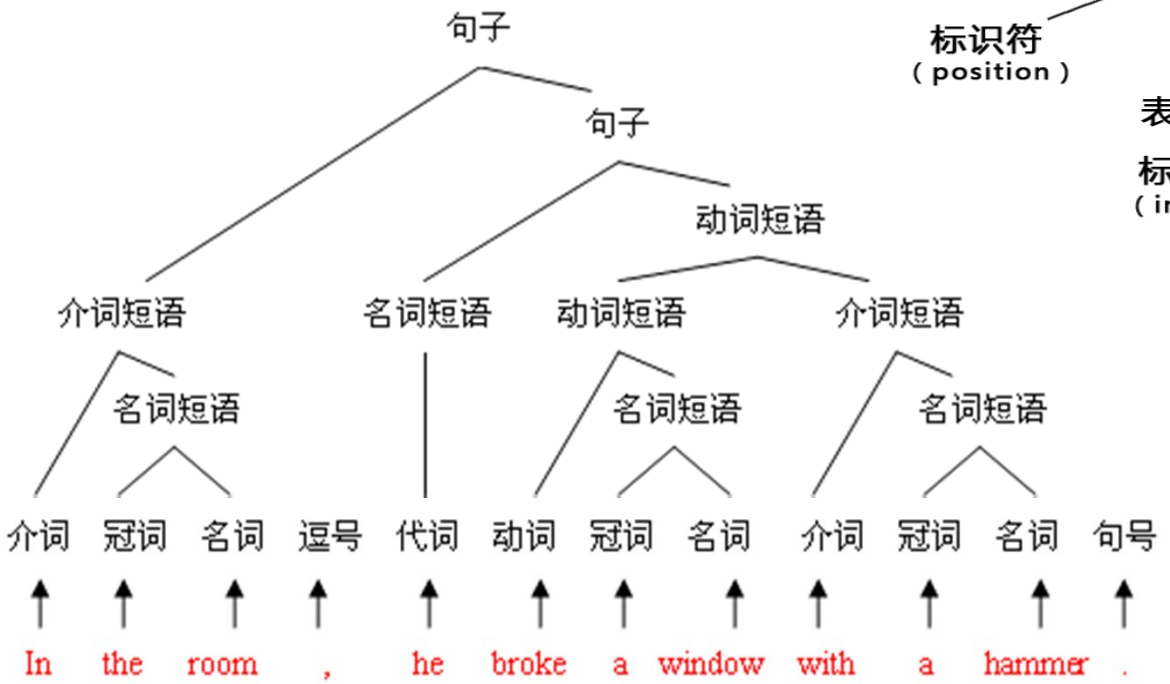
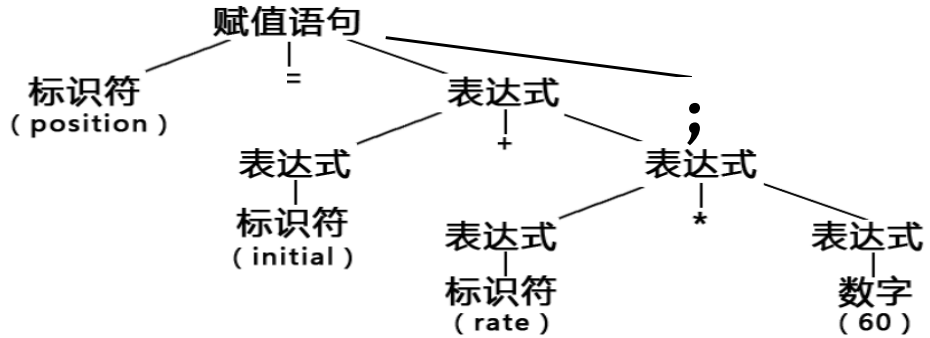
第四章：自顶向下语法分析方法

提纲

回顾





position = initial + rate * 60 ;

<id, position> <=> <id,initial> <+> <id, rate> <*> <num,60> <;>

自顶向下的分析(*Top-Down Parsing*)

- 从分析树的顶部（根节点）向底部（叶节点）方向构造分析树
- 可以看成是从文法开始符号S推导出词串w的过程
  - 例

文法

① $E \rightarrow E+E$

② $E \rightarrow E*E$

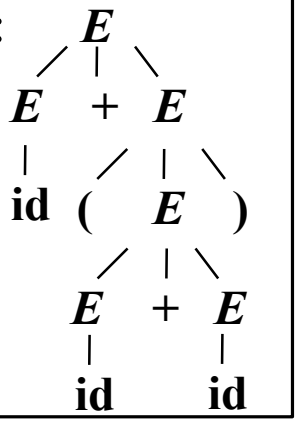
③ $E \rightarrow (E)$

④ $E \rightarrow \mathrm{id}$

输入

$\mathrm{id}+(\mathrm{id}+\mathrm{id})$

分析树：



推导过程：

$E \Rightarrow E+E$

$\Rightarrow E+(E)$

$\Rightarrow E+(E+E)$

$\Rightarrow E+(\mathrm{id}+E)$

$\Rightarrow \mathrm{id}+(\mathrm{id}+E)$

$\Rightarrow \mathrm{id}+(\mathrm{id}+\mathrm{id})$

- 每一步推导中，都需要做两个选择
  - 替换当前句型中的哪个非终结符
  - 用该非终结符的哪个候选式进行替换

最左推导(*Left-most Derivation*)

