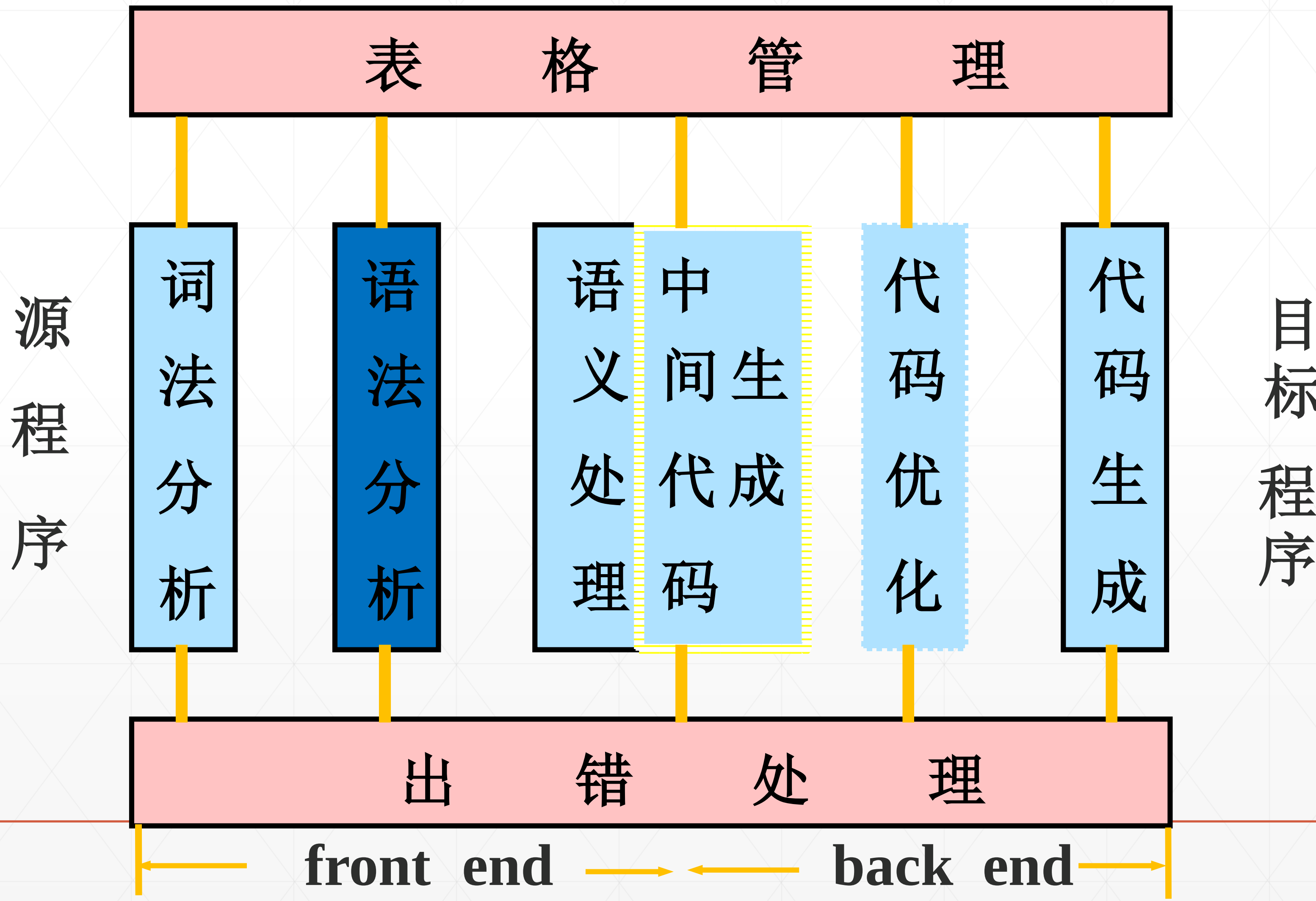
编译原理与设计

北京理工大学 计算机学院

语法分析：自上而下的分析方法



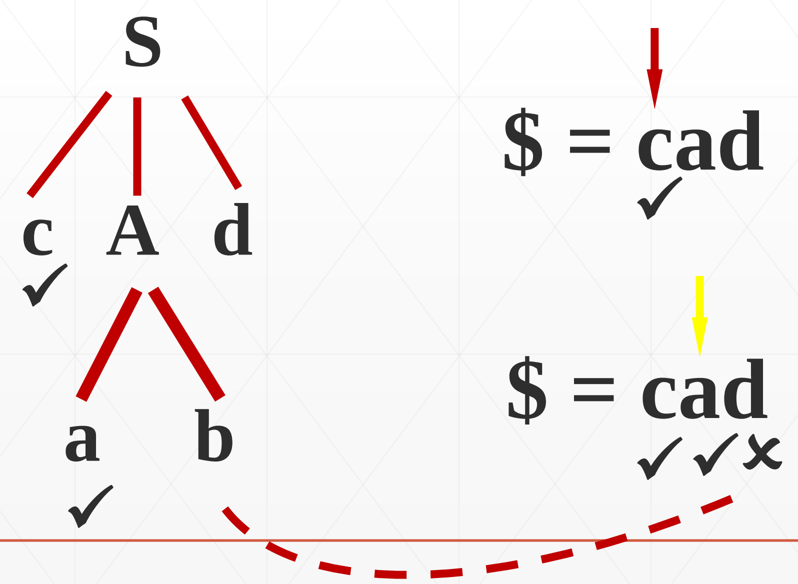
自顶向下分析方法

- 一般自顶向下方法

例：设有如下文法 G和字符串 \$ = cad

(1) $S \rightarrow cAd$

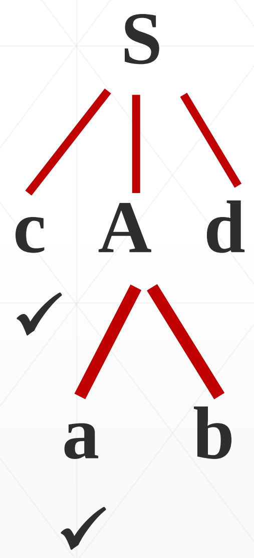
(2) $A \rightarrow ab \mid a$



自顶向下分析方法:一般自顶向下方法

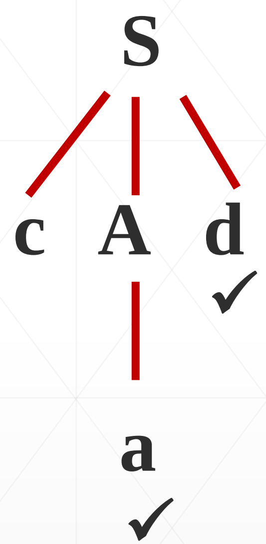
(1) $S \longrightarrow cAd$

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



\$ = cad

自顶向下分析方法:一般自顶向下方法



(1) $S \rightarrow cAd$

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

$\$ = cad$

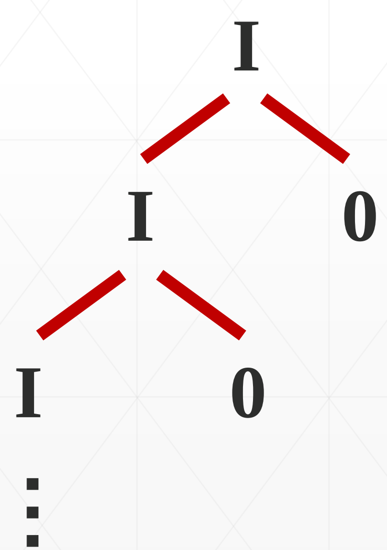
分析的本质是一种带回溯的自上而下分析，是一试探推导的过程，即反复使用不同的产生式去谋求匹配输入串的过程。算法效率低开销大。

