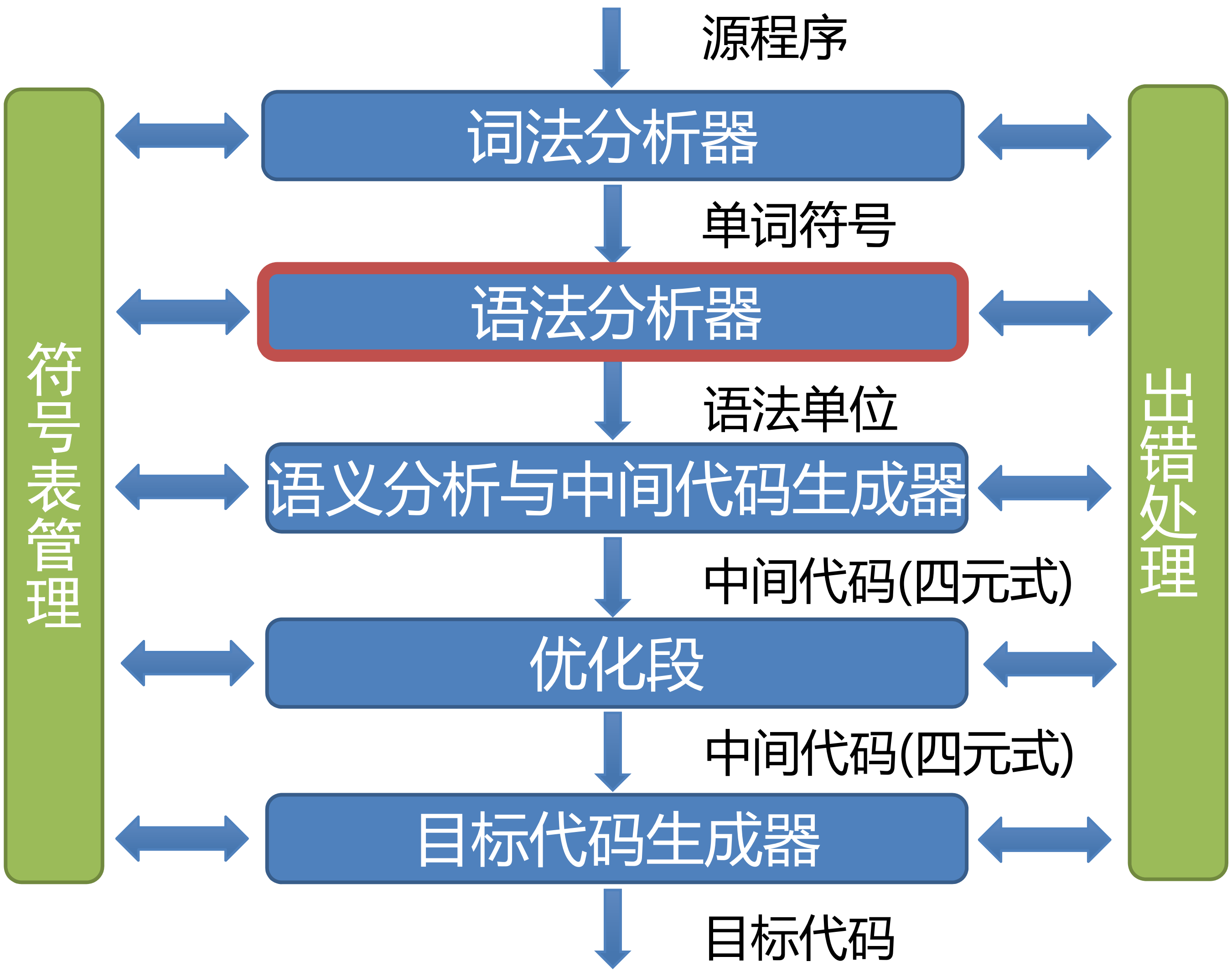
编译原理

自上而下分析的基本问题

编译原理

语法分析基本概念

编译程序总框


源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间代码生成器
中间代码(四元式)
优化段
中间代码(四元式)
目标代码生成器
目标代码
符号表管理
出错处理

语法分析的前提

- 对语言的语法结构进行描述
  - 采用正规式和有限自动机描述和识别语言的单词符号
  - 用上下文无关文法来描述语法规则

上下文无关文法

- 一个上下文无关文法G是一个四元式
  $G=(V_T, V_N, S, P)$，其中
  - V_T：终结符集合(非空)
  - V_N：非终结符集合(非空)，且$V_T \cap V_N=\varnothing$
  - S：文法的开始符号，$S\in V_N$
  - P：产生式集合(有限)，每个产生式形式为
    - $P\to\alpha$， $P\in V_N$， $\alpha \in (V_T \cup V_N)^*$
  - 开始符S至少必须在某个产生式的左部出现一次

上下文无关文法

- 定义：称$\alpha A\beta$**直接推出**$\alpha\gamma\beta$，即

$$\alpha A\beta \Rightarrow \alpha\gamma\beta$$

仅当$A \rightarrow \gamma$是一个产生式，

且α，$\beta \in (V_T \cup V_N)^*$。

- 如果$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$，则我们称这个序列是从$\alpha_1$到$\alpha_n$的一个**推导**。若存在一个从$\alpha_1$到$\alpha_n$的推导，则称$\alpha_1$可以**推导出**$\alpha_n$

句子、句型和语言

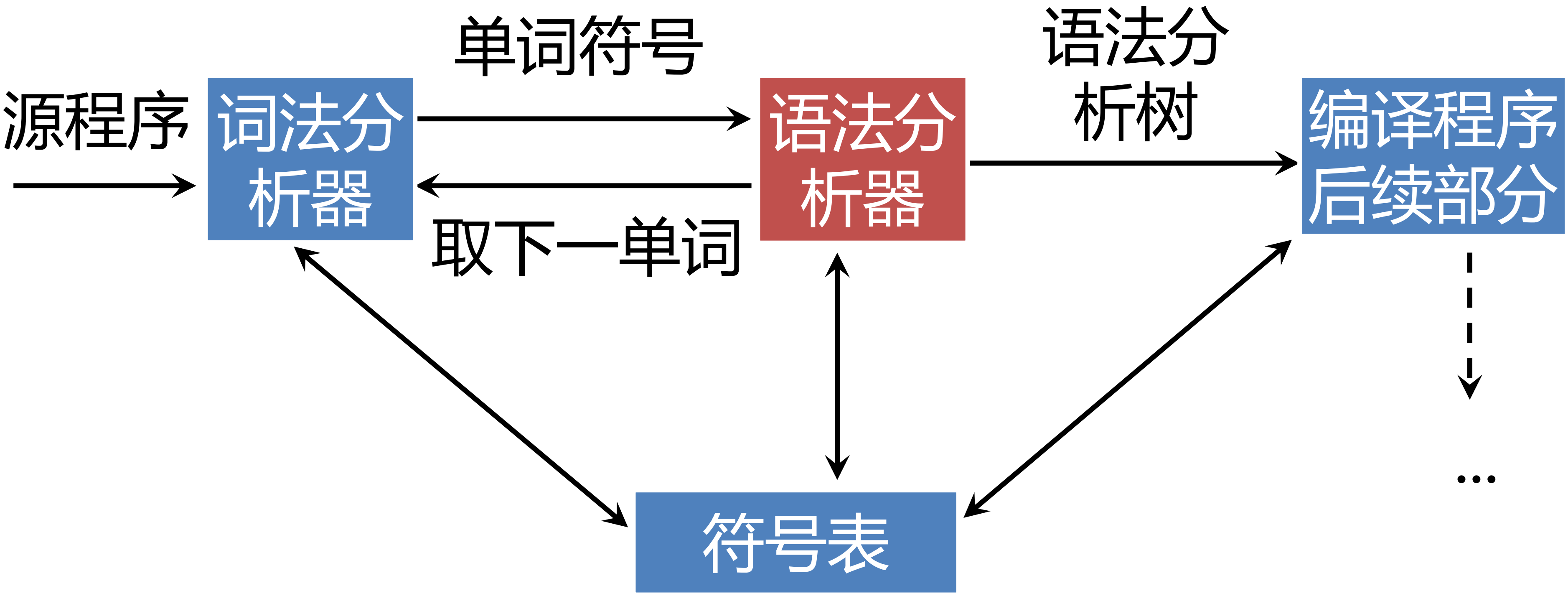
- 定义：假定G是一个文法，S 是它的开始符号。如果$S \overset{*}{\Rightarrow} \alpha$，则$\alpha$称是一个句型。
- 仅含终结符号的句型是一个句子。
- 文法G所产生的句子的全体是一个语言，将它记为 L(G)。

$$L(G) = \{\alpha \mid S \overset{*}{\Rightarrow} \alpha, \alpha \in V_T^*\}$$

语法分析的任务

- 语法分析的任务
  - 分析一个文法的句子的结构
- 语法分析器的功能
  - 按照文法的产生式(语言的语法规则)，识别输入符号串是否为一个句子(合式程序)

