编译原理

1. 文法和语言

\*对程序设计语言给出精确无二义的语法描述

1. 文法和语言的形式定义
2. 符号和符号串

\*字母表：元素的非空有穷集合

\*符号：字母表中的元素

\*符号串：符号组成的有穷序列

\*空串：ε

1. 符号串的运算

\*长度：符号串s的长度记为|s|，空串长度为0.

\*连接：x=ab，y=cd，则xy=abcd；εa = aε

\*方幂：符号串自身连接n次得到的符号串a^n

\*集合：集合A中所有元素都是某字母表上的符号串，则称A为该字母表的符号串集合。

\*集合乘积：若集合A={ab,cde}，B = {0,1}，则 AB ={ab1,ab0,cde0,cde1}

\*闭包：使用 \* 表示上的一切符号串（包括ε）组成的集合。



\*正闭包：上的除ε外的所有符号串组成的集合记为+ 。



1. 文法的定义

\*文法用来生成方式描述语言

定义：四元组（，，Ｐ，Ｓ）

：非终结符号集（变量集）

：终结符号集（与相比无公共元素）

Ｐ：规则的集合

Ｓ：称为开始符号的一个非终结符，至少要在一条产生式中作为左部出现。

规则（产生式、生成式）：形如α→β或α∷=β的(α，β)有序对

例：文法G=（VN，VT，P，S）VN = { S }, VT ={ 0, 1 }

P={ S→0S1, S→01 } S为开始符号

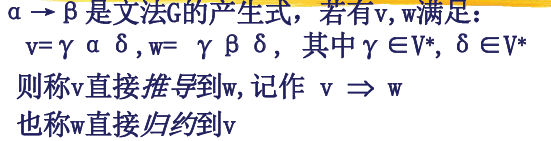
1. 文法的写法

\*元符号：

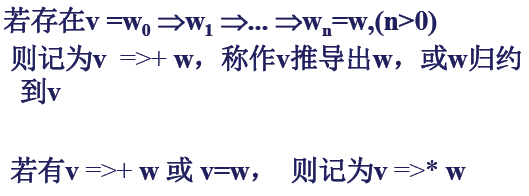
大写字母：非终结符 小写字母：终结符

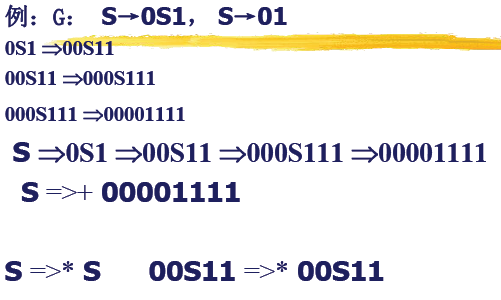
1. 推导的定义

直接推导：

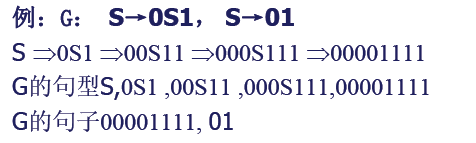


例：



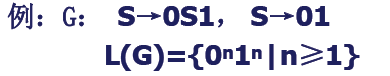


1. 句型、句子的定义



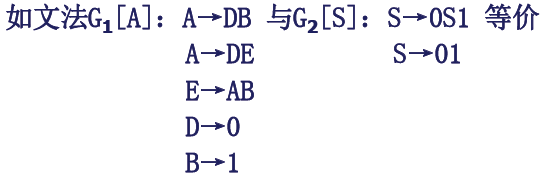
1. （文法生成的）语言的定义

\*由文法G生成的语言记为L(G)，是文法G的一切句子的集合。



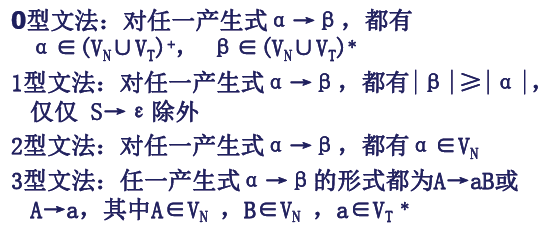
1. 文法的等价

若L(G1)=L(G2)，则称文法G1和G2是等价的



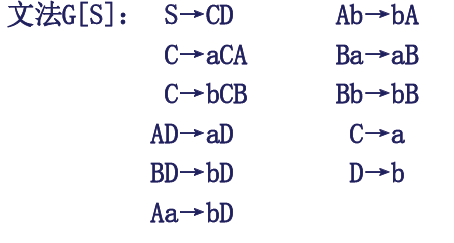
1. 文法的类型

1、

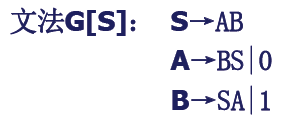


举例：

（1）1型（上下文有关）文法



（2）2型（上下文无关）文法

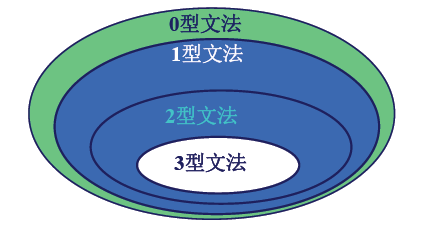


（3）3型文法



1. 四类文法关系

\*将产生式做进一步限制而定义的



1. 文法和语言的关系

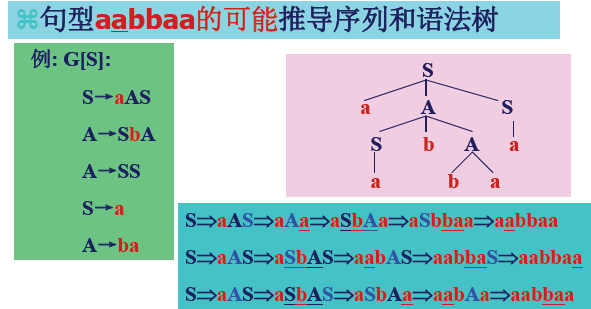
0型文法——0型语言

1型文法——1型（上下文有关）语言 ...

