第四章 自上而下语法分析

廖力
xobjects@seu.edu.cn
3793235

引言

1、语法分析的地位

– 是编译程序的核心部分。

2、语法分析的任务

– 识别由词法分析得出的单词序列是否是给定文法的句子。

3、语法分析的理论基础

– 上下文无关文法和下推自动机

引言

4、语法分析的方式

1）自上而下语法分析

- 反复使用不同产生式进行推导以谋求与输入符号串相匹配。

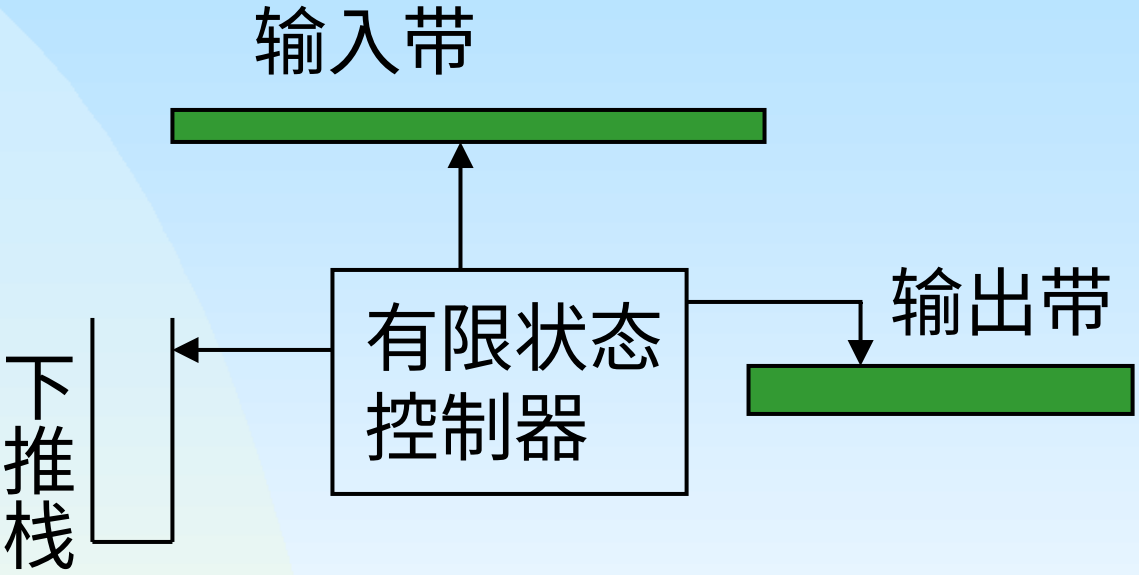
2）自下而上语法分析

- 对输入符号串寻找不同产生式进行归约直到文法开始符号。

注：这里所说的输入符号指词法分析所识别的单词。

第一节　下推自动机(PDA)

一、下推自动机模型



注：1）PDA和FA的模型相比，多了一个下推栈。

2）PDA的动作由三个因素来决定：当前状态、读头所指向符号、下推栈栈顶符号。

3）一个输入串能被PDA所接受，仅当输入串读完，下推栈变空；或输入串读完，控制器到达某些终态。

第一节　下推自动机(PDA)

一、下推自动机模型

注：4）正规文法和有限自动机仅适合于描述和识别高级语言的各类单词，语句可用上下文无关文法来描述，而下推自动机又恰好能识别上下文无关文法所能描述的语言，因此上下文无关文法及其对应的下推自动机就成为编译技术中语法分析的理论基础。

第一节　下推自动机（PDA)

二、下推自动机的形式定义

1、定义：

PDA是一个七元组：$M=(Q,\Sigma,H,\delta,q_0,z_0,F)$

- H：下推栈内字母表
- z_0 ：下推栈的栈初始符，是H中的元素。
- δ ：$Q\times(\Sigma\cup\{\varepsilon\})\times H$到$Q\times H^*$的有限子集映射。

例如：$\delta(q,a,z)=\{(q_1,r_1),(q_2,r_2)\ldots\ldots(q_n,r_n)\}$,其意义是：设控制器当前状态为q,下推栈顶符号为a,输入符号为a(可为空)，则状态转换到序偶集$\{(q_1,r_1),(q_2,r_2)\ldots\ldots(q_n,r_n)\}$($q_i$为下一状态, r_i为代替z的栈顶符号串)

- $F\subseteq Q$是控制器的终态集

第一节　下推自动机（PDA)

二、下推自动机的形式定义

注：1)由此定义的PDA肯定是不确定的PDA。这给语法分析会带来不确定性。我们在构造PDA M的算法的时候，要对PDA做一些限制。

2) PDA采用“ ⊢”来表示PDA做了一步动作。

3)有时，下推自动机还配置输出带，以记录推导或归约过程所用的产生式编号。

back

- 设CFG $G=(V_N, \Sigma, P, S)$,相应的不确定的PDA $M=(Q, \Sigma, H, \delta, q_0, z_0, F)$，其中：
- 1)$Q=F=\{q_0\}$；
- 2)$H=V_N \cup \Sigma$；
- 3) $z_0=S$；
- 4) δ映射关系为：
  - 对于形如$A\rightarrow\omega$的产生式，有$\delta(q, \varepsilon, A)=(q, \omega)$,这称为推导。
  - 有 $\delta(q, a, a)=(q, \varepsilon)$称为匹配，其中$a\in\Sigma$。
- 5)输入串能为PDA所接受，仅当输入串读完，下推栈为空。

第一节 下推自动机（PDA)

例：试给出接受语言L＝{$a^n c b^n$|n>=0}的下推自动机。

解： 1、生成L语言相应文法的产生式为S → aSb|c

2、生成相应的PDA M=(Q ,Σ,H,δ,q_0,z_0,F)

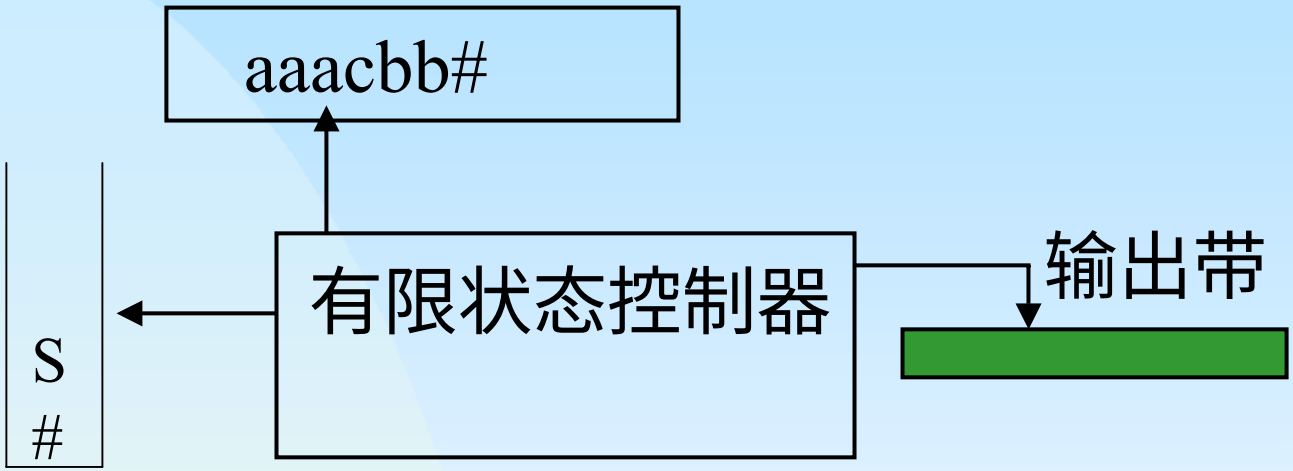
PDA M=({q},{a,b,c},{S,a,b,c}, δ ,q,S,{q})

