第5章 自上而下语法分析

编译原理　陈炬桦

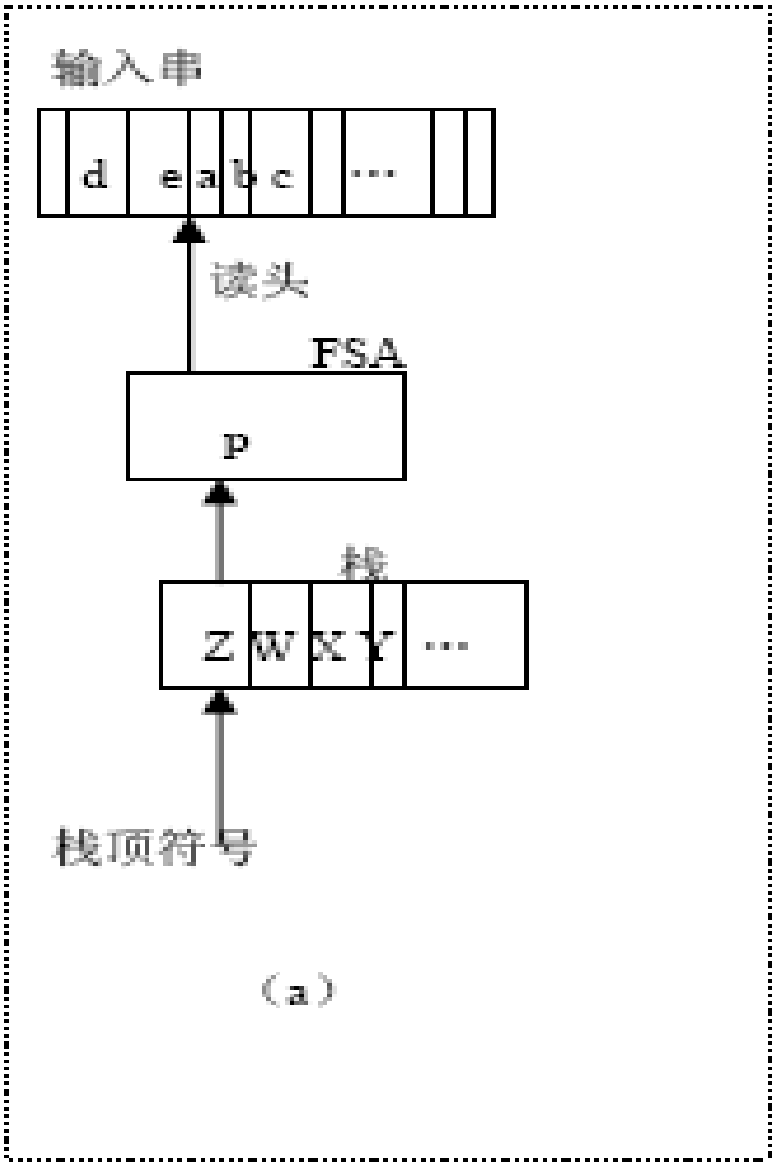
isscjh@mail.sysu.edu.cn

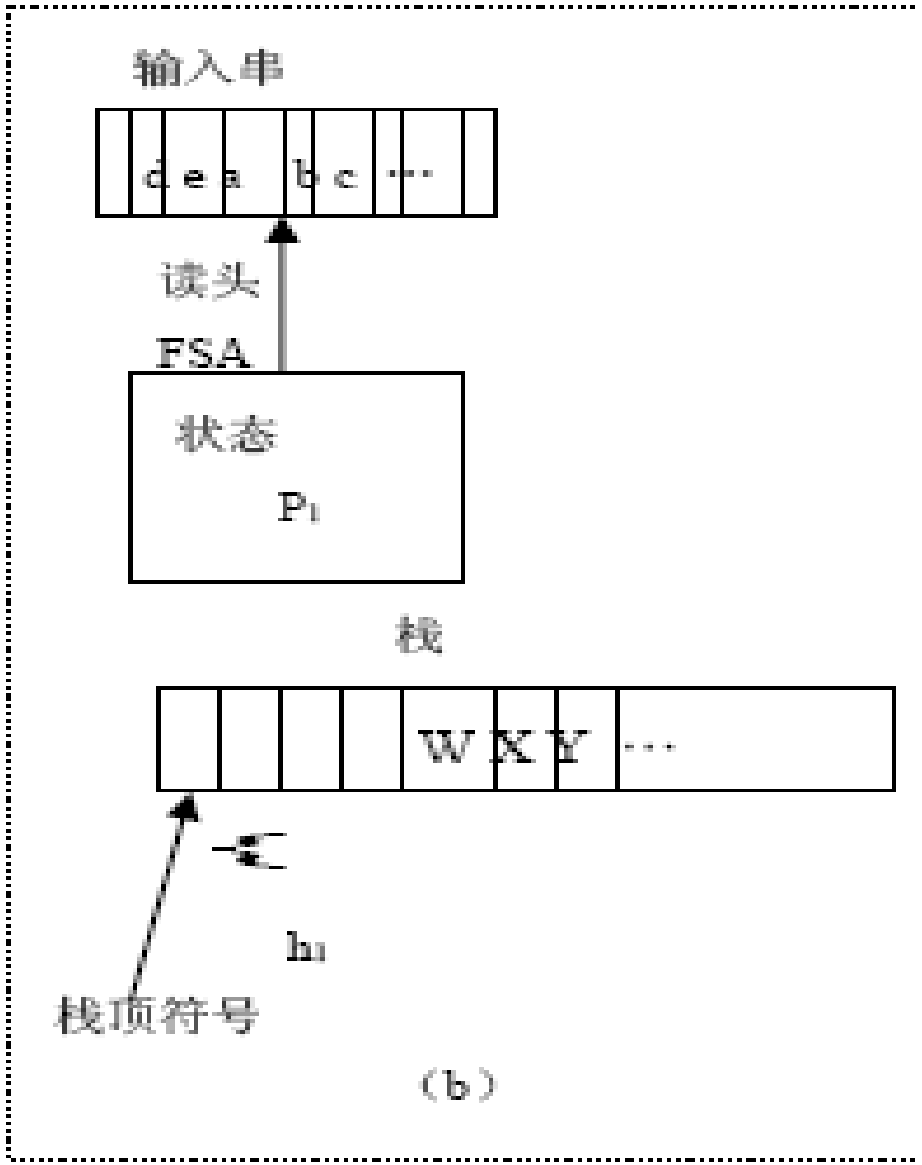
- 语法分析是继词法分析之后编译过程的第2阶段。它的主要任务是对词法分析的输出结果——单词序列进行分析，识别合法的语法单位。
- 语法分析最常用的方法有：优先方法、递归下降法、LL方法和LR方法。
- 所谓自上而下分析是指从树的根结开始朝着句子向下进行分析、构造语法树的。
- 本章中,我们通过讨论一个一般的非确定的自上而下分析器来讨论上下无关语言的自上而下分析器的设计。