- 在最左推导中，总是选择每个句型的最左非终结符进行替换
  - 例

文法

① $E \to E + E$

② $E \to E * E$

③ $E \to (E)$

④ $E \to \mathrm{id}$

输入

$\mathrm{id} + (\mathrm{id} + \mathrm{id})$

推导过程

$E \Rightarrow_{lm} E + E$

$\Rightarrow_{lm} \mathrm{id} + E$

$\Rightarrow_{lm} \mathrm{id} + (E)$

$\Rightarrow_{lm} \mathrm{id} + (E + E)$

$\Rightarrow_{lm} \mathrm{id} + (\mathrm{id} + E)$

$\Rightarrow_{lm} \mathrm{id} + (\mathrm{id} + \mathrm{id})$

最左推导

最右归约

- 如果$S \Rightarrow^*_{lm} \alpha$，则称$\alpha$是当前文法的最左句型(*left-sentential form*)

最右推导(*Right-most Derivation*)

- 在最右推导中，总是选择每个句型的最右非终结符进行替换
  - 例

文法

① $E \to E + E$

② $E \to E * E$

③ $E \to (E)$

④ $E \to \mathrm{id}$

输入

$\mathrm{id} + (\mathrm{id} + \mathrm{id})$

推导过程

$E \Rightarrow_{rm} E + E$

$\Rightarrow_{rm} E + (E)$

$\Rightarrow_{rm} E + (E + E)$

$\Rightarrow_{rm} E + (E + \mathrm{id})$

$\Rightarrow_{rm} E + (\mathrm{id} + \mathrm{id})$

$\Rightarrow_{rm} \mathrm{id} + (\mathrm{id} + \mathrm{id})$

最右推导

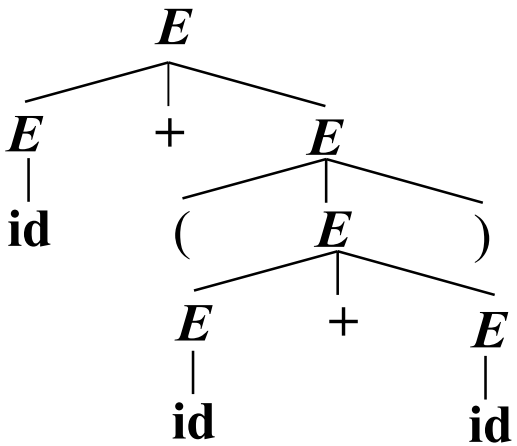
最左归约

- 在自底向上的分析中，总是采用最左归约的方式，因此把最左归约称为规范归约，而最右推导相应地称为规范推导

最左推导和最右推导的唯一性

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow E+(E) \\
&\Rightarrow E+(E+E) \\
&\Rightarrow E+(\mathrm{id}+E) \\
&\Rightarrow \mathrm{id}+(\mathrm{id}+E) \\
&\Rightarrow \mathrm{id}+(\mathrm{id}+\mathrm{id})
\end{aligned}
$$

$$
\begin{aligned}
E &\Rightarrow E+E \\
&\Rightarrow \mathrm{id}+E \\
&\Rightarrow \mathrm{id}+(E) \\
&\Rightarrow \mathrm{id}+(E+E) \\
&\Rightarrow \mathrm{id}+(E+\mathrm{id}) \\
&\Rightarrow \mathrm{id}+(\mathrm{id}+\mathrm{id})
\end{aligned}
$$



$$
\begin{aligned}
E &\Rightarrow_{lm} E+E \\
&\Rightarrow_{lm} \mathrm{id}+E \\
&\Rightarrow_{lm} \mathrm{id}+(E) \\
&\Rightarrow_{lm} \mathrm{id}+(E+E) \\
&\Rightarrow_{lm} \mathrm{id}+(\mathrm{id}+E) \\
&\Rightarrow_{lm} \mathrm{id}+(\mathrm{id}+\mathrm{id})
\end{aligned}
$$

$$
\begin{aligned}
E &\Rightarrow_{rm} E+E \\
&\Rightarrow_{rm} E+(E) \\
&\Rightarrow_{rm} E+(E+E) \\
&\Rightarrow_{rm} E+(E+\mathrm{id}) \\
&\Rightarrow_{rm} E+(\mathrm{id}+\mathrm{id}) \\
&\Rightarrow_{rm} \mathrm{id}+(\mathrm{id}+\mathrm{id})
\end{aligned}
$$

Exercise

1．如果文法G是无二义的，则它的任何句子α( )。

A.最左推导和最右推导对应的语法树必定相同

B.最左推导和最右推导对应的语法树可能不同

C.最左推导和最右推导必定相同

D.可能存在两个不同的最左推导，但它们对应的语法树相同

自顶向下的语法分析采用最左推导方式

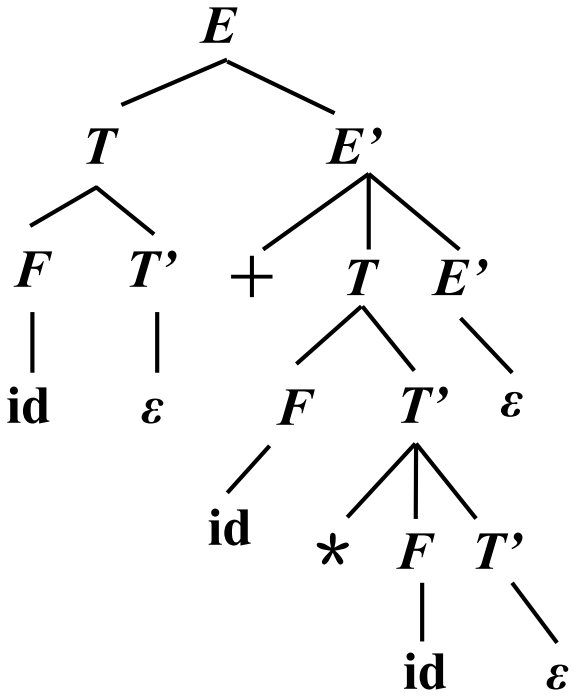
- 总是选择每个句型的最左非终结符进行替换
- 根据输入流中的下一个终结符，选择最左非终结符的一个候选式