其中δ 为：1) δ(q,a,a)= (q, ε)

2) δ(q,b,b)= (q, ε)

3) δ(q,c,c)= (q, ε)

4) δ (q, ε S)={(q,aSb),(q,c)}



其分析过程：

$(q,aacbb,S) \vdash_1 (q,aacbb,aSb) \vdash (q,acbb,Sb)$
$\vdash_1 (q,acbb,aSbb) \vdash (q,cbb,Sbb) \vdash_2 (q,cbb,cbb)$
$\vdash (q,bb,bb) \vdash (q,b,b) \vdash (q, \varepsilon, \varepsilon)$

第二节 自上而下分析法的一般问题

一、自上而下语法分析定义

- 从文法的开始符号开始，反复使用不同产生式进行推导以谋求与输入符号串相匹配。

注：此处的输入符号串是指词法分析结果的一串二元式。

二、一般方法

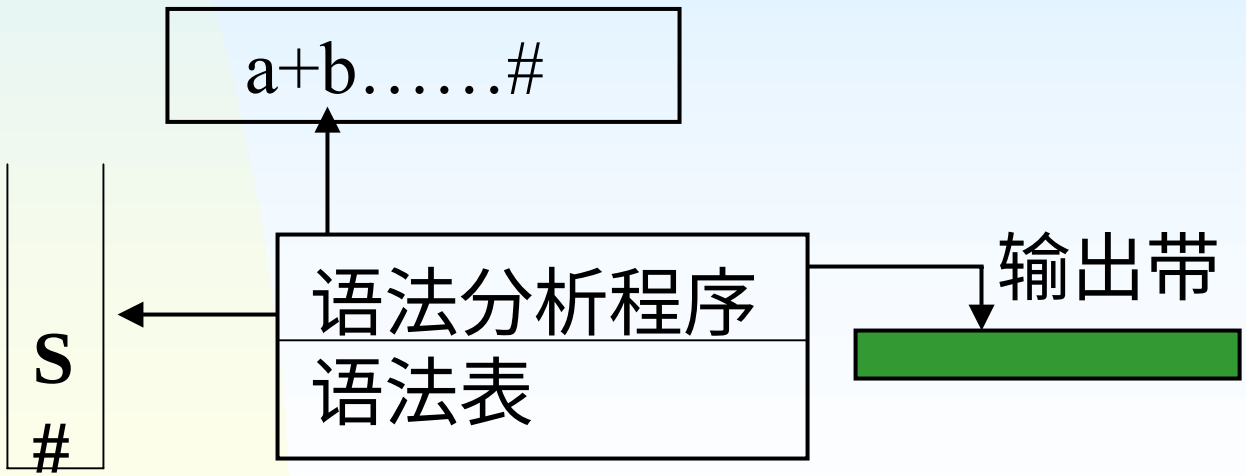
试探法：带回溯的自上而下分析法。

第二节 自上而下分析法的一般问题

二、一般方法

1、基本构成

设下推栈的初始状态包含两个符号：‘#S’，其中‘#’为栈底，‘S’为文法开始符号。整个分析过程在语法分析程序控制下进行。在语法分析中用到的文法产生式的表，称为语法表。



第二节 自上而下分析法的一般问题

二、一般方法

2、算法

（1）若栈顶符号x是非终结符，查询语法表，找出一个以x作为左部的产生式，x出栈，并将其右部反序入栈，且输出带记下产生式编号——**推导**。

（2）若栈顶符号x是终结符，且读头下的符号也是x，则x出栈，读头指向下一个符号——**匹配**。

（3）若栈顶符号x是终结符，但读头下的符号不是x，则匹配失败。这说明可能前面推导时选错了候选式，退回到上次推导现场(包括栈顶符号、读头的指针和输出带上信息)——**回溯**。