5.1 非确定的下推自动机

- 下面所要构造的非确定的自上而下分析器属于一般的下推自动机（PDA）类。
- 所谓一个下推或栈自动机（Stack Automaton）,非形式地说，应包含：
  - ①一个输入符号串；
  - ②一个读头，它从左至右移动，每次读进一个输入符号；
  - ③一个有穷状态自动机，用于控制整个系统的操作；
  - ④一个后进先出下推栈，

下推自动机的非空移动:


输入串
d e a b c …
读头
FSA
p
栈
Z W X Y …
栈顶符号
（a）


输入串
d e a b c …
读头
FSA
状态
P_1
栈
W X Y …
h_1
栈顶符号
（b）

5.1.1 PDA形式定义

形式上说，一个PDA是一个七元组：

$$(Q,\sum,H,\delta,q_0,Z_0,F)$$

- 其中，Q 是状态的有穷集，它的每个元素称为一个状态；
- ∑ 是有穷的字母表，它的每个元素是一个输入符号；
- H 是有穷的下推栈字符表，它的每个元素称为一个栈符号。
- $q_0 \in Q$ 是该PDA的初态；
- $Z_0 \in H$ 是下推栈的初始符号；
- $F \subset Q$ 是一个终态集(或接收状态集)；它的每个元素称为终态；(可空)。
- δ 是一个转换函数，它将三元组(q,a,Z)映象成对偶集

{（p1,h1）,(p2,h2),…}，即，δ(q,a,Z)= {（p1,h1）,(p2,h2),…} .

5.1.3 上下文无关语言与PDA

联系PDA和上下文无关语言的一个重要定理是：

- 定理5.1 对每一个上下文无关语言L，存在一个恰好识别L的非确定的PDA M，反之亦然。
- 这个定理在编译系统中的实际重要性在于：现有的大多数高级程序设计语言都可用上下文无关文法描述。因此定理5.1隐含了：识别这个语言的机械识别器必是PDA。
- 定理5.1包含两方面含义：
  - 给定一个上下文无关语言，存在一个识别它的PDA M；
  - 反过来，给定一个PDA M，可以根据它构造出一个等价的上下文无关文法。

    前者对编译程序的构造很有吸引力，但后者则不然。

算法5.1 从CFG到NDPDA

- 给定 CFG G=（N，∑，P，S）

  可以构造 一个相应的非确定的PDA M：

  $M=(Q,\sum',H,\delta,q_0,Z_0,F)$

  它只有一个状态q和下面的转换规则：
- ① 对P中每一个形如$A\rightarrow w$的产生式，$\delta(q,\varepsilon,A)$包含(q,w);
- ② 对每个$a\in\sum$，$\delta(p,a,a)$包含(q,ε) 且
  - $Q=\{q\}$
  - $\sum'=\sum$
  - $H=N\cup\sum$
  - $q_0=q$
  - $Z_0=S$
  - F为终态集(可空)。

  这个PDA停止于空栈。

例5.2 考虑文法

$$S \rightarrow 0S1|c$$

该文法描述语言0*c1*,其中0的个数和1的个数相等。转换规则是：

- 1.$\delta(q,0,0)=(q,\varepsilon)$
- 2.$\delta(q,1,1)=(q,\varepsilon)$
- 3.$\delta(q,c,c)=(q,\varepsilon)$
- 4.$\delta(q,\varepsilon,S)=\{(q,0S1),(q,c)\}$(其中ε可与任何合法输入符号匹配)

给定输入串00c11，所构造的PDA用下面的移动序列来接收它(注意，我们可从构形中省掉状态，因为它总是相同的)：

$(q,00c11,S) \vdash_{4a} (q,00c11,0S1) \vdash_{1} (q,0c11,S1)$
$\vdash_{4a} (q,0c11,0S11) \vdash_{1} (q,c11,S11)$
$\vdash_{4b} (q,c11,c11) \vdash_{3} (q,11,11)$
$\vdash_{2} (q,1,1) \vdash_{2} (q,\varepsilon,\varepsilon)$ (接收)

5.2 消除左递归方法

5.2.1 文法的左递归性

- 文法的左递归性属文法递归性的一种，在一文法中，所有形如
$$A \rightarrow xAy \quad x,y \in (\sum \cup N)^*, \quad A \in N$$
称为递归产生式(或自嵌入产生式)。
- 若其中$x=\varepsilon$，则有
$$A \rightarrow Ay$$
称之为直接左递归产生式。
- 若其中$y=\varepsilon$，则有
$$A \rightarrow xA$$
称之为直接右递归产生式。
- 若一文法中至少含有一条递归产生式，或在用该文法推导符号串的过程中，存在$A \Rightarrow A\ldots$或$A \Rightarrow \ldots A$或$A \Rightarrow \ldots A\ldots$形式的推导，则称该文法是(直接)递归的。

- 为了避免无限循环，自上而下分析法的文法不应含有左递归。有则消除。
- **文法的左递归性**

  直接左递归：$U \rightarrow Ux \mid y$

  间接左递归：$U \overset{+}{\Rightarrow} Ux$
- 例：G22[A]：

  $A \rightarrow [B$

  $B \rightarrow X] \mid BA$

  $X \rightarrow Xa \mid Xb \mid a \mid b$

正规表达式的形式定义

- **正规表达式** 的文法定义G[E]:

  E→E+T | T ⇔ E →TE’, E’ →+T| ε

  T→TF | F

  F→P* | P ⇔ F →PF’,F’ →*| ε

  P→(E)| I

➢正规表达式为由选择，链接，重复的式子

✓式中 “+” 为选择“|”

✓ 式中 “.” 链接符省略

✓ 式中 I 为 a∈Σ 的符号

5.2.2 用扩展的BNF表示法消除左递归

在前面，文法的产生式都是采用巴科斯范式（BNF）描述的，它使得文法更严谨、简洁和清晰。为了消除文法的左递归，需对巴科斯范式进行扩展，增加以下元符号：

- ① 花括号{ }

  $\{x\}$：表示符号串x出现零次或多次。

  $\{x\}_m^n$：n表示符号串x能重复出现的最大次数，m表示符号串x能重复出现的最小次数。
- ② 方括号[ ]

  方括号用来表示可选项。[x] = x或ε，表示符号串x可出现一次或不出现。可以用来定义某些高级语言中的“条件语句”。
- ③ 圆括号（ ）

  利用圆括号可提出一个非终结符的多个产生式右部的公共因子。例如，

  A→xy|xw|…|xz 可写成 A→x(y|w|…|z)