自顶向下分析方法:一般自顶向下方法

例:设有文法 G和输入字符串 $

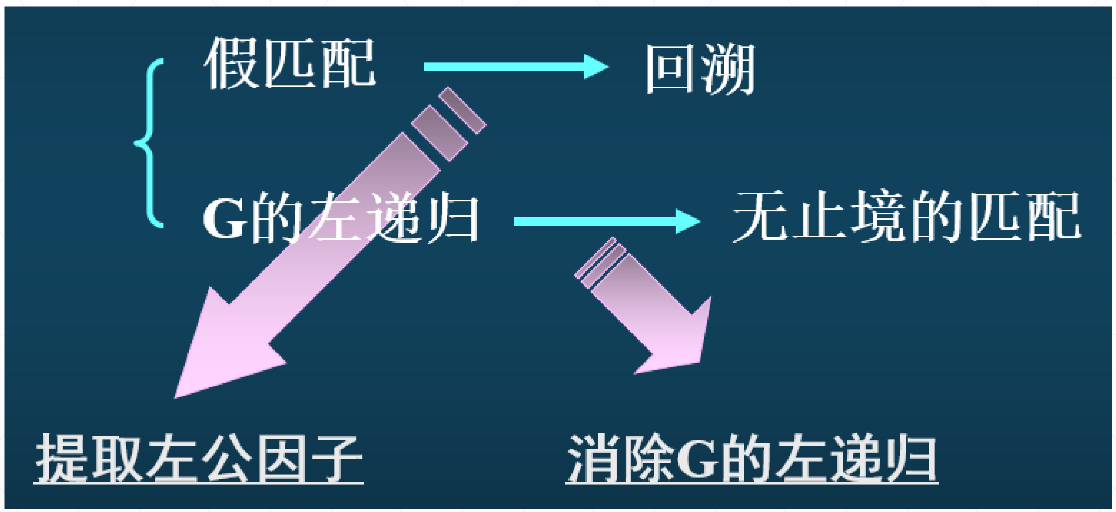
G:　I → I0 | Ia | a　　　$L(G)=a(a|0)^*$

$:　a00



按照自上而下分析法对输入串$产生分析树，则对非终结符的最左推导会使分析树无休止的延伸，使自上而下分析陷入死循环。

自顶向下分析方法:不确定性的原因



自顶向下分析方法:消除文法左递归

文法的直接左递归表现在其含有$A \to A\alpha$ ($\alpha \in (V_T \cup V_N)^*$)形式的产生式规则，则在语法分析的最左推导中会呈现 A=>A……的形式，间接左递归文法会呈现$A \stackrel{+}{\Rightarrow} A$…… 的形式。

自顶向下分析方法:直接左递归的消除

假定关于非终结符P的规则为

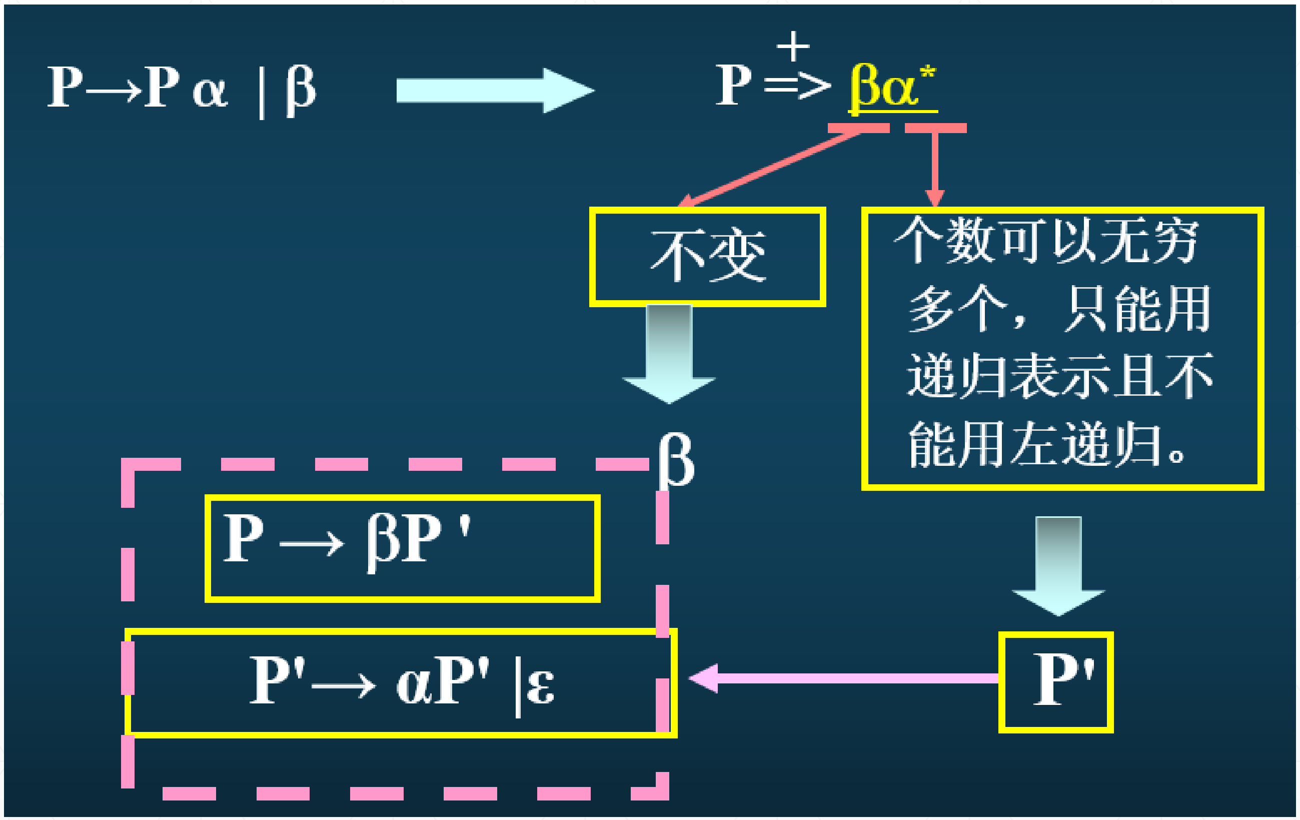
$$P \rightarrow P\alpha \mid \beta \quad \alpha, \beta \in (V_T \cup V_N)^*$$

其中，β 不以P开头。则可以把P的规则改写成如下等价的非直接左递归形式

$$P \rightarrow \beta P'$$

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

自顶向下分析方法：直接左递归消除



自顶向下分析方法：直接左递归的消除

例: 设有简单表达式文法G(E):

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

对G(E)消除二义性后，得到文法G(E)'：

$$\boxed{\begin{aligned} &E \rightarrow E+T \mid T \\ &T \rightarrow T*F \mid F \\ &F \rightarrow (E) \mid i \end{aligned}}$$

自顶向下分析方法:直接左递归的消除

对G(E)消除二义性后，得到文法G(E)'：

$$E \rightarrow E+T \mid T$$

$$T \rightarrow T*F \mid F$$

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

继续消除文法G(E)'的左递归，得到文法G(E)"

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

自顶向下分析方法：直接左递归的消除

假定关于非终结符P的规则为

$$P \rightarrow P\alpha_1 | P\alpha_2 | \ldots | P\alpha_n | \beta_1 | \beta_2 | \ldots | \beta_n$$

其中，每个α_i不等于ε，β_1，β_2，…，β_n 不以P开头。则可以把P的规则改写成如下等价的非直接左递归形式

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

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

自顶向下分析方法:直接左递归的消除

例：设有文法G:

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

对左递归文法G改写后的文法G'为

$$I \to aI' \mid bI'$$

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

自顶向下分析方法：间接左递归的消除

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

$$A \rightarrow Ba \mid a$$
$$B \rightarrow Cb \mid b$$
$$C \rightarrow Ac \mid c$$