第二节 自上而下分析法的一般问题

二、一般方法

2、算法

（4）回溯后选取另一候选式进行推导，若没有候选式可选，则进一步回溯。若回溯到开始符号又已无候选式可选，则识别失败。

（5）若栈内仅剩下“#”，且读头也指向“#”，则识别成功。

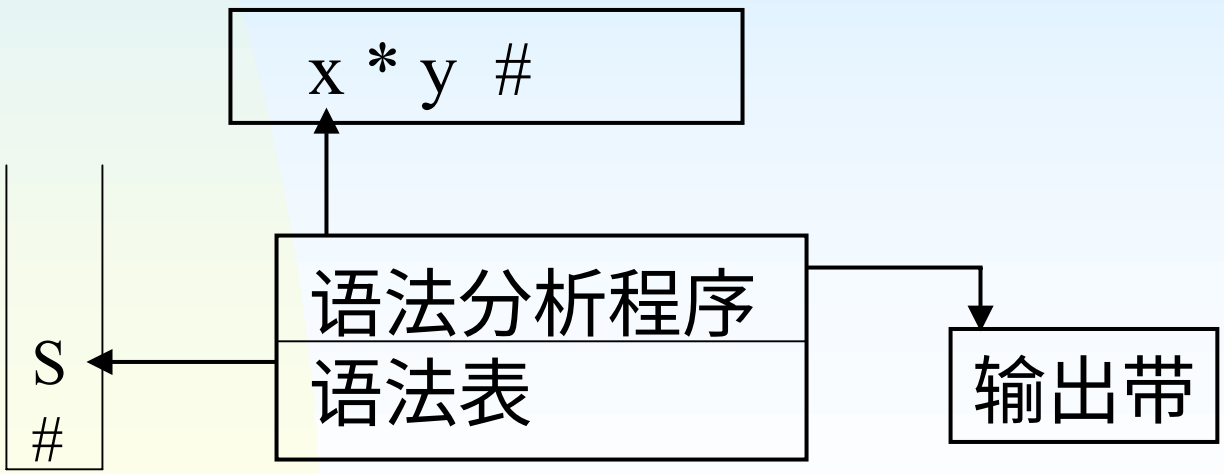
第二节 自上而下分析法的一般问题

二、一般方法

例：文法产生式如下，请分析符号串x*y#的过程

1) S → xAy

2) A →** 3)A → *



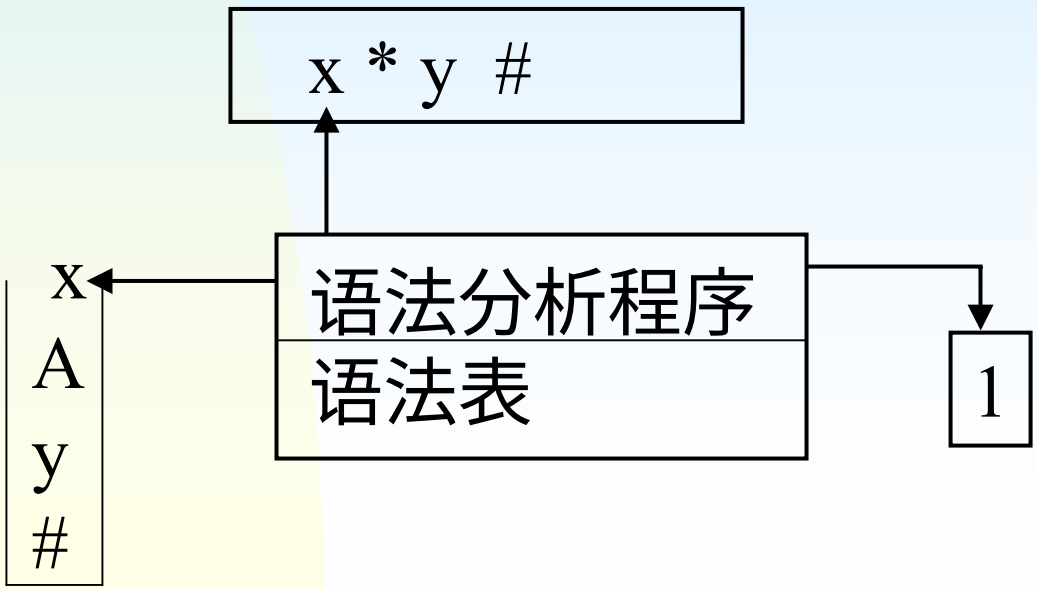
第二节 自上而下分析法的一般问题

二、一般方法

例 文法产生式如下，请分析符号串x*y#的过程

1) S → xAy

2) A →**　　3)A → *



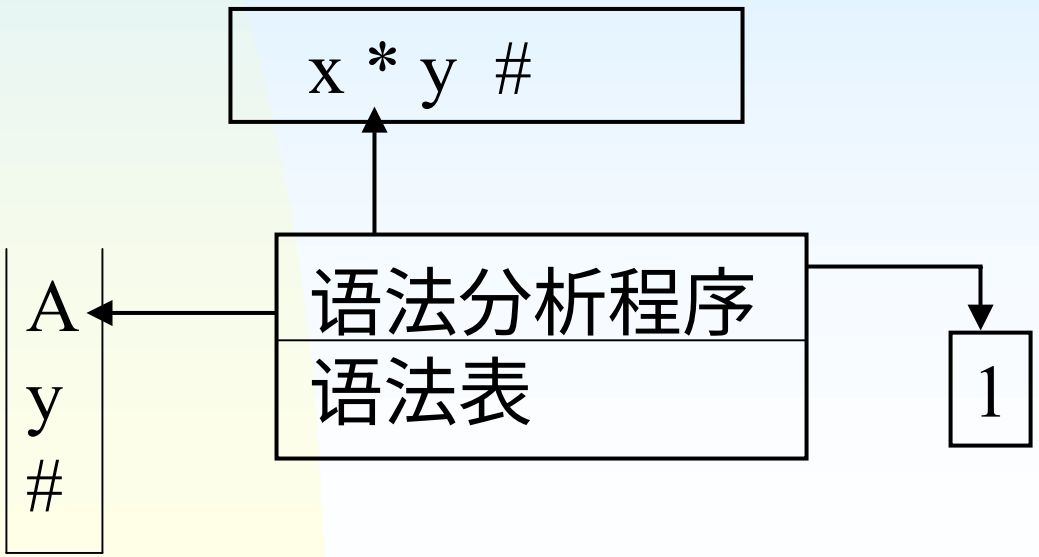
第二节 自上而下分析法的一般问题

二、一般方法

例　文法产生式如下，请分析符号串x*y#的过程

1) S → xAy

2) A →**　　3)A → *



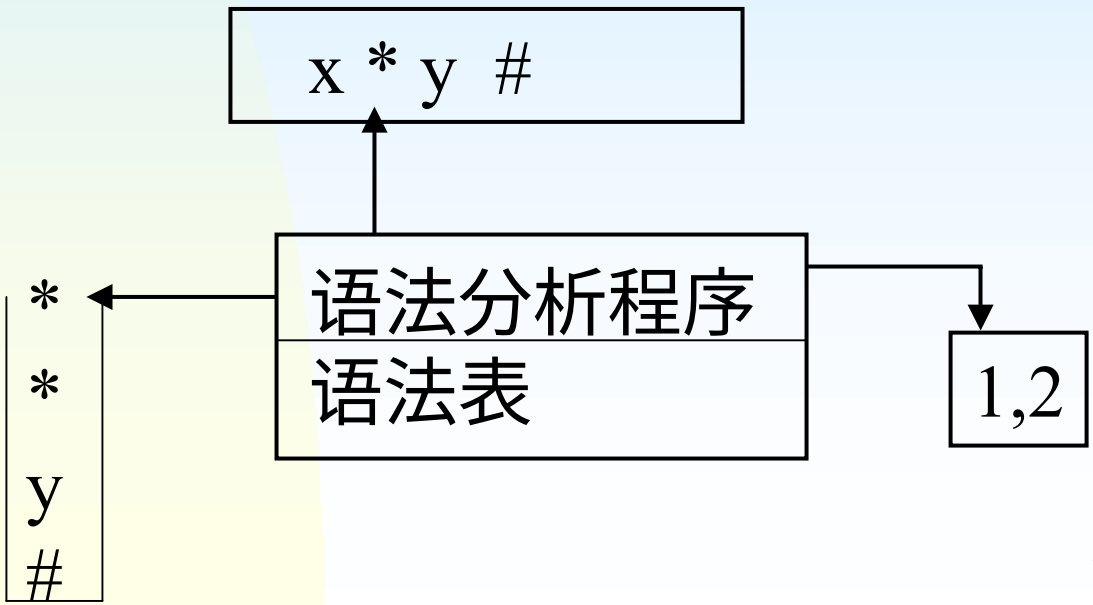
第三节 自上而下分析法的一般问题

二、一般方法

例 文法产生式如下，请分析符号串x*y#的过程

1) S → xAy

2) A →**　　3)A → *



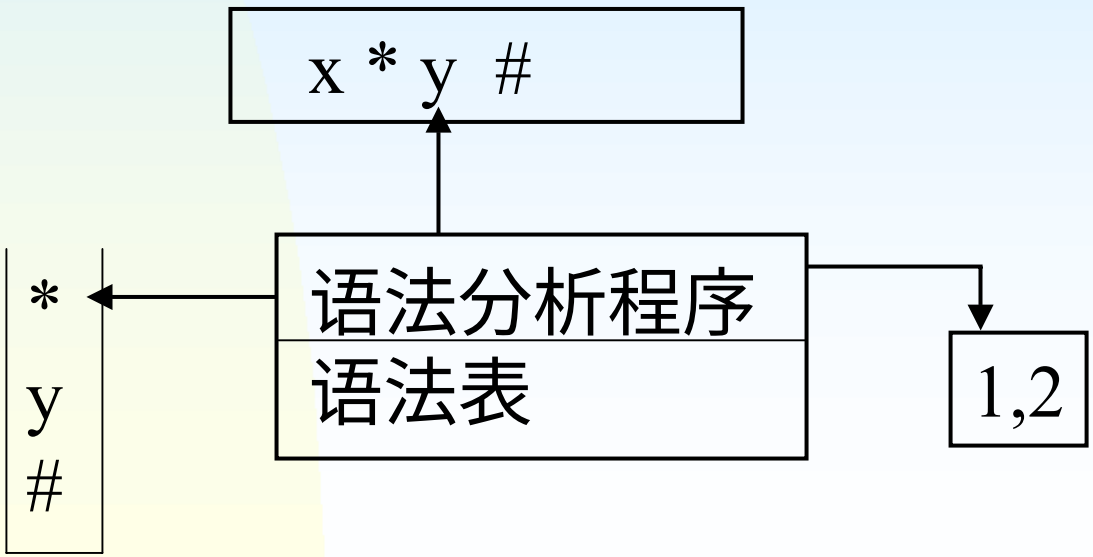
第三节 自上而下分析法的一般问题

二、一般方法

例　文法产生式如下，请分析符号串x*y#的过程

1) S → xAy

2) A →**　　3)A → *



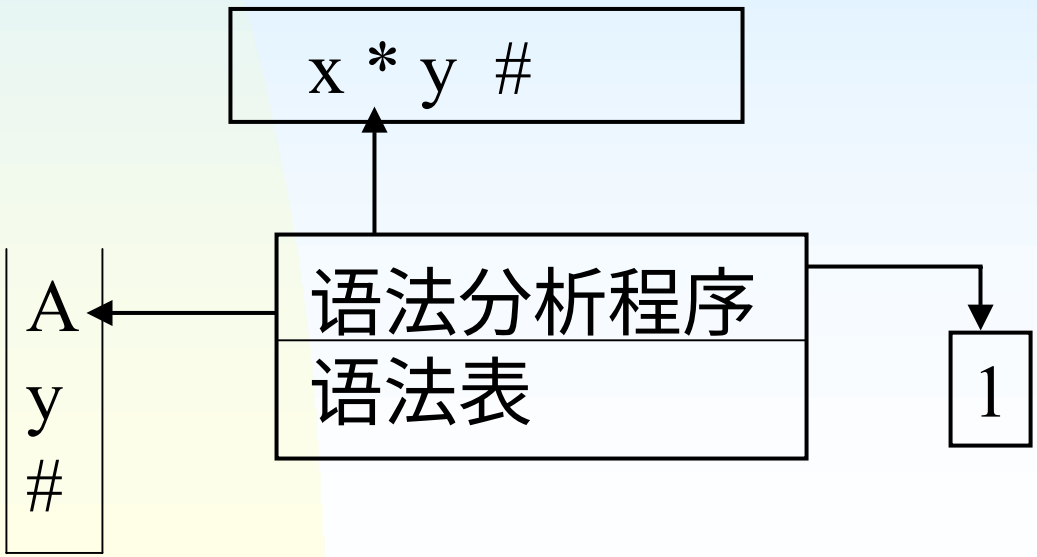
第二节 自上而下分析法的一般问题

二、一般方法

例　文法产生式如下，请分析符号串x*y#的过程

1) S → xAy

2) A →**　　3)A → *



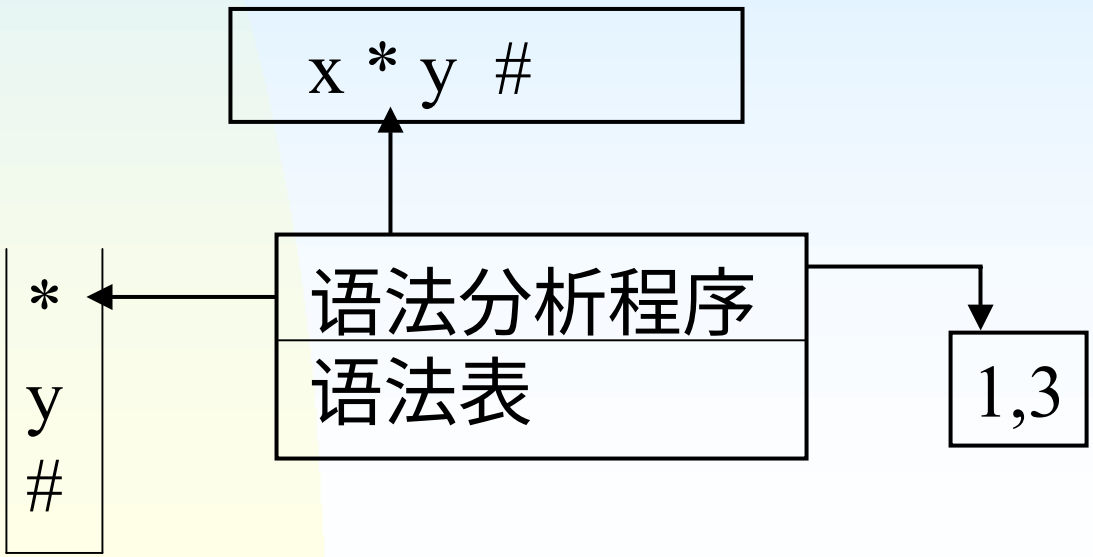
第三节 自上而下分析法的一般问题

二、一般方法

例　文法产生式如下，请分析符号串x*y#的过程

1) S → xAy

2) A →**　　3)A → *



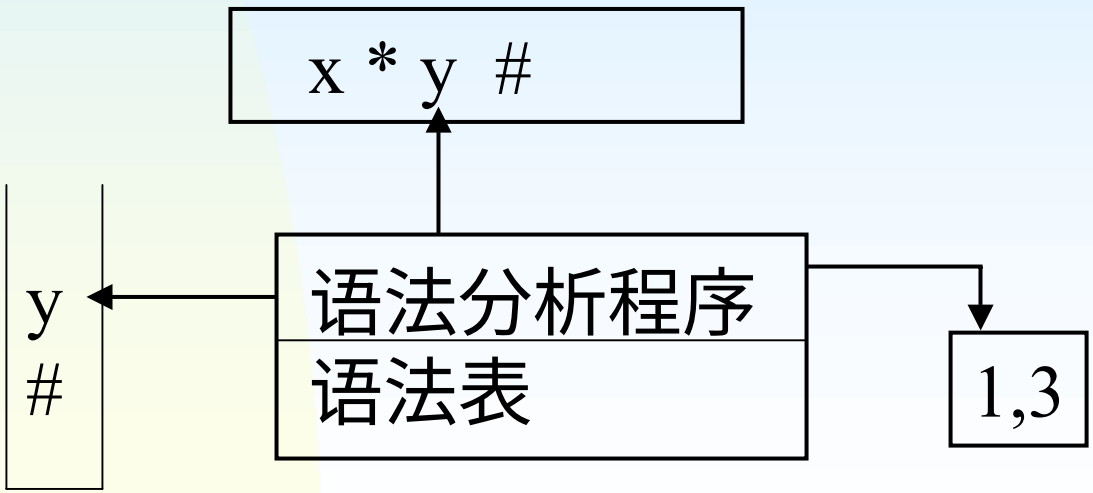
第二节 自上而下分析法的一般问题

二、一般方法

例　文法产生式如下，请分析符号串x*y#的过程

1) S → xAy

2) A →**　　3)A → *



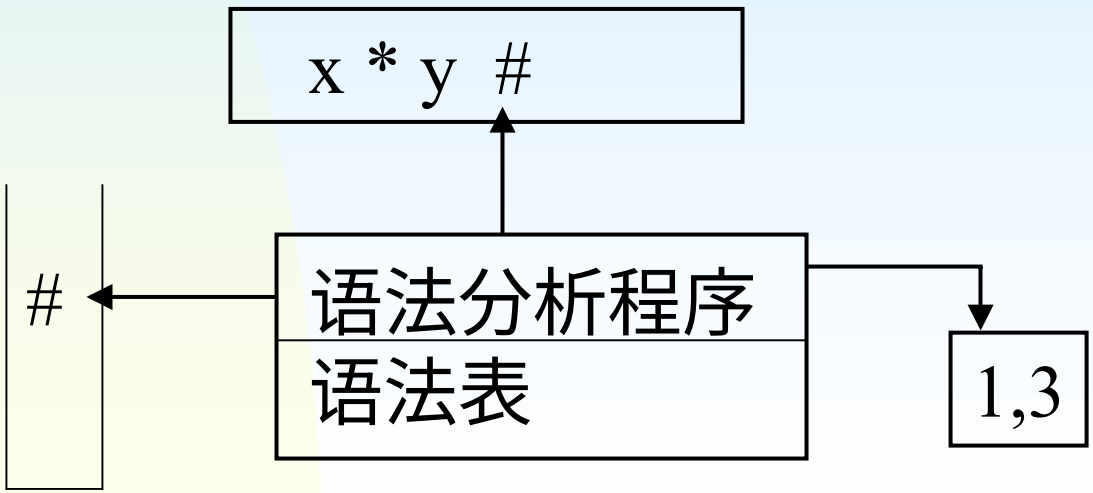
第二节 自上而下分析法的一般问题

二、一般方法

例　文法产生式如下，请分析符号串x*y#的过程

1) S → xAy

2) A →**　　3)A → *



第二节 自上而下分析法的一般问题

二、一般方法

3、带回溯的自上而下分析法的缺陷

1）如果文法存在左递归，语法分析会无限循环下去。

2）若产生式存在多个候选式，选择哪个进行推导完全是盲目的。

3）回溯会引起时间和空间的大量消耗。

4）如果被识别的语句是错的，算法无法指出错误的确切位置。

第二节 自上而下分析法的一般问题
三、不带回溯的自上而下分析算法

