Lecture 5: Top-Down Parsing

Xiaoyuan Xie 谢晓园
xxie@whu.edu.cn
计算机学院E301

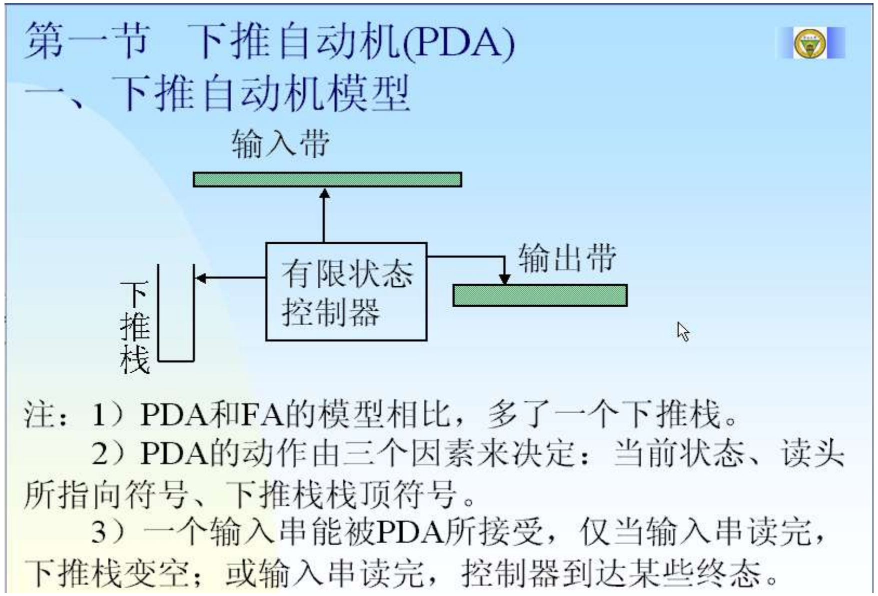
回顾

- CFG，CFG v.s. RG

5.1 重要定义及计算

5.1 重要定义及计算

- **自顶向下分析可以被看作是为输入串构造语法分析树的问题，也可以看作一个寻找输入串的最左推导的过程**
  - 下推自动机PDA



5.1 重要定义及计算

- **自顶向下分析可以被看作是为输入串构造语法分析树的问题，也可以看作一个寻找输入串的最左推导的过程**
  - Q2：在推导的每一步，对非终结符号A，应用哪个产生式，以可能产生于输入串相匹配的终结符号串

5.1 重要定义及计算

■ 例子：最左推导

P:
(1) Z → aBeA
(2) A → Bc
(3) B → d
(4) B → bB
(5) B → ε

a	b	e	c

读头 从左向右移动读头，读入字符串

输入（读头所在）	栈内符号	分析	执行推导	匹配
abec	Z	Z的哪一个产生式以a开头？－(1)	aBeA	a
bec	BeA	B的哪一个产生式以b开头？－(4)	bBeA	b
ec	BeA	B的哪一个产生式能够以e开头？－(5)	εeA	e
c	A	A的哪一个产生式能够以c开头？－(2)(5)		

5.1 重要定义及计算

- **例子：最左推导**

P:
(1) Z → aBeA
(2) A → Bc
(3) B → d
(4) B → bB
(5) B → ε

输入（读头所在）	栈内符号	分析	执行推导	匹配
c	A	A的哪一个产生式能够以c开头？-(2)	Bc	
c	Bc	A的哪一个产生式能够以c开头？-(5)	εc	c

a	b	e	c

读头

从左向右移动读头，读入字符串

5.1 重要定义及计算

- **Q2如何解决？**
  - 在推导的每一步，对非终结符号A，应用哪个产生式，以可能产生于输入串相匹配的终结符号串
    - E.g. 有E→ aT|abT|acT, 先推导至 E+a，下一步该用aT, bT, 还是cT去替换T？--- 取决于读头下的字符

5.1 重要定义及计算

- **Q2解决的关键**
  - 对句型中的*非终极符*选择*合适的产生式*进行下一步推导；
  - 选择产生式时，参考输入的token序列信息更有效，否则有可能产生回溯
  - 向前看几个输入符号？对一般的程序设计语言而言，向前看1个输入符号，即只看当前token（向前看一个token）就足够了

Q1：选择最左非终极符进行替换更合适，便于处理；

5.1 重要定义及计算

- **Q2解决的关键-预测分析技术：**
  - 以开始符号作为初始的当前句型

    => 每次为最左边的非终结符号选择适当的产生式(Q1)

    ⇒ 通过查看下面k个输入符号(token)来对句型中的*非终极符*选择*合适的产生式*进行下一步推导

    ⇒ 试图从开始符号推导出输入符号串，
  - 对一般的程序设计语言而言，向前看1个token就足够(k=1)

5.1 重要定义及计算

- **怎么实现向前看一个token?**
  - Predict($A\rightarrow \alpha$)
    - 面向产生式定义的，对产生式可能推导出的串进行预测，作为对非终极符替换时选择产生式的依据；{a | $S \Rightarrow^{+} \alpha A\beta \Rightarrow \alpha\, \gamma\, \beta \Rightarrow^{*} \alpha\ a\ \beta'$ }, 其中 $\alpha \in V_T^*$, S 是开始符.
  - First(α)
    - 面向符号串(产生式的右部)定义的，当$\varepsilon \notin$First(α)时，直接作为产生式($A \rightarrow \alpha$)的预测符；
  - Follow(A)
    - 面向非终极符(产生式的左部)定义的,当$\varepsilon \in$First(α)时，构成产生式($A \rightarrow \alpha$)的预测符集的一部分

5.1 重要定义及计算

- **First(α)**
  - 输入α：由任意文法符号(V_T or V_N)组成的符号串(即产生式的右部)
  - 返回：可从α推导得到的串的首符号集合
  - Formal Definition: First(α) = $\{a \mid \alpha \Rightarrow^* a\beta, a \in V_T\}$, 特别地，如果 $\alpha \Rightarrow^* \varepsilon$ 则 First(α)= First(α) $\cup$ $\{\varepsilon\}$

5.1 重要定义及计算

- **First(α)**
  - 计算方法

    [1] 如果 $\alpha = \varepsilon$, 则有，First(α) = $\{\varepsilon\}$

    [2] 如果 $\alpha = a$, $a \in V_{T,}$ 则有，First(α) = $\{a\}$

    [3] 如果 $\alpha = A$, $A \in V_{N,}$ $A \rightarrow \beta 1 \mid \beta 2 \mid \ldots\ldots \mid \beta n$，则有，
    First(α) = = $\cup$ First(βi) $1 \le i \le n$

