# 编译原理复习总结

## 第一章 总论

1.编译程序的构造分为两部分:前端与后端。前端完成分析，包括词法分析，语法分析与语义分析几个部分。后端则完成综合，它进行目标代码生成与代 A 码优化。

2.词法分析：完成词法分析工作的部分称词法分析程序，又称扫描程序。词法分析程序从左到右逐个字符地扫描源程序正文的字符，根据词法规则识别具有独立意义的各个最小语法单位——符号(或称单词)，如标识符、无正负号数与界限符等，并把它们转换成通常是等长的内部形式(称属性字)，以供下一阶段语法分析程序使用。往往还进行语法分析前的其他工作，如删除注解、进行宏功能与条件编译等各项预处理工作。

3.语法分析：完成语法分析的部分称为语法分析程序，又称识别程序。它读入由词法分析程序识别出的符号，根据给定的语法规则，识别出各个语法结构。在进行语法分析时，识别出语法结构的同时也就检查了语法的正确性。

4.语义分析：一个程序涉及数据结构和控制结构，分别两方面语义分析。数据结构分析，检查进行运算时类型是否正确等，因此需确定类型，即确定标识符所代表数据对象的数据类型，进而检查运算分量类型的一致性和运算的合法性。控制结构分析,根据程序设计语言所规定的语义,对它们进行相应的语义处理。例如：加法运算检查两运算分量都有定义，且类型一致能进行加法运算后，生成执行加法运算的目标代码。

内部中间表示代码是语义分析的产物，如抽象语法树、逆波兰表达式、四元式序列与三元式序列。

语义分析完成确定类型、类型检查、识别含义与相应的语义处理，并进行一些静态语义检查等四项功能。

5.前端一般是与机器无关的，而后端一般是与机器相关的。相应于各个要完成的基本工作将分别有词法分析程序、语法分析程序、语义子程序、目标代码生成程序与目标代码优化程序，有机地结合起来完成对源程序的编译。

6.每个阶段读入整个输入并进行处理的过程称为遍(或趟)。第一遍的输入是源程序,相应语言记为 SL(源语言),最后一遍输出的是目标程序,相应语言记为 TL(目标语言)，作为中间各遍输出的中间表示，相应的语言记为 L1,L2,…。

## 第二章 文法与语言

1.允许有不包含任何符号的符号串，这种符号串称为空符号串，简称空串，用 ε 表示。

2.符号串 x 中所包含符号的个数称为符号串 x 的长度，用|x|表示。

如：|abc|=3,| ε |=0。

3.设 x 是某字母表上的符号串，把 x 自身联结 n 次，即 z=xx…xx(n 个 x)，称为符号串 x 的 n 次幂，记为 $z=x^n$。

如：x=ab, $x^0=\varepsilon$ , $x^3$=ababab。$x^n=xx^{n-1}==x^{n-1}x$ 且$|x^n|=n|x|$。

4.字母表的闭包与正闭包：

设有字母表 A，由它作方幂 $A^0, A^1, A^2, \cdots, A^n, \cdots$。A 的闭包定义如下：

A 的闭包 $A^*=A^0 \cup A^1 \cup A^2 \cup \cdots \cup A^n \cup \cdots$。

由于 $A^n(n=0,1,2,\cdots)$ 中所有符号串的长度为 n，因此字母表 A 的闭包 $A^*$ 为字母表上一切长度为 $n(n\geqslant 0)$ 的符号串所组成的集合。

如果不允许包含空串 ε，则得到字母表 A 的正闭包。

A 的正闭包 $A^+=A^1\cup A^2\cup\cdots\cup A^n\cup\cdots$。

显然，$A^*=A^0\cup A^+$，且 $A^+=AA^*=A^*A$。

如：设字母表 $\Sigma=\{a,b,c\}$，依次写出长度为 1,2,…的符号串，可得到 Σ 的正闭包 $\Sigma^+$：

$\Sigma^+=\{a,b,c,aa,ab,ac,ba,bb,bc,\cdots\}$，$\Sigma^+$ 中添入空串 ε 即得 $\Sigma^*$。

5. 由于一个字母表的正闭包包含了该字母表中的符号所能组成的一切符号串，而语言是该字母表上的某个符号串集合，因此，在某个字母表上的语言是该字母表的正闭包的子集，且是真子集。

6. XXX 是语言定义的目标，称它为识别符号或开始符号。识别符号作为非终结符，必定至少在一个规则中作为左部出现。

7. 文法 G[Z]是有穷非空的重写规则集合，其中 Z 是识别符号，而 G 是文法名。一般来说，以识别符号为左部的重写规则作为文法的第一个重写规则。

8. 给定了一个文法 G，它的一切非终结符号所组成的集合通常记为 $V_N$，它的一切终结符号所组成的集合通常记为 $V_T$。显然 $V_N\cap V_T=\emptyset$。