1、消除左递归

- 1)什么是左递归
  - 左递归：文法存在产生式$P \overset{+}{\rightarrow} Pa$
  - 直接左递归： $P \rightarrow Pa$
  - 间接左递归： $P \rightarrow Aa$, $A \overset{+}{\rightarrow} Pb$
- 2)消除左递归
  - 消除直接左递归
  - 消除间接左递归

第二节 自上而下分析法的一般问题
三、不带回溯的自上而下分析算法

2、消除直接左递归

设有文法$G=(V_N, V_T, P, S)$，其中产生式P为

$$P \rightarrow P\alpha|\beta\ ,\ \alpha \in V^+\ ,\ \beta \in V^+ \text{ 且}\beta\text{不以P开头}$$

将它转换为等价式：

$$P \rightarrow \beta P' \qquad P' \rightarrow \alpha P'|\varepsilon$$

一般地：

将$P \rightarrow P\alpha_1|P\alpha_2|\ldots|P\alpha_m|\beta_1|\beta_2|\ldots|\beta_n$

转换为：$P \rightarrow \beta_1 P`|\beta_2 p`|\ldots\ldots|\beta_n P`$

$$P` \rightarrow \alpha_1 P`|\alpha_2 P`|\ldots\ldots|\alpha_m P`|\varepsilon$$

例：文法G: (1)E → E+T|T

(2)T → T*F|F

(3)F →(E)| i

转化为G`:

(1) → E →TE` E` →+TE`| ε

(2) → T →FT` T` →*FT`| ε

(3) → F →(E)| i

例：文法G:P → PaPb|BaP

转化为：P →BaPP`

P` →aPbP`| ε

注：只有最左边的P参加变换。

第二节 自上而下分析法的一般问题
三、不带回溯的自上而下分析算法

1、消除左递归

- 1)什么是左递归
  - 左递归：文法存在产生式$P \overset{+}{\rightarrow} Pa$
  - 直接左递归： $P \rightarrow Pa$
  - 间接左递归： $P \rightarrow Aa$, $A \overset{+}{\rightarrow} Pb$
- 2)消除左递归
  - 消除直接左递归
  - 消除间接左递归

第二节 自上而下分析法的一般问题

三、不带回溯的自上而下分析算法

3、消除左递归算法

1）把文法G的所有非终结符按任意顺序排列成 $P_1,P_2,\ldots,P_n$,然后按此顺序执行步骤2。

2） For (I=1,I<=n,I++)

{for (k=1,k<=I-1,k++)

{把形如$P_i \rightarrow P_k\gamma$的规则改写为

$P_i \rightarrow \delta_1\gamma|\delta_2\gamma|\ldots\ldots|\delta_n\gamma$

/* 其中$P_k \rightarrow \delta_1|\delta_2|\ldots\ldots|\delta_n$*/ }

