第3讲 语法分析

学习的主要内容和目标

- **学习的主要内容**
  - 语法分析的基本思想
  - 自顶向下的语法分析实现方法
- **学习的目标**
  - 理解程序设计语言的语法
  - 理解LL(1)文法的概念
  - 掌握递归下降分析实现机制

语法分析概述

- 程序设计语言的语法以及语法分析

```
int b=3;
 int main()
 {
 int a=2,sum;
 sum=a+b;
 return 0;
 }
```

语法分析概述

- 语法分析
  - 判定输入序列是否符合特定语法规则，并确定输入序列的语法结构。

词法记号的特点是什么?

- 实现语法分析的基本策略
  - 判定从词法分析器获得的词法记号序列是否符合语法的定义，并确定输入序列的语法结构

语法分析概述

- “程序设计语言”的语法
  - 可以使用上下文无关文法来描述的语言规则
- 语法分析的实现
  - 输入词法记号序列；
  - 词法记号之间的上下文无关约束
- 语法分析的数学抽象
  - 对任意上下文无关文法$G=(V_T,V_N,P,S)$和任意$w \in V_T^*$，判断$w \in L(G)$?

确定的自顶向下分析

➢回顾推导：*aaab*

S

$\Rightarrow AB$

$\Rightarrow aAB$

$\Rightarrow aaAB$

$\Rightarrow aaaAB$

$\Rightarrow aaaB$

$\Rightarrow aaab$

（1）$S \rightarrow AB$

（2）$A \rightarrow aA$

（3）$A \rightarrow a$

（4）$B \rightarrow b$

（5）$B \rightarrow bB$

确定的自顶向下分析

➢ 无回溯推导的关键和难点？

✧ *aaab*

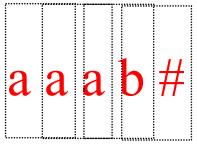
（1）$S \rightarrow AB$
（2）$A \rightarrow aA$
（3）$A \rightarrow a$
（4）$B \rightarrow b$
（5）$B \rightarrow bB$

确定的自顶向下分析

- 每一步推导都是确定的选择
  - 最左推导：
    - ✓总是对最左边的非终结符进行展开
  - 对当前非终结符选择一个确定的候选式
  - 从左向右扫描，向前查看确定数目的符号
  - 分析成功的结果：得到唯一的最左推导

确定的自顶向下分析

➢对 *aaab* 的预测分析



✧ 向前查看 2个符号

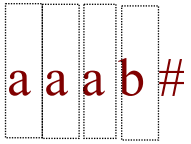
（1）$S \to AB$
（2）$A \to aA$
（3）$A \to a$
（4）$B \to b$
（5）$B \to bB$

$$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaAB \Rightarrow aaaB \Rightarrow aaab$$

只要向前查看 2 个符号，就可预测分析L(G)中所有句子

确定的自顶向下分析

➢对 aaab的预测分析



✧ 向前查看1个符号

文法 G[S]:

（1）$S \rightarrow AB$

（2）$A \rightarrow aA$

（3）$A \rightarrow a$

（4）$B \rightarrow b$

（5）$B \rightarrow bB$

S

$\Rightarrow AB$

$\Rightarrow aAB$

$\Rightarrow aaAB$

$\Rightarrow aaaB$

$\Rightarrow ?$

LL(1)文法

➢引入的目的

- ✧ 实现无回溯的语法分析
- ✧ 文法的子类
  - ✓ 在最左(L)推导中，从左(L)到右扫描输入序列并且读1个单词，就可以为每一步推导中当前非终结符号做唯一、确定的选择。

LL(1)文法

➢First 集合

设 $G=(V_T, V_N, P, S)$ 是上下文无关文法，对 $\alpha \in (V_T \cup V_N)^*$，

① $First(\alpha) = \{ a \mid \alpha \overset{*}{\Rightarrow} a\beta, a \in V_T, \beta \in (V_T \cup V_N)^* \}$

② 若$\alpha \overset{*}{\Rightarrow} \varepsilon$则规定$\varepsilon \in First(\alpha)$

LL(1)文法

➢计算First 集合

文法 G[S]:

(1) $S \to AB$
(2) $A \to Da \mid \varepsilon$
(3) $B \to cC$
(4) $C \to aADC \mid \varepsilon$
(5) $D \to b \mid \varepsilon$