9. 文法 G 的字汇表 $V=V_N\cup V_T$，即字汇表是出现于文法规则中的一切符号所组成的集合。

10. 文法中的规则可以帮助分析或构造句子，句子(程序)总是只由终结符所组成。

11. 文法是一种以有穷的方式描述潜在地无穷的(终结)符号串集合的手段。

12. 生成句子的基本思想是：从识别符号开始，把当前产生的符号串中的非终结符号替换为相应规则右部的符号串，如此反复，直到最终全由终结符号组成。

13. 应用文法产生句子的步骤：

步骤 1 从识别符号开始，把它替换为以它为左部的某个规则的右部；

步骤 2 每次把替换所得符号串中最左的非终结符号替换为相应规则的右部符号串；

步骤 3 重复步骤 2 直到再无非终结符号可替换，最终所得的就是全由终结符号构成的句子。

14. 设 G[Z]是一文法，如果符号串 x 是从识别符号 Z 推导所得的，即

$Z\Rightarrow{}^* x \quad x\in V^+$

则称符号串 x 是该文法 G 的一个句型。

如果一个句型 x 仅由终结符号所组成，即

$Z\Rightarrow{}^* x \quad x\in V_T^+$

则称该句型 x 是该文法 G 的一个句子或一个字。

15. 从识别符号出发进行推导所得到的一切符号串都是相应文法的句型，且识别符号是最简单的句型。句子则是全由终结符号组成的句型。

16. 语言的形式定义：

$L(G[Z])=\{x\mid Z\Rightarrow{}^* x,\text{且 } x\in V_T^+\}$

由 $Z\Rightarrow{}^* x$，因而 x 是文法 G[Z]的一个句型。

$x\in V_T^+$，即 x 全由终结符号所组成，所以 x 是文法 G[X]的句子。

由文法描述的语言是该文法的一切句子的集合。它是所有终结符号串所组成的一个真子集，即

L(G)包含于 $V_T^+$。

构成一个语言的句子集合可以是有穷的，也可以是无穷的。

17. 如果一个规则形如 U::=…U…，则称该规则是递归的。如果规则形如 U::=U…，则称左递归的；如果形如 U::= …U，则称右递归的。

如果对于某文法，存在非终结符号 U，对于它，U=>+ …U…，则称该文法递归于 U；如果 U=>+ U…，则称该文法左递归于 U；如果 U=>+ …U，则称该文法右递归于 U。

18. 如果对于某文法 G，P 中的每个规则具有下列形式：

u::=v

其中 $u\in V^+$，$v\in V^*$，则称该文法 G 为(Chomsky)0 型文法或短语结构文法，缩写为 PSG。其相应语言称为(Chomsky)0 型语言或短语结构语言，又称递归可枚举集。

19. 如果对于某文法 G，P 中的每个规则具有下列形式：

xUy::=xuy

其中 $U\in V_N$，x、$y\in V^*$，$u\in V^+$，则称该文法 G 为(Chomsky)1 型文法或上下文有关文法，也称上下文敏感文法，缩写为 CSG。其相应语言称为(Chomsky)1 型语言或上下文有关语言。

注：如果对于某文法 G，P 中的每个规则 u::=v 都满足|v|≥|u|，则文法 G 是 1 型文法。

20. 如果对于某文法 G，P 中的每个规则具有下列形式：

U::=u

其中 $U\in V_N$，$u\in V^+$，则称该文法 G 为(Chomsky)2 型文法或上下文无关文法，缩写为 CFG。其相应语言称为(Chomsky)2 型语言或上下文无关语言。

注：对于上下文无关文法，在推导中应用规则 U::=u 时，无须考虑非终结符 U 所在的上下文，总能把 U 重写(替换)为符号串 u，或把 u 直接规约为 U。定义中 $u\in V^+$，所以不允许有形如 U::= ε 的规则，这类规则称为 ε 规则。

21. 如果对于某文法 G，P 中的每个规则具有下列形式：

U::=T　或　U::=WT　(左线性)　U::=TW(右线性)

其中 $T\in V_T$，U、$W\in V_N$，则称该文法 G 为(Chomsky)3 型文法或正则文法(或正规文法)，有时又称有穷状态文法，缩写为 RG。其相应语言称为(Chomsky)3 型语言或正则语言(或正规语言)。

注：按照定义，单个非终结符只能重写(替换)作单个终结符号或重写(替换)作单个非终结符号跟以单个终结符号。

22. 对于语言 $L1=\{a^ib^jc^k|i,j,k\geq 1\}$ 可定义如下的相应文法 G1：

G1=({A,B,C},{a,b,c},P1,S)

其中 P1:

S::=Bc|Sc　B::=Ab|Bb　A::=Aa|a

按定义，G1 是 3 型文法，L1 是 3 型语言。

对于语言 $L2=\{a^ib^ic^k|i,k\geq 1\}$ 可定义如下的相应文法 G2：

G2=({S,A},{a,b,c},P2,S)

其中 P2:

S::=Ac|Sc　A::=ab|aAb

按定义，G2 是 2 型文法，L2 是 2 型语言。

对于语言 $L3=\{a^ib^ic^i|i\geq 1\}$ 可定义如下的相应文法 G3：

G3=({S,B,C,D},{a,b,c},P3,S)

其中 P3:

S::=aSBC　　S::=abC

CB::=CD　　CD::=BD　　BD::=BC

bB::=bb　　bC::=bc　　cC::=cc

按定义，G3 是 1 型文法，L3 是 1 型语言。

23. 一般地，存在有不是上下文有关的短语结构语言，存在有不是上下文无关上下文有关语言，存在有不是正则的上下文无关语言。

即 3 型语言类包含于 2 型类语言包含于 1 型语言类包含于 0 型语言类

24. 左递归的存在将导致自顶向下分析技术的失败，因此当采用自顶向下的分析技术时，总是必须首先进行消去左递归的文法等价替换。

25. 规则左递归的消去

一般改写规则如下：

如果 $U::=U_{x1}|U_{x2}|\cdots|U_{xm}|y_1|y_2|\cdots|y_n$，则

$U::=y_1U`|y_2U`|\cdots|y_nU`$

$U`::=x_1U`|x_2U`|\cdots|x_mU`|\varepsilon$

通常总让 ε 作为最后的选择。

26. 文法左递归的消去

一般改写规则如下：

步骤 1 把文法 G 的非终结符号排序成 $U_1,U_2,\cdots,U_n$

步骤 2 以上列顺序执行下列程序：

```
for (i=1; i<=n; i++)
{  for (j=1; j<=i-1; j++)
   {  把形如 Ui::=Ujr 的规则改成:Ui::=xj1r|xj2r|…|xjkr
      Uj::=xj1|xj2|…|xjk是对于 Uj的一切规则
   }
   消除关于 Ui的规则左递归
}
```

步骤 3 化简由步骤 2 得到的文法，即删去那些无用规则

例 1：试用不同的方法消去文法 G：I::=Ia|Ib|c 的规则左递归。

解：步骤 1　判定文法是规则左递归

步骤 2　消去规左递归。

步骤 3

方法 1　改写规则左递归成右递归。

等价文法 G` 为：

I::=cI`　　I`::=(a|b)I`|ε

方法 2　改写成扩充 BNF 表示法.

应用规则 1 提因子有：I::=I(a|b)|c，

应用规则 2 有 I::=c{a|b}

等价文法 G` 为：I::=c{a|b}

例 2：试消去文法 G[W]：W::=AO　A::=AO|W1|0 的文法左递归与规则左递归。

解：步骤 1 判定文法是文法左递归还是规则左递归

步骤 2 判定文法是文法左递归，所以按相应算法消去文法左递归如下。

步骤 2.1：把终结符排序成 $U_1$=W, $U_2$=A(n=2)

步骤 2.2：执行循环

i=1 j=1:j>i－1 不执行关于 j 的循环，

且关于 $U_1$=W 不存在规则左递归。

i=2 j=1,有规则 A::=W1|A0|0 形如 $U_2$::=U…,把 $U_1$::=$r_1$，

即，把 W::=A0 代入得：A::=A01|A0|0 即，

A::=A(01|0)|0

j=2, j>i-1 消去关于 $U_2$=A 的规则左递归有

A::=0A`, A`::=(01|0)A`|ε

步骤 3 最后得到消去左递归的等价文法 G`[W]：

W::=A0 A::=0A` A`::= (01|1)A`|ε

说明：如果在第二步中，把原文法等价变换成扩充表示法，则最终的等价文法是

G`[W]: W::=A0 A::=0{01|0}

例 3：试消去文法 G[S]：

S::=Qc|Rd|c Q::=Rb|Se|b R::=Sa|Qf|a

解: 步骤 1 首先判定是文法左递归还是规则左递归

步骤 2 是文法左递归，按相应算法处理如下。

步骤 2.1 把非终结符号排序成

$U_1$=S $U_2$=Q $U_3$=R (n=3)

步骤 2.2 执行循环：

i=1 j=1: j>i－1，不执行关于 j 的循环，

且关于 $U_1$=S 不存在规则左递归。

i=2 j=1,有规则 Q::=Se|Rb|b,形如 $U_2$::=$U_1$…,把 $U_1$::=$r_1$,即 S::=Qc|Rd|c 代入，得：Q::= (Qc|Rd|c)e| Rb|b，整理后

Q::= Qce|Rde| Rb|ce|b

j=2, j>i-1 对 $U_2$其消去规则左递归，得

Q::=(R(de|b)|ce|b)Q` Q`::=ceQ`|ε

(按扩充表示法，有 Q::=（Rb|Rde|ce|b）{ce} )

i=3 j=1,有规则 R::=Sa|Qf|a，形如 $U_3$::=$U_1$…形，把 $U_1$::=$r_1$，即，S::=Qc|Rd|c 代入，得：R::=(Qc|Rd|c)a|Qf|a，整理后

R::= Rda| Qca| Qf| ca|a

注意: j 循环还未结束,不能消去 Ui=R 的规则左递归!

j=2,有规则 R::= Rda| Qca| Qf| ca|a，形如 $U_3$::=$U_2$…， 把 $U_2$::=$r_2$，即 Q::=(R(b|de)|ce|b)Q`代入，得

R::=Rda| (R(b|de)|ce|b)Q`(ca|f)| ca|a，

整理后 R::=R((b|de)Q`(ca|f)|da)|(ce|b)Q`(ca|f)|ca|a

(按扩充表示法代入的是 Q::=(Rb|Rde|ce|b){ce}) )

j=3, j>i-1 消去关于 $U_3$=R 的规则左递归,得：

R::= ((ce|b)Q`(ca|f)|ca|a)R`

R`::=((b|de)Q`(ca|f)|da)R`|ε

(当按扩充表示法时是:R::=((ce|b)Q`(ca|f)|ca|a){(b|de)Q`(ca|f)|da} )