消除P_i规则的直接左递归；}

3)删去从文法开始符号不可达的非终结符产生式。

第二节 自上而下分析法的一般问题
三、不带回溯的自上而下分析算法

3、消除左递归算法

注：1)此算法适用于消除不含形如P →P的产生式，也不含以ε为右部的产生式的文法。

2)这里第二步所做的工作是：

a)若产生式出现直接左递归则用消除直接左递归的方法消除掉。

b)若产生式右部最左符号是非终结符且其序号大于左部的非终结符，则不处理。

c)若序号小于左部的非终结符，则将这序号小的非终结符用其右部串来取代，然后消除新的直接左递归。

例：对下面文法消除左递归：

(1) $S \rightarrow Qc|c$ (2)$Q \rightarrow Rb|b$ (3) $R \rightarrow Sa|a$

解：1）对非终结符重新排序：R、Q、S

2）对R：S的序号大于R的序号，不处理。

对Q: R的序号1小于Q的序号2,所以改写

$Q \rightarrow Rb|b$，将$R \rightarrow Sa|a$的右部取代R，得到：

$Q \rightarrow Sab|ab|b$,记为：(2`)式； 此时， S的序号大于Q的序号,不再处理。

对S： Q的序号2小于S的序号3,所以改写

$S \rightarrow Qc|c$，将$Q \rightarrow Sab|ab|b$的右部取代Q，得到

$S \rightarrow Sabc|abc|bc|c$;出现直接左递归，变换为：

$S \rightarrow (abc|bc|c)S`$

$S` \rightarrow abcS`| \varepsilon$

3）由于 $R \rightarrow Sa|a$和 $Q \rightarrow Sab|ab|b$中的R,Q对开始符号S来说都是不可达非终结符，所以删除它们。

故最后消除左递归后文法为：

$$S \rightarrow (abc|bc|c)S`$$

$$S` \rightarrow abcS`| \varepsilon$$

注：1）若非终结符排列顺序不同，改写后的文法也不同，但它们是等价的。

2）开始符号不能改变。

第二节 自上而下分析法的一般问题
三、不带回溯的自上而下分析算法

4、消除回溯

1)产生回溯的原因

进行推导时，若产生式存在多个候选式，选择哪个候选式进行推导存在不确定性。

2)消除回溯的基本原则

对文法的任何非终结符，若能根据当前读头下的符号，准确的选择一个候选式进行推导，那么回溯就可以消除。

注：之所以会产生回溯是因为在推导匹配的过程中存在虚假匹配。

第二节 自上而下分析法的一般问题
三、不带回溯的自上而下分析算法

4、消除回溯

3)消除回溯的方法

预测与提左因子

4)预测

根据读头下符号选择候选式，使其第一个符号与读头下符号相同，或该候选式可推导出的第一个符号与读头下符号相同。这相当于向前看了一个符号，所以称为预测。

注：使用了预测之后，选择候选式不再是盲目的了，所以也就无需回溯。

第三节 自上而下分析法的一般问题
三、不带回溯的自上而下分析算法

4、消除回溯

5)求候选式的终结首符集

令G是一个不含左递归的文法，对G的所有非终结符的各候选式α可求出它的终结首符First(α).