例

- 文法
  - ① $E \rightarrow T\ E'$
  - ② $E' \rightarrow +\ T\ E' \mid \varepsilon$
  - ③ $T \rightarrow F\ T'$
  - ④ $T' \rightarrow *\ F\ T' \mid \varepsilon$
  - ⑤ $F \rightarrow (\ E\ ) \mid \mathrm{id}$
- 输入

  id + id * id

  ↑ ↑ ↑ ↑ ↑ ↑



自顶向下语法分析的通用形式

- 递归下降分析 (*Recursive-Descent Parsing*)
  - 由一组过程组成，每个过程对应一个非终结符
  - 从文法开始符号*S*对应的过程开始，其中递归调用文法中其它非终结符对应的过程。如果*S*对应的过程体恰好扫描了整个输入串，则成功完成语法分析

```
    void A( ) {
1)      选择一个A产生式，A →X1 X2 … Xk ;
2)      for ( i = 1 to k ) {
3)          if ( Xi是一个非终结符号)
4)              调用过程 Xi( ) ;
5)          else if ( Xi 等于当前的输入符号a)
6)              读入下一个输入符号;
7)          else /* 发生了一个错误 */ ;
        }
    }
```

可能需要回溯(*backtracking*)，导致效率较低

预测分析 (*Predictive Parsing*)

- 预测分析是递归下降分析技术的一个特例，通过在输入中向前看固定个数（通常是一个）符号来选择正确的*A*-产生式。
  - 可以对某些文法构造出向前看*k*个输入符号的预测分析器，该类文法有时也称为***LL(k)* 文法类**
- 预测分析不需要回溯，是一种确定的自顶向下分析方法

提纲

问题1

左递归文法会使递归下降分析器陷入无限循环

- 例
  - 文法G

$$E \to E + T \mid E - T \mid T$$
$$T \to T * F \mid T / F \mid F$$
$$F \to (E) \mid \text{id}$$

$$E \Rightarrow E + T$$
$$\Rightarrow E + T + T$$
$$\Rightarrow E + T + T + T$$
$$\Rightarrow \dots$$

  - 输入

id + id * id

含有$A \to A\alpha$形式产生式的文法称为是直接左递归的(*immediate left recursive*)

如果一个文法中有一个非终结符A使得对某个串α存在一个推导$A \Rightarrow^{+} A\alpha$，那么这个文法就是左递归的

经过两步或两步以上推导产生的左递归称为是间接左递归的

消除直接左递归

$A \to A\alpha \mid \beta (\alpha \neq \varepsilon, \beta$不以$A$开头$)$ $r=\beta\alpha^*$

$A \to \beta A'$

$A' \to \alpha A' \mid \varepsilon$

事实上，这种消除过程就是把左递归转换成了右递归

$A \Rightarrow A\alpha$
$\Rightarrow A\alpha\alpha$
$\Rightarrow A\alpha\alpha\alpha$
…
$\Rightarrow A\alpha \ldots\alpha$
$\Rightarrow \beta\,\alpha \ldots\alpha$

➢ 例 $E \to E + T \mid T$ （α = $+T$，β = T）

$T \to T * F \mid F$ （α = $*F$，β = F）

$F \to (E) \mid \mathrm{id}$

$E \to T\,E'$

$E' \to +\,T\,E' \mid \varepsilon$

$T \to F\,T'$

$T' \to *\,F\,T' \mid \varepsilon$

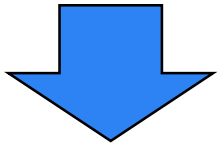
$F \to (E) \mid \mathrm{id}$

$A' \Rightarrow \alpha A'$
$\Rightarrow \alpha\alpha A'$
$\Rightarrow \alpha\alpha\alpha A'$
…
$\Rightarrow \alpha \ldots\alpha A'$
$\Rightarrow \alpha \ldots\alpha$

消除直接左递归的一般形式

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$$

$$(\alpha_i \neq \varepsilon, \beta_j \text{不以} A \text{开头})$$



$$A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$$

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

消除左递归是要付出代价的——引进了一些非终结符和ε_产生式

消除间接左递归

$$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$$
$$(\alpha_i \neq \varepsilon, \beta_j \text{不以} A \text{开头})$$
$$A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_m A'$$
$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_n A' \mid \varepsilon$$

- 例

$$S \to Aa \mid b$$

$$S \Rightarrow Aa$$
$$\Rightarrow Sda$$

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

- 将S的定义代入A-产生式，得：

$$A \to Ac \mid Aad \mid bd \mid \varepsilon$$

- 消除A-产生式的直接左递归，得：

$$A \to bdA' \mid A'$$
$$A' \to cA' \mid adA' \mid \varepsilon$$

消除左递归算法

- 输入：不含循环推导（即形如$A \Rightarrow^{+} A$的推导）和ε-产生式的文法G
- 输出：等价的无左递归文法
- 方法：

1) 按照某个顺序将非终结符号排序为A_1，A_2，…，A_n .
2) for (从1到n的每个i) {
3) for (从1到i -1的每个i) {
4) 将每个形如$A_i \rightarrow A_j \gamma$的产生式替换为产生式组 $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \ldots \mid \delta_k \gamma$，其中$A_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$，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) }

问题2

同一非终结符的多个候选式存在共同前缀，或含有左递归将导致回溯现象

- 例
  - 文法G

    $S \rightarrow aAd \mid aBe$

    $A \rightarrow c$

    $B \rightarrow b$
  - 输入

    $a\ b\ c$

    ↑
- 例
  - 文法G

    $S \rightarrow b \mid Aa$

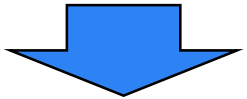
    $A \rightarrow ba$
  - 输入

    $b\ a\ a$

    ↑

提取左公因子(*Left Factoring*)