语法分析器在编译器中的地位



语法分析的方法

- 自下而上(Bottom-up)
  - 从输入串开始，逐步进行归约，直到文法的开始符号
  - 归约：根据文法的产生式规则，把串中出现的产生式的右部替换成左部符号
  - 从树叶节点开始，构造语法树
  - 算符优先分析法、LR分析法
- 自上而下(Top-down)
  - 从文法的开始符号出发，反复使用各种产生式，寻找"匹配"的推导
  - 推导：根据文法的产生式规则，把串中出现的产生式的左部符号替换成右部
  - 从树的根开始，构造语法树
  - 递归下降分析法、预测分析程序

编译原理

自上而下分析面临的问题

自上而下分析

- 基本思想
  - 从文法的开始符号出发，向下推导，推出句子
  - 针对输入串，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

自上而下分析示例

- 假定有文法G(S):

  (1) S→xAy

  (2) A→**|*

  分析输入串x*y(记为α)。

x*y　　　　**S**

IP

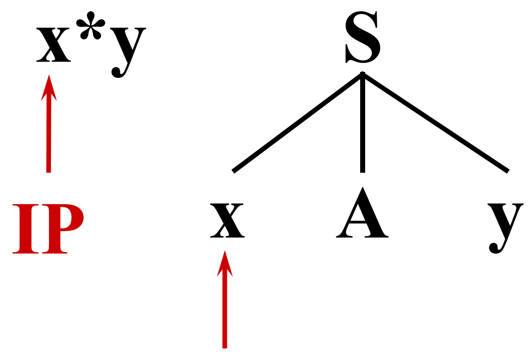
自上而下分析示例

- 假定有文法G(S):

  (1) S→xAy

  (2) A→**|*

  分析输入串x*y(记为α)。



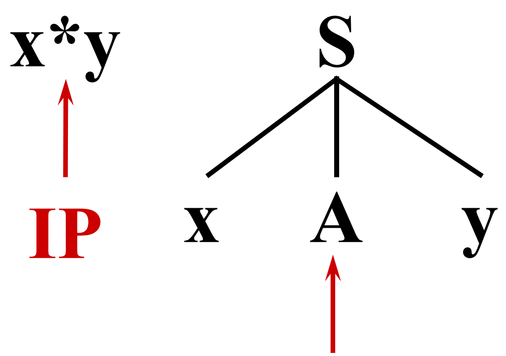
自上而下分析示例

- 假定有文法G(S):

  (1) S→xAy

  (2) A→**|*

  分析输入串x*y(记为α)。



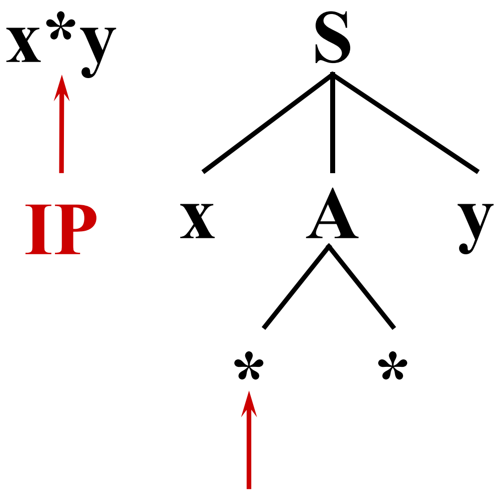
自上而下分析示例

- 假定有文法G(S):

  (1) S→xAy

  (2) A→**|*

  分析输入串x*y(记为α)。



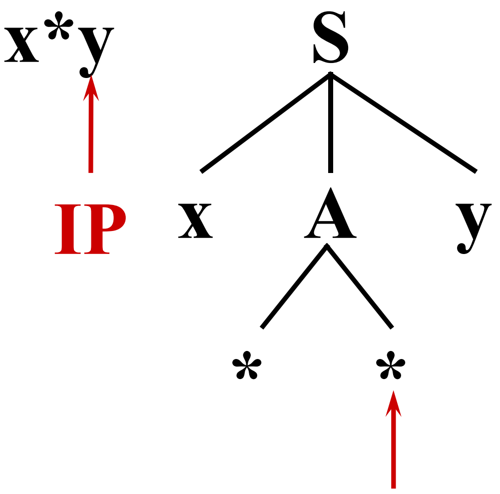
自上而下分析示例

- 假定有文法G(S):

  (1) S→xAy

  (2) A→**|*

  分析输入串x*y(记为α)。



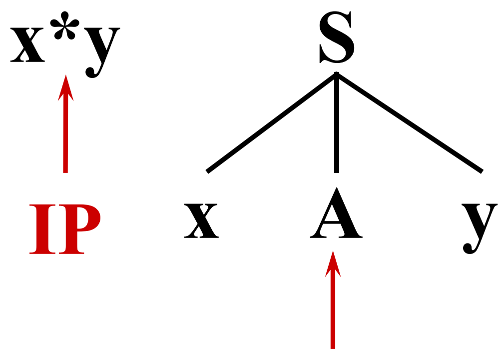
自上而下分析示例

- 假定有文法G(S):

  (1) S→xAy

  (2) A→**|*

  分析输入串x*y(记为α)。



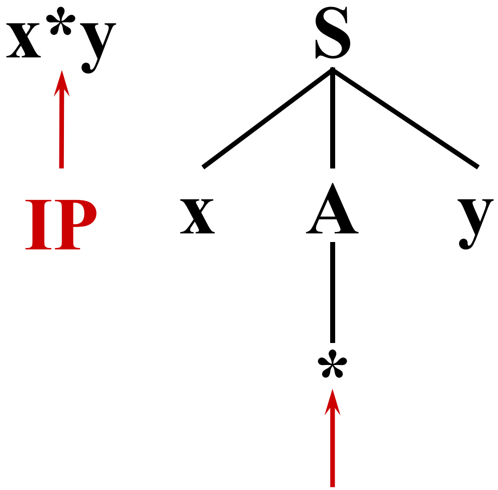
自上而下分析示例

- 假定有文法G(S):

  (1) S→xAy

  (2) A→**|*

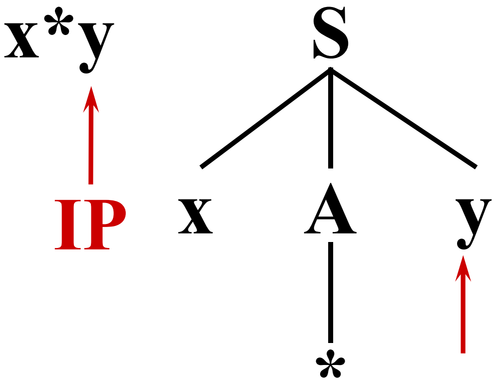
  分析输入串x*y(记为α)。



自上而下分析示例

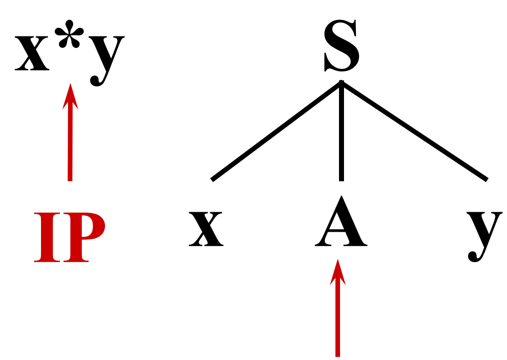
- 假定有文法G(S):
  (1) S→xAy
  (2) A→**|*

  分析输入串x*y(记为α)。



多个产生式候选带来的问题

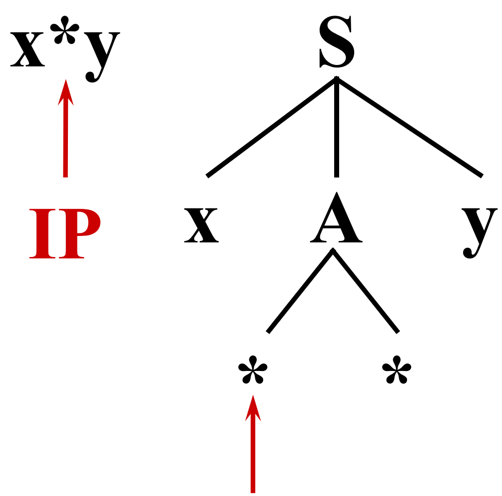
- 回溯问题
  - 分析过程中，当一个非终结符用某一个候选匹配成功时，这种匹配可能是暂时的
  - 出错时，不得不“回溯”



$A \rightarrow ** \mid *$

多个产生式候选带来的问题

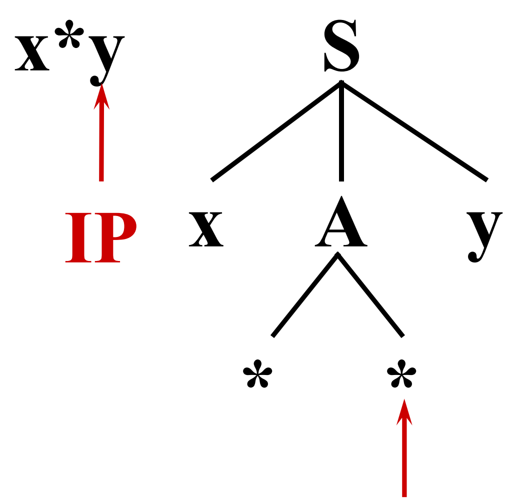
- 回溯问题
  - 分析过程中，当一个非终结符用某一个候选匹配成功时，这种匹配可能是暂时的
  - 出错时，不得不“回溯”