5.1 重要定义及计算

- **First(α)**
  - 计算方法

[4] 如果 $\alpha = X_1 X_2 \ldots\ldots X_{i-1} X_i \ldots\ldots X_n$ 其中$X_i \in V_T \cup V_N$

(1) 先使用下列算法，为每个X计算每个First(X)

(a) 若 $X \in V_T$，则 $FIRST(X) = \{X\}$

(b) 若 $X \in V_N$，且有产生式 $X \to a\cdots$，$a \in V_T$ 则 $a \in FIRST(X)$。

(c) 若 $X \in V_N$，$X \to \varepsilon$，则 $\varepsilon \in FIRST(X)$。

(d)若 $X \in V_N$，$Y_1, Y_2 \cdots Y_i$ 都 $\in V_N$，而有产生式 $X \to Y_1 Y_2 \cdots Y_n$。当 $Y_1, Y_2 \cdots, Y_{i-1}$ 都 $\stackrel{*}{\Rightarrow} \varepsilon$ 时，(其中 $1 \leqslant i \leqslant n$)，则 $FIRST(Y_1) - \{\varepsilon\}$，$FIRST(Y_2) - \{\varepsilon\}$，$\cdots FIRST(Y_{i-1}) - \{\varepsilon\}$，$FIRST(Y_i)$都包含在 $FIRST(X)$中。

(e)当(d)中所有 $Y_i \stackrel{*}{\Rightarrow} \varepsilon$，$(i=1,2,\cdots n)$

则 $FIRST(X) = FIRST(Y_1) \cup FIRST(Y_2) \cup \cdots \cup FIRST(Y_n) \cup \{\varepsilon\}$。

反复使用上述(a)～(e)步直到每个符号的 FIRST 集合不再增大为止。

5.1 重要定义及计算

- **First(α)**
  - 计算方法

    [4] 如果 $\alpha = X_1\ X_2 \ldots\ldots X_{i-1}\ X_i \ldots\ldots X_n$ 其中$X_i \in V_T \cup V_N$

    (2) 根据First(X_i)计算Frist(α)

    当 X_1 不能 $\stackrel{*}{\Rightarrow} \varepsilon$，则置 $FIRST(\alpha)=FIRST(X_1)$。

    若对任何 j

    $(1 \leqslant j \leqslant i-1, 2 \leqslant i \leqslant n), \varepsilon \in FIRST(X_j)$

    则 $FIRST(\alpha)=\bigcup_{j=1}^{i-1}(FIRST(X_j)-\{\varepsilon\}) \cup FIRST(X_i)$

    当所有 $FIRST(X_j)(1 \leqslant j \leqslant n)$都含有 ε 时，则 $FIRST(\alpha)=\bigcup_{j=1}^{n}(FIRST(X_j)) \cup \{\varepsilon\}$

5.1 重要定义及计算

First(α)例子

P:

(1) $E \rightarrow TE'$
(2) $E' \rightarrow + TE'$
(3) $E' \rightarrow \varepsilon$
(4) $T \rightarrow FT'$
(5) $T' \rightarrow * F T'$
(6) $T' \rightarrow \varepsilon$
(7) $F \rightarrow (E)$
(8) $F \rightarrow i$
(9) $F \rightarrow n$

First(E'T'E) =?
First(T'E') = ?

E	{i, n , (}
E'	{ + , ε }
T	{ i, n , (}
T'	{ *, ε }
F	{ i, n , (}

$S_\varepsilon = \{E', T'\}$

First(E'T'E) = {+,*,i,n,(}
First(T'E') = {+,*,ε}

5.1 重要定义及计算

- **Follow(A)**
  - 输入A：非终结符号
  - 返回：可能在某些句型中紧跟在A右边的终结符号集合
  - 例如 $S \Rightarrow^* \alpha A a \beta$，终结符a就在Follow(A)中，c就在First(A)中

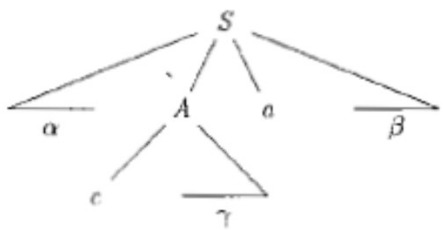


图 4-15　终结符号 c 在 FIRST(A) 中且 a 在 FOLLOW(A) 中

5.1 重要定义及计算

- **Follow(A)**
  - Formal Definition:

$FOLLOW(A)=\{a \mid S \overset{*}{\Rightarrow} \mu A\beta$ 且 $a \in V_T, a \in FIRST(\beta), \mu \in V_T^*, \beta \in V^+\}$

若 $S \overset{*}{\Rightarrow} \mu A\beta$，且 $\beta \overset{*}{\Rightarrow} \varepsilon$，则 $\# \in FOLLOW(A)$。 $\# \in Follow(S)$

也可定义为：$FOLLOW(A)=\{a \mid S \overset{*}{\Rightarrow} \cdots Aa \cdots, a \in V_T\}$

若有 $S \overset{*}{\Rightarrow} \cdots A$，则规定 $\# \in FOLLOW(A)$

这里我们用‘#’作为输入串的结束符，或称为句子括号，如：#输入串#。

5.1 重要定义及计算

- **Follow(A)**
  - 计算方法

    [1] 对于每个$A\in V_N$：令Follow(A):={ }；

    [2] 对于开始符S ：令Follow(S):= {#} ；

    [3] 对于A的每个产生式$A\rightarrow\alpha E\beta$ ($E\in V_N$) ：

    (i) 如果$\varepsilon\notin First(\beta)$, 则$Follow(E):=Follow(E)\cup First(\beta)$;

    (ii) 如果$\varepsilon\in First(\beta)$, 则$Follow(E):=Follow(E)\cup Follow(A)\cup(First(\beta)-\{\varepsilon\})$;

    [4] 重复[3]，直至每个Follow集合收敛。

5.1 重要定义及计算

- **Follow(A)例子**

P:
(1) E → TE'
(2) E' → + TE'
(3) E' → ε
(4) T → FT'
(5) T' → * F T'
(6) T' → ε
(7) F → (E)
(8) F → i
(9) F → n

First(X)

E	{i, n , (}
E'	{ + , ε }
T	{ i, n , (}
T'	{ *, ε }
F	{ i, n , (}

Follow(X)

E	{#,)}
E'	{#,)}
T	{+,), #}
T'	{+,), #}
F	{*, +,), #}

5.1 重要定义及计算

- **Predict($A \to \alpha$)**
  - 输入$A \to \alpha$：产生式
    - Predict($A \to \alpha$) = First(α), if $\varepsilon \notin$ First(α);
    - Predict($A \to \alpha$) = First(α)- $\{\varepsilon\} \cup$ Follow(A), if $\varepsilon \in$ First(α);