- 例
  - 文法G
    - $S \rightarrow aAd \mid aBe$
    - $A \rightarrow c$
    - $B \rightarrow b$



通过改写产生式来推迟决定，等读入了足够多的输入，获得足够信息后再做出正确的选择

  - 文法G'
    - $S \rightarrow aS'$
    - $S' \rightarrow Ad \mid Be$
    - $A \rightarrow c$
    - $B \rightarrow b$

提取左公因子算法

- 输入：文法G
- 输出：等价的提取了左公因子的文法
- 方法：

对于每个非终结符A，找出它的两个或多个选项之间的最长公共前缀α。如果$\alpha \neq \varepsilon$，即存在一个非平凡的(*nontrivial*)公共前缀，那么将所有A-产生式

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

替换为

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

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

其中，γ_i 表示所有不以α开头的产生式体；A'是一个新的非终结符。不断应用这个转换，直到每个非终结符的任意两个产生式体都没有公共前缀为止

Exercise

1．采用自上而下分析，不必(　)。

A.消除回溯　　B.消除左递归

C.消除右递归 D.提取公共左因子

2．在自上而下的语法分析中，应从(　)开始分析。

A.句型　　B.句子

C.文法开始符号 D.句柄

3．语法分析器的输入是（　）。

A.Token序列　　B.源程序

C.目标程序　　D.符号表

4．在递归子程序方法中，若文法存在左递归，则会使分析过程产生(　)。

A.回溯　　B.非法调用

C.有限次调用　　D.无限循环

提纲

S_文法

假如允许S_文法包含ε产生式，
将会产生什么问题？

- 预测分析法的工作过程
  - 从文法开始符号出发，在每一步推导过程中根据当前句型的最左非终结符A和当前输入符号a，选择正确的A-产生式。为保证分析的确定性，选出的候选式必须是唯一的。
- *S*_文法（简单的确定性文法，*Korenjak & Hopcroft*，1966）

每个产生式的右部都以终结符开始

同一非终结符的各个候选式的首终结符都不同

*S*_文法不含ε产生式

例

➢ 文法

① $S \to aBC$
② $B \to bC$
③ $B \to dB$
④ $B \to \varepsilon$
⑤ $C \to c$
⑥ $C \to a$
⑦ $D \to e$

➢ 输入	a d a	a d e
➢ 推导	S $\Rightarrow aBC$ $\Rightarrow adBC$ $\Rightarrow adC$ $\Rightarrow ada$	S $\Rightarrow aBC$ $\Rightarrow adBC$ $\Rightarrow adC$

可以紧跟B后面出现的终结符：c、a

➢ 什么时候使用ε产生式是合理的？

➢ 如果当前某非终结符A与当前输入符a不匹配时，若存在$A \to \varepsilon$，可以通过检查a是否可以出现在A的后面，来决定是否使用产生式$A \to \varepsilon$（若文法中无$A \to \varepsilon$，则应报错）

非终结符的后继符号集

- 非终结符A的后继符号集
  - 可能在某个句型中紧跟在A后边的终结符a的集合，记为$FOLLOW(A)$

$FOLLOW(A)=\{a \mid S \Rightarrow^{*} \alpha A a \beta, a \in V_T, \ \alpha,\beta \in (V_T \cup V_N)^{*}\}$

例

(1) $S \rightarrow aBC$

(2) $B \rightarrow bC$ ← 输入 b

(3) $B \rightarrow dB$ ← d

(4) $B \rightarrow \varepsilon$ ← $\{a, c\}$

(5) $C \rightarrow c$

(6) $C \rightarrow a$

$FOLLOW(B) = \{a,c\}$

如果A是某个句型的最右符号，
则将结束符“$”添加到$FOLLOW(A)$中

产生式的可选集

- 产生式$A\rightarrow\beta$的可选集是指可以选用该产生式进行推导时对应的输入符号的集合，记为$SELECT(A\rightarrow\beta)$
  - $SELECT(A\rightarrow a\beta)=\{a\}$
  - $SELECT(A\rightarrow\varepsilon)=FOLLOW(A)$
- q_文法
  - 每个产生式的右部或为ε，或以终结符开始
  - 具有相同左部的产生式有不相交的可选集

q_文法不含右部以非终结符开始的产生式

串首终结符集

- **串首终结符**
  - 串首第一个符号，并且是终结符。简称首终结符
- 给定一个文法符号串α，α的**串首终结符集*FIRST(α)***被定义为可以从α推导出的所有串首终结符构成的集合。如果$\alpha \Rightarrow^* \varepsilon$，那么ε也在*FIRST(α)*中
  - 对于$\forall\alpha\in(V_T\cup V_N)^+$, $FIRST(\alpha)=\{a \mid \alpha \Rightarrow^* a\beta,\ a\in V_T,\ \beta\in(V_T\cup V_N)^*\}$;
  - 如果$\alpha \Rightarrow^* \varepsilon$，那么$\varepsilon\in FIRST(\alpha)$

计算文法符号X的$FIRST(X)$

- $FIRST(X)$：可以从X推导出的所有串首终结符构成的集合
- 如果$X \Rightarrow^* \varepsilon$，那么$\varepsilon \in FIRST(X)$

- 例

① $E \to TE'$　　$FIRST(E) = \{ \quad (\quad id \quad \}$

② $E' \to +TE' \mid \varepsilon$　　$FIRST(E') = \{ \quad + \quad \varepsilon \quad \}$

③ $T \to FT'$　　$FIRST(T) = \{ \quad (\quad id \quad \}$

④ $T' \to *FT' \mid \varepsilon$　　$FIRST(T') = \{ \quad * \quad \varepsilon \quad \}$