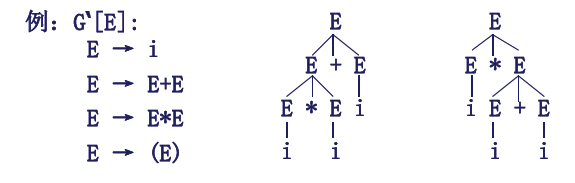
\*\*有不是1型语言的0型语言，有不是2型语言的1型语言，有不是3型语言的2型语言。

1. 上下文无关文法及其语法树

\*语法树：句型推导的直观表示，对于给定文法，对于G的任何句型都能构造与之关联的语法树（推导树）。



\*\*一个句型不一定只有唯一的一棵语法树，最右（左）推导也不一定唯一。



1. 规范推到与规范句型

最左（最右）推导：在推导的任何一步，都是对α的最左（最右）非终结符进行替换。

规范推导：最右推导

规范句型：由规范推导所得的句型

1. 二义文法
2. 定义：若一个文法存在某个句子对应两棵不同的语法树，则称这个文法是二义的

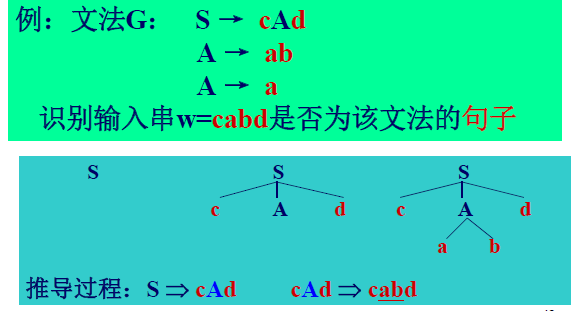
若一个文法存在某个句子有两个不同的最左（右）推导，则称这个文法是二义的

1. 文法的二义性和语言的二义性

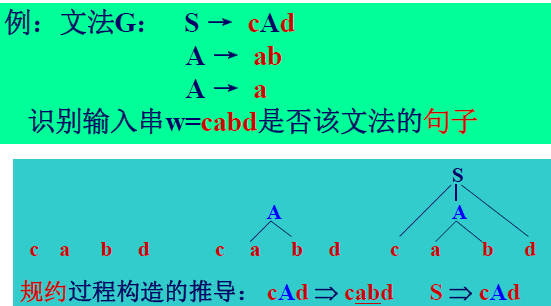
\*两个概念不相同，可能有两个不同的文法G和G’，其中G是二义的，但是却有L(G)=L(G’)，也就是说这两个文法所产生的语言是相同的。

\*文法存在二义性需将其改造成不含二义性的文法

1. （上下文无关文法）句型的分析
2. 定义：识别一个符号串是否为某文法的句型，是某个推导的构造过程。
3. 分析程序（识别程序）：完成句型分析的程序
4. 识别算法：分析算法
5. 从左到右的分析算法：总是从左到右识别输入的符号串，首先识别符号串中的最左符号，进而依次识别右边的一个符号，直到分析结束。
6. 自上而下的分析算法：从文法的开始符号出发，反复使用文法的产生式，寻找与输入符号串匹配的推导（为输入串寻找一个最左推导）。



1. 自下而上的分析算法：从输入符号串开始，逐步进行归约，直至归约到文法的开始符号（自底向上构造语法树）。



1. 文法实用中的一些说明
2. 限制化简文法：文法中不含有有害规则和多余规则

\*有害规则：引起文法二义性的规则

\*多余规则：文法中任何句子的推导都不会用到的规则

\*文法中不含有不可到达和不可终止的非终结符

①不可到达：文法中某些非终结符不在任何规则的右部出现

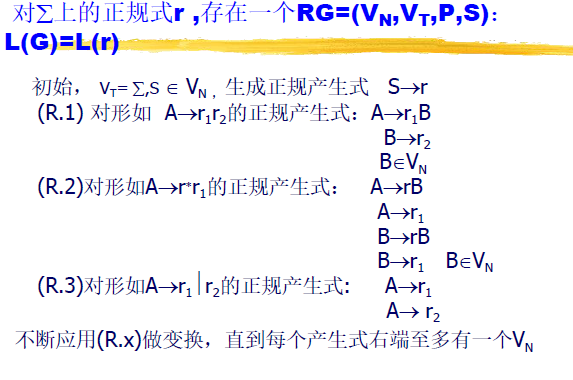
②不可终止：文法中某些非终结符，由它不能推出终结符号串

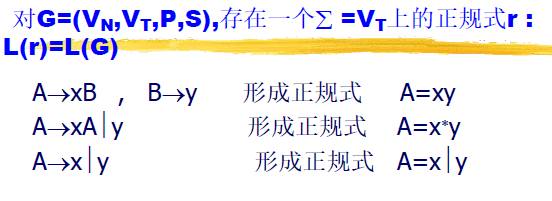
\*任意非终结符A在句子推导中出现需满足条件：

①A必须在某句型中出现

②必须能从A推出终结符号串t

1. 程序设计语言中的单词都能用正规式来定义





1. 三型文法和有穷自动机
2. 词法分析及其自动构造

一、什么是词法分析程序

\*逐个读入源程序字符，并按照构词规则切分成一系列单词

单词：语言中具有独立意义的最小单位，包括保留字、标识符、运算符、标点符号和常量。

单词的描述工具：正规表达式、正规文法

单词的识别系统：有穷自动机

1. 正规式（正则表达式）
2. 定义：说明单词的模式的一种重要的表示法，用以描述单词符号。

\*设字母表为，辅助字母表为，和都是上的正规式，所表示的正规集分别为{}和{}。

\*任何a∈，a是上的一个正规式，它所表示的正规集为{a}；

\*假定e1和e2都是上的正规式，它们所表示的正规集分别为L(e1)，L(e2)，那么，（e1），e1|e2，e1٠e2，e1\*也都是正规式，他们所表示的正规集分别为L(e1)，L(e1)∪L(e2)，L(e1)L(e2)和(L(e1))\*。