$\text{First}(D) = \{b, \varepsilon\}$

$\text{First}(C) = \{a, \varepsilon\}$

$\text{First}(B) = \{c\}$

$\text{First}(A) = \{b, a, \varepsilon\}$

$\text{First}(S) = \{b, a, c\}$

LL(1)文法

➢计算 First 集合

- ✧ 设$X \in V_N \cup V_T$，则First(X) 可按如下步骤计算:
  - ✓ 若$X \in V_T$,则First(X)={X}
  - ✓ 若$X \in V_N$,且有产生式$X \to a\ldots$，($a \in VT$),则把a加入到First(X)中;
  - ✓ 若 $X \to \varepsilon$ 也是一个产生式,则把 ε 也加到First(X)中;
  - ✓ 若$X \to Y_1Y_2\ldots Y_K$ 是一个产生式,对于任何 i:1≤i≤k和任何 j:1≤j≤i-1, First(Y_j)都含有ε, 但 First(Y_i)不含ε, 则把 First(Y_j)中的所有非ε元素和First(Y_i)中的所有元素都加到First(X)中; 特别是若 First(Y_i)都含ε，则把所有First(Y_i) 都加到First(X)中.

LL(1)文法

➢Follow 集合

✧ 设 G =(V_T，V_N，P，S)是上下文无关文法, 对 A∈V_N，

$$\text{Follow}(A)=\{a \mid S \overset{*}{\Rightarrow} \alpha A \beta \text{且} a \in \text{First}(\beta),\ \alpha \in (V_T \cup V_N)^*,\ \beta \in (V_T \cup V_N)^+\}$$

✧ 规定#∈Follow（S）

✧ #代表输入单词序列右边的结束符

LL(1)文法

➢Follow 集合

- ✧ 对于文法的开始符号S，置#于Follow(S) 中;
- ✧ 若$A \to \alpha B \beta$是一个产生式,则把 First(β)–{ε}加至 Follow(B) 中;
- ✧ 若 $A \to \alpha B$ 是一个产生式, 或$A \to \alpha B \beta$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 .

LL(1)文法

➢计算Follow 集合

文法 G[S]:

(1) $S \to AB$
(2) $A \to Da \mid \varepsilon$
(3) $B \to cC$
(4) $C \to aADC \mid \varepsilon$
(5) $D \to b \mid \varepsilon$

First(D) = {b, ε}　　Follow(S) = {#}

First(C) = {a, ε}　　Follow(C) = {#}

First(B) = {c}　　Follow(A) = {c,b,a, #}

First(A) = {b,a, ε}　　Follow(B) = {#}

First(S) = {b,a,c}c　　Follow(D) = {a,#}

LL(1)文法

➢LL(1)文法定义

✧ 文法G是LL(1)，当且仅当对G的每个非终结符A的任何两个候选式 $A\to\alpha|\beta$满足：

✓ First(α)∩First(β)=ϕ，即α和β推导不出同一个终结符为首的符号串，也不同时推导出ε

✓ 若$\beta\overset{*}{\Rightarrow}\varepsilon$，则First(α)∩Follow(*A*)＝ϕ,即α所能推出的串的首符号不应在Follow(*A*）中

LL(1)文法

➢ 验证如下文法G(S)不是LL(1)文法

文法 G[S]:

(1) $S \to AB$
(2) $A \to Da \mid \varepsilon$
(3) $B \to cC$
(4) $C \to aADC \mid \varepsilon$
(5) $D \to b \mid \varepsilon$

First(D) = {b, ε}
First(C) = {a, ε}
First(B) = {c}
First(A) = {b,a, ε}
First(S) = {b,a,c}c

First(Da) = {b,a}
Follow(A) = {c,b,a, #}
First(b) = {b}
Follow(D) = {a, #}
First($aADC$) = {a}
Follow(C) = {#}

Follow(S) = {#}
Follow(C) = {#}
Follow(A) = {c,b,a, #}
Follow(B) = {#}
Follow(D) = {a,#}

因为$A \to Da \mid \varepsilon$的两个候选式有冲突

左递归和左公因子的消除

➢ 左递归带来的问题

✧ 考虑下列文法识别 ba^n 的分析过程

文法 G[S]:

(1) $S \to Sa$

(2) $S \to b$

$$
\begin{aligned}
& S \\
\Rightarrow\ & Sa \\
\Rightarrow\ & Saa \\
\Rightarrow\ & Saaa \\
& \cdots\cdots \\
\Rightarrow\ & Sa^n \\
\Rightarrow\ & ba^n
\end{aligned}
$$

✧ 需要向前查看n+1个符号，才能确定这样的推导序列

左递归和左公因子的消除

➢递归特征和定义

- ✧ 递归： $A \overset{+}{\Rightarrow} \alpha A \beta$
- ✧ 左递归： $A \overset{+}{\Rightarrow} A \beta$
- ✧ 右递归： $A \overset{+}{\Rightarrow} \alpha A$
- ✧ 直接递归： $A \Rightarrow \alpha A \beta$
- ✧ 间接递归： $A \overset{>1}{\Rightarrow} \alpha A \beta$

左递归和左公因子的消除

➢ 左递归消除

✧ 对如下产生式，如果α，β不以 P 打头

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

✧ 可改写为：

$$P \to \beta Q$$
$$Q \to \alpha Q \mid \varepsilon$$

其中Q为新增加的非终结符

左递归和左公因子的消除

➢ 左递归消除

✧ 原文法 G[E]:

$$E \to E + T \mid T$$

$$T \to T * F \mid F$$

$$F \to (E) \mid a$$

✧ 消除左递归后的文法 G'[E]:

$$E \to TR$$

$$R \to + TR \mid \varepsilon$$

$$T \to FP$$

$$P \to * FP \mid \varepsilon$$

$$F \to (E) \mid a$$

左递归和左公因子的消除

➢ 左递归消除

✧ 原文法 G[S]:

$$S \to PQ \mid a$$

$$P \to QS \mid b$$

$$Q \to SP \mid c$$

✧ “代入”

$$Q \to SP \mid c$$

$$Q \to PQP \mid aP \mid c$$

$$Q \to QSQP \mid bQP \mid aP \mid c$$

$$Q \to bQPR \mid aPR \mid cR$$

$$R \to SQPR \mid \varepsilon$$

结果:

$$S \to PQ \mid a$$

$$P \to QS \mid b$$

$$Q \to bQPR \mid aPR \mid cR$$

$$R \to SQPR \mid \varepsilon$$

左递归和左公因子的消除

➢左公因子

✧ 向前查看3个符号来预测分析,文法 G[S]:

$$S \to abA \mid abB$$
$$A \to a$$
$$B \to b$$

✧ 向前查看?个符号来预测分析,文法 G[S]:

$$S \to aAb \mid aAc$$
$$A \to a \mid aA$$

左递归和左公因子的消除

➢左公因子

✧ 对形如 $P \to \alpha\beta \mid \alpha\gamma$ 的一对产生式，可用如下三个产生式替换：

$$P \to \alpha Q$$
$$Q \to \beta \mid \gamma$$

其中Q为新增加的未出现过的非终结符

左递归和左公因子的消除

➢左公因子

✧ 一般含有左公因子的产生式形如

$$P \to \alpha\beta_1 | \alpha\beta_2 | \ldots | \alpha\beta_m | \gamma_1 | \gamma_2 | \ldots | \gamma_n$$

其中，每个γ不以α开头.提取左公共因子， 产生式改写成:

$$P \to \alpha Q | \gamma_1 | \gamma_2 | \ldots | \gamma_n$$

$$Q \to \beta_1 | \beta_2 | \ldots | \beta_m$$

左递归和左公因子的消除

➢ 左公因子

✧ 对文法 G[S]:

$S \rightarrow \underline{\text{if}}\ C\ \text{t}\ S \mid \underline{\text{if}}\ C\ \text{t}\ S\ e\ S$

$C \rightarrow b$

提取左公因子后，可改写为文法G'[S]:

$S \rightarrow \underline{\text{if}}\ C\ \text{t}\ S\ A$

$A \rightarrow \varepsilon \mid e\ S$

$C \rightarrow b$

实现LL(1)文法的预测分析

- 递归下降分析程序
- 非递归分析

递归下降LL(1)分析程序

➢ 基本特点

- ✧ 每个非终结符都对应一个子程序：
- ✧ 每个子程序若干分支，每个分支的行为根据语法描述来明确：
  - ✓ 每遇到一个终结符，则判断当前读入的单词是否与该终结符相匹配，若匹配，再读取下一个单词继续分析；不匹配，则进行出错处理
  - ✓ 每遇到一个非终结符，则调用相应的子程序