A => Ba => Cba => Acba
B => Cb => Acb => Bacb
C => Ac => Bac => Cbac

自顶向下分析方法:间接左递归的消除

- 算法：消除文法左递归

① 对文法G的所有非终结符按任一种顺序排列，例如A_1，A_2，…，A_n。

② for (i=1; i=n; i++)

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

{

把形如$A_i \to A_j\gamma$ 的产生式改写成

$A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \ldots\ldots \mid \delta_k\gamma$

其中$A_j \to \delta_1 \mid \delta_2 \mid \ldots\ldots \mid \delta_k$ 是关于A_j的全部规则；

消除A_i规则中的直接左递归；

}

③ 简化由②所得的文法，即去掉多余的规则。

自顶向下分析方法:间接左递归的消除

- 令文法$G(A)$的非终结符排序为C，B，A。
- 对于C，不存在直接左递归，把C代入B，B的规则变为：$B \to Acb \mid cb \mid b$
- 代换后的B不含直接左递归，将其代入A，A的规则变为：$A \to Acba \mid cba \mid ba \mid a$
- A存在直接左递归，消除A的直接左递归有

$$A \to Ba \mid a$$
$$B \to Cb \mid b$$
$$C \to Ac \mid c$$

$$A \to cbaA' \mid baA' \mid aA'$$
$$A' \to cbaA' \mid \varepsilon$$

自顶向下分析方法：间接左递归的消除

$$A \rightarrow Ba \mid a$$
$$B \rightarrow Cb \mid b$$
$$C \rightarrow Ac \mid c$$

$$A \rightarrow cbaA' \mid baA' \mid aA'$$
$$A' \rightarrow cbaA' \mid \varepsilon$$
$$\boxed{\begin{array}{l} B \rightarrow Acb \mid cb \mid b \\ C \rightarrow Ac \mid c \end{array}}$$

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

自顶向下分析方法：消除回溯

$P \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$　　当前\$：$\ldots a_i \ldots$

在一般自上而下分析中，对于一个V_N进行推导并试图去匹配句子剩余符号时，若V_N含有两个或两个以上的候选式，是依次一个一个地去试探，试图找出一个合乎要求的α_i。先选α_1，与当前输入a_i匹配成功则替换，否则选α_2，依此类推。

自顶向下分析方法:消除回溯

设G是二型文法且文法G不含左递归，则G中的非终结符的每个候选式α的终结首符集FIRST(α)为

$$FIRST(\alpha) = \{ a \mid \alpha \Rightarrow a \ldots\ldots, \; a \in V_T \}$$

若 α =>ε, 则ε∈FIRST(α) 。

<u>不带回溯的条件：</u>

如果对文法G的一个产生式A，设A为

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

如果它的每个候选式α_i均不存在$\alpha_i \overset{*}{\Rightarrow} \varepsilon$的情况，

而且FIRST(α_i)两两彼此互不相交。

自顶向下分析方法:消除回溯

如果对文法G的一个产生式A，设A $(A\in V_N)$为

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

满足不带回溯条件：

据当前输入字符a，若 $a\in FIRST(\alpha_i)$ ，其中 α_i 是 $\alpha_1\ldots\alpha_n$ 中之一，选取 $A\rightarrow\alpha_i$ 进行推导。

自顶向下分析方法:消除回溯

例： 设有文法G:

$$S \rightarrow Ap \mid Bq$$
$$A \rightarrow a \mid cA$$
$$B \rightarrow b \mid dB$$

对S: FIRST(Ap)={a, c}　FIRST(Bq)={b, d}

且 FIRST(Ap) ∩ FIRST(Bq)=φ

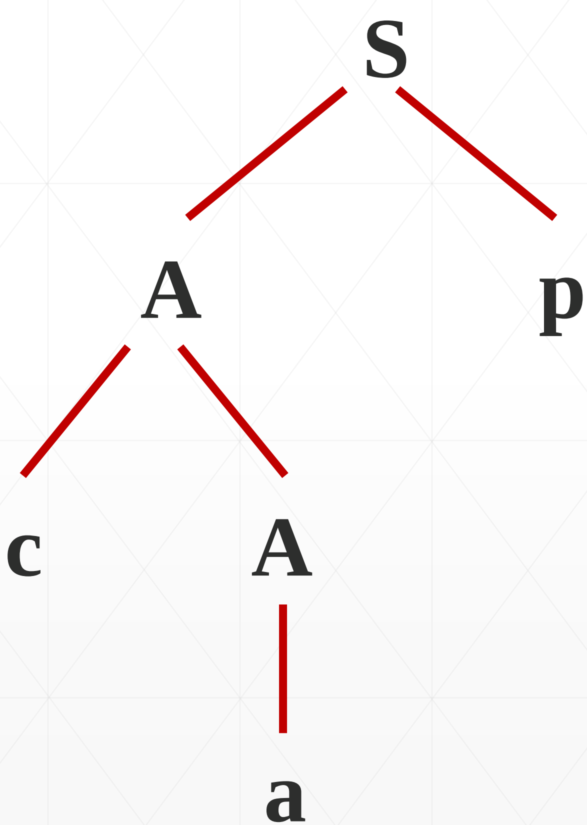
对A: FIRST(a)={a}　FIRST(cA)={c}

且 FIRST(a) ∩ FIRST(cA)=φ

对B: FIRST(b)={b}　FIRST(dB)={d}

且 FIRST(b) ∩ FIRST(dB)= φ

自顶向下分析方法:消除回溯



$S \rightarrow Ap \mid Bq$

$A \rightarrow a \mid cA$

$B \rightarrow b \mid dB$

FIRST(Ap)={a, c}　FIRST(Bq)={b, d}

且 FIRST(Ap) ∩ FIRST(Bq)=φ

FIRST(a)={a}　FIRST(cA)={c}

且 FIRST(a) ∩ FIRST(cA)=φ

FIRST(b)={b}　FIRST(dB)={d}

且 FIRST(b) ∩ FIRST(dB)= φ

$: cap

$: cap

$: cap

自顶向下分析方法：消除回溯

对产生式A的多个候选式的FIRST(α_i)的相互两个彼此交集≠Φ，是因为α_i中有公共左因子，可以通过提取左公因子来改造文法。

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

$$\delta(\beta_1 \mid \beta_2 \mid \ldots \mid \beta_n)$$

可以使用下面规则改写文法G为G':

$$A \rightarrow \delta A'$$

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

自顶向下分析方法:消除回溯

- 若文法G为

$$A \rightarrow \underbrace{\delta_1\beta_1 | \delta_1\beta_2 | \ldots | \delta_1\beta_n}_{\delta_1(\underbrace{\beta_1 | \beta_2 | \ldots | \beta_n}_{A'})} | \underbrace{\delta_2\alpha_1 | \delta_2\alpha_2 | \ldots | \delta_2\alpha_m}_{\delta_2(\underbrace{\alpha_1 | \alpha_2 | \ldots | \alpha_m}_{A''})}$$

- 使用下面规则改写文法G为G′

$$A \rightarrow \delta_1 A' | \delta_2 A''$$

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

$$A'' \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_m$$

自顶向下分析方法

- 递归下降分析器(Recursive-Descent Parser)

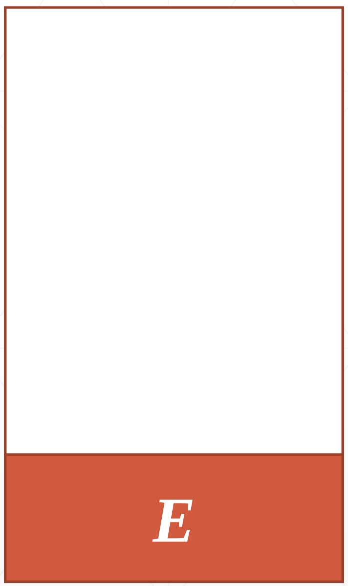
对文法的每个非终结符号，都根据其产生式的各个候选式的结构，为其编写一个对应的子程序（或函数），该子程序完成相应的非终结符对应的语法成份的识别和分析任务。