步骤 3 最后消去了左递归的等价文法 G`[S]：

S::=Qc|Rd|c

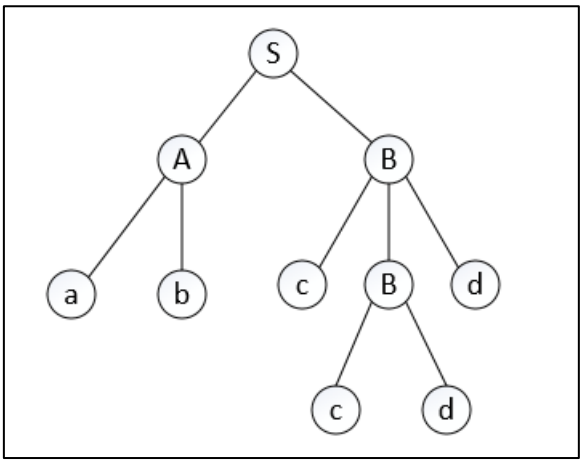
Q::=(R(b|de)|ce|b)Q`　　Q`::=ceQ`|ε
R::= ((b|ce)Q`(ca|f)|ca|a)R`
R`::=((b|de)Q`(ca|f)|da) R`|ε
(按扩充表示法时是 G`[S]:
S::=Qc|Rd|c
Q::=(Rb|Rde|ce|b){ce}
R::= ((ce|b)Q`(ca|f)|ca|a){(b|de)Q`(ca|f)|da}
)

27. 语法分析树

文法 G[S]：S::=AB　A::=aAb|ab　B::=cBd|cd

关于句子 abccdd 的推导：S=>AB=>AcBd=>Accdd=>abccdd

画出语法分析树：




(1)结点：每个符号对于于一个结点，该结点就以相应符号为其名，如：结点 S、A。

(2)边：两结点间的连线称为边。

(3)根结点：没有从上向下进入它的边，而只有从它向下射出的边的结点，这里即是识别符号所对应的结点 S。

(4)分支：从某结点向下射出的边连同边上的结点称为分支。分支的结点是射出该分支的结点的名字。分支的各个结点称为分支结点。兄弟结点中，最右边的分支结点最’年轻’。

(5)子树：语法树的某结点连同从它向下射出的部分称为该语法树的子树，该结点称为子树的根结点。

(6)语法树中再没有分支从它向下射出的结点称为末端结点。所有分支结点都是末端结点的分支称为末端分支。末端结点不一定是终结符号，非终结符号也可能称为末端结点。

从推导构造语法分析树性质：

分支的分支结点符号串是相应句型中相对于分支名字结点的简单短语。

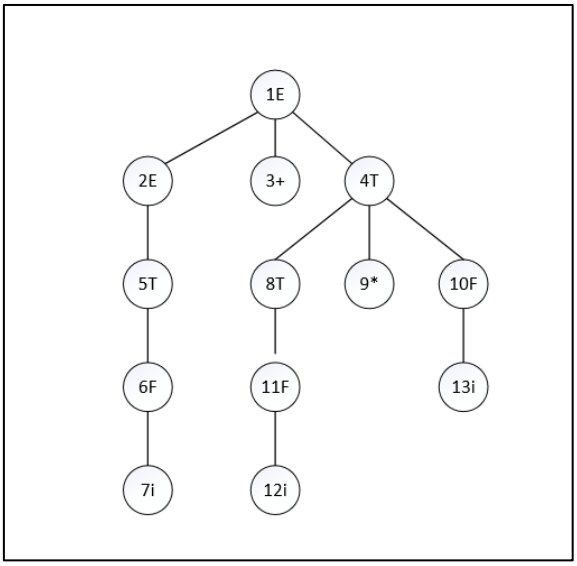
子树的末端结点符号串是相应句型中相对于子树根结点的短语。

重要用途：利用语法分析树寻找句型中的简单短语和短语

28. 语法分析树的存储表示与输出

```
Typedef struct
{
int 结点序号;
```

int 文法符号序号;
int 父结点序号;
int 左兄结点序号;
int 右子结点序号;
}语法树结点类型;
语法树结点类型 语法分析树[MaxNodeNum];




| 结点序号 | 文法符号 | 父结点 | 左兄结点 | 右子结点 |
|---|---|---|---|---|
| 1 | E | 0 | 0 | 4 |
| 2 | E | 1 | 0 | 5 |
| 3 | + | 1 | 2 | 0 |
| 4 | T | 1 | 3 | 10 |
| 5 | T | 2 | 0 | 6 |
| 6 | F | 5 | 0 | 7 |
| 7 | i | 6 | 0 | 0 |
| 8 | T | 4 | 0 | 11 |
| 9 | * | 4 | 8 | 0 |
| 10 | F | 4 | 9 | 13 |
| 11 | F | 8 | 0 | 12 |
| 12 | i | 11 | 0 | 0 |
| 13 | i | 10 | 0 | 0 |

例：设 G[E]:
E::=T|E+T|E-T
T::=F|T*F|T/F
F::=(E)|i
1) 试给出关于(i)、i*i-i 与(i+i)/i 的推导。
2) 证明 E+T*F*i+I 是该文法的句型，然后列出它的一切短语与简单短语。
解：1) E => T => F => (E) => (T) => (F) => (i)
E => E-T => T-T => T*F-T => F*F-T => i*F-T => i*i-T => i*i-F => i*i-i (最左推导)
E => E-T => E-F => E-i =>T-i => T*F-i => T*i-i => F*i-i =>i*i-