1. 正规式中的符号

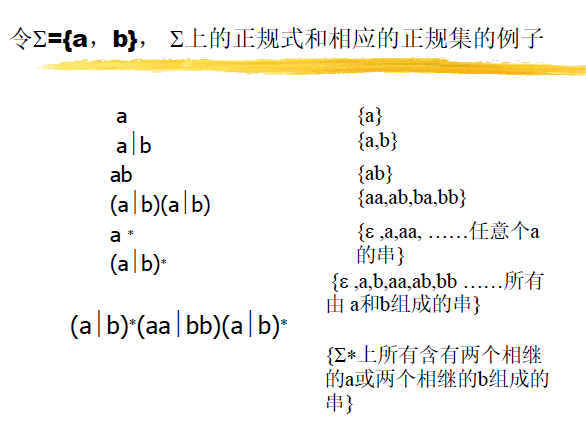
|：或

.：连接

\*：闭包（任意有限次的自重复连接）

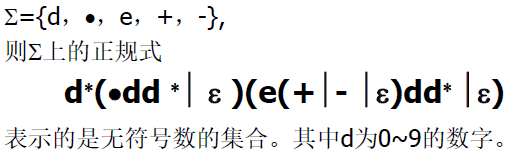
算符优先顺序：\*，.，|

结合规则：左结合



例一：字母打头的字母数字串：定义的正规集

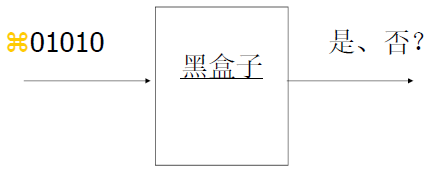
例二：无符号数集合：



例三：={0,1}上，至少有两个连续的0的0和1串的集合：（0|1）\*00（0|1）\*

例四：以11结尾的0和1串的集合：(0|1)\*11

1. 若两个正规式e1和e2所表示的正规集相同，则说e1和e2等价（e1=e2）
2. 重要性：程序设计语言中的单词都能用正规式来定义
3. 有穷自动机（黑盒子）



1. 定义：一种识别装置，能准确的识别正规集，即识别正规式所表示的集合。
2. 分类：确定的有穷自动机（DFA）、不确定的有穷自动机（NFA）

\*\*区别：NFA可以转换成DFA，NFA和DFA的主要区别在于：

1）DFA没有输入空串之上的转换动作，DFA是NFA的特例

2）对于DFA，一个特定的符号输入，有且只能得到一个状态，而NFA就有可能得到一个状态集；

\*\*联系：无论是NFA还是DFA，它们都是定义正规集合的工具。他们定义集合的能力相同。

1. 确定的有穷自动机（DFA）

\*定义：一个DFA是一个五元组：M=（K，，f，S，Z）

K：有穷集，每一个元素称为一种状态

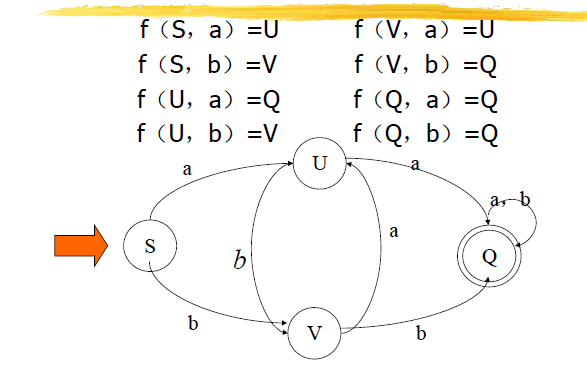
：有穷字母表，每个元素称为一个输入符号（输入符号表）

f：转换函数，输入符号后，从当前状态转入到下一状态。

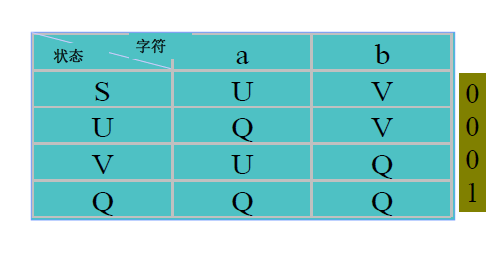
S：∈K，是唯一的初态

Z：∈K，终态

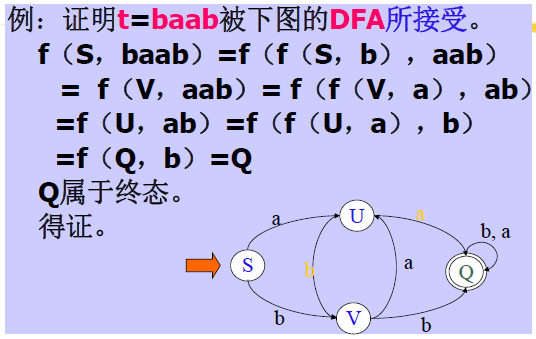
\*状态图表示



\*矩阵表示



\*例题：



\*\*DFA M所能接受的符号串的全体记为L(M).