$A \rightarrow ** \mid *$

多个产生式候选带来的问题

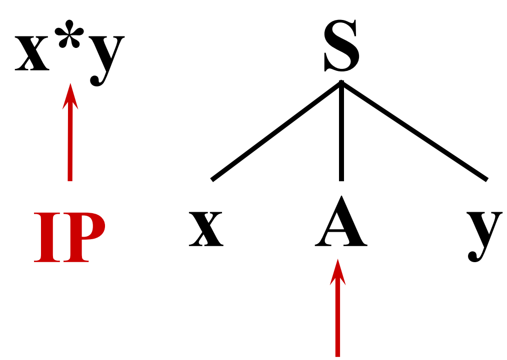
- 回溯问题
  - 分析过程中，当一个非终结符用某一个候选匹配成功时，这种匹配可能是暂时的
  - 出错时，不得不“回溯”



$A \to ** | *$

多个产生式候选带来的问题

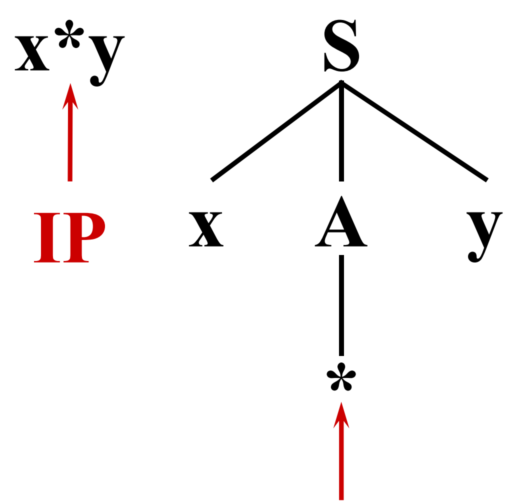
- 回溯问题
  - 分析过程中，当一个非终结符用某一个候选匹配成功时，这种匹配可能是暂时的
  - 出错时，不得不“回溯”



$A \rightarrow {*}{*} \mid {*}$

多个产生式候选带来的问题

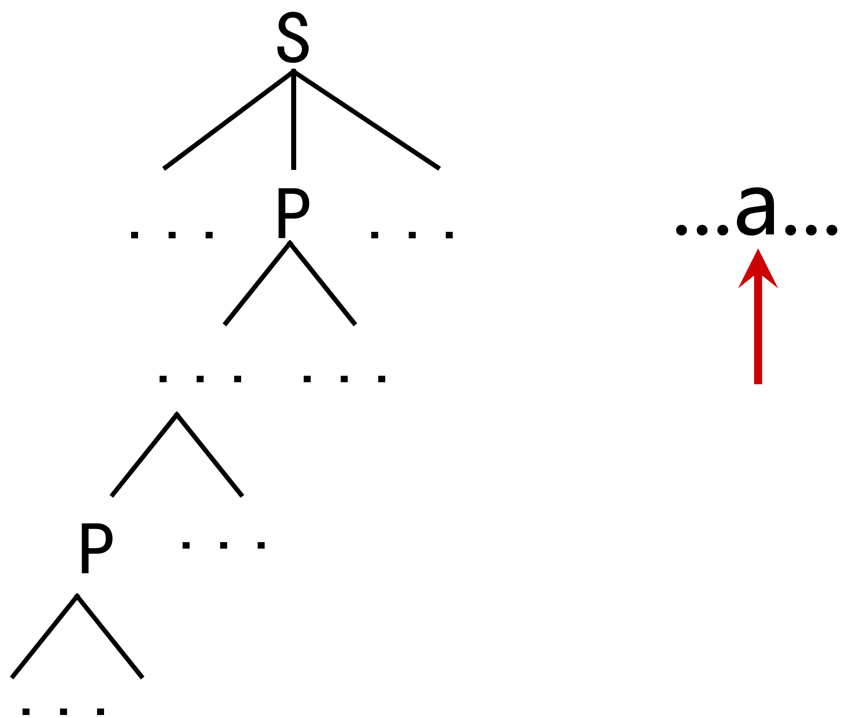
- 回溯问题
  - 分析过程中，当一个非终结符用某一个候选匹配成功时，这种匹配可能是暂时的
  - 出错时，不得不“回溯”



$A \to ** | *$

自上而下分析面临的问题

- 文法左递归问题
  - 一个文法是含有左递归的，如果存在非终结符P



自上而下分析面临的问题

- 回溯问题
- 文法左递归问题

小结

- 语法分析的基本概念
  - 自下而上分析
  - 自上而下分析
- 自上而下分析面临的问题
  - 回溯问题
  - 文法左递归问题

编译原理

LL(1)文法

自上而下分析

- 面临的问题
  - 文法左递归问题
  - 回溯问题
- 构造不带回溯的自上而下分析算法
  - 消除文法的左递归性
  - 消除回溯

自上而下分析

- 面临的问题
  - 文法左递归问题
  - 回溯问题
- 构造不带回溯的自上而下分析算法
  - 消除文法的左递归性
  - 消除回溯

编译原理

消除文法的左递归

直接左递归的消除

- 见诸于产生式中的直接左递归

$P \to P\alpha \mid \beta$ （β不以P开头）

$$
\begin{aligned}
P &\Rightarrow P\,\alpha \\
&\Rightarrow P\,\alpha\,\alpha \\
&\ldots\ldots \\
&\Rightarrow P\,\alpha\ldots\alpha \\
&\Rightarrow \beta\,\alpha\ldots\alpha
\end{aligned}
$$

- 左递归变右递归

$P \to \beta P'$

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

$$
\begin{aligned}
P &\Rightarrow \beta\,P' \\
&\Rightarrow \beta\,\alpha\,P' \\
&\Rightarrow \beta\,\alpha\,\alpha\,P' \\
&\ldots\ldots \\
&\Rightarrow \beta\,\alpha\ldots\alpha\,P' \\
&\Rightarrow \beta\,\alpha\ldots\alpha
\end{aligned}
$$

直接左递归的消除

- 假定P关于的全部产生式是

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

(每个α都不等于ε，每个β都不以P开头)

- 左递归变右递归

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

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

习题

- 给定文法G(E):

  $E \rightarrow E + T \mid T$

  $T \rightarrow T*F \mid F$

  $F \rightarrow (E) \mid i$

请消除其直接左递归。

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

变换成:

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

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

G(E):

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

间接左递归的消除

- 给定文法G(S):
  $S \to Qc|c$
  $Q \to Rb|b$
  $R \to Sa|a$

没有直接左递归，但S、Q、R都是左递归的

$$S \Rightarrow Qc \Rightarrow Rbc \Rightarrow Sabc$$

间接左递归的消除

- 一个文法消除左递归的条件
  - 不含以ε为右部的产生式
  - 不含回路

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

间接左递归的消除

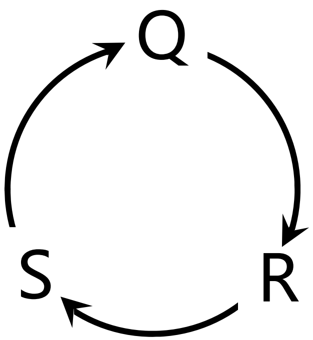
- 给定文法G(S):

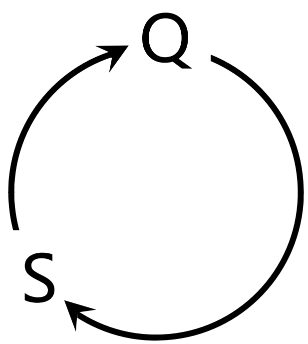
  S→Qc|c

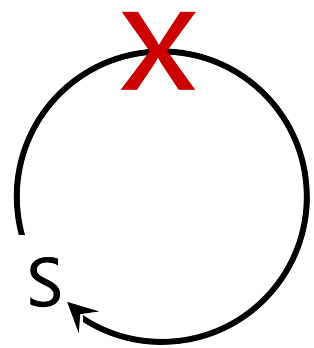
  Q→Rb|b

  R→Sa|a

$$S \Rightarrow Qc \Rightarrow Rbc \Rightarrow Sabc$$







S → Qc | c
Q → Rb | b
R → Sa | a

S → Qc | c
Q → Sab | ab | b
R → Sa | a

S → Sabc | abc | bc | c
Q → Sab | ab | b
R → Sa | a

消除左递归的算法

1. 把文法G的所有非终结符按任一种顺序排列 P_1，P_2，…，P_n；按此顺序执行：

2. FOR i:=1 TO n DO

    BEGIN

P_i 的规则改造成$P_i \to a\ldots \mid P_{i+1}\ldots \mid P_{i+2}\ldots \mid \ldots \mid P_{i+k}\ldots$