递归下降LL(1)分析程序

- 非终结符A对应的递归下降子程序
- 产生式$A \rightarrow u_1 \mid u_2 \mid \ldots$

```
void ParseA()
{
    switch (lookahead)  {
        case First(u1):
            /* code to recognize u1 */
            break;
        case First(u2):
            /* code to recognize u2 */
            break;
        ...
        case Follow(A): /* when A ⇒* ε */
            /* usually do nothing here */
            break;
        default:
            printf("syntax error \n");
            exit(0);
    }
}
```

递归下降LL(1)分析程序

➢递归下降分析程序实例

文法G（S）：

$S \rightarrow aAS \mid bB \mid d$

$A \rightarrow BbS \mid \varepsilon$

$B \rightarrow c$

	FIRST	FOLLOW
S	*a*, *b*, *d*	#, *a*, *b*, *d*
A	*c*, ε	*a*, *b*, *d*
B	*c*	*b*, #, *a*, *d*

递归下降LL(1)分析程序

➢递归下降分析程序实例

$S \rightarrow aAS \mid bB \mid d$

```
void ParseS( ){
  if (token==a) {
      gettoken( );
      ParseA( );
      ParseS( );
  }
  else if (token ==b) {
      gettoken( );
      ParseB( );
  }
  else if (token==d)
      gettoken( );
  else {
      printf("syntax error \n");
      exit(0);
  }
}
```

递归下降LL(1)分析程序

➢递归下降分析程序实例

$A \rightarrow BbS \mid \varepsilon$

```
void ParseA( ){
    if (token==c) {
        ParseB( );
        if (token==b) gettoken ( );
        else{
            printf("syntax error \n”);
            exit(0);
        }
        ParseS( );
    }
else if (token ==a|| token ==b|| token ==d){}
 else{
    printf("syntax error \n”);
    exit(0);
 }
}
```

递归下降LL(1)分析程序

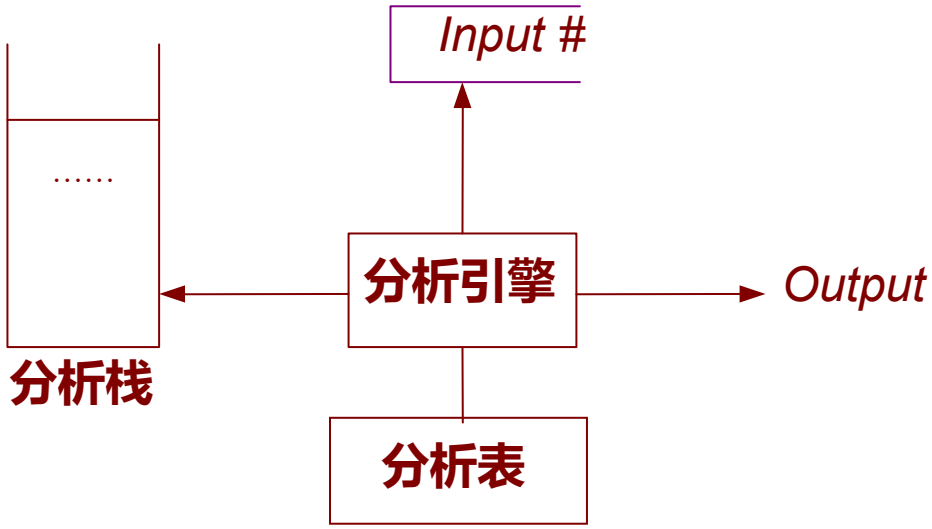
➢递归下降分析程序实例

$B \rightarrow c$

```
void ParseB( ){
    if (token==c) {
        gettoken();
    }
    else{
        printf("syntax error \n”);
        exit(0);
    }
}
```

表驱动预测分析程序

➢ 基本框架



表驱动预测分析程序

➢预测分析表

文法 G（S）：

$S \rightarrow aAS \mid bB \mid d$

$A \rightarrow BbS \mid \varepsilon$

$B \rightarrow c$

	a	b	c	d	#
S	$S \rightarrow aAS$	$S \rightarrow bB$		$S \rightarrow d$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	$A \rightarrow BbS$	$A \rightarrow \varepsilon$	
B			$B \rightarrow c$		