⑤ $F \to (E) \mid \mathrm{id}$　　$FIRST(F) = \{ \quad (\quad id \quad \}$

算法

- 不断应用下列规则，直到没有新的终结符或ε可以被加入到任何*FIRST*集合中为止
  - 如果X是一个终结符，那么$FIRST(X)=\{X\}$
  - 如果X是一个非终结符，且$X\rightarrow Y_1\dots Y_k\in P(k\geq 1)$，那么如果对于某个$i$，$a$在$FIRST(Y_i)$中且$\varepsilon$在所有的$FIRST(Y_1),\dots,FIRST(Y_{i-1})$中(即$Y_1\dots Y_{i-1}\Rightarrow^{*}\varepsilon$)，就把$a$加入到$FIRST(X)$中。如果对于所有的$j=1,2,\dots,k$，$\varepsilon$在$FIRST(Y_j)$中，那么将$\varepsilon$加入到$FIRST(X)$
  - 如果$X\rightarrow\varepsilon\in P$，那么将$\varepsilon$加入到$FIRST(X)$中

计算串$X_1X_2\ldots X_n$的$FIRST$集合

- 向$FIRST(X_1X_2\ldots X_n)$加入$FIRST(X_1)$中所有的非ε符
- 如果ε在$FIRST(X_1)$中，再加入$FIRST(X_2)$中的所有非ε符号；如果ε在$FIRST(X_1)$和$FIRST(X_2)$中，再加入$FIRST(X_3)$中的 所有非ε符号，以此类推
- 最后，如果对所有的i，ε都在$FIRST(X_i)$中，那么将ε加入到$FIRST(X_1X_2\ldots X_n)$ 中

计算非终结符A的$FOLLOW(A)$

- $FOLLOW(A)$：可能在某个句型中紧跟在A后边的终结符a的集合

  $FOLLOW(A)=\{a \mid S \Rightarrow^{*} \alpha A a \beta, a \in V_T, \ \alpha,\beta \in (V_T \cup V_N)^{*}\}$

- 如果A是某个句型的的最右符号，则将结束符“$”添加到$FOLLOW(A)$中

例

① $E \rightarrow TE'$　　$FIRST(E) = \{ (, id \}$　　$FOLLOW(E) = \{ \$,) \}$

② $E' \rightarrow +TE' \mid \varepsilon$　　$FIRST(E') = \{ +, \varepsilon \}$　　$FOLLOW(E') = \{ \$,) \}$

③ $T \rightarrow FT'$　　$FIRST(T) = \{ (, id \}$　　$FOLLOW(T) = \{ +, \$,) \}$

④ $T' \rightarrow *FT' \mid \varepsilon$　　$FIRST(T') = \{ *, \varepsilon \}$　　$FOLLOW(T') = \{ +, \$,) \}$

⑤ $F \rightarrow (E) \mid id$　　$FIRST(F) = \{ (, id \}$　　$FOLLOW(F) = \{ *, +, \$,) \}$

算法

- 不断应用下列规则，直到没有新的终结符可以被加入到任何*FOLLOW*集合中为止
  - 将$放入*FOLLOW*(*S*)中，其中*S*是开始符号，$是输入右端的结束标记
  - 如果存在一个产生式$A \rightarrow \alpha B \beta$，那么*FIRST*(*β*)中除ε之外的所有符号都在*FOLLOW*(*B*)中
  - 如果存在一个产生式$A \rightarrow \alpha B$，或存在产生式$A \rightarrow \alpha B \beta$且*FIRST*(*β*)包含ε，那么*FOLLOW*(*A*)中的所有符号都在*FOLLOW*(*B*)中

产生式的可选集的重新定义

- 产生式$A\rightarrow\beta$的可选集是指可以选用该产生式进行推导时对应的输入符号的集合，记为$SELECT(A\rightarrow\beta)$
  - $SELECT(A\rightarrow a\beta)=\{a\}$
  - $SELECT(A\rightarrow\varepsilon)=FOLLOW(A)$

q_文法不含右部以非终结符开始的产生式

- 产生式$A\rightarrow\alpha$的可选集$SELECT$
  - 如果$\varepsilon\notin FIRST(\alpha)$，那么$SELECT(A\rightarrow\alpha)=FIRST(\alpha)$
  - 如果$\varepsilon\in FIRST(\alpha)$，那么$SELECT(A\rightarrow\alpha)=(FIRST(\alpha)-\{\varepsilon\})\cup FOLLOW(A)$

LL(1)文法

LL(1)文法

- 一个上下文无关文法是LL(1)文法的充要条件是，对每个非终结符A的任意两个不同产生式$A \to \alpha$，$A \to \beta$，满足：

$$SELECT(A \to \alpha) \cap SELECT(A \to \beta) = \Phi$$

其中，α、β不能同时$\Rightarrow^{*} \varepsilon$

同一非终结符的各个产生式的可选集互不相交

可以为LL(1)文法构造预测分析器

$LL(1)$文法

- 一个上下文无关文法是$LL(1)$文法的充要条件是，对每个非终结符A的任意两个不同产生式$A \to \alpha$，$A \to \beta$，满足：

$$SELECT(A \to \alpha) \cap SELECT(A \to \beta) = \Phi$$

其中，α、β不能同时$\Rightarrow^{*} \varepsilon$

$LL(1)$文法不含左递归

- 第一个“L”表示从左向右扫描输入
- 第二个“L”表示产生最左推导
- “1”表示在每一步中只需要向前看一个输入符号来决定语法分析动作

例：表达式文法各产生式的*SELECT*集

X	*FIRST*(*X*)	*FOLLOW*(*X*)
E	(id	$)
E '	+ ε	$)
T	(id	+) $
T '	* ε	+) $
F	(id	* +) $