5.1 重要定义及计算

■ Predict(A → α)例子

P:
(1) E → TE' → First(TE')={i, n,(}
(2) E' → + TE' → First(+TE')={+}
(3) E' → ε → Follow(E')={#,)}
(4) T → FT' → First(FT')={i,n,(}
(5) T' → * F T' → First(*FT')={*}
(6) T' → ε → Follow(T')={),+, # }
(7) F → (E) → First((E))={ (}
(8) F → i → First(i)={i}
(9) F → n → First(n)={n}

first集

E	{i, n , (}
E'	{ + , ε }
T	{ i, n , (}
T'	{ *, ε }
F	{ i, n , (}

Follow集

E	{#,)}
E'	{#,)}
T	{+,), #}
T'	{+,), #}
F	{*, +,), #}

5.1 重要定义及计算

- Predict($A \rightarrow \alpha$)作用：实现了向前看一个token
- 回顾例子

5.1 重要定义及计算

- **例子：最左推导**

每次看栈内首个非终结符所有产生式的Predict

P:
(1) Z → aBeA
(2) A → Bc
(3) B → d
(4) B → bB
(5) B → ε

a	b	e	c

读头　从左向右移动读头，读入字符串

输入（读头所在）	栈内符号	分析	执行推导	匹配
abec	Z	Z的哪一个产生式以a开头？－(1)	aBeA	a
bec	BeA	B的哪一个产生式以b开头？－(4)	bBeA	b
ec	BeA	B的哪一个产生式能够以e开头？－(5)	εeA	e
c	A	A的哪一个产生式能够以c开头？－(2) (5)		

5.1 重要定义及计算

■ 例子：最左推导

P:
(1) Z → aBeA
(2) A → Bc
(3) B → d
(4) B → bB
(5) B → ε

a	b	e	c

读头

从左向右移动读头，读入字符串

输入（读头所在）	栈内符号	分析	执行推导	匹配
c	A	A的哪一个产生式能够以c开头？-(2)(5)	Bc	
c	Bc	A的哪一个产生式能够以c开头？-(2)(5)	εc	c

5.1 重要定义及计算

- **预测分析技术不是万能的**
  - 目的：一种确定性的、无回溯的分析技术, 在每一步都能够选择正确的产生式
  - 不是所有二型文法都能满足这个要求
    - 有多个可能的产生式时预测分析法无能为力 --- 可能需要回溯

注意：不同Parsing技术只能分析CFG的一个子集

5.2 文法的改造

5.2 文法的改造

- **不带回溯的自顶向下语法分析条件**
  - 对任意非终极符A,
    - 对A的任意两条产生式，$predict(A\rightarrow\beta_k)\cap predict(A\rightarrow\beta_j)=\varnothing$，当$k\neq j$
    - 即同一个非终极符的任意两个产生式的predict集合互不相交
- **这个条件保证:针对当前的符号和当前的非终极符,可以选择唯一的产生式来进行推导**
- **当文法的产生式存在左递归或公共前缀时，一定不会满足上述条件，需要对其进行变换，使之可以应用自顶向下的语法分析方法进行分析**

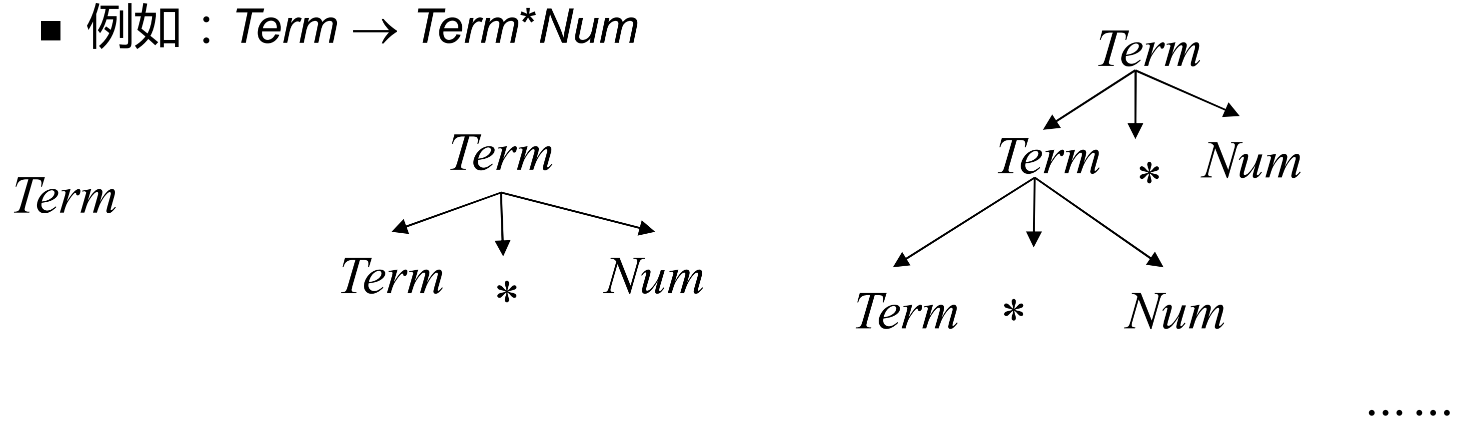
5.2 文法的改造

- **消除左递归**
  - 左递归的定义
    - 文法中一个非终结符号A使得对某个串α，存在一个推导 $A \stackrel{+}{\Rightarrow} A\alpha$ 则称这个文法是左递归的
    - 如果存在$A \rightarrow A\alpha$,则称为直接左递归；否则为间接左递归
    - 其实不是导致回溯的原因，而是导致死循环的原因

5.2 文法的改造

■ 消除左递归

- 为什么要消除左递归？- 左递归+top-down parsing = infinite loop
  - 例如：*Term* → *Term***Num*



5.2 文法的改造

- **消除直接左递归**
  - 假设非终结符A存在直接左递归的情形：

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

    $\beta_i\alpha_i^*$

  - 首先对A产生式分组：
    - 所有α_i不等于ε，β_i不以A开头 可以替换为

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

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

    - 即由A生成的串总是以某个β_i开头，然后跟上零个或者多个α_i的重复

5.2 文法的改造

■ 消除直接左递归示例

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

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

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

$$\begin{aligned} &E \rightarrow E + T \mid T \\ &T \rightarrow T * F \mid F \\ &F \rightarrow (E) \mid \mathbf{id} \end{aligned} \quad \Longrightarrow \quad \begin{aligned} &E \rightarrow TE' \\ &E' \rightarrow +\ TE' \mid \epsilon \\ &T \rightarrow FT' \\ &T' \rightarrow *\ FT' \mid \epsilon \\ &F \rightarrow (\ E\ ) \mid \mathbf{id} \end{aligned}$$