利用下面的两条规则，可把包含直接左递归的产生式转换成用扩展BNF表示法表示的产生式。

- ① 提公因子

  每当一条产生式中有公因子可提的时候，就把它提出来，若原产生式是　　$A\rightarrow x|xy$

  则可写成 $A\rightarrow x(y|\varepsilon)$，这里把ε当作最后一个候选式。

- ② 若　$A\rightarrow x|y|\ldots|z|Av$

  是一组产生式，且它只有一个直接左递归的右部位于最后，则可把这组产生式变换成如下形式：

  $$A\rightarrow(x|y|\ldots|z)\{v\}$$

- 也就是说，使用上述规则①，可把产生式改写成相对于某个非终结符而言，至多只含一个直接左递归的右部；然后，利用上述规则②消除这个直接左递归。

5.2.3 直接改写法

- 设产生式

  $U \rightarrow Ux \mid y$

  此产生式称为直接左递归形式。其中，x和y是两个符号串，y的首字符不是U。
- 产生式为直接左递归形式，可直接改写为一个等价的非直接左递归形式

  $U \rightarrow yU'$

  $U' \rightarrow xU' \mid \varepsilon$
- 其中U′是新引进的非终结符号。显然，这种形式与原形式是等价的，即从A推出的符号串是相同的。
- 直接左递归更一般的形式

  $U \rightarrow Ux1 \mid Ux2 \mid \ldots \mid Uxm \mid y1 \mid y2 \mid \ldots \mid yn$

  其中，　xi ≠　　(i=1, 2, …, m), yi (i=1, 2, …, n)的头字符都不是U。

  $U \rightarrow y1U' \mid y2U' \mid \ldots \mid ynU'$

  $U' \rightarrow x1U' \mid x2U' \mid \ldots \mid xmU' \mid \varepsilon$

5.2.4 消除所有左递归的算法

若一个文法不含形如AA的推导，也不含有以ε为右部的产生式，那么，执行下面的算法将保证消除该文法中的所有左递归:

- ① 将文法G的所有非终结符整理成某一顺序U_1，U_2，…，U_n。
- ② for i:= 1 to n do
- begin
- for j:= 1 to i–1 do
- begin
- 把产生式$U_i \rightarrow U_j\alpha$替换成
- $U_i \rightarrow \beta_1\alpha \mid \beta_2\alpha \mid \ldots \mid \beta_m\alpha$
- (其中$U_j \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$ 1是该文法中关于Uj的所有产生式)
- end;
- 消除Ui产生式中的直接左递归
- end;
- ③ 化简改写之后的文法，删除多余产生式。

5.3 LL(k)文法

- LL(k)文法是上下文无关文法的一个真子集。同时，LL(k)文法也是允许采用确定的从左至右扫描(输入串)和自上而下分析技术的最大一类文法。
- LL系指：自左至右扫描(输入串)，自上而下进行最左推导。
- 给定一文法，判断它是否是一个LL(k)文法并不是很容易的事。我们将给出一个判断条件。根据这个条件就能推断一个给定的文法是否是LL(k)文法。此外，由于分析表是分析器的核心，因此，我们将给出构造LL(k)分析器的分析表的方法。

5.3.1 LL(1)文法的判断条件

- 按照前面所述方法改造，消除左递归后的文法不一定就是一个LL(1)文法，还必须借助于某种判断条件才能确定。为此，我们将引进三个集合：FIRST,FOLLOW和SELECT。
- 先定义集合FIRST和FOLLOW。
  - 设α是文法G的一个符号串，$\alpha \in (V_N \cup V_T)^*$，定义

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

    特别，若有$\alpha \overset{*}{\Rightarrow} \varepsilon$，则$\varepsilon \in FIRST(\alpha)$。即，FIRST(α)是从α可推导出的所有首终结符或可能的ε。
  - 设S是文法的识别符号，$U \in V_N$，定义

    $$FOLLOW(U) = \{b \mid S \overset{*}{\Rightarrow} xUby，b \in V_T，x, y \in (V_N \cup V_T)^*\}$$

    若$S \overset{*}{\Rightarrow} xU$，则规定“$”∈FOLLOW(U)。即，FOLLOW(U)是文法的所有句型中紧接在U之后出现的终结符或$($不是文法符号，而是一个特定的结束符)。

再来看看SELECT集合。

- 假定$U\rightarrow\alpha$是文法G的任一产生式，其中$U\in V_N$，$\alpha\in(V_N\cup V_T)^*$，集合SELECT($U\rightarrow\alpha$)的构造如下：

$$SELECT(U\rightarrow\alpha)=\begin{cases}FIRST(\alpha), & \text{当}\alpha\text{不可空}\\ FIRST(\alpha)\cup FOLLOW(U), & \text{否则}\end{cases}$$

- 利用上面的三个集合可建立LL(1)文法的判断条件如下：

  令G是一个CFG，当且仅当对于G中每个具有相同左部的产生式(即一个非终结符的所有产生式)

  $$U\rightarrow\alpha_1|\alpha_1|\ldots|\alpha_n$$

  都有 SELECT ($U\rightarrow\alpha_i$)$\cap$SELECT($U\rightarrow\alpha_j$)=$\varnothing$ ($i\neq j$, i, j=1, 2, …, n)，则文法G是一个LL(1)文法。

5.3.2 集合FIRST、FOLLOW与SELECT的构造

1. 设$X \in (V_N \cup V_T)$，FIRST(X)的构造

- ① 若$X \in V_T$，则FIRST(X)={X}；
- ② 若$X \in V_N$，它的产生式为$X \to a\ldots$，$a \in V_T$，则$a \in FIRST(X)$；若它有产生式$X \to \varepsilon$，则$\varepsilon \in FIRST(X)$；
- ③ 如果它有产生式$X \to Y\ldots$，$Y \in V_N$，则$FIRST(Y)\backslash\{\varepsilon\} \subset FIRST(X)$；如果它有产生式$X \to Y_1Y_2\ldots Y_k$（其中，$Y_1$，$Y_2$，…，$Y_{i-1}$都是非终结符，且$Y_1Y_2\ldots Y_{i-1} \overset{*}{\Rightarrow} \varepsilon$），则$FIRST(Y_i)\backslash\{\varepsilon\} \subset FIRST(X)$；如果$Y_1Y_2\ldots Y_k \overset{*}{\Rightarrow} \varepsilon$，则$\varepsilon \in FIRST(X)$。

2. 设$\alpha\in(V_N\cup V_T)^*$，$\alpha=X_1X_2\ldots X_n$，FIRST(α)的构造

- ① 若$\alpha=\varepsilon$，显然FIRST(α)=$\{\varepsilon\}$;
- ② 若$\alpha\neq\varepsilon$，则FIRST(X_1)\$\{\varepsilon\}\subset$FIRST($\alpha$);
- ③ 若$X_1X_2\ldots X_{i-1}\overset{*}{\Rightarrow}\varepsilon$，则FIRST($X_i$)–$\{\varepsilon\}\subset$FIRST($\alpha$);