结论：上一个符号串集V是正规的，当且仅当存在一个上的确定有穷自动机M，使得V=L(M)。

\*程序模拟

DFA M=（K，Σ，f，S，Z）的行为的模拟程序

K:=S；

c:=getchar;

while c<>eof do

{

K:=f(K,c);

c:=getchar;

};

if K is in Z then return (‘yes’)

else return (‘no’)

②不确定的有穷自动机（NFA）

\*定义：一个NFA是一个五元组：M=（K，，f，S，Z）

K：有穷非空集，每一个元素称为一种状态

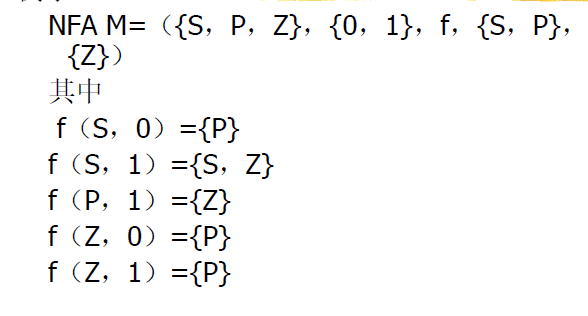
：有穷字母表，每个元素称为一个输入符号（输入符号表）

f：K x

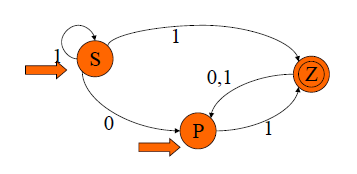
S：∈K，是唯一的初态

Z：∈K，终态

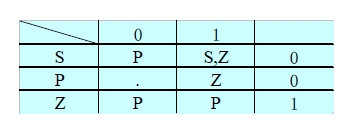
\*例：



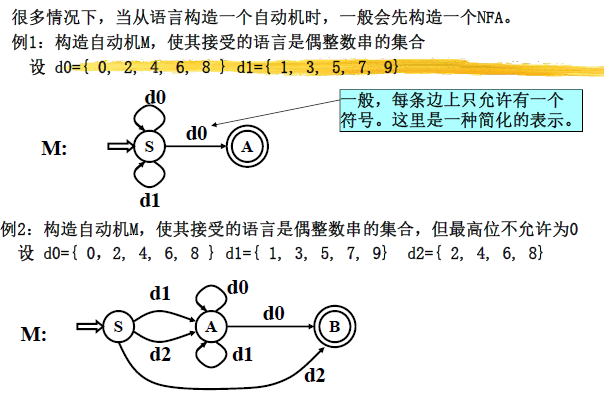
\*状态图表示

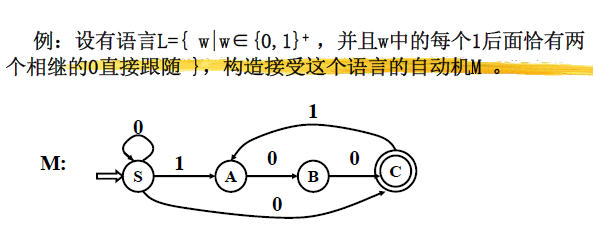


\*矩阵表示



\*应用：





\*\*Σ\*上的符号串t被NFA M接受若∈Σ\*，f(S0，t)=P，其中S0 ∈S，P∈Z，

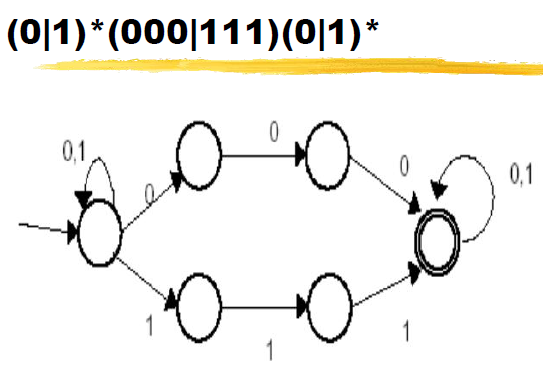
则称t为NFA M所接受（识别），或理解为：

对于Σ﹡中的任何一个串t，若存在一条从某一初态到某一终态的道路，且这条道路

上所有弧的标记字依序连接成的串(不理睬那些标记为ε的弧)等于t，则称t可为NFA M

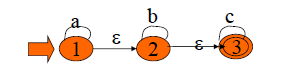
所识别(读出或接受)。若M的某些结既是初态又是终态，或者存在一条从某个初态到

某个终态的道路,其上所有弧的标记均为ε，那么空字可为M所接受。



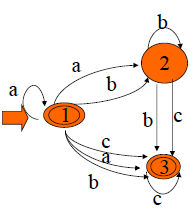
③具有ε转移的不确定有穷自动机

\*例：



\*定理：对任何一个具有ε转移的不确定的有穷自动机NFA N，一定存在一个不具有转移的不确定的有穷自动机NFA Ｍ ，使得L(M)=L(N)。