5.2 文法的改造

- **消除直接左递归示例**
  - 文法G：P→ PaPb|BaP
    - 分析：α = aPb, β=BaP
    - 改写：P→ βP′

      P′ → αP′ |ε
    - 改写后：P→ BaPP′

      P′ → aPbP′ |ε

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

$$A \to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A'$$

$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon$$

多个P怎么办？只看引起左递归的那个P

5.2 文法的改造

■ 消除间接左递归

- 消除立即左递归的方法并不能消除因为两步或多步推导而产生的左递归
  - 左递归：$S \xrightarrow{+} Sa$
    - 直接左递归：$S \rightarrow Sa$
    - 间接左递归：$S \rightarrow Aa, A \xrightarrow{+} Sb$ （ $A \xrightarrow{+} Aab$ ）
  - 例如文法：S → Aa | b， A → Sd | ε

    S => Aa => Sda

5.2 文法的改造

- **消除间接左递归**
  - 所谓间接左递归，就是经过多步替换，会出现直接左递归
    - 按照某种顺序替换，使得直接左递归出现，消除
  - 分析：把A排个序，$A_1 \rightarrow A_2a$, $A_2 \rightarrow A_3b$, … $A_k \rightarrow A_mc$
    - 如果存在产生式右部m=产生式左部k，则出现直接左递归
    - 如果所有产生式右部m全部>产生式左部k，则既没有直接左递归也没有间接左递归
    - 如果存在$A_i \rightarrow A_j a$, $A_j \xrightarrow{+} A_i b$, i<j，则多步替换后会出现$A_i$的直接左递归，因为形成了回路闭环

S → Aa | b,   A → Sd | ε
S => Aa => Sda

5.2 文法的改造

消除间接左递归

- 再例如：S→Qc|c，Q→Rb|b，R→Sa|a
  - 经过若干次推导就显现出其左递归性了，这就是间接左递归文法。
- 解决思路（消除所有左递归）
  - 首先消除可见的直接左递归
  - 然后按照A_i排序依次进行替换，直到出现闭环，则意味着显现出了直接左递归，=>消除，此时意味着包括A_i在内的所有所有A_j (j<i)都不存在直接左递归，并且$A_1 \to A_2$a, ... $A_j \to A_i$c中不存在任何m<k（即，A_i之前不存在任何回路）
  - 扫描完所有Ai之后，实现了所有$A_1 \to A_2$a, $A_2 \to A_3$b, ... $A_k \to A_m$c, 不存在m<k, 意味着不可能会出现回路闭环，那么左递归消除完毕

5.2 文法的改造

■ 消除所有左递归（通用算法）

输入： 没有环或 ϵ 产生式的文法 G。

输出： 一个等价的无左递归文法。

方法： 对 G 应用图 4-11 中的算法。请注意，得到的非左递归文法可能具有 ϵ 产生式。 □

```
1)  按照某个顺序将非终结符号排序为 A₁, A₂, ... , Aₙ.
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 Aᵢ → Aⱼγ 的产生式替换为产生式组 Aᵢ → δ₁γ | δ₂γ | ··· | δₖγ,
            其中 Aⱼ → δ₁ | δ₂ | ··· | δₖ 是所有的 Aⱼ 产生式
5)      }
6)      消除 Aᵢ 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

5.2 文法的改造

■ 消除所有左递归（通用算法）解释

- 把文法所有非终结符按某一顺序排序, 如A1, A2, …, An
- 先看A1有没有直接左递归，如果有，则消除
- 然后从i=2开始，对每一个Ai, 看其产生式是否有Ai→Aj (i>j), 如果有，则用Aj所有产生式替换
  - 都替换完成后，对当前Ai，则只有Ai→At (i<=t)，然后消除Ai的直接左递归，则当前Ai，只剩下Ai→At (i<t)
  - 注意此时因为j<i, 所以Aj已经处理完，即不存在Aj→Ak (j>=k)，而只有j<k
- 当完成An上述操作后，意味着所有产生式只有Ai→Aj (i<j)，即不可能形成闭环回路，则不会存在任何左递归