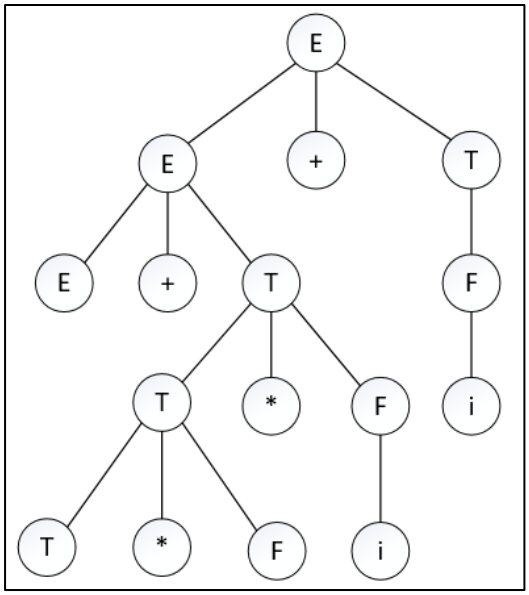
i(最右推导)

E => T => T/F => F/F => (E)/F => (E+T)/F=> (T+T)/F => (F+T)/F => (i+T)/F => (i+F)/F=> (i+i)/F=> (i+i)/i(最左推导)

E => T => T/F => T/i => F/i => (E)/i => (E+T)/i => (E+F)/i => (E+i)/i=>(T+i)/i=>(F+i)/i=>(i+i)/i(最右推导)

2) E => E+T => E+T+T => E+T*F+T => E+T*F*F+T => E+T*F*i+T => E+T*F*i+F => E+T*F*i+i

E => E+T => E+F => E+i =>E+T+i=> E+T*F+i => E+T*i+i => E+T*F*i+i




所以 句型 E+T*F*i+i 中

相对于 E 的短语有 E+T*F*i+i 和 E+T*F*i

相对于 T 的短语有 T*F*i、T*F 和 i

相对于 F 的短语有 i

所以 句型 E+T*F*i+i 中

相对于 T 的简单短语有 T*F

相对于 F 的简单短语有 i

例：试给出一个产生语言 L={w|w∈{0,1}⁺，且 w 不包含两个相邻的 1}的正则文法。

解：G[S]：S::=0|1|0S|1A　　A::=0|0S

例:设计一个语言是不以零打头的奇整数组成的集合,试写出产生该语言的文法。

解：G[S]： S::=J|IBJ　　B::=0B|IB| ε　I::=J|2|4|6|8　J::=1|3|5|7|9

例：设有文法 G[S]：S::=aAcB|BdS　B::=aScA|cAB|b　A::=BaB|aBc|a

试对下列符号串：1）aabcccab　2）ababccbb

进行句型分析，识别是否是文法 G[S]的句子。当是句子时，给出 最左推导、最右推导与相应的语法分析树。

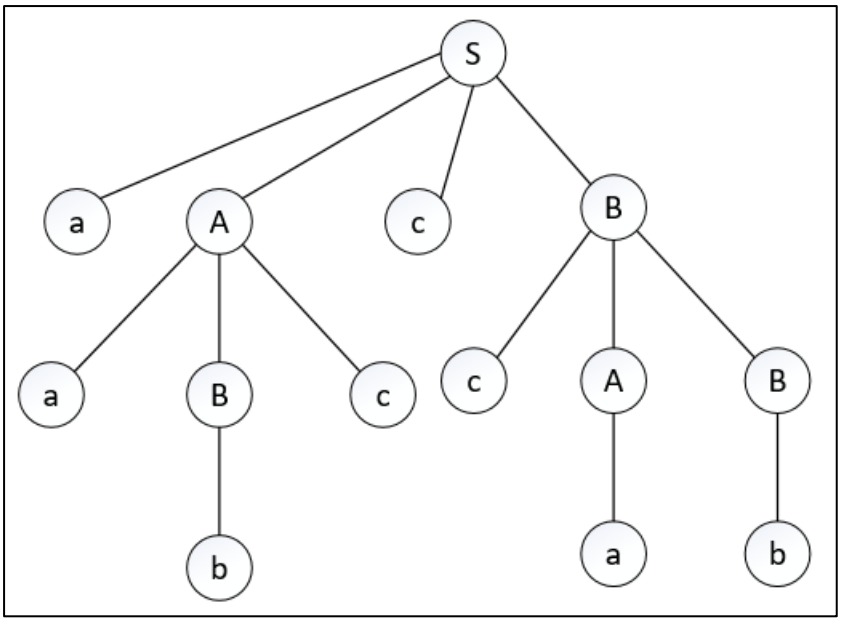
解：1）建立最左推导如下：

S=> aAcB => aaBccB=>aabccB =>aabcccAB=>aabcccaB=>aabcccab

最右推导如下:

S=>aAcB=>aAccAB=>aAccAb=>aAccab=>aaBcccab=> aabcccab

即,S=>* aabcccab，因此，aabcccab 是该文法的句子。




2) ababccbb 不是该文法句子，因为不能为它构造推导或语法分析树。

## 第三章 词法分析

1.一个确定的有穷状态自动机 DFA 是五元组(K, Σ,M,S,F)，其中，

K 是有穷非空的状态集合

Σ是有穷非空的输入字母表

M 是从 K×Σ到 K 的映像

S 是开始状态，S∈K

F 是非空的终止状态集合，F 真包含于 K。

例：DFA D=({S,Z,A,B},{a,b},M,S,{Z})