  若$X_1X_2\ldots X_n\overset{*}{\Rightarrow}\varepsilon$，则$\varepsilon\in$FIRST($\alpha$)。

3. 设U $\in V_N$，FOLLOW(U)的构造

- ① 若U是文法的开始(识别)符号，则$\$\in$FOLLOW(U);
- ② 若有产生式A→xUy，则FIRST(y)–{ε}⊂FOLLOW(U);
- ③ 若有产生式A→xU，或A→xUy，y $\overset{*}{\Rightarrow}$ ε，则FOLLOW(A)⊂ FOLLOW(U)。

例5.9 设文法G_{32} [S]:

- S→ A
- A→ BA′
- A′ → iBA′ | ε
- B→ CB′
- B′ → +CB′ | ε
- C →) A*| (

- ∵ S是识别符号，且未出现在任何产生式右部，∴FOLLOW(S)={$}。
- ∵ S→ A , ∴$∈FOLLOW(A),
- ∵ C→)A* , ∴*∈FOLLOW(A),
- ∴ FOLLOW(A) = {*, $};
- ∵ A→ BA′, ∴FOLLOW(A′)=FOLLOW(A) = {*, $},
- ∵ A′→ iBA′|ε,
- ∴ FIRST(A′)\{ε} = {i, ε}\{ε}={i}⊂ FOLLOW(B),
- FOLLOW(A′) = {*,$}⊂FOLLOW(B),
- ∴ FOLLOW(B) = {i, *, $};
- ∵ B→ CB′,
- ∴ FOLLOW(B′)=FOLLOW(B)={i, *, $},
- ∵ SELECT (A′→iBA′)∩SELECT(A′→ε)= FIRST(iBA′)∩(FIRST(ε)∪FOLLOW(A′))= {i}∩{*, $, ε}=∅,
- SELECT (B′→ +CB′)∩SELECT(B′→ε) = FIRST(+CB′)∩(FIRST(ε)∪FOLLOW(B′))= {+}∩{i, *, $, ε}=∅,
- SELECT (C→)A*)∩SELECT(C→ ( )= FIRST( )A*)∩FIRST (( )= { } }∩{ (}=∅。
- ∴ 此文法G是一个LL(1)文法。

需要注意的是，对于LL(1)文法，利用SELECT集，我们还可以得到十分有助于句子分析的结果：

- 在分析过程中的某一步，当非终结符U正处于栈顶时，若a∈SELECT (U→α)是当前的输入符号，则可立即选用产生式U→α；若a∈SELECT (U→β)是当前的输入符号，则可选用产生式U→β。
- 由于LL(1)文法的任何一对具有相同左部的产生式的SELECT集是不相交的，因此，根据现行的非终结符(已在栈顶)和当前的输入符号就能唯一地确定分析过程中的下一步动作，也就是说，我们完全可以为LL(1)文法构造一个确定的分析器。

5.4 确定的LL(1)分析器的构造

- 逻辑上说，一个LL(1)分析器由两大部分组成：一个总控算法和一张分析表。LL(1)分析器的总控算法基本上是一样的，只是分析表各不相同。由于总控程序比较容易实现，因此，我们常常把构造分析器的任务只看作是构造分析表。
- 分析表是一个M[U,a]形式的矩阵，其中U为非终结符，a为终结符或$。矩阵元素M[U,a]中可能存放关于U的产生式，表示当U面临输入符号a时所应选用的产生式；M[U,a]中也可能存放一个ERROR，表示此时输入串中存在语法错误。

5.4.1 构造分析表M的算法

对于构造分析表M的算法，我们可以有两种形式的描述，先来看看以FIRST和FOLLOW集合定义的算法：

- ① 对文法的每一条产生式U→α，若a∈FIRST(α)，则M[U, a]=′U→α′;
- ② 若ε∈FIRST(α)，则M[U, b]= ′U→α′，其中，b∈FOLLOW(U);
- ③ 分析表M的其它元素均为出错标志error，通常用空白表示。

如果用SELECT集合来定义算法，则有以下步骤：

- ① 对文法G的每个产生式U→α，计算SELECT（U→α）。
  - 若SELECT（U→α）=a,a∈∑,则置M[U,a]为产生式U→α;
  - 若SELECT（U→α）={a_1,a_2,…a_n}, a_1,a_2,…an都是终结符号或$或ε，则置M[U,$a_1$]=M[U,$a_2$]=…=M[U,$a_n$]产生式为U→α。
- ② 对所有尚未定义的M[U,a],置上ERROR（用空白表示）。

例5.11 对于例5.9中的文法G [S]，构造分析表。

- 对于产生式S→ A，
  ∵ FIRST(A)={(,)}，
  ∴ M[S, ( ] = ′S→ A′，M[S,)] = ′S→A′。
- 对于产生式A→BA′，
  ∵ FIRST(B)={(,)}，
  ∴ M[A, C] = ′A→ BA″，M[A,)] = ′ A→ BA″。
- 对于产生式A′→ iBA′，
  ∵ FIRST (iBA′) = {i}，
  ∴ M[A′, i]=′ A′→ iBA″。
- 对于产生式A′→ ε，
  ∵ FOLLOW(A′)={* , $}，
  ∴ M[A′, *] = ′A′→ ε′, M[A′, $]=′A′→ ε′。
- 对于产生式B→CB′，
  ∵ FIRST(C) = {(,)}，
  ∴ M[B, C] = ′B→ CB″，M[B,)] = ′B→CB″。
- 对于产生式B′→ +CB′，
  ∵ FIRST(+CB′) = {+}，
  ∴ M[B′, +] = ′ B′→ +CB″。
- 对于产生式B′→ ε，
  ∵ FOLLOW(B′) = {i, *, $}，
  ∴ M[B′, i] = ′ B′→ ε′，M[B′, *] = ′B′→ ε′，
  M[B′, $] = ′B′→ε′。
- 对于产生式C→)A*，
  ∵ FIRST ( )A*) = {) }，
  ∴ M[C,)] = ′C→)A*′。
- 对于产生式C→ (，
  ∵ FIRST ( )) = { (}，
  ∴ M[C, (] = ′C→ (′。

分析表M

$S \rightarrow A$

$A \rightarrow BA'$

$A' \rightarrow iBA' \mid \varepsilon$

$B \rightarrow CB'$

$B' \rightarrow +CB' \mid \varepsilon$

$C \rightarrow)A^* \mid ($

非终结符	终结符					
	I	+	*	(	)	$
S				S→A	S→A	
A				A→BA′	A→BA′	
A′	A′→iBA′		A′→ε			A′→ε
B				B→CB′	B→CB′	
B′	B′→ε	B′→+CB′	B′→ε			B′→ε
C				C→(	C→)A*	