```
按照某个顺序将非终结符号排序为 A1, A2, ..., An.
for ( 从 1 到 n 的每个 i ) {
    for ( 从 1 到 i − 1 的每个 j ) {
        将每个形如 Ai → Ajγ 的产生式替换为产生式组 Ai → δ1γ | δ2γ | ··· | δkγ,
        其中 Aj → δ1 | δ2 | ··· | δk 是所有的 Aj 产生式
    }
    消除 Ai 产生式之间的立即左递归
}
```

5.2 文法的改造

- **消除左递归示例:**

$S \rightarrow A\,b$ $A \rightarrow S\,a \mid b$

1:S 2:A
$A \rightarrow Aba \mid b$
$A \rightarrow bA'$ $A' \rightarrow baA' \mid \varepsilon$

5.2 文法的改造

- 消除左递归示例:

$S \rightarrow Aa \mid b,$
$A \rightarrow Ac \mid Sd \mid \varepsilon$

1:S
2:A

$S \rightarrow Aa \mid b,$
$A \rightarrow Ac \mid Aad \mid bd \mid \varepsilon$

$S \rightarrow Aa \mid b,$
$A \rightarrow bdA' \mid A'$
$A' \rightarrow cA' \mid adA' \mid \varepsilon$

5.2 文法的改造

- ## 消除左递归示例

$$
\begin{aligned}
S &\rightarrow Qc \mid c \\
Q &\rightarrow Rb \mid b \\
R &\rightarrow S\,a \mid a
\end{aligned}
$$

1:R
2:Q
3:S

$R \rightarrow Sa \mid a$

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

$S \rightarrow Qc \mid c \rightarrow Sabc \mid abc \mid bc \mid c$

$S \rightarrow (abc \mid bc \mid c)S'$

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

5.2 文法的改造

消除左递归示例

变换非终结符顺序

排序不同，

变换后文法也不同

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

1:S
2:Q
3:R

$S \to Qc \mid c$
$Q \to Rb \mid b$
$R \to Sa \mid a \to (Qc|c)a \mid a \to Qca \mid ca \mid a$
$\to (Rb|b)ca \mid ca \mid a$

$S \to Qc \mid c$
$Q \to Rb \mid b$
$R \to (bca \mid ca \mid a)R'$
$R' \to bcaR' \mid \varepsilon$

5.2 文法的改造

- **消除左递归算法要求**
  - 没有环或ε产生式的文法
    - 环：形如$A \overset{+}{\Rightarrow} A$ 的推导
    - ε产生式：A → ε
  - 需要文法的改造(等价)
    - 文法的简化
    - 构造无ε产生式的上下文无关文法

5.2 文法的改造

- **文法的简化：产生式个数越少越好**
  - 产生式在推导过程中永远不会被用到 (无用产生式)– 删除
  - 形如A→A 的产生式 – 删除
  - 产生式在推导过程中不能生成终结符 – 删除

类似于FA中不能到达终结状态的状态

5.2 文法的改造

- **文法的简化：产生式个数越少越好**
  - 删除无用产生式：A $\rightarrow\alpha$, A不出现在任何句型中
    - 生成会出现在句型中的非终极符集合SS
      - SS = {S}
      - $\forall$ $S_i \in$ SS,
      - $S_i \rightarrow \alpha 1$,......, $S_i \rightarrow \alpha n$
      - 把$\alpha 1$,......, αn中的所有非终极符加入SS中;
      - 重复直到没有新的非终极符加入;
    - 不属于SS的非终极符,它们的产生式是无用产生式,删除掉;

类似于FA中的S到A不可达

5.2 文法的改造

- **文法的简化：产生式个数越少越好**
  - 删除无用产生式例子

$S \to \varepsilon \mid A\,a\,BB$
$A \to B\,B \mid a$
$B \to \varepsilon \mid b$
$C \to c$

SS = {S}
SS = {S, A, B}
SS = {S, A, B}

$C \to c$是无用产生式

5.2 文法的改造

- **文法的简化：产生式个数越少越好**
  - 删除特型产生式 $A \rightarrow B$
    - 对每个非终极符A,构造 $S_A = \{B \mid A \Rightarrow^+ B, B \in V_N\}$
    - 如果 $C \in S_A$,而且 $C \rightarrow \alpha$ 不是特型产生式,则增加 $A \rightarrow \alpha$;
    - 删除特型产生式
    - 删除无用产生式

类似于FA中的A接受ε 到达B

5.2 文法的改造

- **文法的简化：产生式个数越少越好**
  - 删除特型产生式例子

$S \to \varepsilon \mid A$
$A \to B \mid a$
$B \to b$

$S_S = \{A,B\}$	$S \to a$ $S \to b$
$S_A = \{B\}$	$A \to b$
$S_B = \{\}$	

$S \to \varepsilon \mid a \mid b$
$A \to a \mid b$
$B \to b$

5.2 文法的改造

- **文法的简化：产生式个数越少越好**
  - 删除在推导过程中不能生成终结符的产生式
  - 例子

类似于FA中的
A不能到达终结状态

S→ Be;
S→ Ec;
A→Ae;
A→e
A→A
B→Ce
B→Af
C→Cf
D→f

S→Be
A→Ae
A→e
B→Af

5.2 文法的改造

- **构造无ε产生式的CFG：空产生式的存在常常给语法分析带来困难**
  - 定义：无ε产生式的CFG满足：要么P中不含有ε产生式，要么只有S→ ε; 若存在S→ ε, 则S不出现在任何产生式右部