        FOR j:=1 TO i-1 DO

$P_j \to a\ldots \mid P_{j+1}\ldots \mid P_{j+2}\ldots \mid \ldots \mid P_{j+k}\ldots$

            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)

$P_i \to a\ldots \mid P_j\ldots \mid P_{i+1}\ldots \mid \ldots \mid P_{i+k}\ldots$

        消除关于P_i规则的直接左递归性

    END

$P_i \to a\ldots \mid P_{i+1}\ldots \mid P_{i+2}\ldots \mid \ldots \mid P_{i+k}\ldots$

3. 化简由2所得的文法，去除从开始符号出发永远无法到达的非终结符的产生规则。

习题

- 消除文法G(S)的左递归

$S \rightarrow Qc \mid c$

$Q \rightarrow Rb \mid b$

$R \rightarrow Sa \mid a$

$S \longrightarrow abcS' \mid bcS' \mid cS'$
$S' \longrightarrow abcS' \mid \varepsilon$

消除左递归的算法

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

- 注意，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。但不难证明，它们都是等价的。
- 例如，若对文法的非终结符排序选为S、Q、R，那么，最后所得的无左递归文法是：

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

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

- 上述两个文法是等价的。

自上而下分析

- 面临的问题
  - 文法左递归问题
  - 回溯问题
- 构造不带回溯的自上而下分析算法
  - 消除文法的左递归性
  - 消除回溯

编译原理

消除回溯

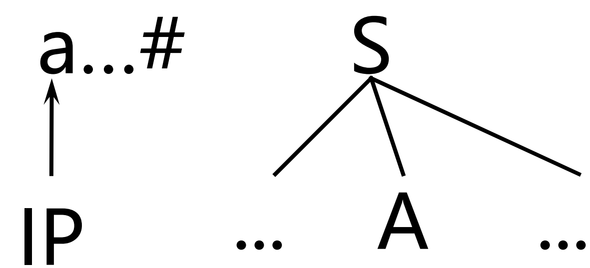
自上而下分析

- 面临的问题
  - 文法左递归问题
  - 回溯问题
- 构造不带回溯的自上而下分析算法
  - 消除文法的左递归性
  - 消除回溯

消除回溯

- 为了消除回溯必须保证
  - 对文法的任何非终结符，当要它去匹配输入串时，能够根据它所面临的输入符号准确地指派它的一个候选去执行任务，并且此候选的工作结果应是确信无疑的。

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



FIRST集合

- 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α定义它的终结首符集FIRST(α)为：

$$FIRST(\alpha) = \left\{ a \middle| \alpha \overset{*}{\Rightarrow} a \ldots, a \in V_T \right\}$$

特别是，若$\alpha \overset{*}{\Rightarrow} \varepsilon$，则规定$\varepsilon \in FIRST(\alpha)$。

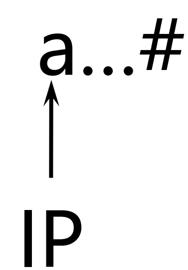
FIRST集合

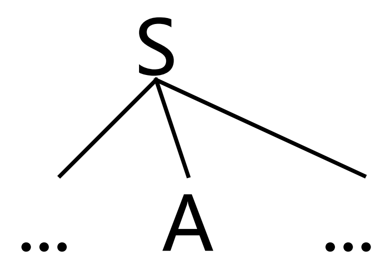
- 如果非终结符A的所有候选首符集两两不相交，即A的任何两个不同候选α_i和α_j

$$\mathrm{FIRST}(\alpha_i)\cap\mathrm{FIRST}(\alpha_j)=\phi$$

- 当要求A匹配输入串时，A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去执行任务。这个候选就是那个终结首符集含a的α。

$$A\rightarrow\alpha_1\,|\,\alpha_2\,|\,\ldots\,|\,\alpha_n$$





提取公共左因子

- 假定关于A的规则是

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

  (其中，每个γ不以δ开头)

  那么，可以把这些规则改写成

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

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

- 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

ε 候选

- 考虑文法G(E)：
  $E \rightarrow TE'$
  $E' \rightarrow +TE' \mid \varepsilon$
  $T \rightarrow FT'$
  $T' \rightarrow *FT' \mid \varepsilon$
  $F \rightarrow (E) \mid i$

- 分析：i + i

E

i + i #

IP

G(E):

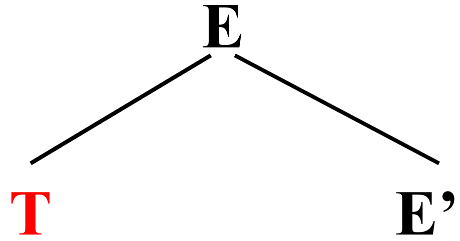
E→TE′

E′→+TE′ | ε

T→FT′

T′→*FT′ | ε

F→(E) | i



i + i #

$\uparrow$

IP

G(E):

E→TE′

E′→+TE′ | ε

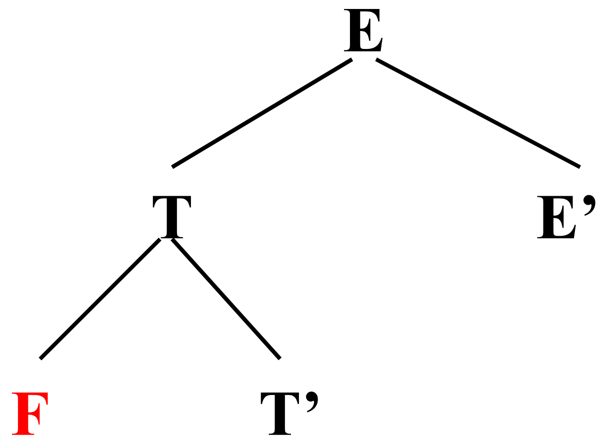
T→FT′

T′→*FT′ | ε

F→(E) | i

i + i #

IP



G(E):

E→TE′

E′→+TE′ | ε

T→FT′

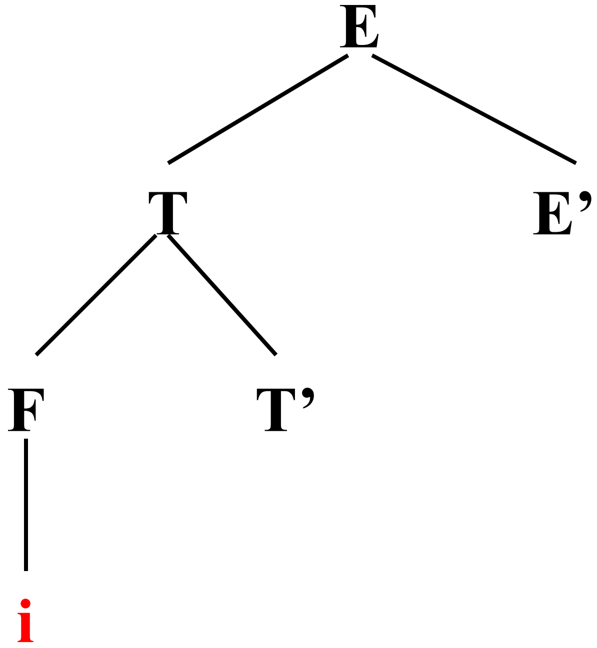
T′→*FT′ | ε

F→(E) | i

i + i #

↑

IP



G(E):

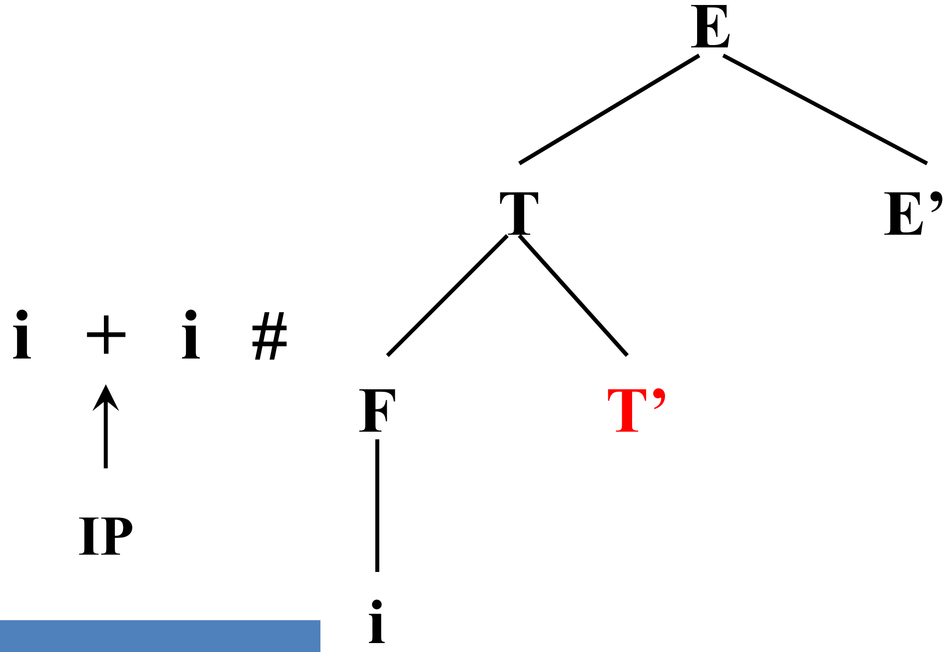
$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$