5.4.2 LL(1)分析器的总控算法

LL(1)分析器就是带有LL(1)分析表的一个PDA，也称预测分析程序。下面给出LL(1)分析器的总控算法：

- ① 最初，分析器把文法的开始符号S置于栈顶(假定输入串以$结束)；
- ② 若栈顶为一终结符，而且与当前输入符号匹配，则读头前进一位置(扫描下一输入符号)，并逐出栈顶符号。否则报错。(匹配动作)
- ③ 若栈顶符号是一非终结符U，且当前的输入符号为a，则查看分析表M，若M[U,a]置有关于U的产生式U→w，则先从栈中逐出U再把w下推进栈；若w=ε，则不推进任何信息进栈，仅逐出栈顶符号；若M[U,a]为空白，则调用出错处理子程序。(应用动作)
- ④ 重复步骤②、③，直至栈变为空。
- ⑤ 该分析器停止于空栈。(接收)

非终结符	终结符					
	I	+	*	(	)	$
S				S→A	S→A	
A				A→BA′	A→BA′	
A′	A′→iBA′		A′→ε			A′→ε
B				B→CB′	B→CB′	
B′	B′→ε	B′→+CB′	B′→ε			B′→ ε
C				C→ (	C→)A*	

步骤	符号栈 S[i]	输入串 str[j]	产生式
1	$S	(i ($	S→ A
2	$A	(i ($	A→ BA′
3	$A′B	(i ($	B→ CB′
4	$A′B′C	(i ($	C→ (
5	$A′B′(	(i ($	
6	$A′B′	i ($	B′→ ε
7	$A′	i ($	A′→ iBA′
8	$A′Bi	i ($	
9	$A′B	($	B→ CB′
10	$A′B′C	($	C→ (
11	$A′B′(	($	
12	$A′B′	$	B′→ ε
13	$A′	$	A′→ ε
14	$	$	OK

5.5 LL(k)文法的几个结论

可以证明：

- ① LL(k)文法是不含左递归的；
- ② LL(k)文法是无二义性的；
- ③ 一文法G是LL(1)，当且仅当对于G的每一非终结符U的任何两个不同产生式$A\rightarrow\alpha|\beta$，下面的条件成立：
  - $FIRST(\alpha) \cap FIRST(\beta)= \varnothing$;
  - α,β中至多只有一个能推出空串ε;
  - 若$\beta \overset{*}{\Rightarrow} \varepsilon$，则$FIRST(\alpha)\cap FOLLOW(A)= \varnothing$。

5.6 递归下降分析程序及其设计

- 若一文法G满足下述两个条件：
  - G中不含有直接左递归；
  - G中的每个非终结符的所有候选式的FIRST集都是两两不相交的。
- 那么，我们就能为G中的每个非终结符编写一个相应的递归过程，把该文法中的所有这样一些递归过程组合起来就有可能构成一个不带回溯的自上而下分析程序。
- 这种分析程序称为递归下降分析程序(Recursive-descent Parser)，它是一种确定的自上而下分析方法,也是编译程序中使用得最为广泛的一类分析程序。可以证明，一个递归下降分析程序能正确工作的必要条件是它的源文法是LL(1)的。

下面，我们形式化描述递归子程序的设计方法。在定义子程序时用到一个全局变量ch，存放输入符号串的当前符号；还用到一个函数READ(ch)，从ch中读输入符号串的下一个符号。

- 设$a \in V_T$，P(a)代表语句
  if ch = a then READ(ch) else error
- 设$\alpha = x_1x_2\ldots x_n$，$x_i \in (V_N \cup V_T)$ (i = 1, 2, … , n)，P(α)代表复合语句
  begin P(x_1); P(x_2); … ; P(x_n) end
- 设$U \in V_N$，产生式 $U \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_m$，定义P(U)：

```
read(ch);
if ch∈FIRST(α1) then P(α1)
 else if ch∈FIRST(α2) then P(α2)
 ……
else if ch∈FIRST(αm) then P(αm)
   else error
```

- 如果非终结符U有空产生式$U \rightarrow \varepsilon$，则改写P(U)为

```
read(ch);
if ch∈FIRST(α1) then P(α1)
 else if ch∈FIRST(α2) then P(α2)
 ……
 else if ch∈FIRST(αm) then P(αm)
 else if ch∈FOLLOW(U) then return
 else error
```