  - 变换方法： 找出所有可以推导出ε的非终极符,记为Sε;
    对于所有产生式
    $A \to X1\ X2 \ldots\ldots Xi\text{-}1XiXi+1 \ldots\ldots Xn$
    $Xi \in S\varepsilon$
    增加$A \to X1\ X2 \ldots\ldots Xi\text{-}1Xi+1 \ldots\ldots Xn$
    直到没有新的产生式产生
    删除对应空产生式, 除了S→ε不可以删除

5.2 文法的改造

构造无ε产生式的CFG例子

$S \to \varepsilon \mid A\ a\ BB$

$A \to\ B\ B \mid a$

$B \to \varepsilon \mid b$

$S_\varepsilon = \{S, B, A\}$

$S \to A\ a\ BB$

$S \to a\ BB$ $\quad$ $S \to aB$ $\quad$ $S \to a$

$S \to A\ a\ B$ $\quad$ $S \to Aa$

$A \to BB$

$A \to B$

变换后得到的文法：

$S \to \varepsilon \mid A\ a\ BB \mid aBB$

$\quad \mid AaB \mid aB \mid Aa \mid a$

$A \to\ B\ B \mid B \mid a$

$B \to b$

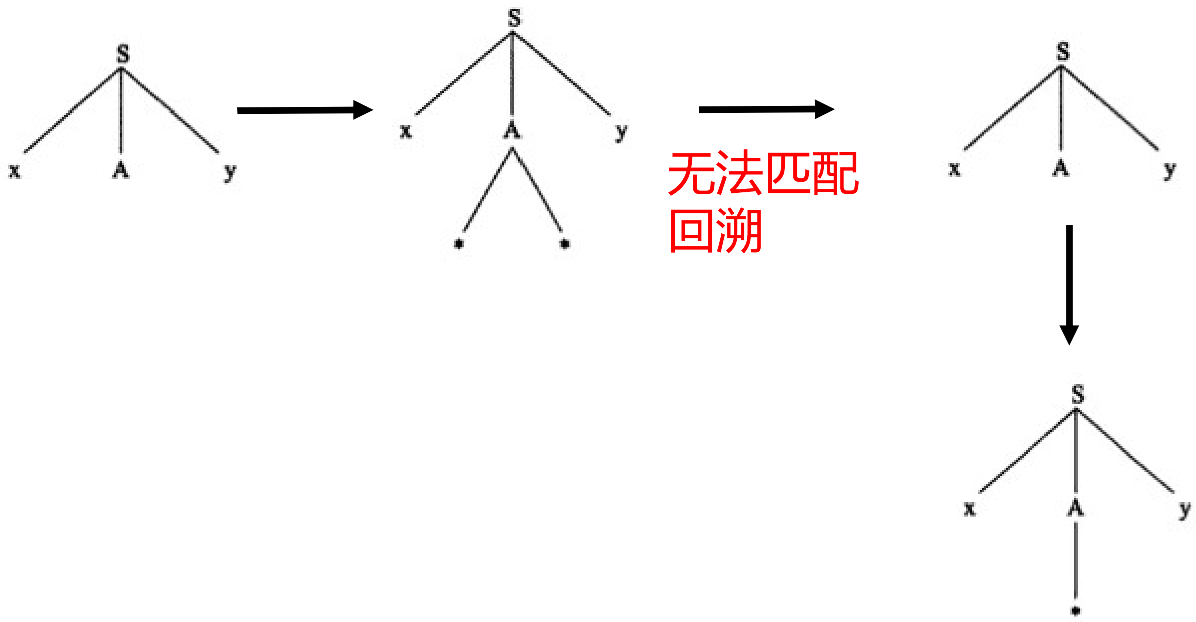
5.2 文法的改造

- **左因子**
  - 若文法中含有形如$A \to \alpha\beta|\alpha\gamma$的产生式，则导致$Predict(A \to \alpha\beta) \cap Predict(A \to \alpha\gamma) \neq \emptyset$，不满足LL(1)文法的充分必要条件，导致回溯
  - 在推导的时候，不知道该如何选择

$$
\begin{aligned}
stmt \to & \ \textbf{if} \ expr \ \textbf{then} \ stmt \ \textbf{else} \ stmt \\
| & \ \textbf{if} \ expr \ \textbf{then} \ stmt
\end{aligned}
$$

5.2 文法的改造

- **回溯示例：S→xAy，A→**|*，输入x*y**



把A第一个产生的子树注销掉
还要把读头退回至进入A时的状态，即x*y中的*符号

5.2 文法的改造

- **左因子**
  - 若文法中含有形如A→αβ|αγ的产生式，则导致Predict(A→αβ)∩Predict(A→αγ)≠∅，不满足LL(1)文法的充分必要条件，导致回溯
  - 在推导的时候，不知道该如何选择

$$
\begin{aligned}
\mathit{stmt} \rightarrow & \ \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt}\ \textbf{else}\ \mathit{stmt} \\
| & \ \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt}
\end{aligned}
$$

5.2 文法的改造

■ 提取左公因子算法

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \Longrightarrow \begin{array}{l} A \rightarrow \alpha A' \\ A' \rightarrow \beta_1 \mid \beta_2 \end{array}$$

- 输入：文法*G*
- 输出：一个等价的提取了左公因子的文法
- 方法：对于每个非终结符号*A*，找出它的两个或多个可选项之间的最长公共前缀α，且$\alpha \neq \varepsilon$，那么将*A*所有的产生式

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

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

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

若在βi、βj、βk … (其中$1 \leq i$，j，$k \leq n$)中仍含有左公共因子，这时可再次提取，这样反复进行提取直到引进新非终结符的有关产生式再无左公共因子为止。

5.2 文法的改造

提取左公因子, 例1:

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

  $E \rightarrow b$

- 对于S而言，最长前缀是$iEtS$, 因此

  $S \rightarrow iEtSS' \mid a$

  $S' \rightarrow eS \mid \varepsilon$

  $E \rightarrow b$

5.2 文法的改造

■ 提取左公因子, 例2

文法G的产生式为：

(1) S→aSb

(2) S→aS

(3) S→ε

对产生式(1)、(2)提取左公因子后得：

S→ aS(b|ε)

S→ε

进一步变换为文法G′：

S→aSA

A→b

A→ε

S→ε

变换为无ε产生式文法G′′：

S→aSA

S→aS

A→b

S→ε

5.2 文法的改造

■ 提取左公因子, 例3

若文法G的产生式为：

(1) A→ad

(2) A→Bc

(3) B→aA

(4) B→bB

产生式(2)的右部以非终结符开始，因此左公共因子可能是隐式的。这种情况下对右部以非终结符开始的产生式，用其相同左部而右部以终结符开始的产生式进行相应替换，即分别用(3)、(4)的右部替换(2)中的B，可得：

(1) A→ad; (2) A→aAc; (3) A→bBc; (4) B→aA; (5) B→bB

提取产生式(1)、(2)的左公共因子得：

A→a(d|Ac); A→bBc; B→aA; B→bB

引进新非终结符A′，去掉'('，')'后得G′为：

(1) A→aA′; (2) A→bBc; (3) A′→d; (4) A′→Ac; (5) B→aA; (6) B→bB

5.2 文法的改造

- **提取左公因子，注意1**
  - 变换后有时会使某些产生式变成无用产生式，在这种情况下必须对文法重新压缩(或化简)

文法G的产生式为：
(1) S→aSd
(2) S→Ac
(3) A→aS
(4) A→b

用产生式(3)、(4)中右部替换产生式(2)中右部的A，文法变为：