自顶向下分析方法:递归下降分析器

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

```
E ( )
  { T; E′}

E′( )
{
  if (c=‘+’)  {n++; T; E′}
 }

T ( )
  { F; T′}

F ( )
 {  if ( c=‘i’ )  n++;
      else { if ( c=‘(’ )
             { n++; E;
               if (c=‘)’ )  n++;
              else error }
      else error1 }}

T′ ( )
{  if (c=‘*’)  {n++; F; T′} }
```

自顶向下分析方法：递归下降分析器



$E \rightarrow TE'$

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

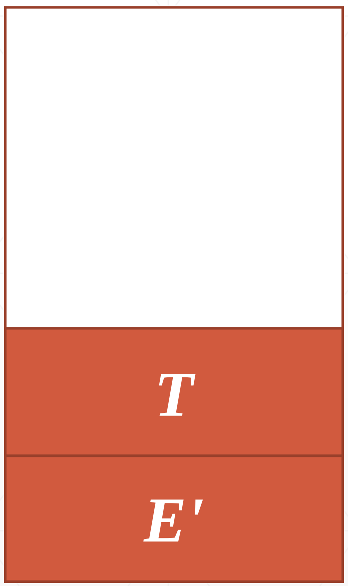
$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

$i + i * i \ \#$

自顶向下分析方法：递归下降分析器



$E \rightarrow TE'$

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

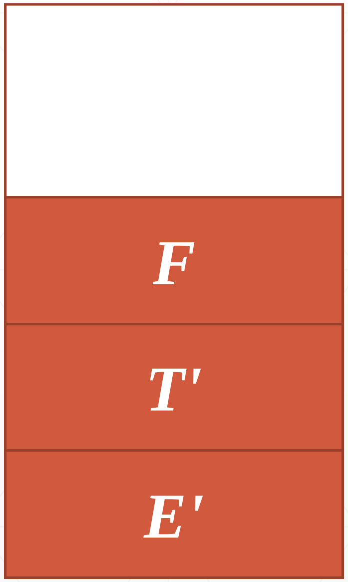
$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

$i + i * i \ \#$

自顶向下分析方法：递归下降分析器



$E \rightarrow TE'$

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

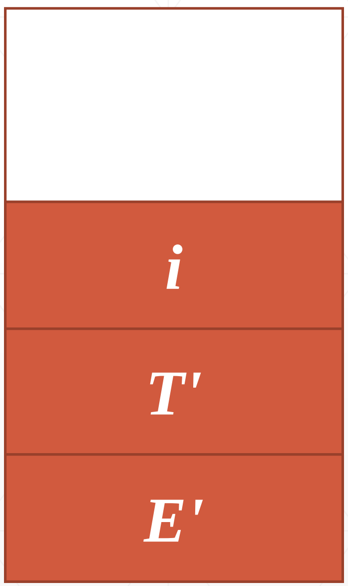
$T \rightarrow FT'$

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

$F \rightarrow (E) | i$

$i + i * i$ #

自顶向下分析方法：递归下降分析器



$E \rightarrow TE'$

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

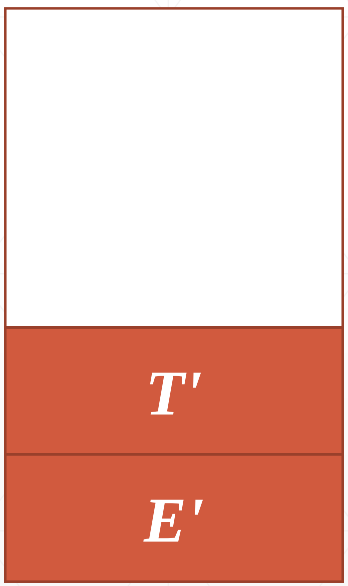
$T \rightarrow FT'$

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

$F \rightarrow (E) | i$

$i + i * i \ \#$

自顶向下分析方法：递归下降分析器



$E \rightarrow TE'$

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

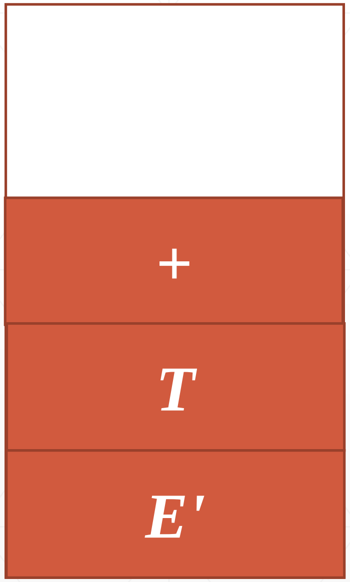
$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

$i + i * i \ \#$

自顶向下分析方法：递归下降分析器



$E \rightarrow TE'$

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

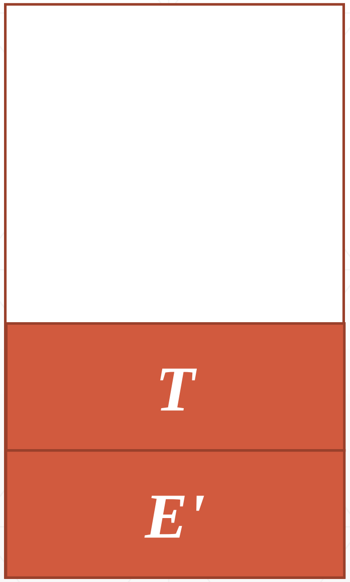
$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

$i + i * i \ \#$

自顶向下分析方法：递归下降分析器



$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

$i + i * i \ \#$

自顶向下分析方法：递归下降分析器

F
T'
E'

$E \rightarrow TE'$

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

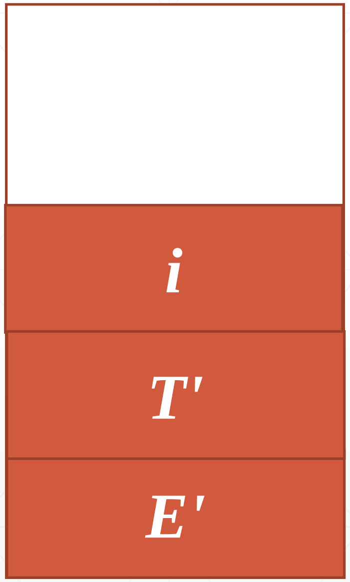
$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