其中， M： M(S,a)=A M(S,b)=B M(A,a)=Z M(A,b)=B

M(B,a)=A M(B,b)=Z M(Z,a)=Z

2. 一个非确定的有穷状态自动机 NFA 是五元组(K, Σ,M,S,F)，其中，

K 是有穷非空的状态集合

Σ是有穷非空的输入字母表

M 是从 K×Σ到 K 的子集所组成集合的映像

S 是非空的开始状态集合，S 真包含于 K

F 是非空的终止状态集合，F 真包含于 K。

例：NFA N=({S A,B,Z},{a,b},M,{S},{Z})

其中， M： M(S,a)={A} M(S,b)={B} M(A,a)={Z} M(A,b)={B}

M(B,a)={A,b} M(B,b)={Z} M(Z,a)={A,Z} M(Z,b)=Ø

3.NFA 与 DFA 的区别：

NFA 可以有若干个开始状态，而 DFA 仅一个。

NFA 的 M 是从从 K×Σ到 K 的子集所组成集合的映像，而不是到 K 的映像，即映像 M 将产生一个状态集合(可能为空集)，而不是单个状态。

4.上例构造 DFA

| 状态\转换为\输入 | a | b |
|---|---|---|
| [S] | [A] | [B] |
| [A] | [Z] | [B] |
| [B] | [AB] | [Z] |
| [Z] | [AZ] | [] |
| [AB] | [ABZ] | [BZ] |
| [AZ] | [AZ] | [B] |
| [BZ] | [ABZ] | [Z] |
| [ABZ] | [ABZ] | [BZ] |

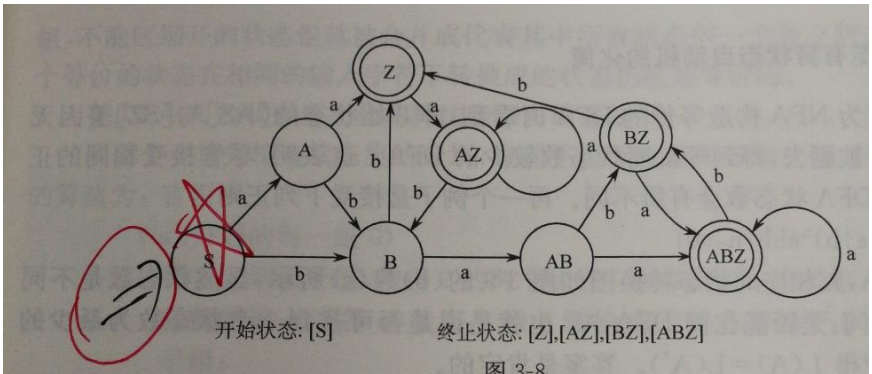



图 3-8

现在 DFA 可写出如下：

DFA $N' = (K', \{a,b\}, M', [S], F')$

其中，$K' = \{[S],[A],[B],[Z],[AB],[AZ],[BZ],[ABZ]\}$

$M'$：$M'([S],a) = [A]$　　$M'([S],b) = [B]$

$M'([A],a) = [Z]$　　$M'([A],b) = [B]$

$M'([B],a) = [AB]$　　$M'([B],b) = [Z]$

$M'([Z],a) = [AZ]$

$M'([AB],a) = [ABZ]$　　$M'([AB],b) = [BZ]$

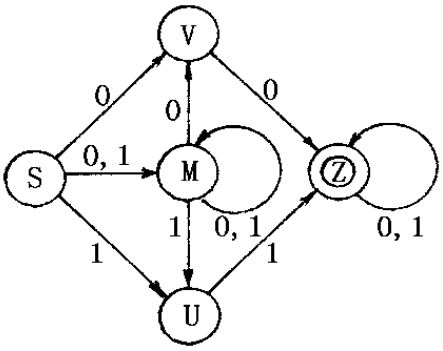
$M'([AZ],a) = [AZ]$　　$M'([AZ],b) = [B]$

$M'([BZ],a) = [ABZ]$　　$M'([BZ],b) = [Z]$

$M'([ABZ],a) = [ABZ]$　　$M'([ABZ],b) = [BZ]$

$F' = \{[Z],[AZ],[BZ],[ABZ]\}$

例：设有 NFA，其状态转换图如图所示，试为其构造 DFA。



步骤 1 首先写出 NFA，然后再确定化。

NFA N＝ ({S,V,M,U,Z},{0,1},M,{S},{Z})

其中: M : M(S,0)={V,M} M(S,1)={M,U}

M(V,0) ={Z} M(V,1)= ε M(M,0)={V,M}

M(M,1)={M,U} M(U,0)= ε M(U,1)={Z}

M(Z,0) ={Z} M(Z,1)={Z}

步骤 2 为确定化，列表如下，以得到 DFA 的一切状态。

| | 0 | 1 |
|---|---|---|
| [S] | [MV] | [MU] |
| [MV] | [MVZ] | [MU] |
| [MU] | [MV] | [MUZ] |
| [MVZ] | [MVZ] | [MUZ] |
| [MUZ] | [MVZ] | [MUZ] |

步骤 3 构造 DFA 如下：

DFA N`=(K`,{0,1}, M`, [S], F`)

其中：K`= {[S],[ MV],[MU],[ MUZ],[ MVZ]}

M`:M`([S],0)=[VM] M`([S],1)=[MU]

M`([MV],0)=[ MVZ] M`([MV],1)=[MU]

M`([MU],0)=[ MV] M`([MU],1)=[MUZ]

M`([MVZ],0)=[ MVZ] M`([MVZ],1)=[MUZ]

M`([MUZ],0)=[ MVZ] M`([MUZ],1)=[MUZ]