(1) S→aSd; (2) S→aSc; (3) S→bc; (4) A→aS; (5) A→b

对(1)、(2)提取左公共因子得：S→aS(d|c)
引入新非终结符A′后变为：
(1) S→aSA′; (2) S→bc; (3) A′→d|c;
(4) A→aS; (5) A→b

A变成不可到达的符号，删除

5.2 文法的改造

- **提取左公因子, 注意2**
  - 存在某些文法不能在有限步骤内提取完左公共因子。

文法G的产生式为：

(1) $S \to Ap|Bq$

(2) $A \to aAp|d$

(3) $B \to aBq|e$

用(2)、(3)产生式的右部替换(1)中产生式的A、B使文法变为：

(1) $S \to aApp|aBqq$; (2) $S \to dp|eq$; (3) $A \to aAp|d$; (4) $B \to aBq|e$

对(1)提取左公共因子则得：$S \to a(App|Bqq)$

再引入新非终符S′结果得等价文法为：

(1) $S \to aS'$; (2) $S \to dp|eq$; (3) $S' \to App|Bqq$; (4) $A \to aAp|d$; (5) $B \to aBq|e$

5.2 文法的改造

■ 提取左公因子, 注意2(cont.)

- 存在某些文法不能在有限步骤内提取完左公共因子。

文法G的产生式为：

(1) $S \to Ap|Bq$
(2) $A \to aAp|d$
(3) $B \to aBq|e$

同样分别用(4)、(5)产生式的右部替换(3)中右部的A、B再提取左公共因子最后结果得：
(1) $S \to aS'$; (2) $S \to dp|eq$; (3) $S' \to aS''$; (4) $S' \to dpp|eqq$; (5) $S'' \to Appp|Bqqq$; (6) $A \to aAp|d$; (7) $B \to aBq|e$

可以看出若对(5)中产生式A、B继续用(6)、(7)产生式的右部替换，只能使文法的产生式愈来愈多无限增加下去，但不能得到提取左公共因子的预期结果。

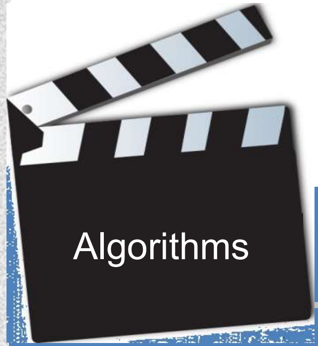
5.2 文法的改造

- **两种实现自顶向下语法分析的方法**
  - 递归下降法
  - 非递归预测分析：LL(1)法
- **使用条件：文法能够适用于top-down分析, 也称为LL(1)文法**

つづく

作 业

- 文法化简
  - 教材P137：4.3.1, 4.3.2



5.3 两种预测分析的实现

5.3 两种预测分析的实现

- **两种实现自顶向下语法分析的方法**
  - 递归下降法
  - 非递归预测分析：LL(1)法
- **使用条件：文法能够适用于top-down分析, 也称为LL(1)文法**

5.3 两种预测分析的实现

- **LL(1)语法充分必要条件**
  - 一个文法G是LL(1)的， 当且仅当G中对A的任意两条产生式，A→α|β满足下面条件：
    - 不存在终结符号a使得α和β都可以推导出以a开头的串
    - α和β中最多只有一个可以推导出空串
    - 如果β=>*ε, 那么α不能推导出任何以Follow(A)中某个终结符开头的串；类似地，如果α =>*ε, 那么β不能推导出任何以Follow(A)中某个终结符开头的串
  - 显然，LL(1)一定是无二义，无左递归，无ε产生式

5.3 两种预测分析的实现

- **LL(1)语法充分必要条件**
  - 解释: 前两个条件等价于First(α)∩First(β)=∅；第三个条件等价于如果ε∈First(β)，那么First(α)∩Follow(A)=∅, 如果ε∈First(α)，那么First(β)∩Follow(A)=∅
  - 其实就是：predict(A→ α)∩ predict(A→ β)=∅
- **不符合要求时，可以尝试文法变换**
  - 消除二义性，消除左递归，提左因子
  - 去除空产生式，消除无用产生式，消除特型产生式

5.3 两种预测分析的实现

例：判断下面文法是不是LL(1)文法：

$S \rightarrow$ if E then S else S

| if E then S

| other

$E \rightarrow b$

解：首先对文法进行改造，提取左公共因子；文法改写为：

[1] $S \rightarrow$ if E then S S'

[2] | other

[3] $S' \rightarrow$ else S

[4] | ε

[5] $E \rightarrow b$

5.3 两种预测分析的实现

第2步：求预测符集

First(S)={ if, other }, First(S`)={ else, ε }

First(E)={ b }

Follow(S)= Follow(S`)={ else,# }

Follow(E)={ then }

Predict([1])={ if }　　Predict([2])={ other }

Predict([3])={ else }　Predict([4])={ else,# }

Predict([5])={ b }

第3步：判定文法是不是LL(1)文法。

因为：1) Predict([1]) ∩ Predict([2])= Φ

但是：2) Predict([3]) ∩ Predict([4])={else}不为空集, 故此文法不是LL(1)文法

5.3 两种预测分析的实现

- **两种实现自顶向下语法分析的方法**
  - 递归下降法
  - LL(1)法

5.3 两种预测分析的实现

递归下降法

- 一个递归下降语法分析程序由一组过程组成，每个非终结符号对应一个过程，程序的执行从开始符号对应的过程开始，如果这个过程的过程体扫描了整个输入串，它就停止并宣布语法分析成功。

$G = (V_T, V_N, S, P)$

预定义函数:

void match(a: V_T)

void read();

全局变量: token: V_T

输入串: str

对于每个非终极符 A， $A \to \alpha_1|......|\alpha_n$

A( )

{ case token of

Predict($A \to \alpha_1$): SubR(α_1) ; break;

......

Predict($A \to \alpha_n$): SubR(α_n) ; break;

default: error; }

和递归定义保持一致

5.3 两种预测分析的实现

■ 递归下降法

- 例子

P:
(1) $Z \rightarrow aBd$ {a}
(2) $B \rightarrow d$ {d}
(3) $B \rightarrow c$ {c}
(4) $B \rightarrow bB$ {b}

a b c d