$i + i * i \ \#$

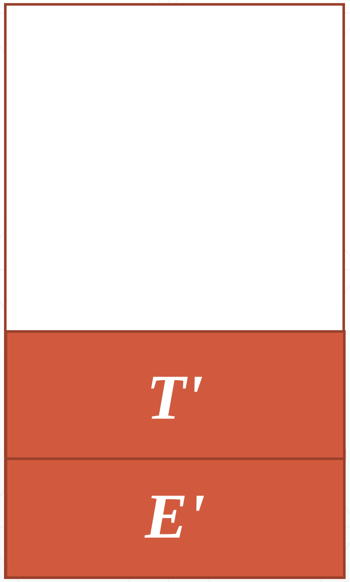
自顶向下分析方法：递归下降分析器



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

$i + i * i$ #

自顶向下分析方法：递归下降分析器



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

$i + i * i \ \#$

自顶向下分析方法：递归下降分析器

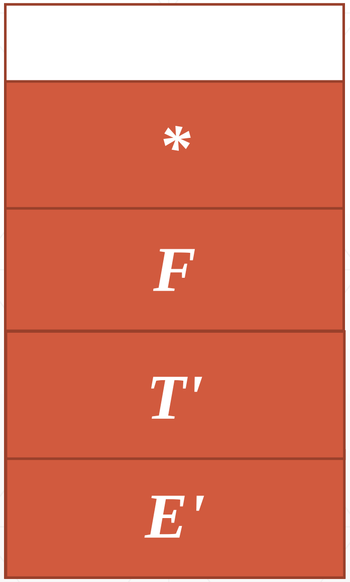
$E \rightarrow TE'$

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

$T \rightarrow FT'$

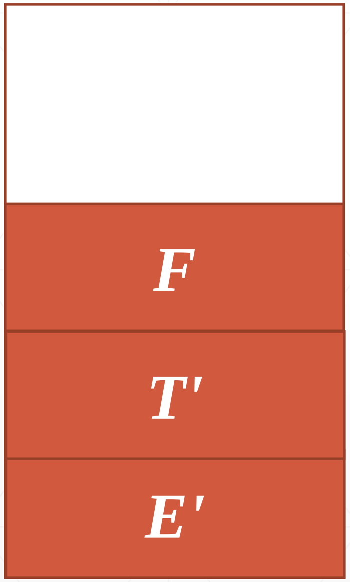
$T' \rightarrow *FT' | \varepsilon$

$F \rightarrow (E) \mid i$



$i + i * i \ \#$

自顶向下分析方法：递归下降分析器



$E \rightarrow TE'$

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

$T \rightarrow FT'$

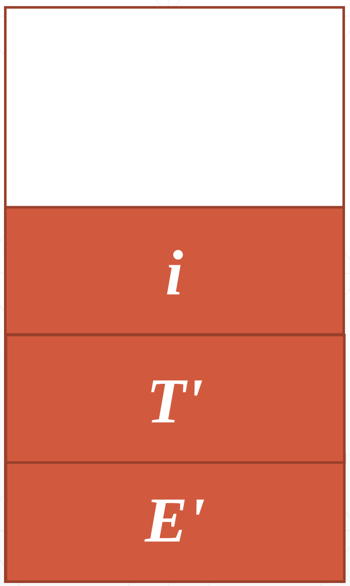
$T' \rightarrow *FT' | \varepsilon$

$F \rightarrow (E) \mid i$

$i + i * i \ \#$

自顶向下分析方法：递归下降分析器

- 递归下降分析器(Recursive-Descent Parser)



$E \rightarrow TE'$

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

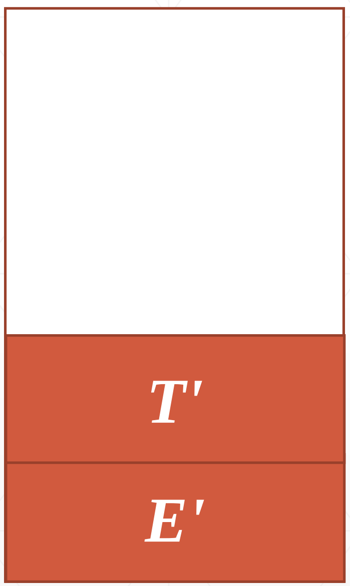
$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

$i + i * i \ \#$

自顶向下分析方法：递归下降分析器



$E \rightarrow TE'$

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

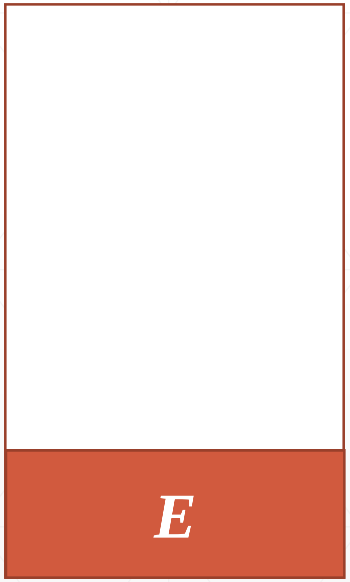
$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

$i + i * i \ \#$

自顶向下分析方法：递归下降分析器



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

(+ *i* #

自顶向下分析方法：递归下降分析器

$E \rightarrow TE'$

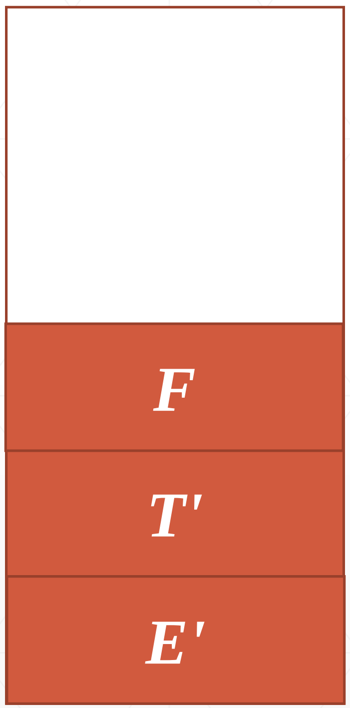
$E' \rightarrow +TE' | \varepsilon$

$T \rightarrow FT'$

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

$F \rightarrow (E) | i$

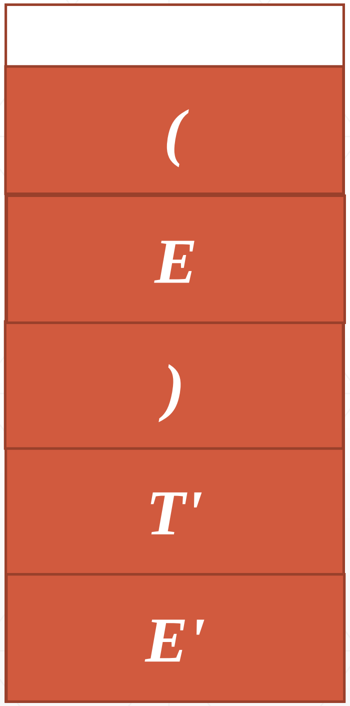
自顶向下分析方法：递归下降分析器



(+ i #

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

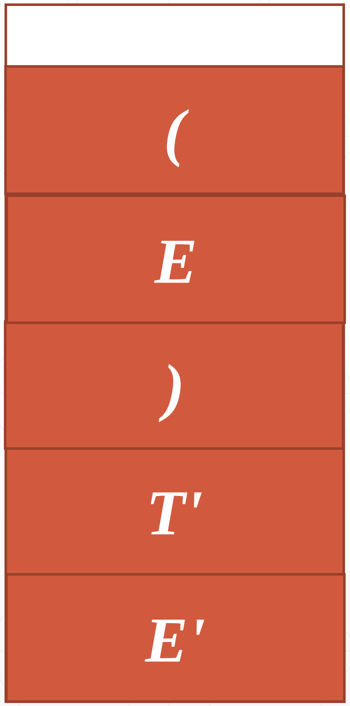
自顶向下分析方法：递归下降分析器



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

(+ *i* #

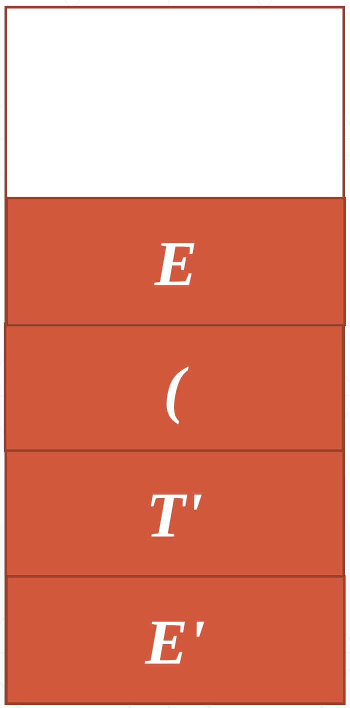
自顶向下分析方法：递归下降分析器



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

(+ *i* #

自顶向下分析方法：递归下降分析器



$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

$(\quad + \quad i \quad \#$

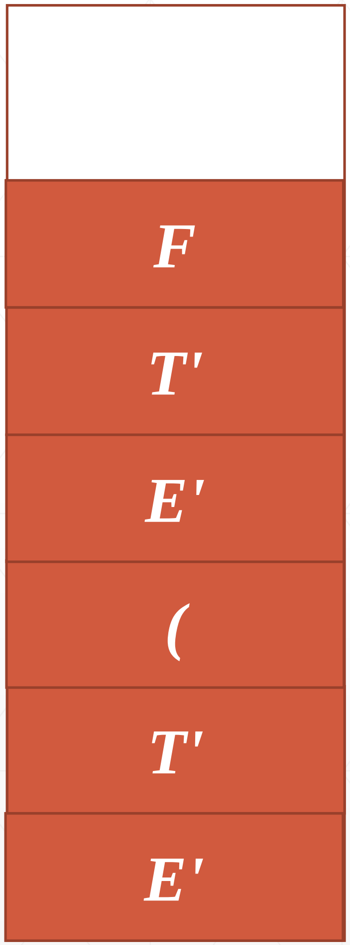
自顶向下分析方法：递归下降分析器

T
E'
(
T'
E'