$First(\alpha)=\{a|\ \alpha \overset{*}{\rightarrow} a\ldots\ldots, a \in V_T\}$

特例:若$\alpha \overset{*}{\rightarrow} \varepsilon$,那么$\varepsilon \in First(\alpha)$。

即， First(α)集合包括了α的所有可能推导的开头终结符或可能的ε。

第二节 自上而下分析法的一般问题
三、不带回溯的自上而下分析算法

4、消除回溯

5)求候选式的终结首符集

设A是非终结符，且$A\rightarrow\alpha|\beta$，当A出现在下推栈的栈顶，且输入符号为a时，应如何选择A的候选式进行推导？这里分为四种情况：

(a)若$a\in First(\alpha)$,而$a\notin First(\beta)$，则选$A\rightarrow\alpha$

(b)若$a \notin First(\alpha)$,而$a \in First(\beta)$，则选$A\rightarrow\beta$

(c) 若$a \notin First(\alpha)$,且$a \notin First(\beta)$,则表示输入有错

(d)若$a\in First(\alpha)$,且$a \in First(\beta)$,则表示终结首符集相交，需改写文法，进行公因子提取。

注：对$First(\alpha)=\{\varepsilon\}$情况，留待讨论。

第二节 自上而下分析法的一般问题
三、不带回溯的自上而下分析算法

4、消除回溯

6)采用预测方法后PDA的运行

- 设A为出现在栈顶的非终结符，如果A的所有候选式的候选首符集都两两不相交，那么，PDA可以根据它所面临的读头符号，准确地指派某候选式替换A;
- 但是，许多文法都存在一些非终结符，其所有候选的首符集并不是两两不相交，这样，仍无法选择用哪个候选式进行推导。

第二节 自上而下分析法的一般问题

三、不带回溯的自上而下分析算法

4、消除回溯

7)提取公共左因子

将某产生式$A\rightarrow\delta\beta_1|\ \delta\beta_2|\ldots.|\ \delta\beta_n|\ \delta$

改写为 $A\rightarrow\delta A`$

$A`\rightarrow\beta_1|\beta_2|\ldots.|\beta_n|\varepsilon$

注：1)通过反复提取左因子，就能把所有非终结符的所有候选首符集变为两两不相交。

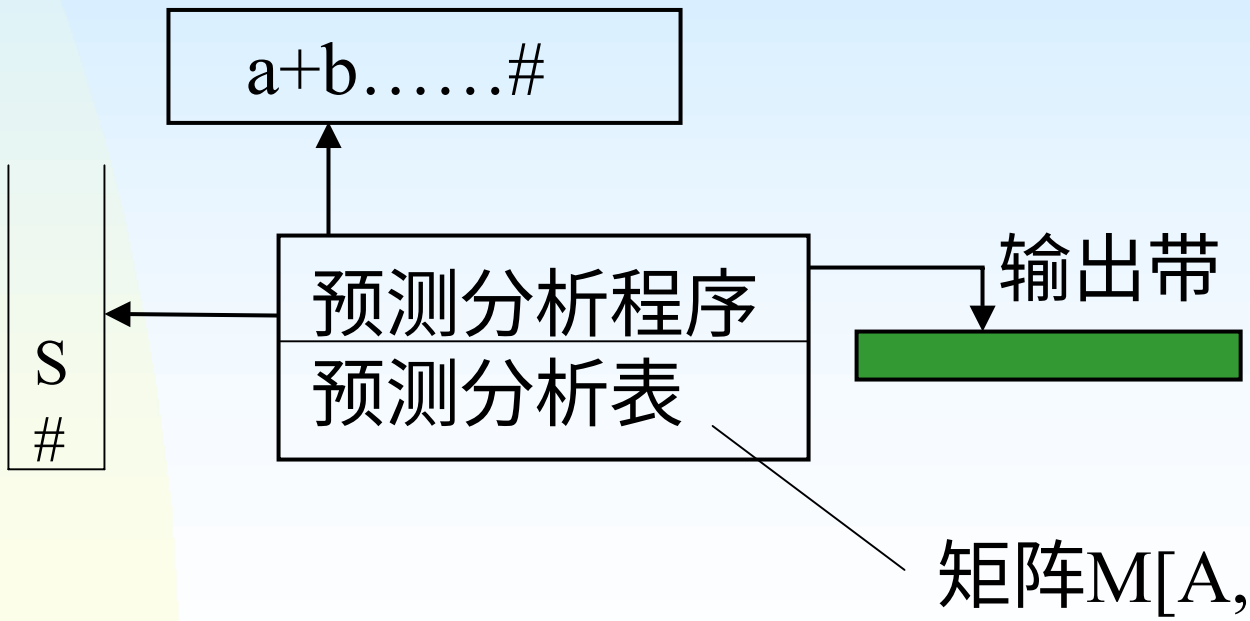
2)反复提取左因子也有一定代价，因为在提取过程中会大量引入非终结符和ε产生式，增加语法分析的复杂性。

