \rightarrow \varepsilon$的产生式，故无需求解形如FIRST(α)∩FOLLOW(A)的值
- 也即，文法G[S]是一个LL(1)文法

# Exercise

- 由G[s]：S→BA  A→BS|d  B→aA|bS|c的

  FIRST(B)={a, b, c};    FOLLOW(B)={a, b, c, d };

  FIRST(A)={a, b, c, d};  FOLLOW(A)={#, a, b, c, d };

  FIRST(S)={a, b, c}。   FOLLOW(S)={#, a, b, c, d }

  可构造LL(1)预测分析表如下：

| | a | b | c | d | # |
|---|---|---|---|---|---|
| S | S→BA | S→BA | S→BA | | |
| A | A→BS | A→BS | A→BS | A→d | |
| B | B→aA | B→bS | B→c | | |

# Exercise

| 栈 | 当前输入符号 | 输入串 | 说明 |
|---|---|---|---|
| #S | a | dccd# | S→BA |
| #AB | a | dccd# | B→aA |
| #AAa | a | dccd# | |
| #AA | d | ccd# | A→d |
| #Ad | d | ccd# | |
| #A | c | cd# | A→BS |
| #SB | c | cd# | B→c |
| #Sc | c | cd# | |
| #S | c | d# | S→BA |
| #AB | c | d# | B→c |
| #Ac | c | d# | |
| #A | d | # | A→d |
| #d | d | # | |
| # | | # | 分析成功 |

# 练习

- 判断下列文法是否LL（1）文法？

(1) S->Abc, A->a | ε, B->b | ε　　对

(2) S->Ab, A->a | B | ε, B->b | ε　　错

(3) S->ABBA, A->a | ε, B->b | ε　　错

(4) S->aSe | B, B->bBe | C, C->cCe | d　　对

# 练习

- 设文法G(A)

**A->iB*e, B->SB | ε, S->[eC] | .i, C->eC | ε**

试用LL（1）分析技术识别输入串**i.i*e**是否为该文法的句子

| | i | * | e | [ | ] | . | # |
|---|---|---|---|---|---|---|---|
| A | **A->iB*e** | | | | | | |
| B | | **B-> ε** | | **B->SB** | | **B->SB** | |
| C | | | **C->eC** | | **C-> ε** | | |
| S | | | | **S->[eC]** | | **S-> .i** | |

# 练习

- 对下面文法：

**Expr -> -Expr**

**Expr -> (Expr) | Var ExprTail**

**ExprTail -> -Expr | ε**

**Var -> id VarTail**

**VarTail -> (Expr) | ε**

（1）构造LL（1）分析表

（2）给出句子**id--id((id))**分析过程

# 练习

- 给出文法G：

**S -> aSb | P**

**P -> bPc | bQc**

**Q -> Qa | a**

消除左递归，提取左公因子以后是不是LL（1）文法？请证明。

# Exercises

- 课本第81-82页，下堂课我们将讲解
  - 1
  - 2 (1) (2) (3)

# 第2题

- **E→TE';**
- **E'→+E | ε ;**
- **T→FT' ;**
- **T' →T | ε ;**
- **F→PF' ;**
- **F'→ *F' |ε ;**
- **P→(E)|a|b|∧**

# 第2题

| | a | b | ∧ | + | * | ( | ) | # |
|---|---|---|---|---|---|---|---|---|
| E | E→TE' | E→TE' | E→TE' | | | E→TE' | | |
| E' | | | | E'→+E | | | E'→ε | E'→ε |
| T | T→FT' | T→FT' | T→FT' | | | T→FT' | | |
| T' | T' →T | T' →T | T' →T | T' →ε | | T' →T | T' →ε | T' →ε |
| F | F→PF' | F→PF' | F→PF' | | | F→PF' | | |
| F' | F'→ε | F'→ε | F'→ε | F'→ε | F'→ *F' | F'→ε | F'→ε | F'→ε |
| P | P→a | P→b | P→∧ | | | P→(E) | | |