5.6.1 框图设计

例5.15 设文法G [S]：

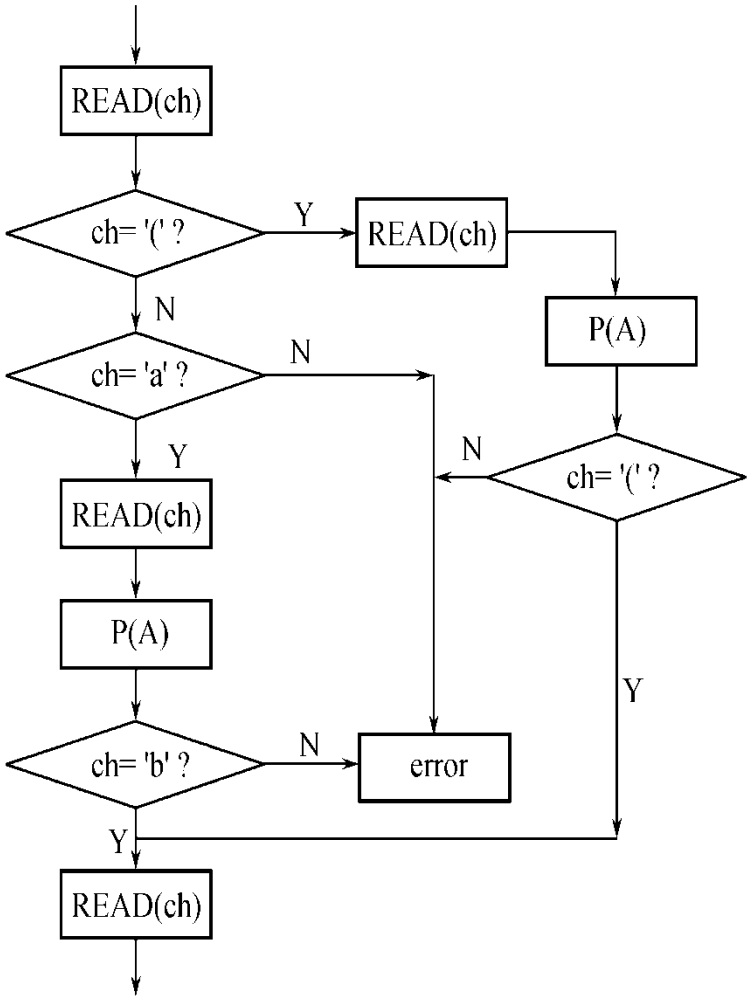
$S \rightarrow (A) \mid aAb$

$A \rightarrow eA' \mid dSA'$

$A' \rightarrow dA' \mid \varepsilon$

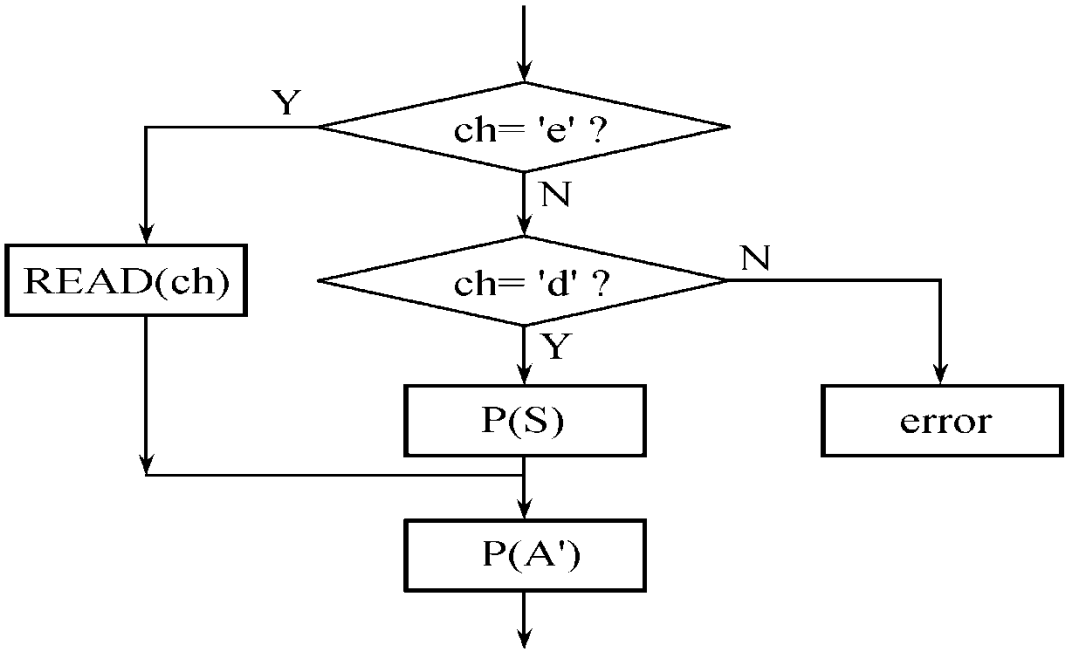
此文法的递归下降程序框图设计如下所示。图中省略了递归入口RI和递归出口RO部分，如果采用低级语言编程，只需要在程序框图的开头和末尾加上这两个部分。

P(S)：



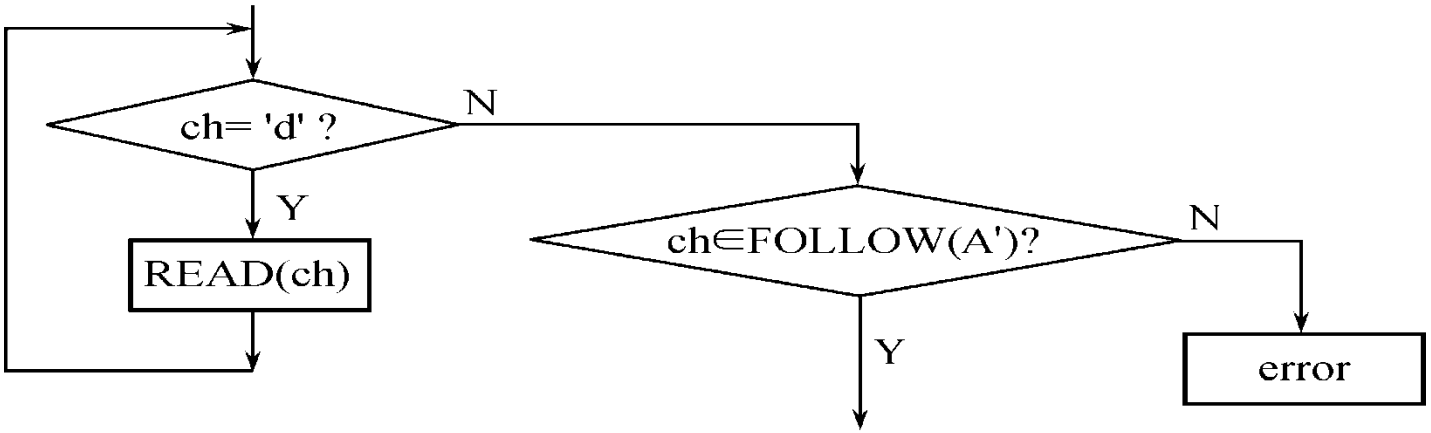
(a)

P(A)：


ch= 'e' ?
Y
N
READ(ch)
ch= 'd' ?
N
Y
P(S)
error
P(A')

(b)

P(A')：


ch= 'd' ?
N
Y
READ(ch)
ch∈FOLLOW(A')?
N
Y
error

(c)

5.6.2 程序设计

例5.16 例5.15中的文法G [S]：

- 子程序P(S)：
- READ(ch)
- if ch = ′(′ then
- begin
- READ(ch); P(A);
- if ch = ′)′ then goto L
- else error
- end
- else if ch ≠ ′a′ then error
- else begin
- READ(ch); P(A);
- if ch = ′b′ then goto L
- else error
- end
- L: READ(ch);
- return

子程序P(A)：

- if ch = ′e′ then
- begin
- READ(ch); goto L
- end
- if ch ≠ ′d′ then error
- P(S);
- L: P(A′);
- return

子程序P（A′）：

- L: if ch = ′d′ then
- begin
- read(ch); goto L
- end
- else if ch = ′b′ then goto L′
- else if ch = ′)′ then goto L′
- else error
- L′: return

5.7 带回溯的自上而下分析法

不带回溯的自上而下分析法，并不适用于所有的上下文无关文法，必须对文法加以限制；而带回溯的自上而下分析法，对上下文无关文法具有通用性，几乎没有限制，但速度慢是这种分析法的致命弱点。

5.7.1 文法在内存中的存放形式

- 带回溯的自上而下分析法，实际上就是穷举所有可能的推导，看是否能推导出待检查的符号串。在穷举所有可能的推导过程中，其唯一的依据是文法的产生式。
- 因此，一个文法若使用带回溯的自上而下分析法，则它的所有产生式都必须先保存在内存中，具体地说，是存放在一个一维数组中。
- 在这个一维数组中，存放每一条产生式的左部和右部，并且在每一个产生式右部的尾符号之后放置一个符号“|”，作为产生式右部的结束符；在一个非终结符的最后一个产生式右部的尾符号之后放置一个符号“$”，作为这个非终结符的所有产生式右部的结束符。