表驱动预测分析程序

- LL(1)文法可以构造出一个M(A,a)最多只包含一个产生式的预测分析表
- LL(1)预测分析表的构造算法
  - ✧ 对文法G的每个产生式A→α执行如下步骤：
    - ✓ 对每个a∈First(α)，把A→α加入M[A,a]
    - ✓ 若ε∈First(α)，则对任何b∈Follow(A)，把A→α加至M[A,b]中把所有无定义的M[A,a]标上“出错标志”
  - ✧ 可以证明：这样为一个文法G构造的的预测分析表不含多重入口，当且仅当该文法是LL(1)的

表驱动预测分析程序

- 算法

a:=gettoken();

push # ;

push *S*;

*X*是栈顶符号;

while(*X*≠#) {

  if (*X*∉V_N and *X*=*a*) {

    pop;

    a:=gettoken();

  }

  elseif M[*X*, *a*]=*error* error();

  elseif M[*X*, *a*]=*X*→$Y_1Y_2\ldots Y_n$ {

    pop;

    for *i*= *n* to 1 do push Y_i;

}

if (*X*=# and *a*=#) return;

表驱动预测分析程序

➢ 表驱动预测分析过程

文法 G（S）：

剩余的输入串

abc#

$S \rightarrow aAS \mid bB \mid d$

$A \rightarrow BbS \mid \varepsilon$

$B \rightarrow c$

	a	*b*	*c*	*d*	#
S	$S \rightarrow aAS$	$S \rightarrow bB$		$S \rightarrow d$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	$A \rightarrow BbS$	$A \rightarrow \varepsilon$	
B			$B \rightarrow c$		

S
#

表驱动预测分析程序

➢ 表驱动预测分析过程

剩余的输入串

abc#

a
A
S
#

文法 G（S）：

$S \rightarrow aAS \mid bB \mid d$

$A \rightarrow BbS \mid \varepsilon$

$B \rightarrow c$

	a	*b*	*c*	*d*	#
S	$S \rightarrow aAS$	$S \rightarrow bB$		$S \rightarrow d$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	$A \rightarrow BbS$	$A \rightarrow \varepsilon$	
B			$B \rightarrow c$		

表驱动预测分析程序

➢ 表驱动预测分析过程

文法 G（S）：

剩余的输入串

bc#

$S \rightarrow aAS \mid bB \mid d$

$A \rightarrow BbS \mid \varepsilon$

$B \rightarrow c$

	a	*b*	*c*	*d*	#
S	$S \rightarrow aAS$	$S \rightarrow bB$		$S \rightarrow d$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	$A \rightarrow BbS$	$A \rightarrow \varepsilon$	
B			$B \rightarrow c$		

A
S
#

表驱动预测分析程序

➢ 表驱动预测分析过程

文法 G（S）：

剩余的输入串

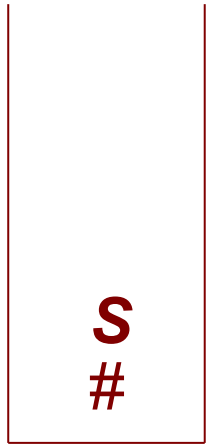
bc#

$S \rightarrow aAS \mid bB \mid d$

$A \rightarrow BbS \mid \varepsilon$

$B \rightarrow c$

	a	*b*	*c*	*d*	#
S	$S \rightarrow aAS$	$S \rightarrow bB$		$S \rightarrow d$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	$A \rightarrow BbS$	$A \rightarrow \varepsilon$	
B			$B \rightarrow c$		



表驱动预测分析程序

➢ 表驱动预测分析过程

剩余的输入串

bc#

文法 G（S）：

$S \rightarrow aAS \mid bB \mid d$

$A \rightarrow BbS \mid \varepsilon$

$B \rightarrow c$

	a	*b*	*c*	*d*	#
S	$S \rightarrow aAS$	$S \rightarrow bB$		$S \rightarrow d$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	$A \rightarrow BbS$	$A \rightarrow \varepsilon$	
B			$B \rightarrow c$		

b
B
#

表驱动预测分析程序

➢ 表驱动预测分析过程

文法 G（S）：

$S \rightarrow aAS \mid bB \mid d$

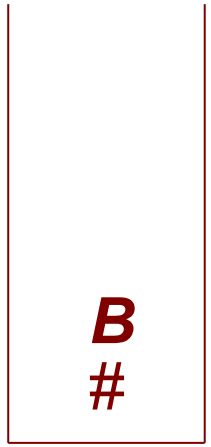
$A \rightarrow BbS \mid \varepsilon$

$B \rightarrow c$

剩余的输入串

$c\#$

	a	b	c	d	#
S	$S \rightarrow aAS$	$S \rightarrow bB$		$S \rightarrow d$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	$A \rightarrow BbS$	$A \rightarrow \varepsilon$	
B			$B \rightarrow c$		