```
Z ( )
{
if (token == a)
 {   match(a);
     B( );
     match(d);
 }
else error();
}
```

```
B ( )
{
 case token of
  d:  match(d);break;
  c:   match(c); break;
  b:{  match(b);
        B( ); break;}
  other: error();
}
```

```
void main()
{read();
  Z();  }
```

5.3 两种预测分析的实现

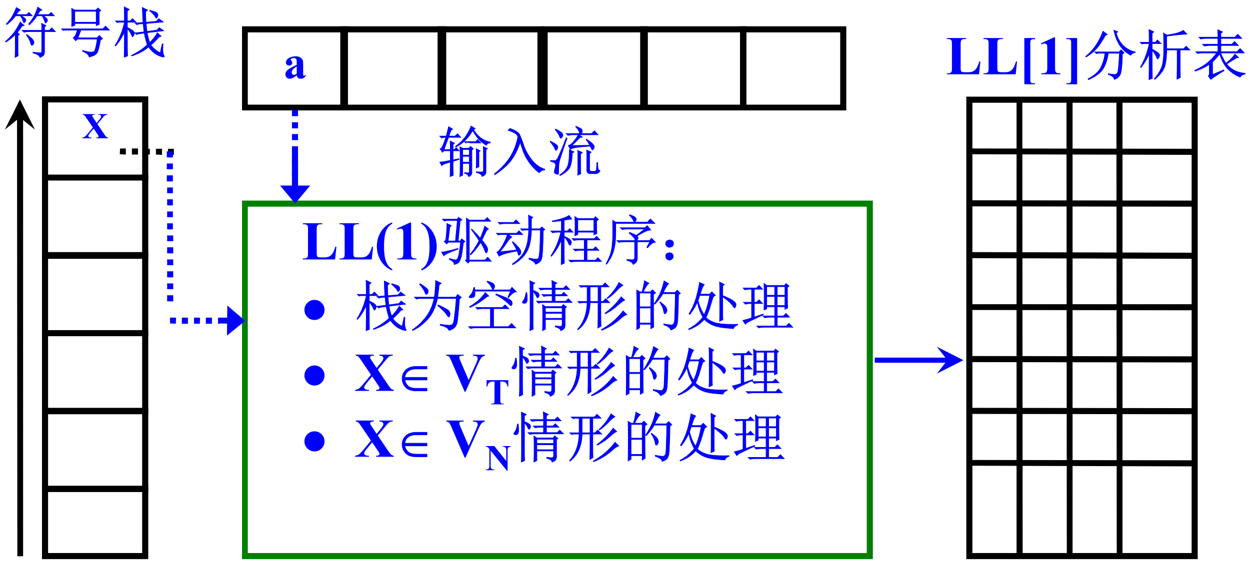
- **两种实现自顶向下语法分析的方法**
  - 递归下降法
  - LL(1)分析法

5.3 两种预测分析的实现

- **LL(1)分析**
  - 非递归思想
    - 采用LL(1)分析表记录每个产生式的预测符
    - 采用LL(1)分析驱动程序控制分析过程
    - 采用符号栈记录需要推导的句型

5.3 两种预测分析的实现

LL(1) 分析机制



- 符号栈：保存**LL(1)**分析的中间结果，当输入流和符号栈同时为空，则接受否则拒绝输入串
- **LL(1)**分析表：**T(A,a),**指引选择哪条产生式

5.3 两种预测分析的实现

LL(1)分析表

如果当前非终极符是X, 当前输入符号是a, 当且仅当 $a \in predict(X \to \alpha)$ 时，用产生式 $X \to \alpha$ 对X进行替换

	a_1	…	a_n	#
A_1				
…	….	….	…	
Am				

对于任意的一个**LL(1)**文法**G = (V_N, V_T, S, P)**

V_T = {a1, …, an}，V_N = {A1, …, Am}

LL(Ai, aj) = Ai→ α, 如果**aj∈predict(Ai→ α)**

LL(Ai, aj) = error(⊥), 如果**aj** 不属于**Ai**的任何一条产生式的预测符集

LL分析表的作用是对当前非终极符和输入符号确定应该选择用哪个产生式进行推导

5.3 两种预测分析的实现

- **LL(1)分析表，例1：**

P:
(1) $Z \rightarrow aBd$
(2) $B \rightarrow d$
(3) $B \rightarrow c$
(4) $B \rightarrow bB$

产生式	Predict集
(1)	{a}
(2)	{d}
(3)	{c}
(4)	{b}

	a	b	c	d	#
Z	(1)				
B		(4)	(3)	(2)	

5.3 两种预测分析的实现

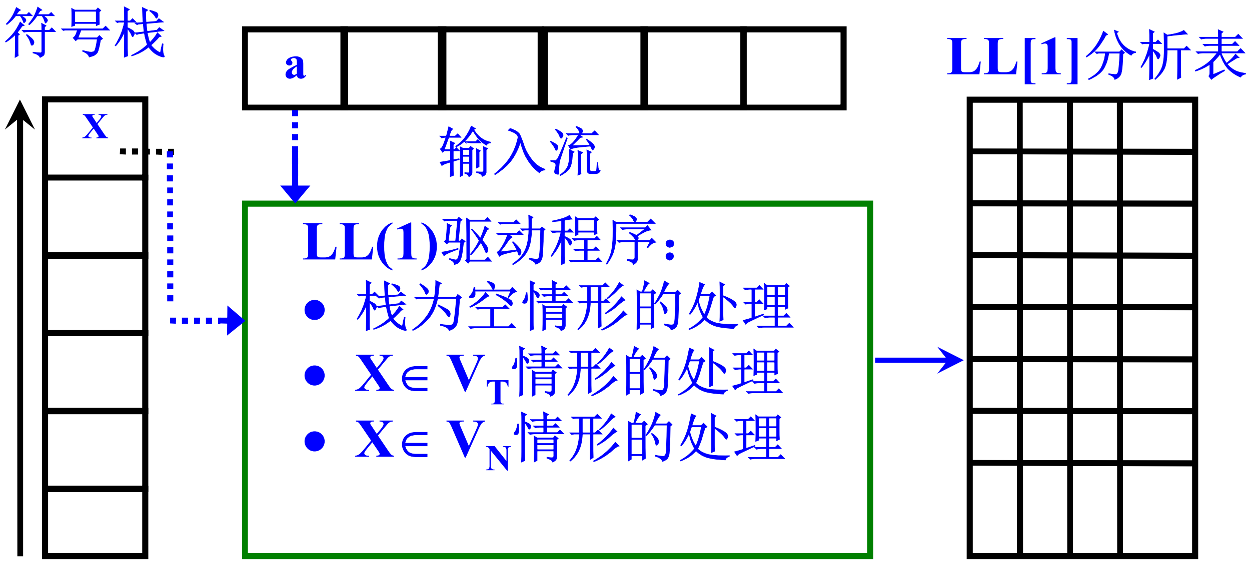
- **LL(1)分析表，例2：**

(1) E → TE'	{ i, n, (}
(2) E' → + TE'	{+}
(3) E' → ε	{#,)}
(4) T → FT'	{i,n,(}
(5) T' → * F T'	{*}
(6) T' → ε	{),+, # }
(7) F → (E)	{ (}
(8) F → i	{i}
(9) F → n	{n}

	+	*	(	)	i	n	#
E			**(1)**		**(1)**	**(1)**	
E'	**(2)**			**(3)**			**(3)**
T			**(4)**		**(4)**	**(4)**	
T'	**(6)**	**(5)**		**(6)**			**(6)**
F			**(7)**		**(8)**	**(9)**	

5.3 两种预测分析的实现

LL(1)
分析
机制



- 符号栈：保存LL(1)分析的中间结果，当输入流和符号栈同时为空，则接受否则拒绝输入串
- LL(1)分析表：T(A,a),指引选择哪条产生式

5.3 两种预测分析的实现

■ LL(1)分析驱动程序

分析格局：<符号栈Stack，输入串inp>
X = top(Stack)，则格局的可能情形有：

X = ε情形的处理，

< ，#> : 成功
< ，a> :出错

X∈ V_T情形的处理:

X = a, <a, a>, 匹配
X = a, <a, b>, 出错

X ∈ V_N情形的处理:

<X, a>, 如果LL [X,a] = i,则将X替换成第i条产生式的右部(替换时产生式右部符号逆序压栈)
<X, a>, 如果LL [X, a] = ⊥, 出错

5.3 两种预测分析的实现

LL(1)分析例子

```
Stmt   ::= (1) if expr then Stmt else Stmt
           (2) while Expr do Stmt |
           (3) begin Stmts end
Stmts ::= (4) Stmt ; Stmts | (5) ε
Expr  ::=(6) id
```

	if	then	else	while	do	begin	end	id	;	$
Stmt	*1*			*2*		*3*				
Stmts	*4*			*4*		*4*	*5*			
Expr								*6*		

empty = error

5.3 两种预测分析的实现

完整算法

```
push S$                        /* S is start symbol */
while Stack not empty
    X := pop(Stack)
    a := peek at next input “token” /* EOF => $ */
    if X is terminal or $
        If X==a, read token a else abort;
    else look at PREDICT(X, a) /* X is nonterminal*/
        Empty       : abort
        rule X→ α   : push α
If not at end of input, Abort else Accept
```

5.3 两种预测分析的实现

- **LL(1)分析-示例**
  - 见LL1-example.pdf

5.3 两种预测分析的实现

- **递归下降法 v.s. LL(1)法**
  - 相同点：
    - 同属于自顶向下分析法，分析条件相同
  - 不同点：
    - 递归下降法对每个非终极符产生子程序，而LL(1)方法则产生LL分析表；
    - 递归下降法能判断每个产生式的结束，而LL(1)方法则不能；
    - 递归下降分析法不用符号栈，而LL(1)方法则用符号栈。

5.3 两种预测分析的实现

- **延伸阅读：预测分析中的错误恢复**
  - 例如，Panic mode: if LL(A, a) = error，哪些字符可以放在A的同步集中？一些 heuristics：
    - 语句开头的符号
    - Follow(A), First(A)
    - ……

作 业

- 教材P147: 4.4.1(3), (5), 4.4.2, 4.4.4 (只针对3,5两个文法进行计算)

Thank you!