表达式文法是 *LL*(1) 文法

(1) $E \rightarrow T\,E'$ *SELECT* (1)= { (id }

(2) $E' \rightarrow +\,T\,E'$ *SELECT* (2)= { + }

(3) $E' \rightarrow \varepsilon$ *SELECT* (3)= { $) }

(4) $T \rightarrow F\,T'$ *SELECT* (4)= { (id }

(5) $T' \rightarrow *\,F\,T'$ *SELECT* (5)= { * }

(6) $T' \rightarrow \varepsilon$ *SELECT* (6)= { +) $ }

(7) $F \rightarrow (\,E\,)$ *SELECT* (7)= { (}

(8) $F \rightarrow$ id *SELECT* (8)= { id }

预测分析表

	产生式	SELECT
E	E→TE′	(id
E′	E′→+TE’	+
	E′→ε	$)
T	T→FT′	(id
T′	T′→*FT’	*
	T′→ε	+) $
F	F→(E)	(
	F→id	id

文法产生式的可选集

文法的预测分析表

非终结符	输入符号					
	id	+	*	(	)	$
E	E→TE′			E→TE′		
E′		E′→+TE’			E′→ε	E′→ε
T	T→FT′			T→FT′		
T′		T′→ε	T′→*FT’		T′→ε	T′→ε
F	F→id			F→(E)		

Exercise

1、LL(1)分析法中“1”的含义是在输入串中查看一个输入符号，其目的是（　　）。

A.确定最左推导

B.确定句柄

C.确定使用哪一个产生式进行展开

D.确定是否推导

2、一个文法G，若(　)，则称它是LL(1)文法。

A.G中不含左递归

B.G无二义性

C.G的LL(1)分析表中不含多重定义的条目

D.G中产生式不含左公因子

Exercise

3、在语法分析处理中，FIRST集合、FOLLOW集合均是(　)。

A.非终结符集

B.终结符集

C.字母表

D.状态集

4、已知文法G[S]:

S→eT|RT

T→DR|ε

R→dR|ε

D→a|bd

求FIRST(S)=（　　）

A.{e }

B.{e，d，a，b}

C.{e，d }

D.{e，d，a，b，ε}

Exercise

5、已知文法G[S]:

S→eT|RT

T→DR|ε

R→dR|ε

D→a|bd

求FOLLOW(D)=（　　）

A.{d，e}

B.{d，ε}

C.{d，$}

D.{a，d}

6、FIRST集中可以含有ε（　）

7、FOLLOW集中可以含有ε（　）

8、SELECT集中可以含有ε(　)

LL(1)文法的分析方法

- 递归的预测分析法（递归下降LL(1)分析）
- 非递归的预测分析法（表驱动的预测分析）

提纲

递归的预测分析法

- 递归的预测分析法是指：在递归下降分析中，根据预测分析表进行产生式的选择
  - 根据每个非终结符的产生式和$LL(1)$文法的预测分析表，为每个非终结符编写对应的过程

```
    void A( ) {
1)    选择一个A产生式，A → X1 X2 … Xk ;
2)    for ( i = 1 to k ) {
3)        if ( Xi是一个非终结符号)
4)            调用过程 Xi( ) ;
5)        else  if ( Xi 等于当前的输入符号a)
6)            读入下一个输入符号;
7)        else /* 发生了一个错误 */ ;
        }
    }
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
program DESCENT;
    begin
        GETNEXT(TOKEN);
        PROGRAM(TOKEN);
        GETNEXT(TOKEN);
       if TOKEN≠'$'  then ERROR;
    end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure PROGRAM(TOKEN);
    begin
    if TOKEN≠'program' then ERROR;

    GETNEXT(TOKEN);
    DECLIST(TOKEN);

    GETNEXT(TOKEN);
    if TOKEN≠':' then ERROR;

    GETNEXT(TOKEN);
    TYPE(TOKEN);

    GETNEXT(TOKEN);
    if TOKEN≠';' then ERROR;

    GETNEXT(TOKEN);
    STLIST(TOKEN);

    if TOKEN≠'end' then ERROR;
    end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure DECLIST(TOKEN);
    begin
        if TOKEN≠'id'  then ERROR;

        GETNEXT(TOKEN);
        DECLISTN(TOKEN);
    end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure DECLISTN(TOKEN);
        begin
            if TOKEN =',' then
              begin
                GETNEXT(TOKEN);
                 if TOKEN≠'id'  then ERROR;

                GETNEXT(TOKEN);
                DECLISTN(TOKEN);
              end
            else  if TOKEN≠':'  then ERROR;
        end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure STLIST(TOKEN);
    begin
        if TOKEN≠'s' then ERROR;
        GETNEXT(TOKEN);
        STLISTN(TOKEN);
    end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

SELECT(4)={:}
SELECT(7)={end}

```
procedure STLISTN(TOKEN);
        begin
          if TOKEN =';' then
            begin
              GETNEXT(TOKEN);
              if TOKEN≠'s'  then ERROR;