表驱动预测分析程序

➢ 表驱动预测分析过程

剩余的输入串

c#

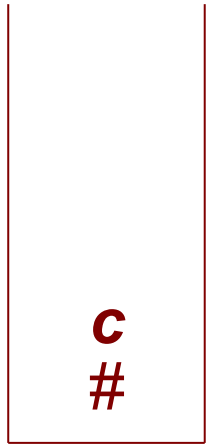
文法 G（S）：

$S \rightarrow aAS \mid bB \mid d$

$A \rightarrow BbS \mid \varepsilon$

$B \rightarrow c$

	a	*b*	*c*	*d*	#
S	$S \rightarrow aAS$	$S \rightarrow bB$		$S \rightarrow d$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	$A \rightarrow BbS$	$A \rightarrow \varepsilon$	
B			$B \rightarrow c$		



表驱动预测分析程序

➢ 表驱动预测分析过程

文法 G（S）：

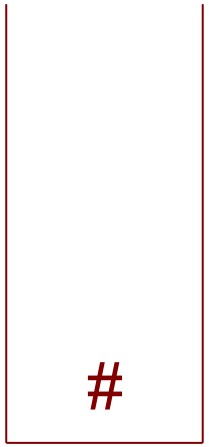
$S \rightarrow aAS \mid bB \mid d$

$A \rightarrow BbS \mid \varepsilon$

$B \rightarrow c$

剩余的输入串

#



	a	b	c	d	#
S	$S \rightarrow aAS$	$S \rightarrow bB$		$S \rightarrow d$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	$A \rightarrow BbS$	$A \rightarrow \varepsilon$	
B			$B \rightarrow c$		

表驱动预测分析程序

步骤	分析栈	余留符号	使用的产生式
0	#	abc#	
1	#S	abc#	S→ aAS
2	#SAa	abc#	
3	#SA	bc#	A→ε
4	#S	bc#	
5	#Bb	bc#	S→ bB
6	#B	c#	
7	#c	c#	B→c
8	#	#	

预测分析中的错误处理

➢ 表驱动LL(1)分析中的错误处理

✧ 错误报告（error reporting）

- ✓ 栈顶的终结符与当前输入符不匹配
- ✓ 非终结符A于栈顶，面临的输入符为a， 但分析表M的M[A,a]为空

预测分析中的错误处理

➢应急方式错误处理

- ✧ 把 Follow(A) 中的所有符号作为A的同步符号, 跳过 输入串中的一些符号直至遇到这些“同步符号”， 把A从栈中弹出，可使分析继续
- ✧ 把First(A)中的符号加到A的同步符号集，当First(A) 中的符号在输入中出现时，可根据A恢复分析

自底向上的语法分析

- $aaab$ 自底向上分析过程

文法 G[S]:

$S \to AB$

$A \to aA \mid a$

$B \to b \mid bB$

$$
\begin{aligned}
& aaab && (A \to a) \\
\Leftarrow\ & aaAb && (A \to aA) \\
\Leftarrow\ & aAb && (A \to aA) \\
\Leftarrow\ & Ab && (B \to b) \\
\Leftarrow\ & AB && (S \to AB) \\
\Leftarrow\ & S
\end{aligned}
$$

自底向上的语法分析

- 移进归约模式

文法 G[S]:

$$S \to AB$$
$$A \to aA \mid a$$
$$B \to b \mid bB$$

步骤	分析栈	余留输入串	动作
(0)		*aaab#*	*Shift*
(1)	*a*	*aab#*	*Shift*
(2)	*aa*	*ab#*	*Shift*
(3)	*aaa*	*b#*	*reduce*
(4)	*aaA*	*b#*	*reduce*
(5)	*aA*	b#	*reduce*
(6)	*A*	*b#*	*shift*
(7)	*Ab*	#	*reduce*
(8)	*AB*	#	*reduce*
(9)	S	#	*accept*