第三节 预测分析程序与LL(1)文法

一、预测分析程序

1、带预测分析的PDA

在PDA中加入预测分析之后，可以消除自上而下分析中出现回溯的现象。此时PDA可以改造为：



第三节 预测分析程序与LL(1)文法

一、预测分析程序

2、带预测分析的PDA

注：1）改造后，整个分析过程都在预测分析程序控制下工作。

2）预测分析程序用了一个预测分析表，它是预测分析程序分析时的主要依据。

第四节 预测分析程序与LL(1)文法

一、预测分析程序

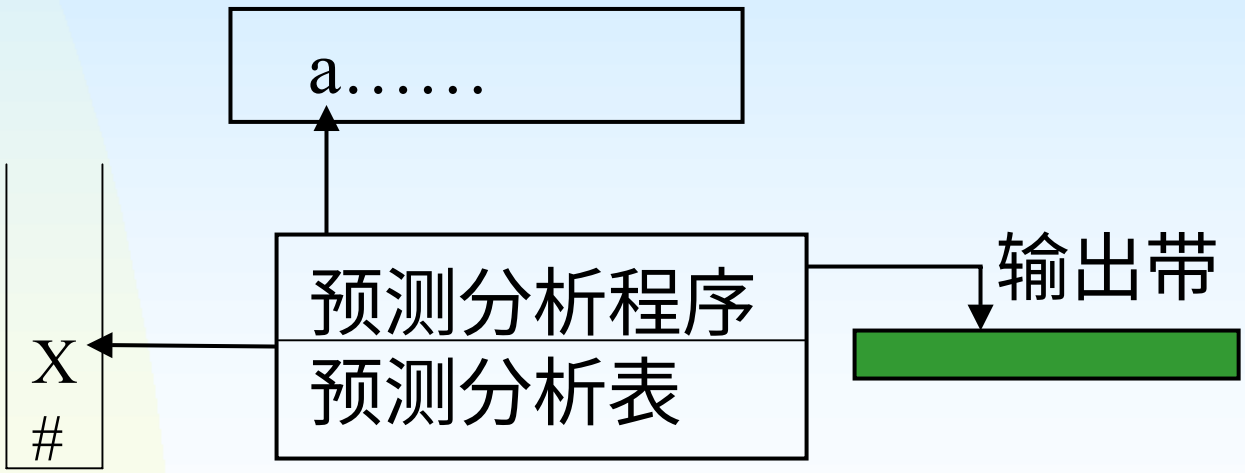
2、预测分析表

预测分析表是一矩阵M[A,a]，其中行标A是非终结符，列标a是终结符或串结束符；矩阵元素M[A,a]是存放A的一个侯选式，指出当前栈顶符号为A且面临读入符号为a时应选的候选式；或者存放“出错标志”，指出A不该面临读入符号a。

第三节 预测分析程序与LL(1)文法

一、预测分析程序

3、预测分析程序算法描述



第三节 预测分析程序与LL(1)文法

一、预测分析程序

3、预测分析程序算法描述

设栈顶符号为X，读入符号为a，则

1)若X=a='#'，则表示识别成功，退出分析程序；

2)若X=a≠'#'，则表示匹配，弹出栈顶符号X，读头前进一格，让读头指向下一个符号，以读入下一个符号；若X是终结符，但X≠a，则调用error处理；

3)若$X \in V_N$，则查预测分析表M。若M[X,a]中存放着关于X的产生式，则弹出X，且将相应产生式右部以自右向左的顺序压入栈，在输出带上记下产生式编号；若M[X,a]中存放着出错标记，则调用相应Error处理。

第三节 预测分析程序与LL(1)文法

二、求串α的终结首符集和非终结符的随符集

1、求串α的终结首符集First(α)

a)定义

假定α是文法G的一个符号串，$\alpha \in V^*$，则

$$First(\alpha)=\{a \mid \alpha \overset{*}{\rightarrow} a\ldots\ldots, a \in V_T\}$$

注：1)若$\alpha \overset{*}{\rightarrow} \varepsilon$,那么$\varepsilon \in First(\alpha)$。

2)First(α)集合是α的所有可能推导出的开头终结符或ε所组成的集合。

b)算法

设$\alpha=X_1X_2\ldots X_n$，其中$X_i \in (V_N \cup V_T), 1 \le i \le n$,为了求α的首符集，分两步：首先求$X_i$的首符集，然后再求α的首符集。

第三节 预测分析程序与LL(1)文法

二、求串α的终结首符集和非终结符的随符集

1、求串α的终结首符集First(α)

b)算法

1）求出文法中每个文法符号的首符集；

(1)若$x \in V_T$，则First(x)={x};

(2)若$X \in V_N$，且有产生式$X \rightarrow a\ldots\ldots$，则将a加到First(X)中;若$X \rightarrow \varepsilon$, 则ε也加入到 First(X)；

第三节 预测分析程序与LL(1)文法

二、求串α的终结首符集和非终结符的随符集

1、求串α的终结首符集First(α)

b)算法

1）求出文法中每个文法符号的首符集；

(3)若$X \to Y_1Y_2\ldots Y_k$，其中$Y_j \in (V_N \cup V_T), 1 \le j \le k$,则按如下算法求First(X)

j=0; FIRST(X)={}; //初始化

REPEAT j=j+1;

FIRST(X)=FIRST(X)∪(FIRST(Y_j)-{ε})

UNTIL ε∉ FIRST(Y_j) 或j=k

IF (j=k 且ε∈ FIRST(Y_k))

THEN FIRST(X)=FIRST(X) ∪ {ε}

第三节 预测分析程序与LL(1)文法

二、求串α的终结首符集和非终结符A的随符集

1、求串α的终结首符集First(α)

b)算法

2）求First(α)

设$\alpha = X_1X_2\ldots X_n$，其中$X_i \in (V_N \cup V_T), 1 \le i \le n$,则按如下算法求First(α)

i=0; FIRST(α)={}; //初始化

REPEAT i=i+1;

FIRST(α)=FIRST(α)∪(FIRST(X_i)-{ε})

UNTIL ε∉ FIRST(X_i) 或i=n

IF (i=n 且ε∈ FIRST(X_n))

THEN FIRST(α)=FIRST(α)∪{ε}

第三节 预测分析程序与LL(1)文法

二、求串α的终结首符集和非终结符A的随符集

2、求非终结符A的随符集Follow(A)

a)定义

假定S是文法G的开始符号，对于G的任何非终结符A，定义

$$Follow(A)=\{a|S \overset{+}{\rightarrow}\ldots Aa\ldots, a \in V_T\}$$

注：1)若 $S \overset{+}{\rightarrow}\ldots A$,则规定：$\# \in Follow(A)$。

2)Follow(A)集合是指在所有句型中紧跟A之后的终结符或#所组成的集合。

第三节 预测分析程序与LL(1)文法

二、求串α的终结首符集和非终结符A的随符集

2、求非终结符A的随符集Follow(A)

b)算法

1)对文法开始符号S，将‘#’加入到Follow(S)中；

2)若B →αAβ是文法G的一个产生式，则将First(β)-ε加入到Follow(A)中;

3)若B →αA是文法G的一个产生式,或B →αAβ是文法G的一个产生式,且β $\overset{+}{\rightarrow}$ε，则将Follow(B)加入到Follow(A)中；

注：这里的文法必须是消除了左递归且提取了左因子后的文法。

第三节 预测分析程序与LL(1)文法

二、求串α的终结首符集和非终结符A的随符集

例：对如下文法G(已加上编号)：

1.E → TE`　　2.E` → +TE`　　3.E` →ε

4.T → FT`　　5.T` → *FT`　　6.T` →ε

7.F → i　　8.F →(E)

求各非终结符号的终结首符集和随符集。

解：First(E)=First(T)=First(F)={(, i}

First(E`)={+, ε}

First(T`)={*, ε}

Follow(E)= Follow(E`)={),#}

Follow(T)= Follow(T`)={+,),#}

Follow(F)={*,+,),#}

第三节 预测分析程序与LL(1)文法

三、构造预测分析表

1、基本思想

1）若$A \to \alpha$ 是一个产生式，$a \in First(\alpha)$,那么当A是栈顶符号且将读入a时，选择α取代A匹配成功的希望最大。故，M[A,a]元素为$A \to \alpha$ 。

2）若$A \to \alpha$ ，而$\alpha = \varepsilon$ ，或$\alpha \overset{+}{\to} \varepsilon$ ；当A是栈顶符号且将读入a时，若$a \in Follow(A)$,则栈顶的A应被ε匹配此时读头不前进，让A的随符与读头下的符号进行匹配，这样输入串匹配成功的可能最大。故M[A,a]元素为$A \to \alpha$ (这里$\alpha = \varepsilon$或$\alpha \overset{+}{\to} \varepsilon$)。