与上例等价的一个NFA：

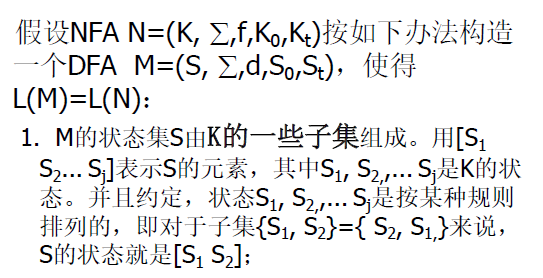


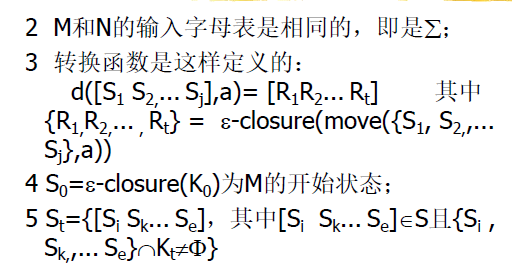
1. NFA确定化算法：NFA转换为DFA

\*\*任意一个NFA都可以等价的变换为DFA（DFA可以不唯一）

1. 子集法

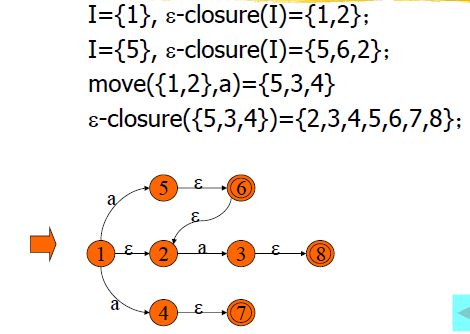
\*方法描述：





\*相关运算：

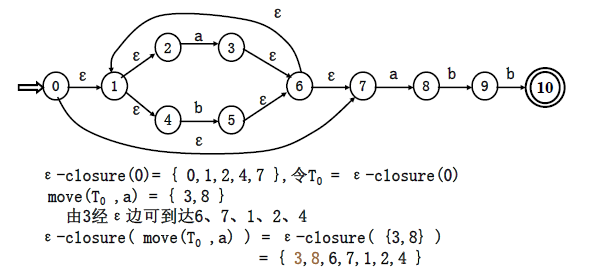
（1）状态集合I的ε-闭包，表示为ε-closure(I)，定义为一状态集，是状态集I中的任何状态S经任意条ε弧而能到达的状态的集合。状态集合I的任何状态S都属于ε-closure(I)。

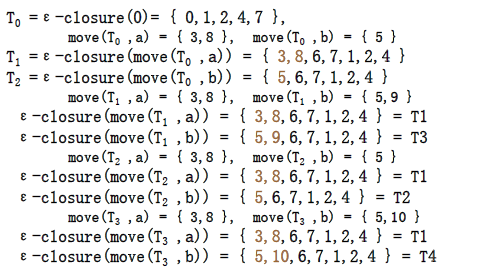


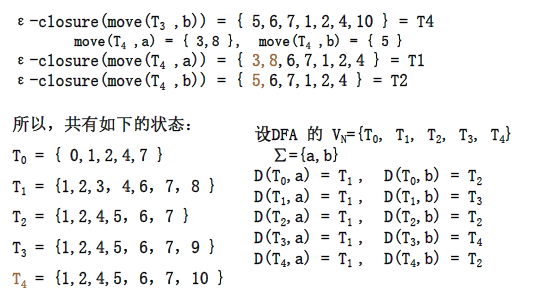
（2）状态集合I的a弧转换，表示为move(I,a)定义为状态集合J，其中J是所有那些可从I中的某一状态经过一条a弧而到达的状态的全体。

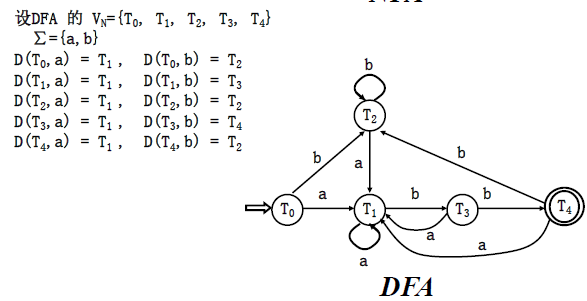
\*举例：

（1）

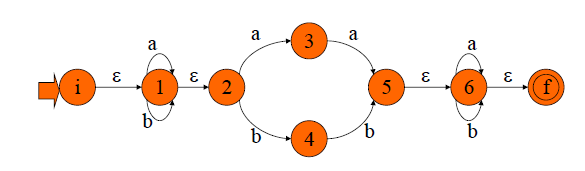


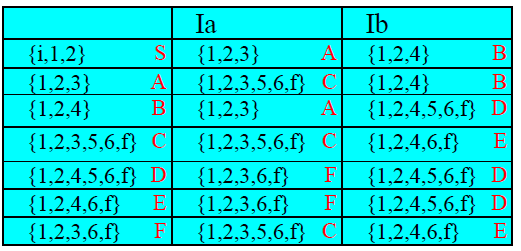




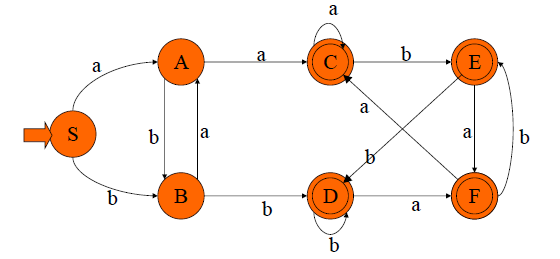


（2）





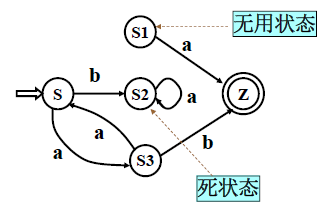
等价DFA：



1. 确定有穷自动机（DFA）的化简

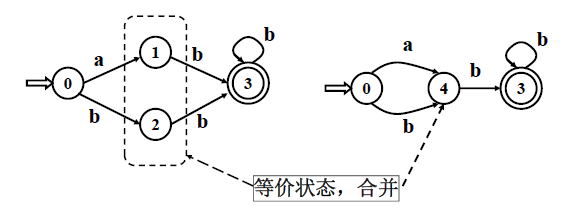
\*定义：化简就是对给定的确定自动机A1，构造另一个确定的自动机A2，使L(A2)=L(A1)。并且A2的状态个数不多于A1的状态个数。即是说，它没有多余状态并且它的状态中没有两个是互相等价的。一个有穷自动机可以通过消除多余状态和合并等价状态而转换成一个最小的与之等价的有穷自动机。