5.7.2 其它信息的存放

- 在带回溯的自上而下分析法中，需设置一个S栈来保存当前目标以及它的“双亲”、“孩子”和“兄长”等信息，一旦试探失败，则可进行回溯。S栈的元素是一个五元组

  （GOAL，i，FATHER，SON，BROTHER）

- 其中，GOAL是目标，i是目标的产生式右部的符号在数组GRAMMAR中的位置，FATHER、SON和BROTHER分别是目标的双亲、第一个孩子和兄长。

5.7.3 带回溯的自上而下分析算法

算法包括六个部分：INIT（初始化）、TEST（检查）、LOOK（查看）、SUCC（成功）、FAIL（失败）和ATRY（再试）。

INIT：

- P:= 1；{P指示当前结点}
- k:= 1；{k是栈S的栈顶指针}
- j:= 1；{j是输入符号串INPUT的下标}
- S[k]:= (Z, 0, 0, 0)； {Z是文法的识别符，作为当前目标，压入S栈中}
- goto TEST； {控制转到TEST部分}

TEST：

- if GOAL in VT then if GOAL = INPUT[j] then begin
- j:= j+1；goto SUCC {读下一个输入符号，转SUCC部分}
- end
- else goto FAIL； i:= GOAL的产生式右部第一个选择的首符号在数组GRAMMAR中的位置；
- goto LOOK； {若GOAL是非终结符，则找到它的产生式右部第一选择的首符号的位置，且转LOOK部分}

LOOK：

- if GRAMMAR [i] = ′|′ then {若是产生式右部结束符}
- if FATHER ≠ 0 then goto SUCC {且有双亲，则转SUCC部分}
- else STOP {若无双亲，即根结点，则分析成功，待查符号串是句子}
- if GRAMMAR [i] = ′$′ then {若是一个非终结符的所有产生式的右部结束符}
- if FATHER ≠ 0 then goto FAIL {且有双亲，则转FAIL部分}
- else STOP
- k:= k+1；{S栈指针加1}{若GRAMMAR [i]既不是“|”，也不是“$”}
- S[k]:=（GRAMMAR[i], O, P, O, SON）；{GRAMMAR[i]作为当前目标进栈，P指示的原目标是当前目标的双亲，原目标的第一个孩子是当前目标的兄长}
- SON:= k；{当前目标是原目标的孩子}
- P:= k；{P指示当前目标}
- goto TEST；

SUCC：

- P:= FATHER；{向双亲报告匹配成功，双亲作为当前目标}
- i:= i+1；goto LOOK；

FAIL：

- P:= FATHER；{向双亲报告匹配失败，双亲作为当前目标}
- SON:= S[SON].BRO; {将原目标的兄长作为当前目标的孩子}
- k:=k–1；{与原目标脱离父子关系}
- goto ATRY；{再试探}

ATRY：

- if SON = 0 then {若没有兄长}
- begin
- while GRAMMAR[i] ≠ ′1′ do {则跳过当前产生式右部}
- i:= i+1;
- i:= i+1; goto LOOK {取产生式右部另一个选择再查看}
- end;
- i:= i–1； {若有兄长}
- P:= SON； {兄长作为当前目标}
- if GOAL in VN then
- goto ATRY;
- j:= j–1；
- goto FAIL;

用扩展的BNF表示法消除左递归

① { } 零次或多次，$\{x\}_m^n$ m~n次

② [ ] 零次或一次，可选项。

③ ( ) 描述提取公因子。

- 例：G[标识符]:

<标识符> →<字母> | <标识符> <字母> | <标识符> <数字>

<数字> →0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<字母> →a|b|c|e|f|g|h|i|j|k|m|n|o|p|q|r|s|t|u|v|w|x|y|z

显然标识符产生式有左递归，可改写为

<标识符> →<字母> { <字母> | <数字>}

直接改写法

- U→Ux | y

  => U→yU’,

  U’→ xU’ | ε

- U→Ux1 | Ux2… | Uxm | y1 | y2 … yn

  => U→U(x1 | x2… | xm) | (y1 | y2 … yn)

  => U→ (y1 | y2 … yn)U’

  U’→ (x1 | x2… | xm)U’ | ε

直接改写法举例

- 例：G22[A]：

  $A \rightarrow [B$

  $B \rightarrow X] \mid BA$

  $X \rightarrow Xa \mid Xb \mid a \mid b$

  改写为： G22[A]：

  $A \rightarrow [B$

  $B \rightarrow X]B'$

  $B' \rightarrow AB' \mid \varepsilon$

  $X \rightarrow aX' \mid bX'$

  $X' \rightarrow aX' \mid bX' \mid \varepsilon$

消除左递归算法

① 将VN符号整理成序 Uk，k=1,…,n

② for i:=1 to n do

begin

j:=i+1 to n do 替换 Ui→Ujα中的Uj

消除Ui产生式中的直接左递归

end

③ 化简，删除多余产生式

消除左递归算法举例

- 例5.4 设文法

  A→Bcd

  B →Ce | f

  C →Ab | c

变换：

B →Ce | f => B →Abe | ce | f

A→Bcd => A→ Abecd | cecd | fcd

消除左递归

A→ cecdA’ | fcdA’

A’→becd A’|ε

B →Abe | ce | f

C →Ab | c

删除多余产生式

LL(k)文法

- 最左推导
- 从左到右扫描输入串
- 向前查看K（≥1）个符号，选择产生式

- 本课程只讨论LL(1)

LL(1)文法的判断条件

- $\alpha \in (VN \cup VT)^*$

  $FIRST(\alpha)=\{a \mid \alpha \overset{*}{\Rightarrow} a\beta，\ a \in VT，\ \beta \in (VN \cup VT)^*\}$
- $U \in VN$

  $FOLLOW(U)= \{b \mid S \overset{*}{\Rightarrow} xUby, b \in VT, x,y \in (VN \cup VT)^*\}$

- 定义5.1 文法G是LL(1)： $U \rightarrow x1 \mid x2 \ldots \mid xn$

  如果 $FIRST(xi) \cap FIRST(xj)=\Phi$

  当$\varepsilon \in FIRST(xi)$时，$FIRST(xi) \cap FOLLOW(U)=\Phi$

集合FIRST、FOLLOW的构造

FIRST(α)的构造

① $\alpha \in V_T\beta$，FIRST(α)= {a}

② $\alpha \in V_N\beta$，$V_N \rightarrow a\ldots$，$a \in FIRST(\alpha)$;

$\alpha \rightarrow \varepsilon$，$\varepsilon \in FIRST(\alpha)$

③ $\alpha \in V_N\beta$，$V_N \rightarrow XY\ldots$，

$FIRST(X)-\{\varepsilon\} \subset FIRST(\alpha)$，

若$X \overset{*}{\Rightarrow} \varepsilon$，则$FIRST(Y)-\{\varepsilon\} \subset FIRST(\alpha)$

集合FIRST、FOLLOW的构造

- FOLLOW（U）的构造

① U是开始符号，则#∈ FOLLOW(U)

② A→xUy，

则FIRST(y)-{ε} ⊂FOLLOW（U）

③ A→xUy，$y \overset{*}{\Rightarrow} \varepsilon$, (包括A→xU)