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

(+ i #

自顶向下分析方法：递归下降分析器



$(\ + \ i \ \#$

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid i$

自顶向下分析方法：递归下降分析器

F
T'
E'
(
T'
E'

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

(+ i #

自顶向下分析方法: LL(1)分析器构造

预测分析器

L L (1)

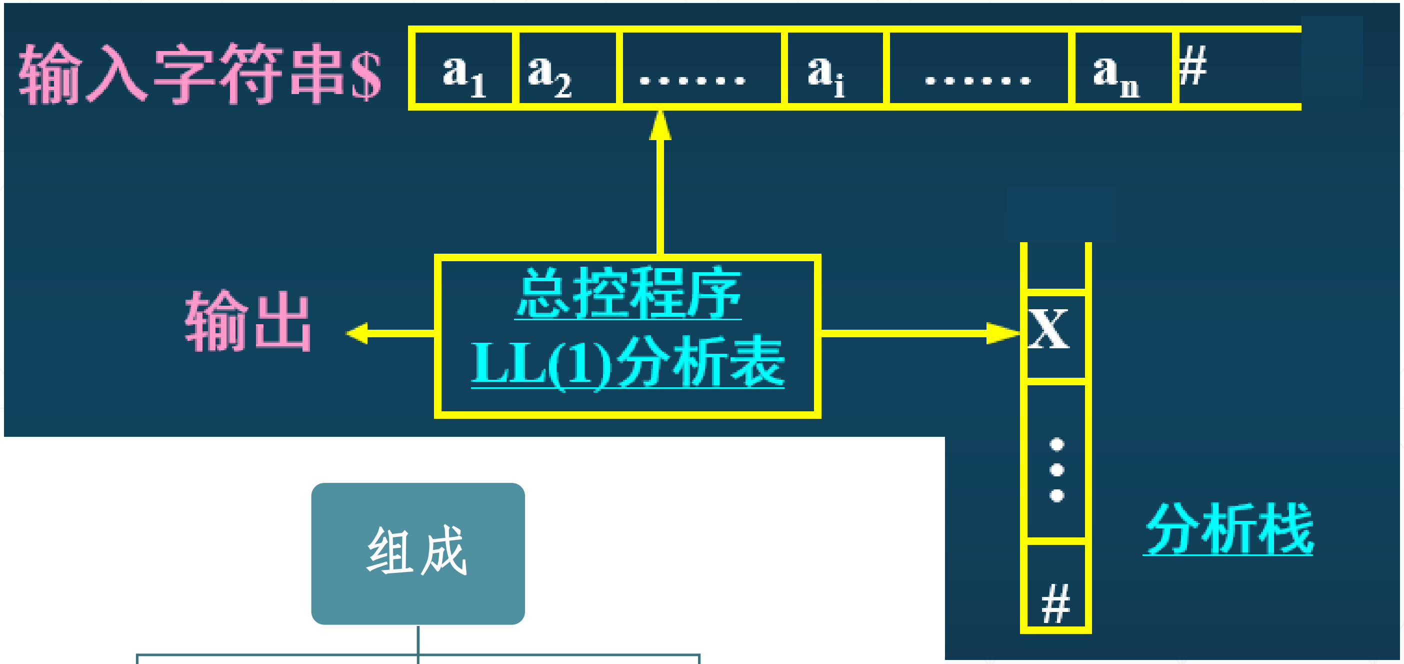
在分析中最多向前看$的1个输入字符

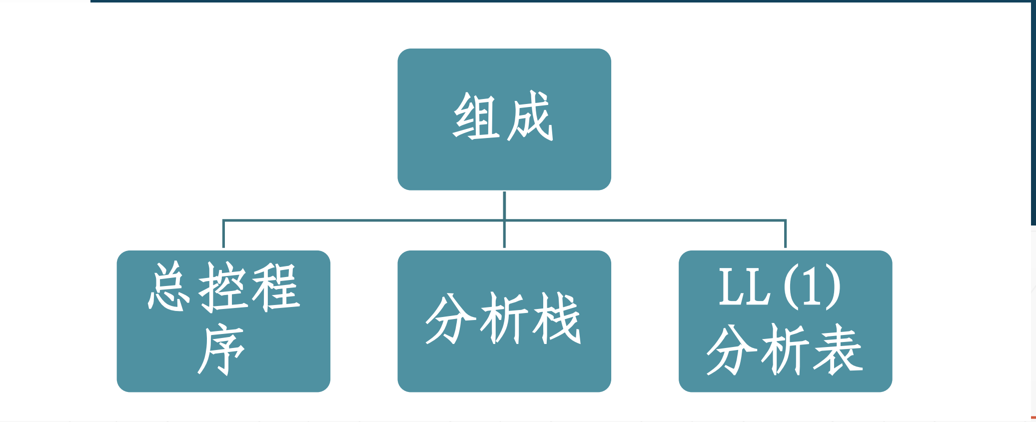
分析模式：最左推导

扫描模式：自左向右

自顶向下分析方法: LL(1)分析器构造

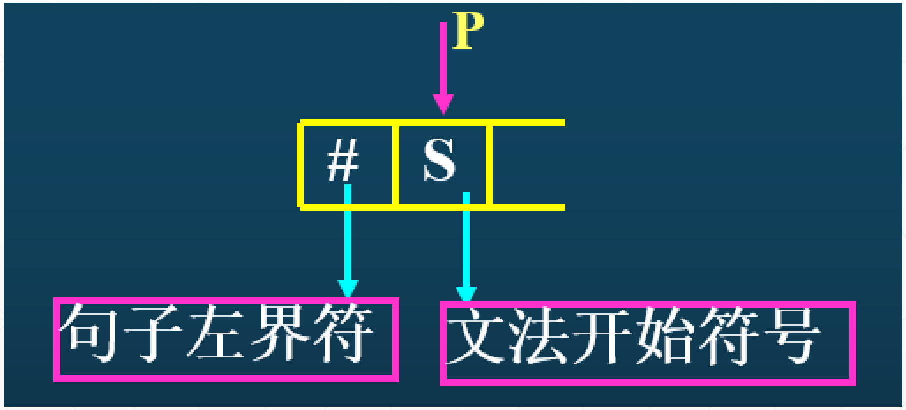
- 逻辑结构





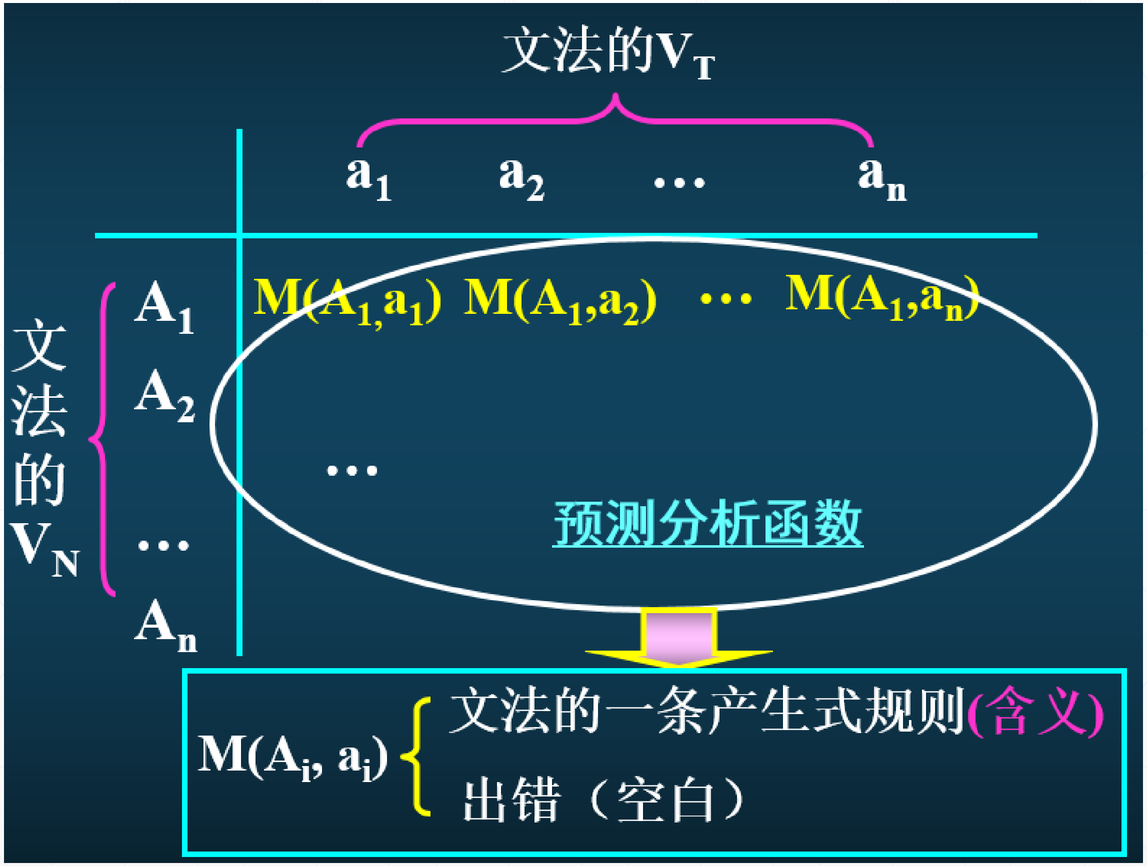
自顶向下分析方法: LL(1)分析器构造

- 分析栈
  - 存放分析过程中的文法符号（待匹配和已经推导的串）。
  - 初始状态为:



自顶向下分析方法: LL(1)分析器构造

- 分析表



自顶向下分析方法: LL(1)分析器构造

- 总控程序
  - 初始化

  - 若当前分析栈顶 X和a_i都是终结符号，则对于
    - $X = a_i = \#$，表示分析成功，停止分析过程；
    - $X = a_i \neq \#$，将 X从分析栈退掉，p指向下一个输入字符；
    - $X \neq a_i$，表示不匹配的出错情况。
  - 若$X \in V_N$，则查分析表。此时对$M(X , a_i)$
    - 若$M(X,a_i)$中为一个产生式规则，则将X从 栈中弹出并将此规则右部的符号序列按倒序推进栈。
    - 若$M(X,a_i)$中为空白，表示出错，调用语法出错处理子程序。

自顶向下分析方法: LL(1)分析器构造

E→TE′

E′→+ TE′ | ε

T→FT′

T′→*FT′ | ε

F→(E) | i

V_N	终 结 符 V_T					
	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)		

V_N	终结符 V_T					
	i	+	*	(	)	#
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	$F\to i$			$F\to(E)$		

步骤	分析栈	余留输入串	所用产生式
1	#E	i+i*i#	E→TE′
2	#E′T	i+i*i#	T→FT′
3	#E′T′F	i+i*i#	F→i
4	#E′T′i	i+i*i#	p++
5	#E′T′	+i*i#	T′→ε
6	#E′	+i*i#	E′→+TE′
7	#E′T+	+i*i#	p++
8	#E′T	i*i#	T→FT′
9	#E′T′F	i*i#	F→i
10	#E′T′i	i*i#	p++
11	#E′T′	*i#	T′→*FT′

自顶向下分析方法: LL(1)分析器构造

V_N	终 结 符 V_T					
	i	$+$	$*$	(	)	#
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	$F \to i$			$F \to (E)$		

步 骤	分析栈	余留输入串	所用产生式
12	#E′T′F*	*i#	