第三节 预测分析程序与LL(1)文法

三、构造预测分析表

2、构造算法

1）假定A $\rightarrow\alpha$是一个产生式，a$\in$ First(α),那么当A是栈顶符号且将读入a时， A $\rightarrow\alpha$就应作为选用的侯选式，A $\rightarrow\alpha$应填入 M[A,a]中。

2）若A $\rightarrow\alpha$，而 $\varepsilon \in$ First(α) ,对每个a$\in$ Follow(A),在M[A,a]元素中应填A $\rightarrow\alpha$(一般填A $\rightarrow \varepsilon$)。

3）把所有无定义的M[A,a]都填上出错标志。

注：1）用此算法可以为任意文法G构造其分析表M。

2）若是二义文法或没有消除左递归和提取左因子的文法，构造出的M包含有重定义项。即，它们的M[A,a]中填有一个以上的产生式。

例：对如下文法G，构造预测分析表

1.E → TE`　　2.E` → +TE`　　3.E` → ε

4.T → FT`　　5.T` → *FT`　　6.T` → ε

7.F → i　　8.F →(E)

解：1）求出各非终结符的First集和Follow集

First(E)=First(T)=First(F)={(, i}

First(E`)={+, ε}

First(T`)={*, ε}

Follow(E)= Follow(E`)={),#}

Follow(T)= Follow(T`)={+,),#}

Follow(F)={*,+,),#}

2）根据算法构造预测分析表

	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)		

第三节 预测分析程序与LL(1)文法

四、LL(1)文法

1、定义

若文法G的预测分析表M中不含有多重定义项，则称G为LL (1)文法。

注：1)LL(1)文法是无二义的，二义文法一定不是LL(1)文法。

2) LL的含义是从左到右扫描输入串，采用最左推导分析句子。

3)数字1表示分析句子时需向前看一个输入符号。

4)有LL(1)就有 LL(k)，LL(k)向前查看k个输入符号，选择候选式更加准确，但M的尺寸会以n^k增长，其中$n=|\Sigma|+1$。对程序设计语言取k＝1就够了。

第三节 预测分析程序与LL(1)文法

四、LL(1)文法

2、证明定理

文法G是LL(1)文法当且仅当对于G的每个非终结符A的任何两个不同产生式$A \rightarrow \alpha|\beta$有：

1）$First(\alpha) \cap First(\beta)=\Phi$

2）若$\varepsilon \in First(\beta)$，则$First(\alpha) \cap Follow(A)= \Phi$

注：1）可以使用这个定理直接根据首符集、随符集来判断文法是否是LL(1)。但判断之前，必须消除左递归和提取公共左因子,因为包含左递归和公共左因子的文法肯定不是LL(1)文法。

2）LL(1)文法只是上下文无关文法的一个子集。

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，消除文法左递归和提取公共左因子；文法改写为：

$S \rightarrow$ if E then S S`|other

$S` \rightarrow$ else S| ε

$E \rightarrow b$

第2步求首符集和随符集

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

First(E)={b}

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

Follow(E)={then}

第3步：根据定理判定文法是不是LL(1)文法。

因为：1) First(if E then S S`) ∩ First(other)= Φ

First(else S) ∩First(ε)= Φ

但2) First(else S) ∩Follow(S`)={else}不为空集

故此文法不是LL(1)文法。

第三节 预测分析程序与LL(1)文法

五、状态表

1、为了节省存储空间并提高算法的效率，可以对预测分析表进行简化。不需把整个产生式放在分析表的各项中，只需要将右部候选式倒序存放其中，甚至在分析表中只保存产生式编号，产生式另存在一个语法表中。

注：1）将右部候选式倒序存放在分析表中是为了替换的时候可以边读边压栈。

2）一般情况下，分析表是稀疏矩阵，即使改为仅填写编号，空间浪费也很大，因此需要改进，引入状态表概念。

第三节 预测分析程序与LL(1)文法

五、状态表

2、将预测分析表改造为状态表

1）预测分析程序的功能：

- INITIAL：将“＃S”入栈；
- NEXT：读入下一个符号；
- PROCESS：按M[X,a]所规定的动作行事。

2） M[X,a]所规定的动作

- 根据当前状态（下推栈栈顶符号X）和输入符号a，执行语义子程序，这时会引起状态变迁，状态变化后程序要转去执行下一步动作。

第三节 预测分析程序与LL(1)文法

五、状态表

2、将预测分析表改造为状态表

3）状态表组成：

- 状态（栈顶符X）
- 输入符号a
- 语义子程序
- 状态变迁（栈顶符号如何变化）
- 下一步动作

第三节 预测分析程序与LL(1)文法

五、状态表

2、将预测分析表改造为状态表

3）状态表组成：

注：a)状态变迁可能涉及到推导和匹配，若相应候选式$Y_1Y_2\ldots Y_N$，Y_1是非终结符，那么这仅是推导的过程，改变栈顶符号后，下一步仍执行PROCESS。

b)若相应候选式$Y_1Y_2\ldots Y_N$，Y_1是终结符a，那么先执行匹配操作，即入栈的是$Y_2\ldots Y_N$，此时需要读入新的符号，下一步就转到NEXT；若Y_1是终结符，但不等于a，或者该项为空，则认为出错，调用ERROR；若相应候选式为ε，则在该项中填写“=>”，动作是栈顶符号出栈，下一步转到PROCESS.

第三节 预测分析程序与LL(1)文法

五、状态表

2、将预测分析表改造为状态表

3）状态表组成：

注：c) “下一步动作”指：NEXT, PROCESS, ERROR, RETURN四个动作。

d)状态表的使用仅仅是数据结构上的改变，没有改变LL(1)分析方法。使用状态表使分析表紧凑，总控程序也不复杂。

第四节 递归下降分析法

一、递归下降分析程序

若一个文法G不含有左递归，而且每个非终结符的所有候选式的首符集都是两两不相交的，那么就能为G中每个非终结符编写一个相应的递归过程。把该文法中所有这样的递归过程组合起来就可能构成一个不带回溯的自上而下分析程序——递归下降分析程序。

第四节　递归下降分析法

二、递归下降分析法实现思想

为文法中每个非终结符编写一个递归过程，每个过程的功能是识别由该非终结符推出的串，当某非终结符的产生式有多个候选式时，按LL(1)形式唯一确定选择哪个候选式进行推导，若遇到某候选式为ε，认为其自动匹配。把这些递归过程组合起来就构成了文法的递归下降分析程序。

注：绝大多数程序设计语言可用CFG来描述，CFG的特点在于其递归性，因此适合使用递归下降程序来分析。

第五节　递归下降分析法

三、递归下降分析法的实现

1、使用LL(1)文法

先将文法消除左递归、提取公共左因子，使之成为LL(1)文法，后将每个非终结符对应一个递归过程，过程体是按照相应产生式的右部符号串顺序编写。每匹配一个终结符，则再读入下一个符号，对产生式右部的每个非终结符，则调用相应的过程。

注：使用递归实际上与下推栈的原理相同。

2、使用BNF范式

先将文法改写为BNF形式，后再书写递归子程序。

第四节　递归下降分析法

四、递归下降分析法的缺点

1）对文法的要求高，必须满足LL(1)文法。

2）高深度的递归调用会影响语法分析的效率，速度慢，占空间多。

小结

1、消除左递归
2、构造LL(1)分析表
First、Follow
3、递归下降分析子程序
使用扩展BNF表示