F`= {[MVZ], [ MUZ]}

## 第四章 语法分析——自顶向下分析技术

1.如果某文法，其预测分析表无多重定义的元素，则称该文法为 LL(1)文法。

2.一个文法 G 是 LL(1)的，当且仅当对于 G 的每个非终结符号 U 的任何两个不同的重写规则 U::=x 与 U::=y，下面的条件成立。

条件 1 FIRST(x)∩FIRST(y)=Ø;

条件 2 x=>* ε 与 y=>* ε 不能同时成立;

条件 3 如果 y=>* ε，则 FIRST(x)∩FOLLOW(U)= Ø。

## 第五章 语法分析——自底向上分析技术

1.算符文法

如果文法 G 中没有形如 U::=…VW… 的规则，其中 U,V,W∈$V_N$，则该文法称为算符文法，缩写为 OG。

即在算符文法中不存在右部分包含两个相邻非终结符号的规则，任何规则右部中的任何两个非终结符号之间必须有终结符号(可能不止一个)。

2.一个文法 G 是 L(k)文法，当且仅当在句子的识别过程中，任一句柄总是由其左部的符号串及其右部的 k 个终结符号所唯一确定。

3. L(k)文法性质：

性质 1 对于任何可用一个 LR(k)文法定义的上下文无关语言都能用一个确定的下推自动机以自底向上方式识别，且该下推自动机在识别过程中以一种能精确定义的方式“使用”该文法，能被称作是识别各个规则。

性质 2 LR(k)文法是无二义性的。

性质 3 当 k 给定时便可能判断一个文法是否 LR(k)文法。

性质 4 一个语言能由 LR(k)文法生成，当且仅当它能由 LR(1)文法生成。

例：文法 G5.5 [E]：

1：E::=E+T　　2：E::=T　　3：T::=T*F

4：T::=F　　5：F::=(E)　　6：F::=i

LR(1)分析表

| 状态 | ACTION | | | | | | GOTO | | |
|---|---|---|---|---|---|---|---|---|---|
| | + | * | ( | ) | i | # | E | T | F |
| 0 | | | S4 | | S5 | | 1 | 2 | 3 |
| 1 | S6 | | | | | acc | | | |
| 2 | r2 | S7 | | r2 | | r2 | | | |
| 3 | r4 | r4 | | r4 | | r4 | | | |
| 4 | | | S4 | | S5 | | 8 | 2 | 3 |
| 5 | r6 | r6 | | r6 | | r6 | | | |
| 6 | | | S4 | | S5 | | | 9 | 3 |
| 7 | | | S4 | | S5 | | | | 10 |
| 8 | S6 | | | S11 | | | | | |
| 9 | r1 | S7 | | r1 | | r1 | | | |
| 10 | r3 | r3 | | r3 | | r3 | | | |
| 11 | r5 | r5 | | r5 | | r5 | | | |

识别 i+i*i

| 步骤 | 状态栈 | 符号栈 | 输入栈 | ACTION | GOTO | 说明 |
|---|---|---|---|---|---|---|
| 1 | 0 | # | i+i*i# | S5 | | |
| 2 | 05 | #i | +i*i# | r6 | 3 | 规约 i |
| 3 | 03 | #F | +i*i# | r4 | 2 | 规约 F |
| 4 | 02 | #T | +i*i# | r2 | 1 | 规约 T |
| 5 | 01 | #E | +i*i# | S6 | | |
| 6 | 016 | #E+ | i*i# | S5 | | |
| 7 | 0165 | #E+i | *i# | r6 | 3 | 规约 i |
| 8 | 0163 | #E+F | *i# | r4 | 9 | 规约 F |
| 9 | 0169 | #E+T | *i# | S7 | | |
| 10 | 01697 | #E+T* | i# | S5 | | |
| 11 | 016975 | #E+T*i | # | r6 | 10 | 规约 i |
| 12 | 01697A | #E+T*F | # | r3 | | 规约 T*F |
| 13 | 0169 | #E+T | # | r1 | | 规约 E+T |
| 14 | 01 | #E | # | acc | | 接受 |

识别(i+i)*i

| 步骤 | 状态栈 | 符号栈 | 输入栈 | ACTION | GOTO | |
|---|---|---|---|---|---|---|
| 1 | 0 | # | (i+i)*i# | S4 | | |
| 2 | 04 | #( | i+i)*i# | S5 | | |
| 3 | 045 | #(i | +i)*i# | r6 | 3 | |
| 4 | 043 | #(F | +i)*i# | r4 | 2 | |
| 5 | 042 | #(T | +i)*i# | r2 | 8 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 048 | #(E | +i)*i# | S6 | | |
| 7 | 0486 | #(E+ | i)*i# | S5 | | |
| 8 | 04865 | #(E+i | )*i# | r6 | 3 | |
| 9 | 04863 | #(E+F | )*i# | r4 | 9 | |
| 10 | 04869 | #(E+T | )*i# | r1 | 8 | |
| 11 | 048 | #(E | )*i# | S11 | | |
| 12 | 048,11 | #(E) | *i# | r5 | 3 | |
| 13 | 03 | #F | *i# | r4 | 2 | |
| 14 | 02 | #T | *i# | S7 | | |
| 15 | 027 | #T* | i# | S5 | | |
| 16 | 0275 | #T*i | # | r6 | 10 | |
| 17 | 027,10 | #T*F | # | r3 | 2 | |
| 18 | 02 | #T | # | r2 | 1 | |
| 19 | 01 | #E | # | acc | | |