13	#E′T′F	i#	$F \to i$
14	#E′T′i	i#	
15	#E′T′	#	$T' \to \varepsilon$
16	#E′	#	$E' \to \varepsilon$
17	#	#	分析成功

自顶向下分析方法: LL(1)分析器构造

- 整个分析过程是分析栈和$构成的二元式不断变化的过程。
- 分析器结构、总控程序不变，不同的源语言仅是LL(1)分析表不同。

自顶向下分析方法: LL(1)分析器构造

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

据FIRST集合的定义：对文法G，若G中产生式形如$A\rightarrow\alpha$ 且没有 $\alpha \Rightarrow \varepsilon$ 的情况，则对文法规则 $A\rightarrow\alpha$

当 $a\in FIRST(\alpha)$

则 $M(A, a) = \{A\rightarrow\alpha\}$

自顶向下分析方法: LL(1)分析器构造

例　设有文法G:

$S \rightarrow Ap \mid Bq \qquad A \rightarrow a \mid cA \qquad B \rightarrow b \mid dB$

对S:

FIRST(Ap)={a, c}　　FIRST(Bq)={b, d}

且 FIRST(Ap) ∩ FIRST(Bq)=φ

对A:

FIRST(a)={a}　　FIRST(cA)={c}

且 FIRST(a) ∩ FIRST(cA)=φ

对B :

FIRST(b)={b}　　FIRST(dB)={d}

且 FIRST(b) ∩ FIRST(dB)= φ

自顶向下分析方法: LL(1)分析器构造

例　设有文法G:

$S \rightarrow Ap \mid Bq \qquad A \rightarrow a \mid cA \qquad B \rightarrow b \mid dB$

对S: FIRST(Ap)={a, c}　　FIRST(Bq)={b, d}

且 FIRST(Ap) ∩ FIRST(Bq)=φ

对A: FIRST(a)={a}　　FIRST(cA)={c}

且 FIRST(a) ∩ FIRST(cA)=φ

对B : FIRST(b)={b}　　FIRST(dB)={d}

且 FIRST(b) ∩ FIRST(dB)= φ

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

自顶向下分析方法: LL(1)分析器构造

若是ε∈FIRST(α)怎么办？此时当面临 a∉FIRST(α)时并不一定出错。

设上下文无关文法G，S是文法的开始符号，对于文法G的任何非终结符A

$FOLLOW(A) = \{a \mid S \Rightarrow \ldots Aa \ldots,\ a \in V_T^*\}$

若$S \Rightarrow \ldots A$，则令 $\# \in FOLLOW(A)$。

FOLLOW(A)的含义是指，在文法G的一切句型中，能够紧跟着A之后的一切终结符或“#”。

自顶向下分析方法: LL(1)分析器构造

- 构造FOLLOW集方法

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

① 对文法的开始符号S，令$\# \in$FOLLOW(S)；

② 若文法G中有形如$A \rightarrow \alpha B\beta$的规则，且$\beta \neq \varepsilon$，则将FIRST(β)中的一切非ε符号加入FOLLOW(B)；