G(E):

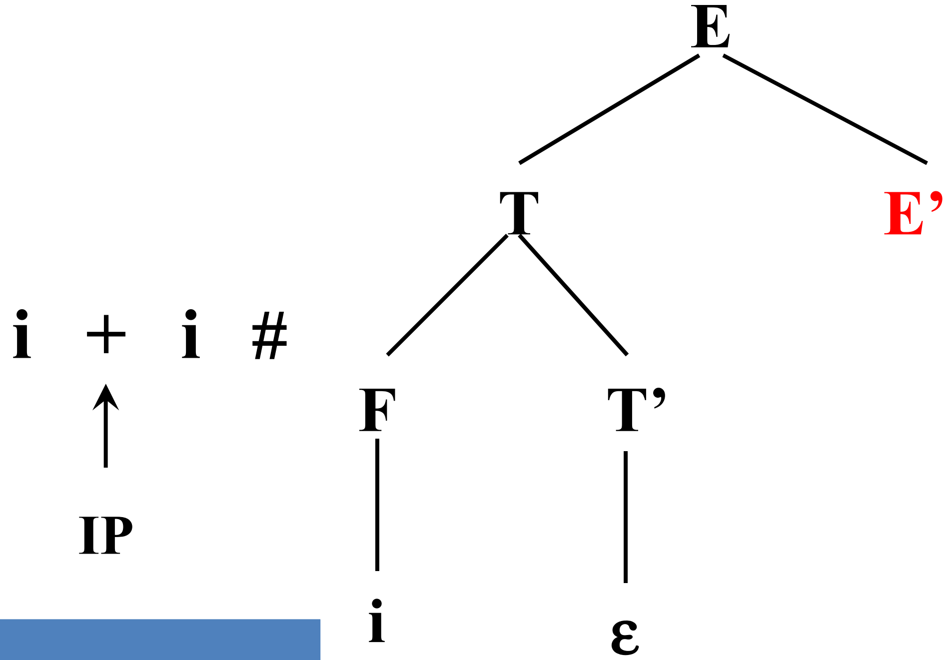
E→TE′

E′→+TE′ | ε

T→FT′

T′→*FT′ | ε

F→(E) | i



G(E):

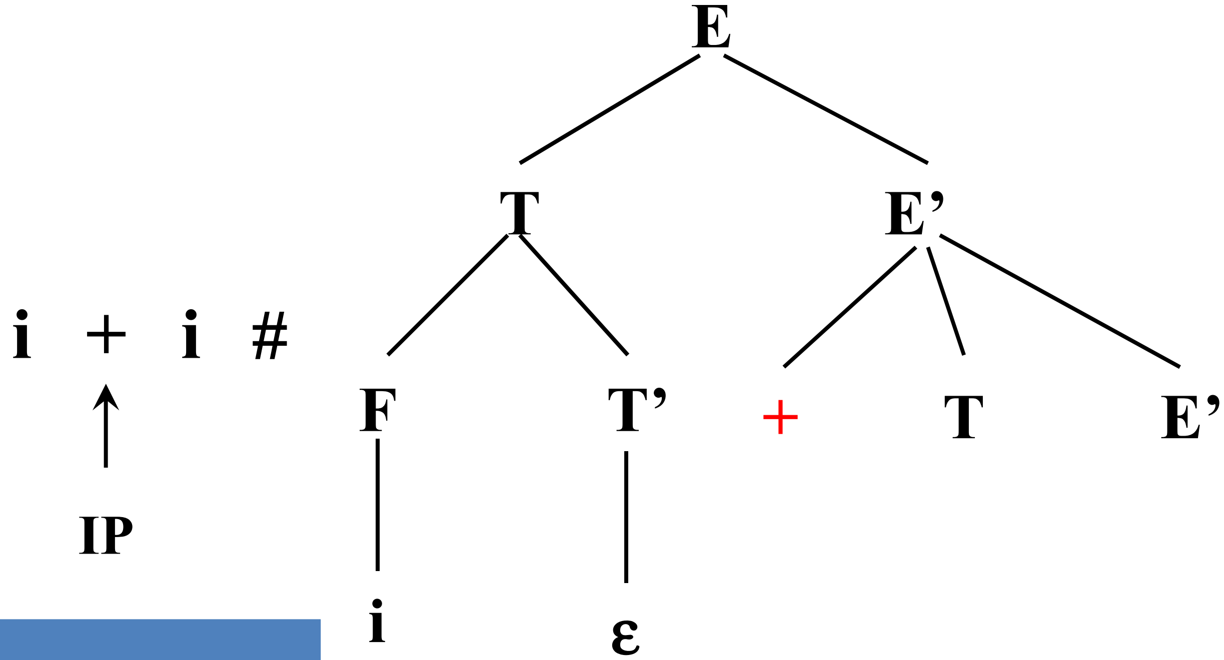
$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$



G(E):

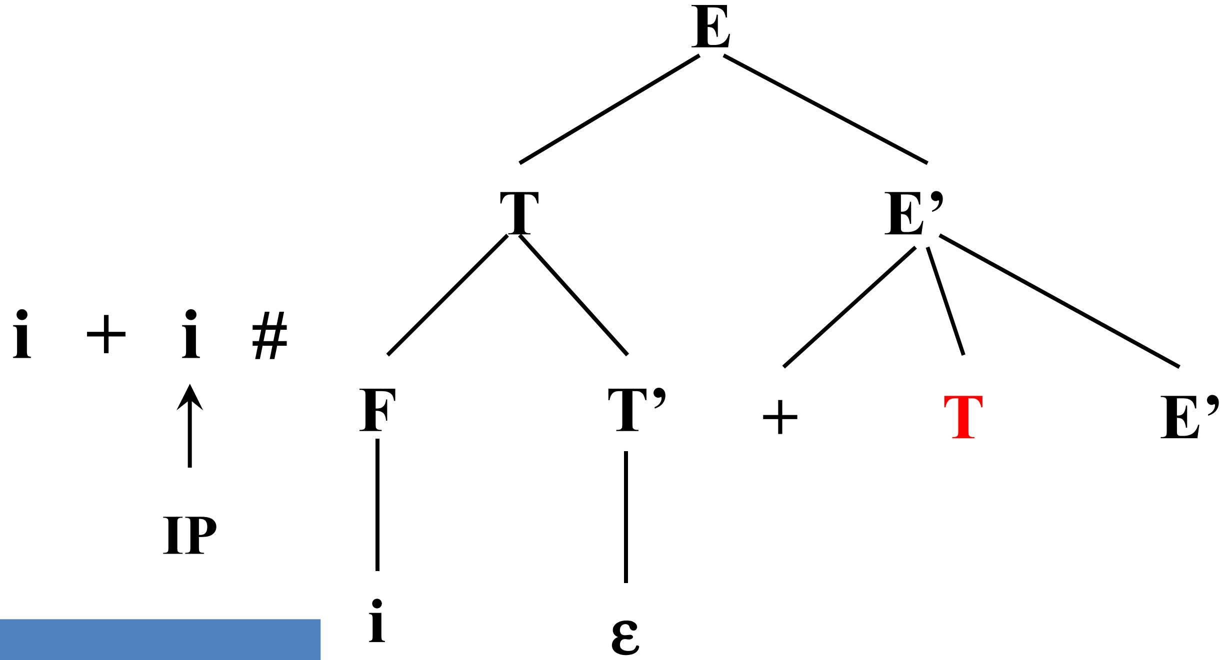
$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$



G(E):