              GETNEXT(TOKEN);
              STLISTN(TOKEN);
            end
          else  if TOKEN≠'end'  then ERROR;

        end
```

例

(1) <PROGRAM> → program <DECLIST> :<TYPE> ; <STLIST> end

(2) <DECLIST> → id <DECLISTN>

(3) <DECLISTN> → , id <DECLISTN>

(4) <DECLISTN> → ε

(5) <STLIST> → s <STLISTN>

(6) <STLISTN> → ; s <STLISTN>

(7) <STLISTN> → ε

(8) <TYPE> → real

(9) <TYPE> → int

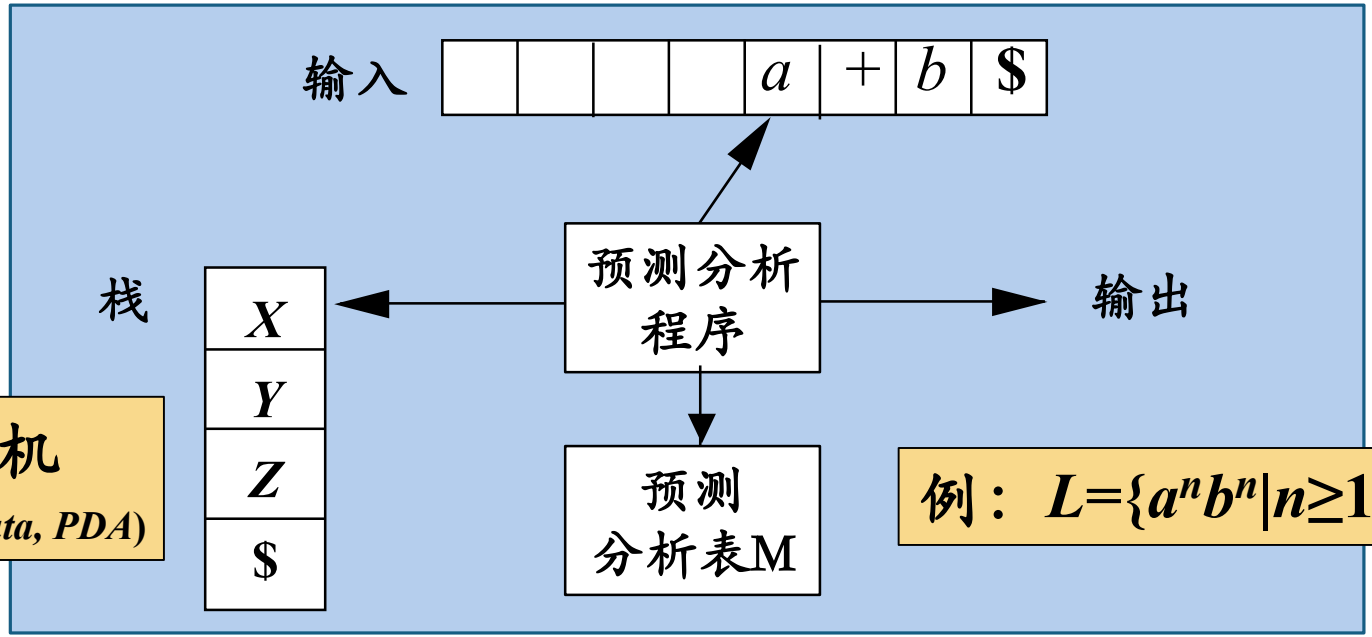
SELECT(4)={:}
SELECT(7)={end}

```
procedure TYPE(TOKEN);
        begin
            if TOKEN≠'real' and TOKEN≠'int'
                then ERROR;
        end
```

提纲

非递归的预测分析法

- 非递归的预测分析不需要为每个非终结符编写递归下降过程，而是根据预测分析表构造一个自动机，也叫表驱动的预测分析



下推自动机

(Push Down Automata, PDA)

例

非终结符	输入符号					
	id	+	*	(	)	$
E	$E\to TE'$			$E\to TE'$		
E'		$E'\to +TE'$			$E'\to\varepsilon$	$E'\to\varepsilon$
T	$T\to FT'$			$T\to FT'$		
T'		$T'\to\varepsilon$	$T'\to *FT'$		$T'\to\varepsilon$	$T'\to\varepsilon$
F	$F\to$id			$F\to(E)$		

如果w是至今为止已经匹配完成的输入部分，那么栈中保存的文法符号序列α满足 $S\Rightarrow^*_{lm} w\alpha$

栈	剩余输入	输出
E $	id+id*id $	
TE' $	id+id*id $	$E\to TE'$
$FT'E'$ $	id+id*id $	$T\to FT'$
id$T'E'$ $	id+id*id $	$F\to$id
$T'E'$ $	+id*id $	
E' $	+id*id $	$T'\to\varepsilon$
$+TE'$ $	+id*id $	$E'\to +TE'$
TE' $	id*id $	
$FT'E'$ $	id*id $	$T\to FT'$
id$T'E'$ $	id*id $	$F\to$id
$T'E'$ $	*id $	
$*FT'E'$ $	*id $	$T'\to *FT'$
$FT'E'$ $	id $	
id$T'E'$ $	id $	$F\to$id
$T'E'$ $	$	
E' $	$	$T'\to\varepsilon$
$	$	$E'\to\varepsilon$

表驱动的预测分析法

- 输入：一个串w和文法G的分析表M
- 输出：如果w在$L(G)$中，输出w的最左推导；否则给出错误指示
- 方法：最初，语法分析器的格局如下：输入缓冲区中是$w\$$，G的开始符号位于栈顶，其下面是$\$$。下面的程序使用预测分析表M生成了处理这个输入的预测分析过程