则FOLLOW(A) ⊂ FOLLOW(U)

② α∈VN，α→a…，a ∈FIRST(α)；
α→ε，ε∈FIRST(α)

③ α∈VN，α→XY…，
FIRST(X)-{ε} FIRST(α)，
若X ε，则FIRST(Y)-{ε} FIRST(α)

- 例G27[S]

S→A

A→BA’

A’→iBA’|ε

B→CB’

B’→+CB’| ε

C→)A*|(

	FIRST	FOLLOW
S	),(	#
A	),(	#,*
A’	i, ε	#,*
B	),(	#,*,i
B’	+, ε	#,*,i
C	),(	#,*,i,+

① U是开始符号，则#∈ FOLLOW(U)

② A→xUy，

则FIRST(y)-{ε}　FOLLOW（U）

③ A→xUy，$y \overset{*}{\Rightarrow} \varepsilon$, (包括A→xU)

则FOLLOW(A) ⊂　FOLLOW(U)

- 例G27[S]

S→A

A→BA’

A’→iBA’|ε

B→CB’

B’→+CB’| ε

C→)A*|(

	FIRST	FOLLOW
S	),(	#
A	),(	#,*
A’	i, ε	#,*
B	),(	#,*,i
B’	+, ε	#,*,i
C	),(	#,*,i,+

构造分析表的算法

- 例G27[S]

  S→A

  A→BA’

  A’→iBA’|ε

  B→CB’

  B’→+CB’| ε

  C→)A*|(

① M[U,a]=’U→α’，a∈FIRST(α);

② M[U,b]=’U→ε’，b∈FOLLOW（U）；

③ M[U,b]=’Error’，空白处;

	FIRST	FOLLOW
S	),(	#
A	),(	#,*
A’	i, ε	#,*
B	),(	#,*,i
B’	+, ε	#,*,i
C	),(	#,*,i,+

VN	VT					
	i	+	*	(	)	#
S				S→A	S→A	
A				A→BA’	A→BA’	
A’	A’→iBA’		A’→ε			A’→ε
B				B→CB’	B→CB’	
B’	B’→ε	B’→+CB’	B’→ε			B’→ε
C				C→(	C→)A*	

符号串‘(i(’分析过程

VN	VT					
	i	+	*	(	)	#
S				S→A	S→A	
A				A→BA’	A→BA’	
A’	A’→iBA’		A’→ε			A’→ε
B				B→CB’	B→CB’	
B’	B’→ε	B’→+CB’	B’→ε			B’→ε
C				C→(	C→)A*	

	符号栈	输入串		符号栈	输入串
1	#S	(i(#	8	#A’Bi	i(#
2	#A	(i(#	9	#A’B	(#
3	#A’B	(i(#	10	#A’B’C	(#
4	#A’B’C	(i(#	11	#A’B’(	(#
5	#A’B’(	(i(#	12	#A’B’	#
6	#A’B’	i(#	13	#A’	#
7	#A’	i(#	14	#	#

5.4 递归下降分析程序及其设计

- 例G27[S]

  S→A

  A→BA’

  A’→iBA’|ε

  B→CB’

  B’→+CB’| ε

  C→)A*|(

- 给每个非终结符设计一个相应的子程序。

```
int fS(){j=1;   return fA(); }
int fA(){if fB() return fA’();
                else return Error;  }
int fA’(){  if (ch[j]= =‘i’)
                {   j=j+1;
                    if fB() return fA’();  else return Error; }
                else return OK; }
…
int fC(){  if(ch[j]= =‘)’)
              {j=j+1; if fA()    {if (ch[j]= =‘*’);
                        {j=j+1 ; return OK} else return Error; }
             else  if(ch[j]= =‘(’) {j=j+1; return OK; }
                    else return Error;
             }
```

- FOLLOW（U）的构造

① U是开始符号，则#∈ FOLLOW(U)

② A→xUy，
则FIRST(y)-{ε}　FOLLOW（U）

③ A→xUy，y　ε，(包括A→xU)
则FOLLOW(A)　FOLLOW(U)

① α∈VT，FIRST(α)= {α}

② α∈VN，α→a…，a ∈FIRST(α)；
α→ε，ε∈FIRST(α)

③ α∈VN，α→XY…，
FIRST(X)-{ε}　FIRST(α)，
若X　ε，则FIRST(Y)-{ε}　FIRST(α)

例：已知文法G[S]：
S→eT | RT
T→DR |ε
R→dR |ε
D→a | bd

	FIRST	FOLLOW
S	a,b,e,d, ε	#
T	a,b, ε	#
R	d, ε	a,b,#
D	a,b	d,#

例：

已知文法G[S]：

S→eT | RT

T→DR |ε

R→dR |ε

D→a | bd

② LL(1)分析表

	FIRST	FOLLOW
S	a,b,e,d, ε	#
T	a,b, ε	#
R	d, ε	a,b,#
D	a,b	d,#

	a	b	e	d	#
S	S→RT	S→RT	S→eT	S→RT	S→RT
T	T→DR	T→DR			T→ε
R	R→ε	R→ε		R→dR	R→ε
D	D→a	D→bd			

- 已知文法G[A]:

A→A∨B | B

B→B∧C | C

C→┐D | D

D→（A）| i

① 消除左递归，计算每个非终结符的FIRST、FOLLOW集。

② 构造G[S]的LL(1)分析表。

- 解：①

A→BA’

A’→∨BA’ | ε

B→CB’

B’→∧C B’ | ε

C→┐D | D

D→（A）| i

	FIRST	FOLLOW
A	┐,(,i	),#
A’	∨, ε	),#
B	┐,(,i	∨,),#
B’	∧, ε	∨,),#
C	┐,(,i	∧,∨,),#
D	(,i	∧,∨,),#

A→BA’

A’→∨BA’ | ε

B→CB’

B’→∧C B’ | ε

C→¬ D | D

D→（A）| i

	FIRST	FOLLOW
A	¬ ,(,i	),#
A’	∨, ε	),#
B	¬ ,(,i	∨,),#
B’	∧, ε	∨,),#
C	¬ ,(,i	∧,∨,),#
D	(,i	∧,∨,),#

	∧	∨	¬	(	)	i	#
A			A→BA’	A→BA’		A→BA’	
A’		A’→∨BA’			A’→ε		A’→ε
B			B→CB’	B→CB’		B→CB’	
B’	B’→∧C B’	B’→ε			B’→ε		B’→ε
C			C→¬ D	C→D		C→D	
D				D→（A）		D→i	

- G[S]:

S→aAbDe|d

A →BSD|e

B →SAc|cD|ε

D →Se|ε

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

	a	b	c	d	e	#
S	aAbDe			d		
A	BSD		BSD	BSD	e	
B	Sac/ε		cD	Sac/ε		
D	Se/ ε	ε	ε	Se/ ε	ε	