1. 多余状态：任何输出串不能到达的那个状态（无用状态），或者从这个状态没有通路到达终态（死状态）



1. 等价状态：兼容性——同是终态或同是非终态

传播性——从s出发读入某个a和从t出发读入某个a到达的状态等价。

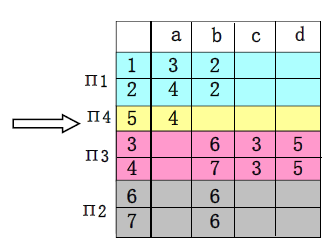
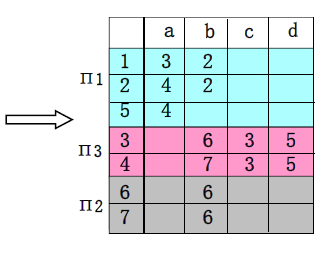
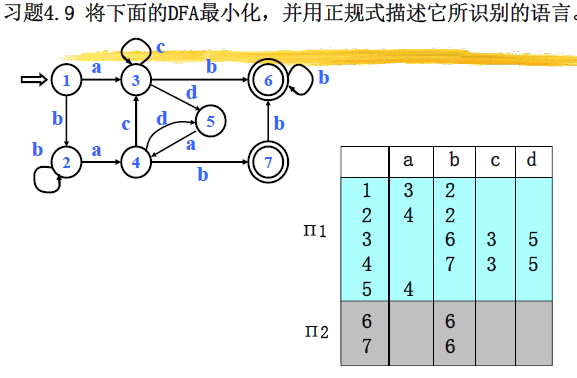


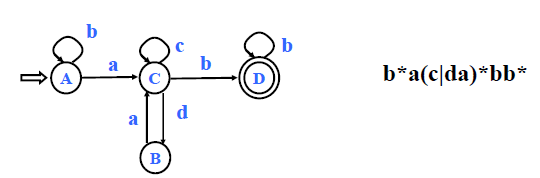
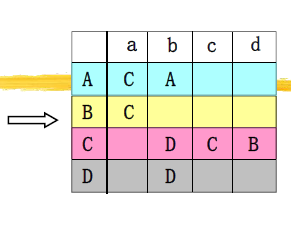
\*最小状态DFA（不计同构的情况下唯一）

1. 分割法

方法：把一个DFA的状态分成一些不相交的子集，使得任何不同的两子集的状态都是可区别的，而同一子集中的任何两个状态都是等价的.

举例：

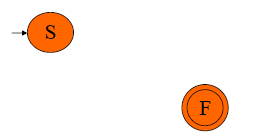




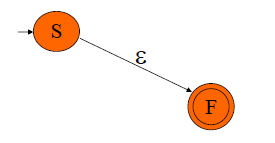
1. 词法分析程序的自动构造

\*\*基于有穷自动机和正规表达式的等价性

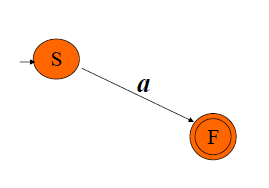
1. 对于正规式R=∅. ,构造的NFA



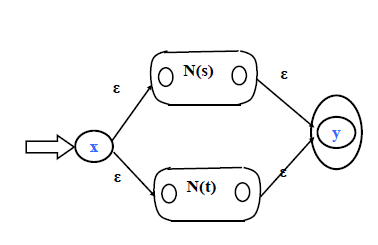
2、对于正规式R=ε,构造的NFA



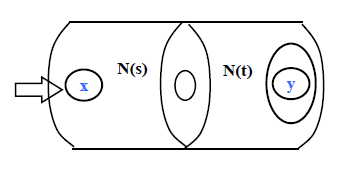
1. 对于正规式R=a ,构造的NFA



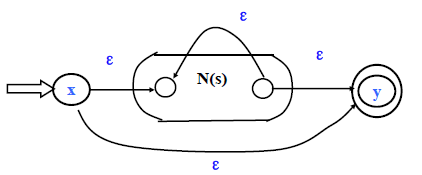
1. 对于正规式r, r= s|t构造的NFA



5、对于正规式r, r=st构造的NFA

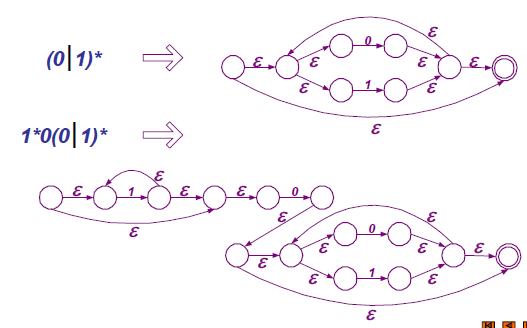
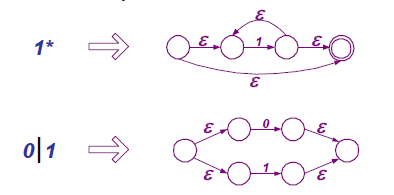


1. 对于正规式r, r=s\*构造的NFA

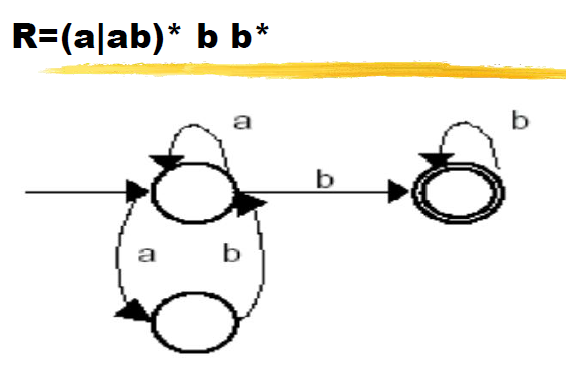


\*\*举例：

1. 正规表达式 1\*0(0|1)\*



1. 由正规式构造NFA



1. 自顶向下语法分析方法
2. 什么是语法分析

\*分析程序的语法结构

\*编译过程的核心部分

\*\*举例：

1. 存在语法错误：if( x==y { … }
2. 存在语义错误：int a=10; double d=2.32;a=a&d ;
3. 存在逻辑错误：

int i, a[5] ;

for( i=0; i<=5; i++ )

a[i]=i\*i ;