```
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≠ $ ) {    /* 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 →Y1Y2 … Yk ) {
                输出产生式X →Y1Y2 … Yk ;
                弹出栈顶符号;
                将Yk，Yk-1 …，Yi 压入栈中，其中Y1位于栈顶。
        }
        令X=栈顶符号
}
```

递归的预测分析法vs.非递归的预测分析法

	递归的预测分析法	非递归的预测分析法
程序规模	程序规模较大， 不需载入分析表	主控程序规模较小， 需载入分析表（表较小）
直观性	较好	较差
效率	较低	分析时间大约正比于待分析程序的长度
自动生成	较难	较易

预测分析法实现步骤

1）构造文法

2）改造文法：消除二义性、消除左递归、消除回溯

3）求每个变量的*FIRST*集和*FOLLOW*集，从而求得每个候选式的*SELECT*集

4）检查是不是*LL*(1) 文法。若是，构造预测分析表

5）对于递归的预测分析，根据预测分析表为每一个非终结符编写一个过程；对于非递归的预测分析，实现表驱动的预测分析算法

提纲

预测分析中的错误检测

- 两种情况下可以检测到错误
  - 栈顶非终结符与当前输入符号在预测分析表对应项中的信息为空
  - 栈顶的终结符和当前输入符号不匹配

预测分析中的错误恢复

- 恐慌模式
  - 忽略输入中的一些符号，直到输入中出现由设计者选定的同步词法单元(*synchronizing token*)集合中的某个词法单元
    - 其效果依赖于同步集合的选取。集合的选取应该使得语法分析器能从实际遇到的错误中快速恢复
      - 例如可以把*FOLLOW(A)*中的所有终结符放入非终结符*A*的同步记号集合
  - 如果终结符在栈顶而不能匹配，一个简单的办法就是弹出此终结符

例

非终结符	输入符号					
	id	+	*	(	)	$
E	$E\to TE'$			$E\to TE'$	synch	synch
E'		$E'\to +TE'$			$E'\to\varepsilon$	$E'\to\varepsilon$
T	$T\to FT'$	synch		$T\to FT'$	synch	synch
T'		$T'\to\varepsilon$	$T'\to *FT'$		$T'\to\varepsilon$	$T'\to\varepsilon$
F	$F\to$id	synch	synch	$F\to(E)$	synch	synch

X	FOLLOW(X)
E	$)
E'	$)
T	+) $
T'	+) $
F	* +) $

*Synch*表示根据相应非终结符的*FOLLOW*集得到的同步词法单元

- 分析表的使用方法
  - 如果M[*A*,*a*]是空，表示检测到错误，根据恐慌模式，忽略输入符号*a*
  - 如果M[*A*,*a*]是*synch*，则弹出栈顶的非终结符*A*，试图继续分析后面的语法成分
  - 如果栈顶的终结符和输入符号不匹配，则弹出栈顶的终结符

非终结符	输入符号					
	id	+	*	(	)	$
E	E→TE′			E→TE′	synch	synch
E′		E′→+TE′			E′→ε	E′→ε
T	T→FT′	synch		T→FT′	synch	synch
T′		T′→ε	T′→*FT′		T′→ε	T′→ε
F	F→id	synch	synch	F→(E)	synch	synch

栈	剩余输入	
E $	+id*+id $	ignore +
E $	id*+id $	
TE′ $	id*+id $	
FT′E′ $	id*+id $	
idT′E′ $	id*+id $	
T′E′ $	*+id $	
*FT′E′ $	*+id $	
FT′E′ $	+id $	error
T′E′ $	+id $	
E′ $	+id $	
+TE′ $	+id $	
TE′ $	id $	
FT′E′ $	id $	
idT′E′ $	id $	
T′E′ $	$	
E′ $	$	
$	$	

- 如果M[*A*,*a*]是空，表示检测到错误，根据恐慌模式，忽略输入符号*a*
- 如果M[*A*,*a*]是*synch*，则弹出栈顶的非终结符*A*，试图继续分析后面的语法成分
- 如果栈顶的终结符和输入符号不匹配，则弹出栈顶的终结符

自顶向下分析小结

- 如何实现自顶向下分析?
  - 递归下降分析法
- 什么样的文法能够实现确定的自顶向下分析?
  - LL(1)文法
- 如何实现确定的自顶向下分析?
  - 递归的方式：　　基于预测分析表对递归下降分析法进行扩展
  - 非递归的方式：

递归的预测分析

- 是对递归下降分析框架的扩展

```
A(Token)
{
    if Token ∈SELECT(A→ α1 )
            code1;
    if Token ∈SELECT(A→ α2 )
            code2;
            …
    if Token ∈SELECT(A→ αn )
            coden;
}
```

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

code*1*　code*2*　code*n*

codei: 设$\alpha_i = X_1X_2\ldots X_k$

```
for (j=1 to k)
    if Xj ∈ VT    if Xj ==Token  then GetNext(Token)
                  else ( Xj ≠Token)     Error( )
    else Xj ∈ VN   Xj ( )
```

非递归的预测分析

1.（20分）已知文法G[S]：

S→aH　　H→aMd|d

M→Ab|ε　　A→aM|e

(1)判断G 是否是LL(1)文法,若是，请构造相应的LL(1)预测分析表(15分);

(2)如果是LL(1)文法，请给出输入串aaabd$的预测过程（5分）

非终结符	FIRST集	FOLLOW集
S	{a}	{$}
H	{a, d}	{$}
M	{a, e, ε}	{d, b}
A	{a, e}	{b}

	a	d	b	e	$
S	S->aH				
H	H->aMd	H->d			
M	M->Ab	M-> ε	M-> ε	M->Ab	
A	A->aM			A->e	

非递归的预测分析

1.（20分）已知文法G[S]：

S→aH　　H→aMd|d

M→Ab|ε　　A→aM|e

(1)判断G是否是LL(1)文法,若是，请构造相应的LL(1)预测分析表；

(2)如果是LL(1)文法，请给出输入串aaabd$的预测过程

	a	d	b	e	$
S	S->aH				
H	H->aMd	H->d			
M	M->Ab	M-> ε	M-> ε	M->Ab	
A	A->aM			A->e	

步骤	分析栈	输入串	推导使用产生式
1	S$	aaabd$	S→aH
2	aH$	aaabd$	‘a’匹配
3	H$	aabd$	H→aMd
4	aMd$	aabd$	‘a’匹配
5	Md$	abd$	M→Ab
6	Abd$	abd$	A→a M
7	aMbd$	abd$	‘a’匹配
8	Mbd$	bd$	M→ε
9	bd$	bd$	‘b’匹配
10	d$	d$	‘d’匹配
11	$	$	分析成功