③ 若G中有形如$A \rightarrow \alpha B$ 或 $A \rightarrow \alpha B\beta$的规则，且$\varepsilon \in$FIRST(β)，则FOLLOW(A)中的全部元素属于FOLLOW(B)。

自顶向下分析方法: LL(1)分析器构造

- 构造FOLLOW集方法

例 设有文法G[S]为:

$S \rightarrow AB \mid bC$ $A \rightarrow \varepsilon \mid b$ $B \rightarrow \varepsilon \mid aD$

$C \rightarrow AD \mid b$ $D \rightarrow aS \mid c$

FOLLOW(S)={#} S => AB=>AaD=>AaaS

FOLLOW(A)={ a, c, # }

FOLLOW(B)={#}

FOLLOW(C)={#}

FOLLOW(D)={#}

$$\left\{\begin{array}{l} S \Rightarrow AB \Rightarrow AaD \\ S \Rightarrow bC \Rightarrow bAD \Rightarrow bAc \\ S \Rightarrow AB \Rightarrow A \end{array}\right.$$

自顶向下分析方法: LL(1)分析器构造

- 构造FOLLOW集方法：关系图法
  - 文法G中的每个符号和“#”对应图中的一个结点，对应终结符和“#”的结点符号本身标记。对应非终结符的结点则用FOLLOW(A)或FIRST(A)标记。
  - 从开始符号S的FOLLOW(S)结点到“#”号的结点连一条箭弧。
  - 如果文法中有产生式A→αBβX,且β=>ε,则从FOLLOW(B)结点到FIRST(X)结点连一条弧，当X∈V_T时，则与X相连。
  - 如果文法中有产生式A→αBβ,且β=>ε,则从FOLLOW(B)结点到FOLLOW(A) 结点连一条箭弧。
  - 对每一FIRST(A)结点如果有产生式A→αXβ,且α=>ε，则 从FIRST(A)到FIRST(X)结点连一条箭弧。
  - 凡是从FOLLOW(A)结点有路径可以到达的终结符或“#”号的结点，其所标记的终结符或“#”号即为FOLLOW(A)的成员。

自顶向下分析方法: LL(1)分析器构造

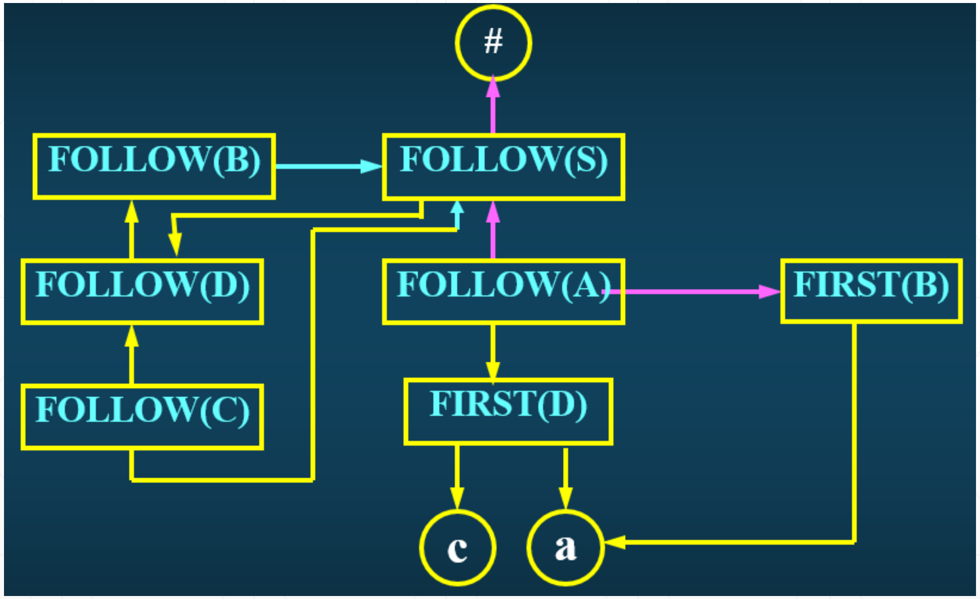
- 构造FOLLOW集方法：关系图法

例 设有文法G[S]为:

S→AB | bC　　A→ε| b

B→ε| aD　　C→AD| b

D→aS | c



自顶向下分析方法: LL(1)分析器构造

- 分析表构造

设对文法G求出FIRST和FOLLOW，则对$\alpha \overset{*}{\Rightarrow} \varepsilon$情况的文法确定惟一候选:

对$A \to \alpha_1 | \alpha_2$，当$\alpha_1$、$\alpha_2$不同时推出为$\varepsilon$时，设$\alpha_2 \overset{*}{\Rightarrow} \varepsilon$

如果满足：

$$FIRST(\alpha_1) \cap (FIRST(\alpha_2) \cup FOLLOW(A)) = \Phi$$

则惟一候选为:

若$a \in FIRST(\alpha_1)$，则置$M(A, a) = A \to \alpha_1$；
若$\varepsilon \in FIRST(\alpha_2)$且$b \in FOLLOW(A)$，则置$M(A, b) = A \to \varepsilon$。

自顶向下分析方法: LL(1)分析器构造

- 分析表构造

已知 $FOLLOW(A) = \{a \mid S \overset{*}{\Rightarrow} \ldots Aa\ldots,\ a \in V_T\}$

若有 $b \in FOLLOW(A)$，则必有 $S \Rightarrow \ldots Ab\ldots (b \in V_T)$ 设 $S \Rightarrow \ldots\ a\gamma Ab\beta\ \ldots$，由于对文法规则存在 $A \rightarrow \alpha$ 且 $\varepsilon \in FIRST(\alpha)$，当 $\alpha \Rightarrow \varepsilon$，则 $A \Rightarrow \varepsilon$。

故有 $S \Rightarrow \ldots a\gamma Ab\beta \ldots \Rightarrow \ldots a\gamma b\beta \ldots$

$\therefore\ M(A, b) = A \rightarrow \varepsilon$(这样在自上而下分析的推导中才可能从栈中退掉A。)

自顶向下分析方法: LL(1)分析器构造

- 分析表构造

输入： 文法 G;G 的 FIRST 和 FOLLOW 集合

输出： 文法 G 的 LL(1)分析表

算法：

```
for 文法 G 的每个产生式 A→γ1 |γ2 |… |γn |
{
  if a∈ FIRST(γi) 置 M[A,a]为"A→γi";
  if ε∈ FIRST(γi)
    for 任何 a∈ FOLLOW(A) {置 M[A,a]为"A→γi"}
}
置所有无定义的 M[A,a]为出错。
```

自顶向下分析方法: LL(1)分析器构造

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

对E： FIRST(TE′)={ (, i }

对T： FIRST(FT ′)={ (, i }

对F： FIRST((E))={ (} FIRST(i)={ i }

∩ = Φ

对E′： FIRST(+ TE′)={+}

FIRST(ε)= { ε }

FOLLOW(E′)= { #,) }

满足：{ +}∩({ ε } ∪ { #,) }) = Φ

对T′： FIRST(*FT′)={ * }

FIRST(ε)={ε}

FOLLOW(T′)={ #,), + }

满足：{* }∩({ε} ∪ { #,), + }) = Φ

	+	*	(	)	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→(E)		F→i	

自顶向下分析方法: LL(1)分析器构造

S→ iCtSS′ | a　　S′ →eS | ε　　C→ b

对S：　FIRST(iCtSS′)={ i }

FIRST(a)={ a }

对S′：　FIRST(eS)={ e }

FIRST(ε)={ε}　FOLLOW(S')={#,e}

FIRST(eS) ∩ (FIRST(ε) ∪ FOLLOW(S')) ≠Φ

	a	b	e	i	t	#
S	S→a			S→iCtSS’		
S′			S′ → ε S′ →eS			S′ →ε
C		C→b				

自顶向下分析方法

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