\*从形式语言来看：分析给定的符号串是否为文法的句子

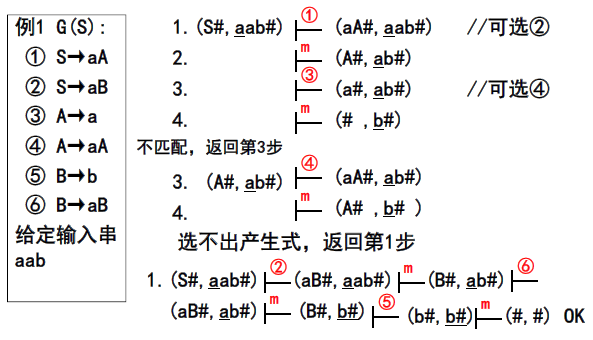
\*分析方法：

1. 推导法（自顶向下）

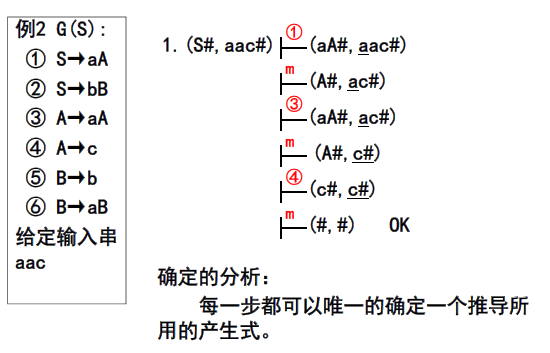
\*\*要解决的关键问题：如何选取推导所用的产生式

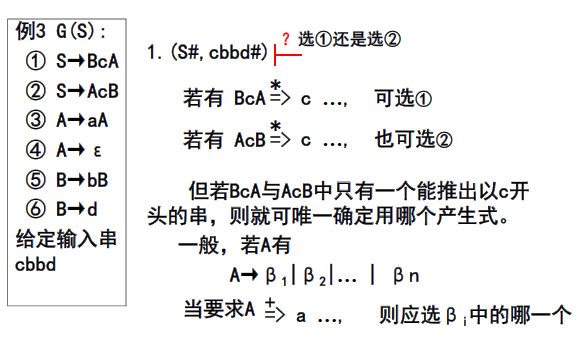
\*\*分类：

①不确定的自顶向下分析（回溯推导分析）

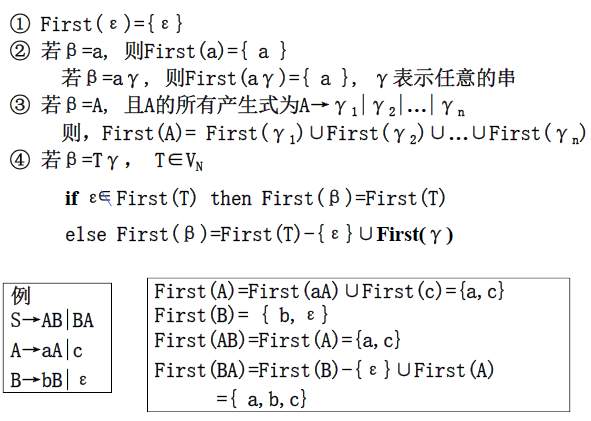


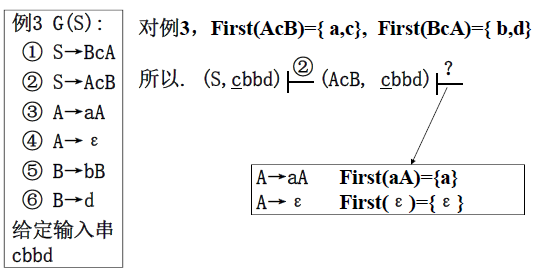
②确定的自顶向下分析

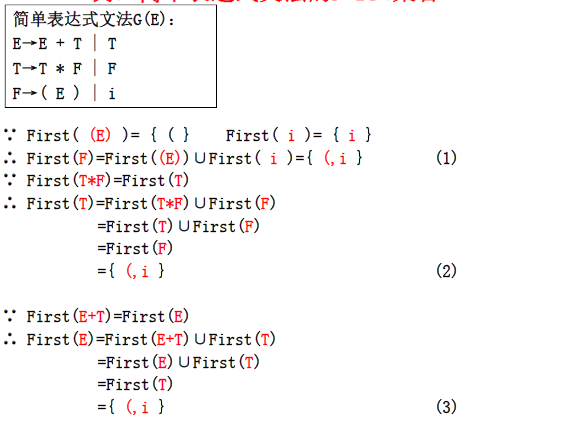




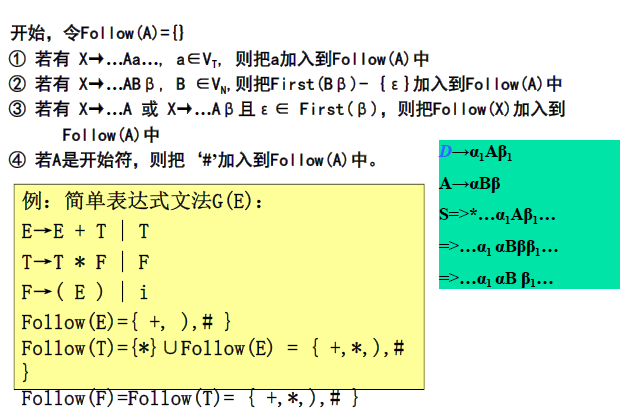
1. 归约法（自底向上）
2. 三个集合
3. First集合







1. Follow集合

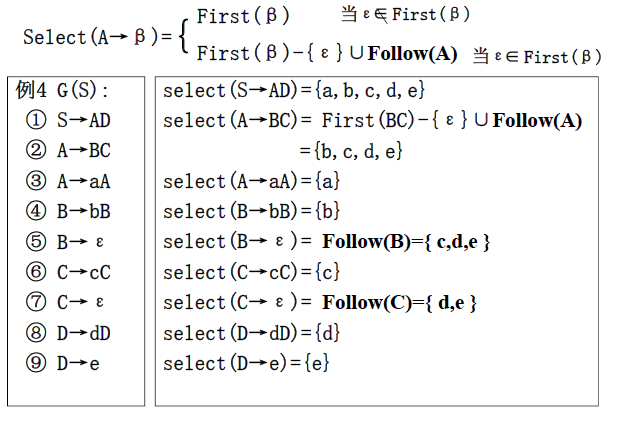


（1）直接收取：注意产生式右部的每一个形如“…Ua…”的组合，把a直接收入到Follow(U)中。

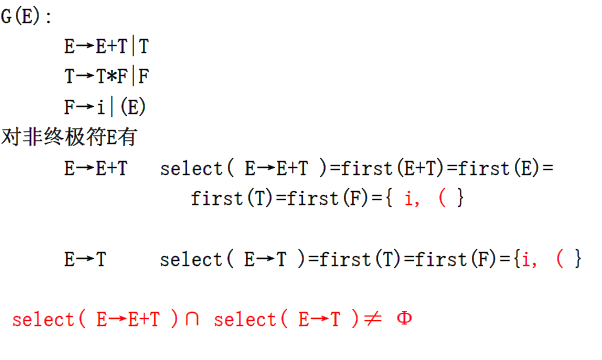
（2）直接收取：对形如“…UP…”(P是非终结符)的组合，把First(P)除ε直接收入到Follow(U)中。

（3）反复传送：对形如P－…U的产生式（其中U是非终结符），应把Follow(P)中的全部内容传送到Follow(U)中。(或 P－…UB且First(B)包含ε，则把First(B)除ε直接收入到Follow(U)中，并把Follow(P)中的全部内容传送到Follow(U)中)