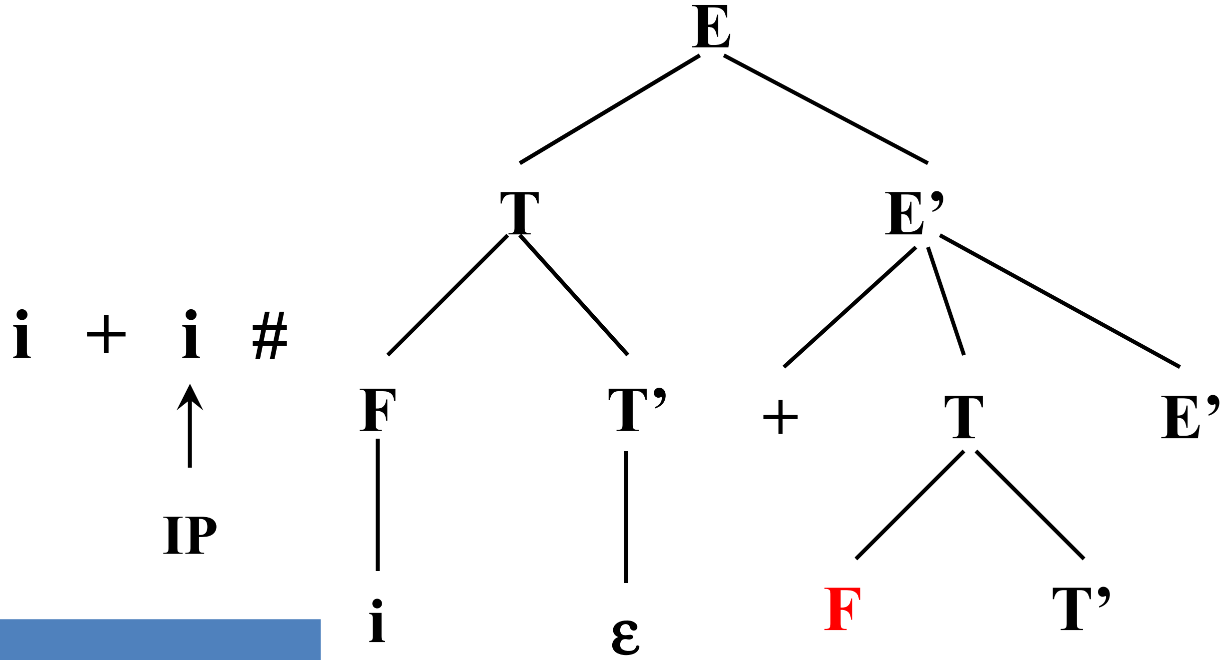
$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$



G(E):

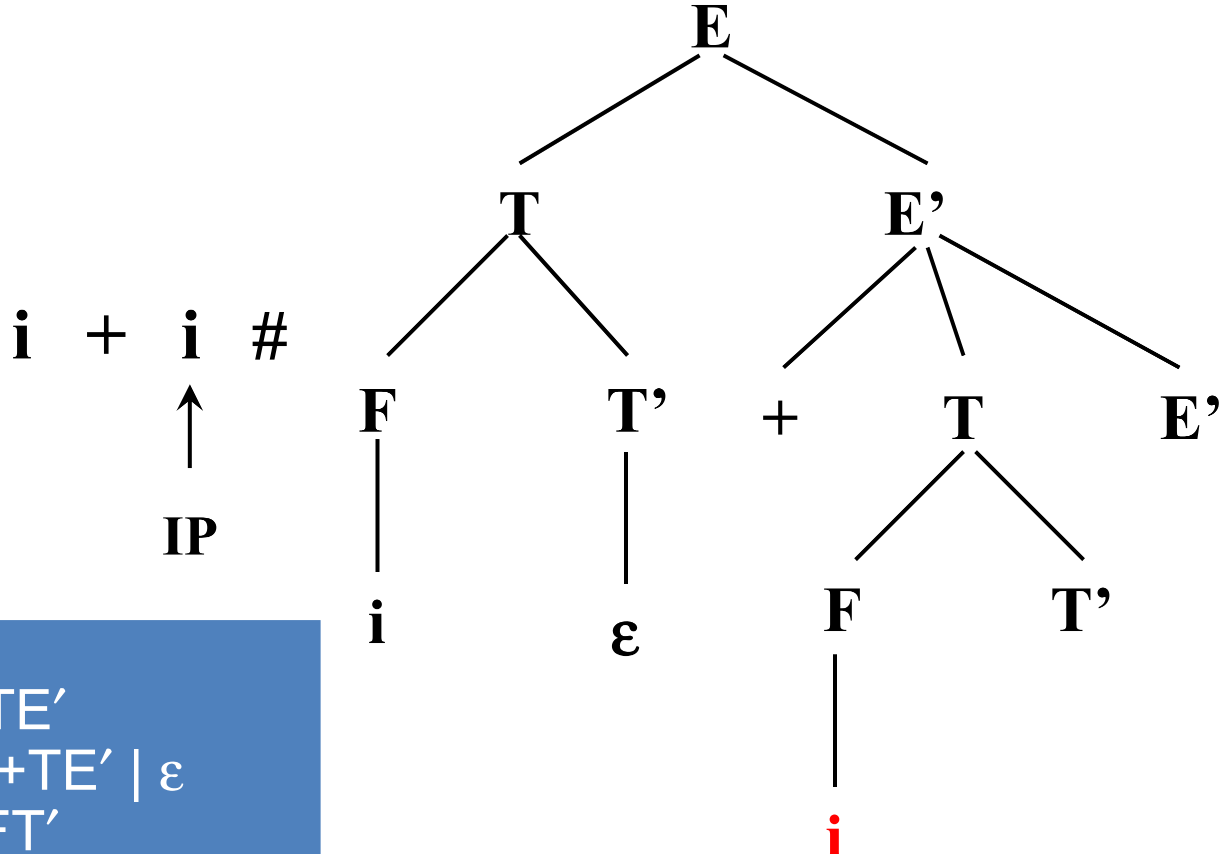
$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$



G(E):

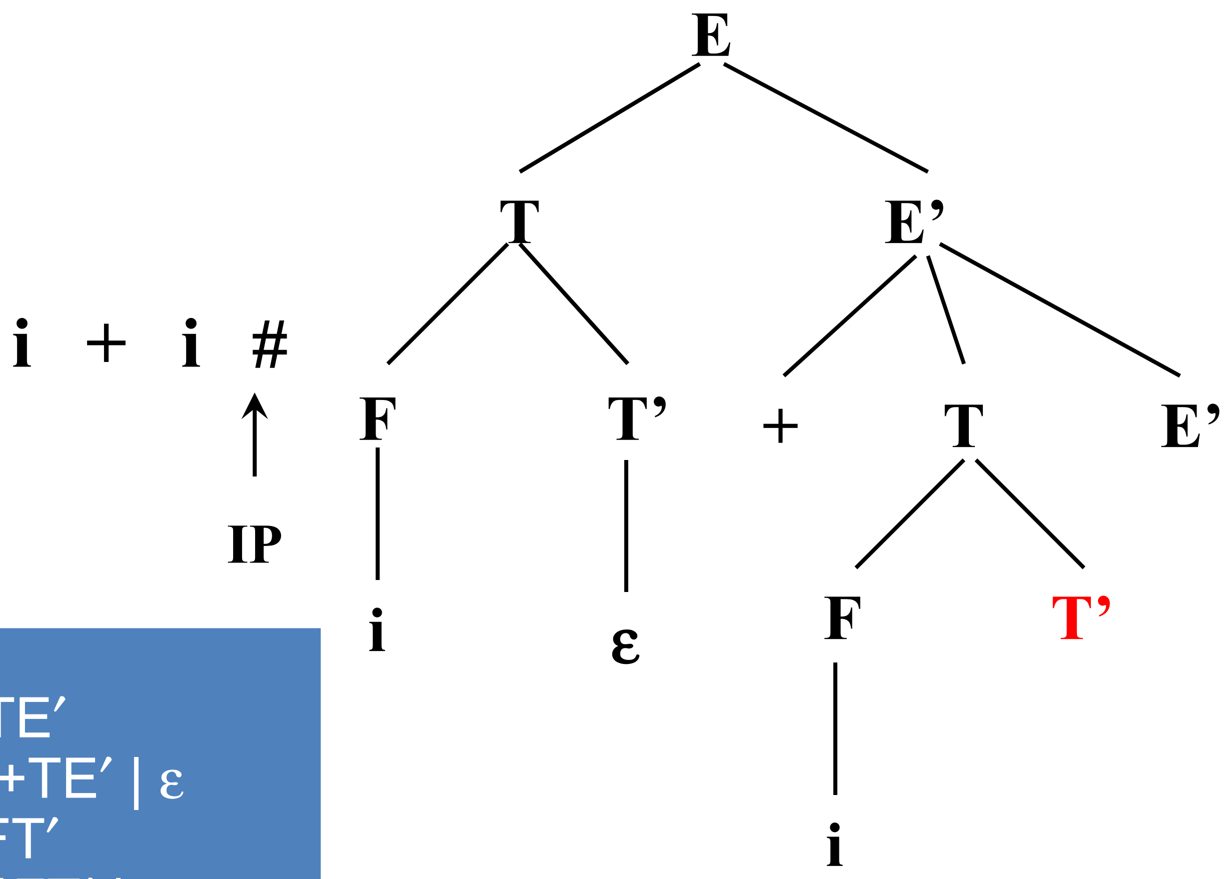
$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$



G(E):