## 第六章 语法分析与目标代码生成

1.语义分析：

对于所写源程序，应进一步分析其含义，在理解含义的基础上，为生成相应的目标代码做好准备，或者直接生成目标代码，这就是语义分析。即分析源程序的含义，并作相应的语义处理。

2.语义分析的基本功能如下：

(1)确定类型：确定标识符所关联数据对象的数据类型。

(2)类型检查：按照语言的类型规则，对运算及进行运算的运算分量进行类型检查，检查运算的合法性与运算分量类型的一致性(相容性)，必要时作相应的类型转换。

(3)识别含义，并作相应的语义处理：根据程序设计语言的语义定义(形式或非形式的)，确认(识别)程序中各构造成分组合到一起的含义，并作相应的语义处理。这时对可执行语句生成中间表示代码或目标代码。

(4)其他一些静态语义检查：语义分析时可进行一些静态语义检查，例如控制流检查。

3.语义分析与目标代码生成分别进行的原因：

原因 1 该工作复杂、繁琐，分开可使难点分解，分开解决。

原因 2 编译一次，多次运行，分开可对中间代码进行优化以产生高效目标代码。

原因 3 通常目标代码与机器有关，语义分析与机器无关，分开可时语义分析程序适用于多个目标代码生成程序。

原因 4 有利于人员组织、有利于整个编译程序的有效开发。

4.逆波兰表示

例：赋值语句 t=(a+b)*c/(d-e)的逆波兰表示是：t a b + c * d e - / = 。

例：A[i]=B[j+k][m]的逆波兰表示是：A I [] B j k + [] m [] = 。

GOF 和 GO 分别表示按假转与无条件转向某序号处。

例：条件语句 if(a<b)max=b;else max=a;的逆波兰表示是：

a b < 11 GOF max b = 14 GO max a =

例：分程序复合语句

{int i;i=1;f=1;a=0;
AGAIN:if(i<10)
{b=f;f=f+a;a=b;i=i+1;goto AGAIN;}
fib=f;}的逆波兰表示如下。

BLOCK i 1 = f 1 = a 0 = AGAIN : i 10 < 36 GOF b f = f f a + = a b = i i 1 + = AGAIN GOL fib f = BLOCKEND

例：while(i<10){b=f;f=f+a;a=b;}逆波兰表示是：

i < 10 = 19 GOF b f = f f a + = a b = 1 GO

5. 例：有程序控制部分如下：

```
max=A[1];i=2;
while(i<=n)
{if(A[i]>max)max=A[i];
i=i+1;}
```

while 语句展开：

```
max=A[1];i=2;
loop:if(i<=n)
{if(A[i]>max)max=A[i];
i=i+1;goto loop;}
```

相应四元式如下：运算符　运算分量　运算分量　结果

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| (1) | =[] | A | 1 | t1 | (7) | =[] | A | i | t3 |
| (2) | := | t1 | | max | (8) | := | t3 | | max |
| (3) | := | 2 | | I | (9) | + | i | 1 | t4 |
| (4) | > | i | n | (12) | (10) | := | t4 | | i |
| (5) | =[] | A | i | t2 | (11) | GO | (4) | | |
| (6) | ≤ | t2 | max | (9) | (12) | | | | |

相应三元式如下：运算符　运算分量　运算分量

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| (1) | =[] | A | 1 | (8) | GOF | (11) | (7) |
| (2) | := | (1) | max | (9) | =[] | A | i |
| (3) | := | 2 | i | (10) | := | (9) | max |
| (4) | ≤ | i | n | (11) | + | i | 1 |
| (5) | GOF | (14) | (4) | (12) | := | (11) | i |
| (6) | =[] | A | i | (13) | GO | (4) | |
| (7) | > | (6) | max | (14) | | | |

例：试把表达式(a+b)*(c-d)-(a*b+c)翻译成

(1)逆波兰表示；(2)四元式序列；(3)三元式序列。

解：(1)ab+cd-*ab*c+-

(2)四元式序列

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| (1) | + | a | a | t1 | (4) | * | a | b | t4 |
| (2) | - | c | d | t2 | (5) | + | t4 | c | t5 |
| (3) | * | t1 | t2 | t3 | (6) | - | t3 | t5 | t6 |

(3)三元式序列

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| (1) | + | a | a | (4) | * | a | b |
| (2) | - | c | d | (5) | + | (4) | c |
| (3) | * | (1) | (2) | (6) | - | (3) | (5) |

例：试写出下列程序片段的四元式表示。

```
positive=0;  negative=0;    ero=0;
for(i=1; i<=100; i=i+1)
{if(A［i］＞0)positive=positive+1;
else if(A［i］==0)zero=zero+1;
else negative=negative+1; }
```

循环展开：

```
i=1;LOOP:if(i>100) goto FINISH;
if(A[i]>0)positive=positive+1;
else if(A［i］==0) zero=zero+1;
else negative=negative+1;
i=i+1;  goto LOOP;
FINISH:
```

逆波兰表示是：

positive 0 = negative 0 = ero 0 = i 1 = loop: i 100 <= 59 GOF A i[] 0 > 30 GOF positive positive 1 + = A i[] 0 == 41 GOF zero zero 1 + = A i[] 0 < 52 GOF negative negative 1 + = i i 1 + = loop GOL