$E \rightarrow TE'$

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

$T \rightarrow FT'$

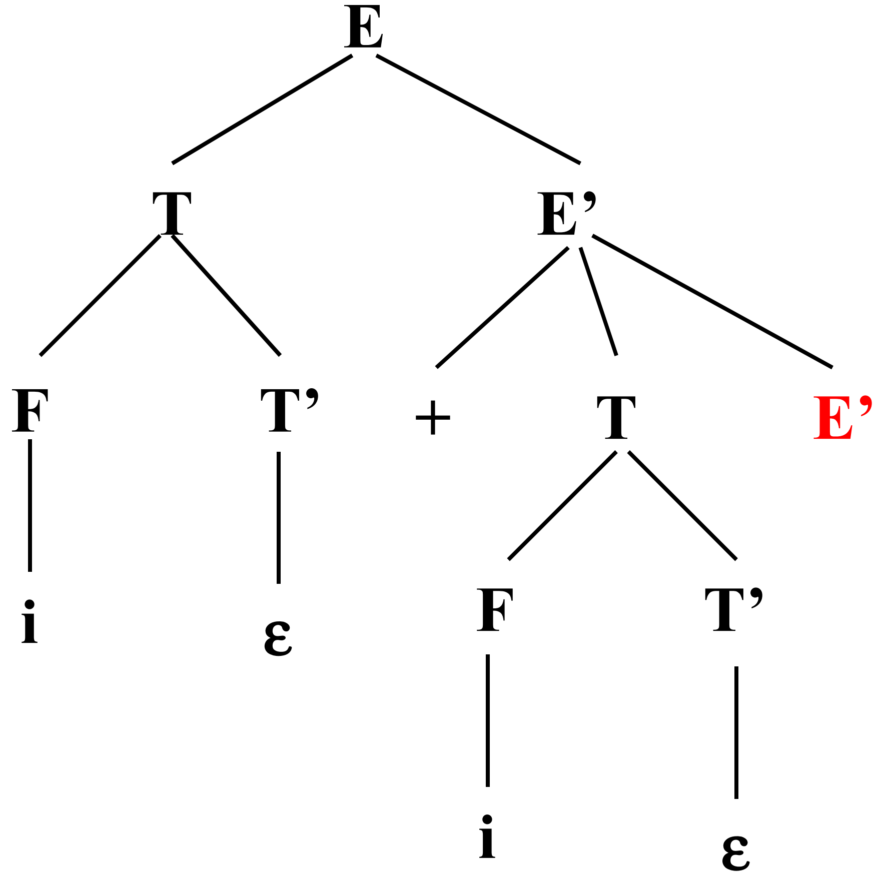
$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid i$

i + i #

↑

IP



G(E):

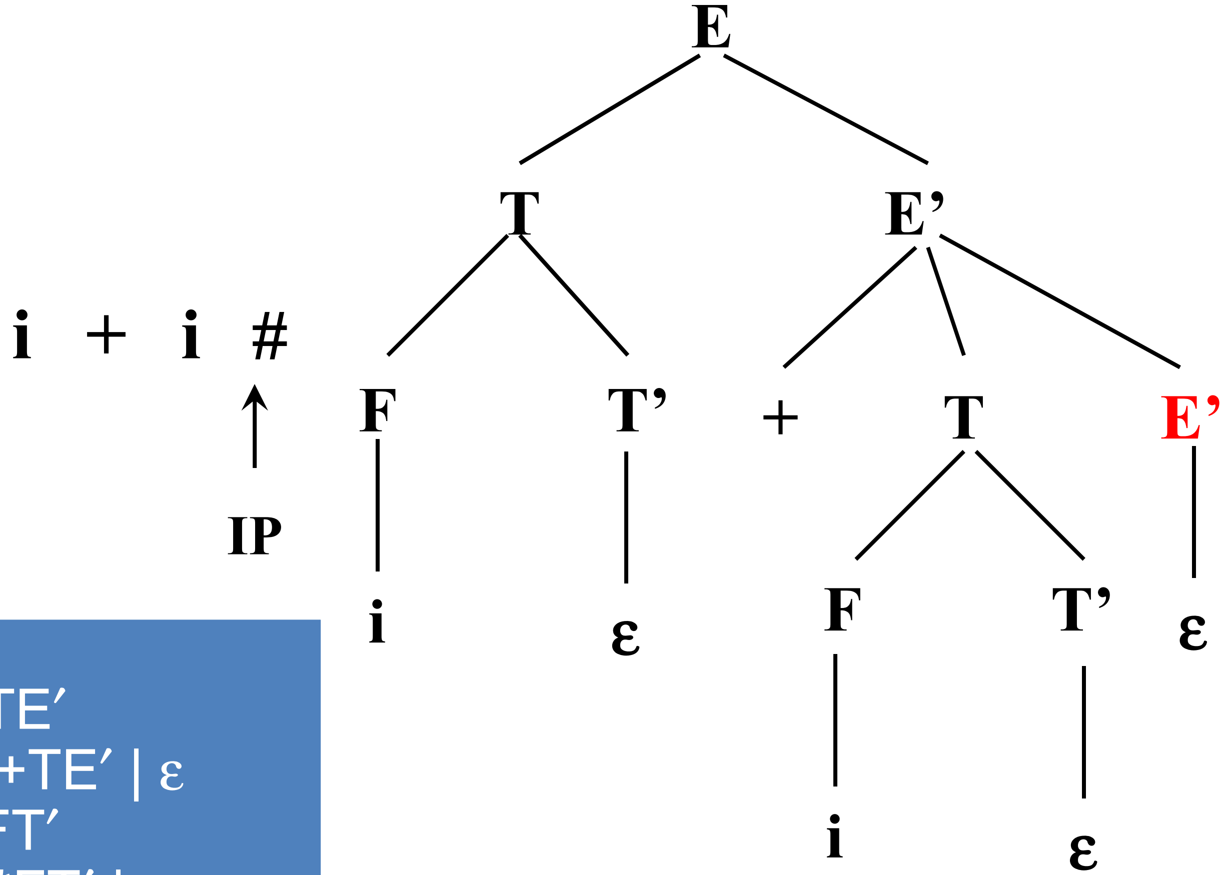
$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$



G(E):

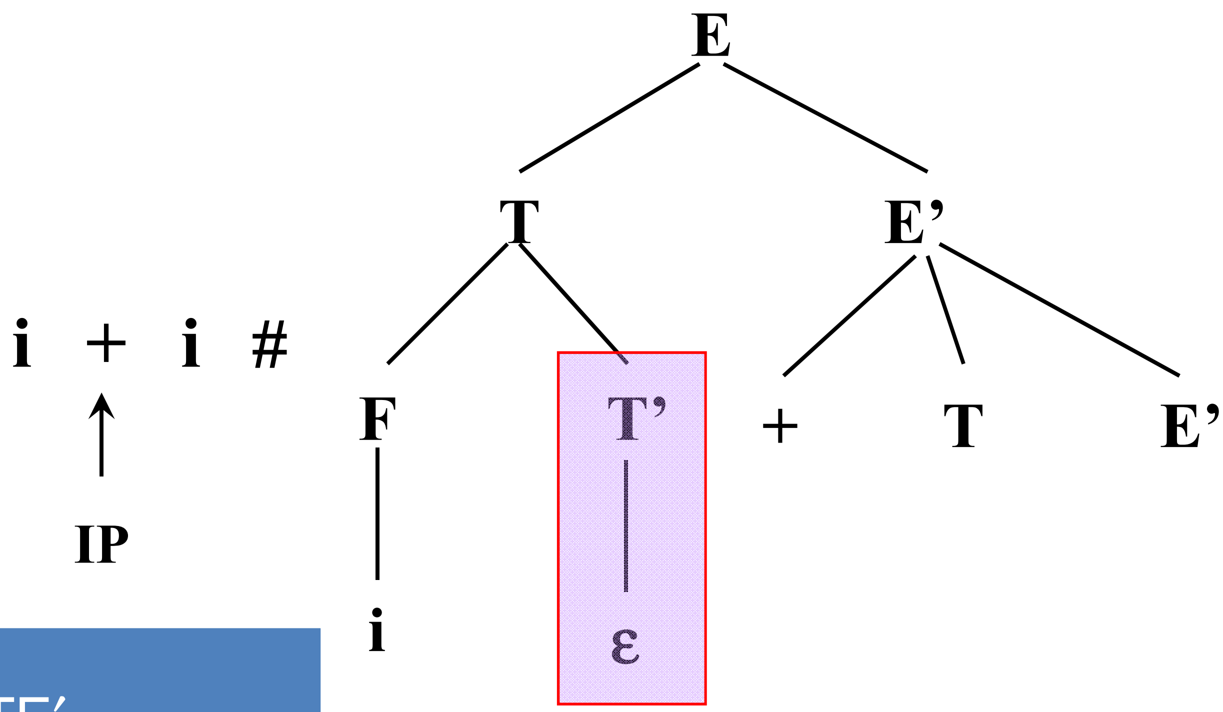
$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$



G(E):

$E \to TE'$

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

$T \to FT'$

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

$F \to (E) \mid i$

$$E \Rightarrow \ldots \Rightarrow i\,T'+\ldots$$

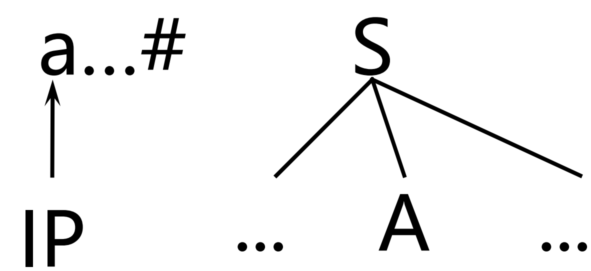
FOLLOW集合

- 假定S是文法G的开始符号，对于G的任何非终结符A，我们定义A的FOLLOW集合

$$FOLLOW(A) = \left\{ a \middle| S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V_T \right\}$$

特别是，若$S \overset{*}{\Rightarrow} \ldots A$，则规定$\# \in FOLLOW(A)$

$A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n | \varepsilon$



LL(1)文法

▶ 构造不带回溯的自上而下分析的文法条件

1. 文法不含左递归
2. 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。即，若

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

   则 $\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \phi \qquad (i \neq j)$

3. 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

$$\text{FIRST}(\alpha_i) \cap \text{FOLLOW}(A) = \phi, \ i = 1,2,\ldots,n$$

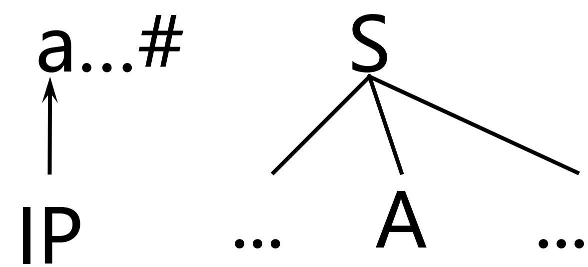
如果一个文法G满足以上条件，则称该文法G为LL(1)文法。

L: 从

L: 最

1:每一步只需向前查看一个符号

LL(1)分析法



- 对于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。
- 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

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

1. 若$a \in FIRST(\alpha_i)$，则指派α_i执行匹配任务；
2. 若a不属于任何一个候选首符集，则：
   (1) 若ε属于某个$FIRST(\alpha_i)$且$a \in FOLLOW(A)$，则让A与ε自动匹配。
   (2) 否则，a的出现是一种语法错误。

编译原理

FIRST和FOLLOW集合的构造

FIRST和FOLLOW集合的构造

$$FIRST(\alpha) = \left\{a \middle| \alpha \overset{*}{\Rightarrow} a \ldots, a \in V_T\right\}$$

$$FOLLOW(A) = \left\{a \middle| S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V_T\right\}$$

构造FIRST(α)

- $FIRST(\alpha) = \left\{ a \middle| \alpha \overset{*}{\Rightarrow} a \ldots, a \in V_T \right\}$
- $\alpha = X$， $X \in V_T \cup V_N$
- $\alpha = X_1 X_2 \ldots X_n$， $X_i \in V_T \cup V_N$

构造每个文法符号的FIRST集合

- 对每一$X \in V_T \cup V_N$，连续使用下面的规则，直至每个集合FIRST不再增大为止：

1. 若$X \in V_T$，则FIRST(X) = {X}。
2. 若$X \in V_N$，且有产生式$X \rightarrow a\ldots$，则把a加入到FIRST(X)中；若$X \rightarrow \varepsilon$也是一条产生式，则把ε也加到FIRST(X)中。
3.
   - 若$X \rightarrow Y\ldots$是一个产生式且$Y \in V_N$，则把FIRST(Y)中的所有非ε-元素都加到FIRST(X)中；
   - 若$X \rightarrow Y_1Y_2\ldots Y_{i-1}Y_i\ldots Y_k$是一个产生式，$Y_1$，…，$Y_{i-1}$都是非终结符，
     - 对于任何j，$1 \le j \le i-1$，FIRST(Y_j)都含有ε(即$Y_1\ldots Y_{i-1} \overset{*}{\Rightarrow} \varepsilon$)，则把FIRST($Y_i$)中的所有非ε-元素都加到FIRST(X)中
     - 若所有的FIRST(Y_j)均含有ε，j = 1，2，…，k，则把ε加到FIRST(X)中。

构造FIRST(α)

- $FIRST(\alpha) = \left\{ a \middle| \alpha \overset{*}{\Rightarrow} a \ldots, a \in V_T \right\}$
- α = X， $X \in V_T \cup V_N$
- α = $X_1X_2 \ldots X_n$， $X_i \in V_T \cup V_N$

构造任何符号串的FIRST集合

- 对文法G的任何符号串$\alpha=X_1X_2\ldots X_n$构造集合FIRST(α)
  1. 置FIRST(α) = FIRST(X_1)\{ε};
  2. 若对任何$1\le j\le i-1$，$\varepsilon\in$ FIRST(X_j)，则把FIRST(X_i)\{ε}加至FIRST(α)中；特别是，若所有的FIRST(X_j)均含有ε，$1\le j\le n$，则把ε也加至FIRST(α)中。显然，若$\alpha=\varepsilon$则FIRST(α) = {ε}。

构造FOLLOW(A)

- $FOLLOW(A) = \left\{ a \middle| S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V_T \right\}$

构造每个非终结符的FOLLOW集合

- 对于文法G的每个非终结符A构造FOLLOW(A)的办法是，连续使用下面的规则，直至每个FOLLOW不再增大为止：
  1. 对于文法的开始符号S，置＃于FOLLOW(S)中；
  2. 若$A \rightarrow \alpha B\beta$是一个产生式，则把$FIRST(\beta)\backslash\{\varepsilon\}$加至FOLLOW(B)中；
  3. 若$A \rightarrow \alpha B$是一个产生式，或$A \rightarrow \alpha B\beta$是一个产生式而$\beta \overset{*}{\Rightarrow} \varepsilon$ (即$\varepsilon \in FIRST(\beta)$)，则把FOLLOW(A)加至FOLLOW(B)中

构造每个非终结符的FOLLOW集合

- 对于文法G的每个非终结符A构造FOLLOW(A)的办法是，连续使用下面的规则，直至每个FOLLOW不再增大为止：
  1. 对于文法的开始符号S，置＃于FOLLOW(S)中；
  2. 若$A\rightarrow\alpha B\beta$是一个产生式，则把$FIRST(\beta)\backslash\{\varepsilon\}$加至FOLLOW(B)中；
  3. 若$A\rightarrow\alpha B$是一个产生式，或$A\rightarrow\alpha B\beta$是一个产生式而$\beta\overset{*}{\Rightarrow}\varepsilon$ (即$\varepsilon\in FIRST(\beta)$)，则把FOLLOW(A)加至FOLLOW(B)中

$A\rightarrow\alpha B\beta$是一个产生式,

$\because\ \forall\ a\in FIRST(\beta)\ )\backslash\{\varepsilon\}$，有$\beta\overset{*}{\Rightarrow} a\ldots$

则有 $S\overset{*}{\Rightarrow}\ \ldots A\ldots\Rightarrow\ldots\alpha B\beta\ldots\overset{*}{\Rightarrow}\ \ldots\alpha Ba\ldots$

$\therefore a\in FOLLOW(B)$

构造每个非终结符的FOLLOW集合

- 对于文法G的每个非终结符A构造FOLLOW(A)的办法是，连续使用下面的规则，直至每个FOLLOW不再增大为止：
  1. 对于文法的开始符号S，置#于FOLLOW(S)中；
  2. 若$A\rightarrow\alpha B\beta$是一个产生式，则把FIRST($\beta$)\{$\varepsilon$}加至FOLLOW(B)中；
  3. 若$A\rightarrow\alpha B$是一个产生式，或$A\rightarrow\alpha B\beta$是一个产生式而$\beta\overset{*}{\Rightarrow}\varepsilon$ (即$\varepsilon\in$ FIRST(β))，则把FOLLOW(A)加至FOLLOW(B)中

若$A\rightarrow\alpha B$是一个产生式

$\because \forall a \in$ FOLLOW(A), 有$S\overset{*}{\Rightarrow} \ldots Aa \ldots$

$\therefore S\overset{*}{\Rightarrow} \ldots Aa \ldots \Rightarrow \ldots \alpha Ba \ldots$

$\therefore a \in$ FOLLOW(B)

$A\rightarrow\alpha B\beta$是一个产生式而$\beta\overset{*}{\Rightarrow}\varepsilon$

$\because \forall a \in$ FOLLOW(A), 有$S\overset{*}{\Rightarrow} \ldots Aa \ldots$

$\therefore S\overset{*}{\Rightarrow} \ldots Aa \ldots \Rightarrow \ldots \alpha B\beta a \ldots \overset{*}{\Rightarrow} \ldots \alpha Ba \ldots$

$\therefore a \in$ FOLLOW(B)

测试

- 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集合

$FIRST(E) = \{ (, i \}$
$FIRST(E') = \{ +, \varepsilon \}$
$FIRST(T) = \{ (, i \}$
$FIRST(T') = \{ *, \varepsilon \}$
$FIRST(F) = \{ (, i \}$

$FOLLOW(E) = \{ \#,) \}$
$FOLLOW(E') = \{ \#,) \}$
$FOLLOW(T) = \{ +, \#,) \}$
$FOLLOW(T') = \{ +, \#,) \}$
$FOLLOW(F) = \{ *, +, \#,) \}$

小结

- 构造不带回溯的自上而下分析算法
  - 消除文法的左递归
  - 提取左公共因子，克服回溯
- LL(1)文法的条件
  - FIRST、FOLLOW集合