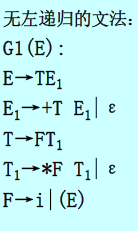
3、select集合



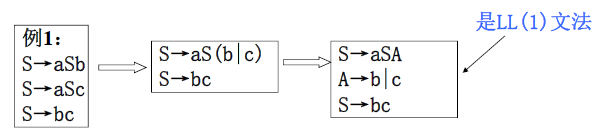
1. LL(1)文法
2. 定义：对文法G的每个非终极符A，设A有n个产生式，A→γ1|γ2|…|γn。若select(A→γi) ∩ select(A→γj)=Φ（1≤ i,j≤n 且i≠j），则称文法G是LL(1)的。
3. 特点：对于LL(1)文法可对其进行确定的推导分析
4. 区分
5. 含左公共因子的文法不是LL(1)文法
6. 带直接左递归的简单表达式文法不是LL(1)文法



1. 消去直接左递归的简单表达式文法是LL(1)文法



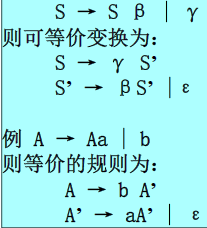
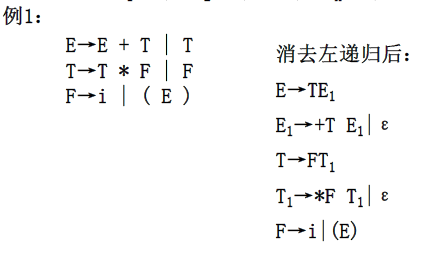
1. 非LL(1)文法构建等价LL(1)文法
2. 提左公因子（不是所有都可行）



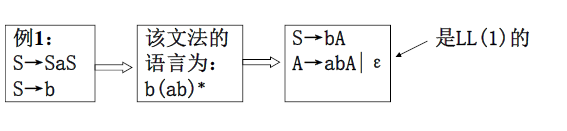
1. 消去直接左递归（不是所有都可行）

\*直接左递归产生式：S->Sβ

\*若S有一个直接左递归产生式，则S必须还有非直接左递归产生式

3) 重新构造等价的文法

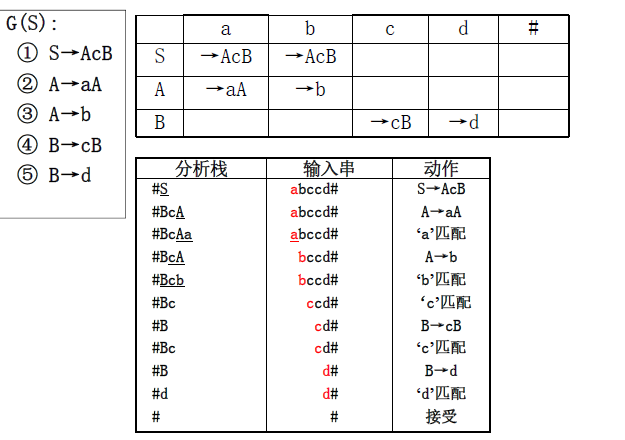


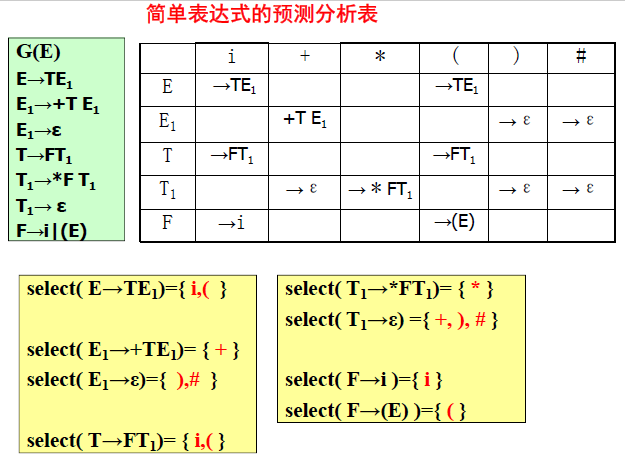
1. 二义性文法

\*定义：如果对于文法的同一个句子存在两个不同的语法树，则称该句子是二义性的。包含二义性句子的文法称为二义性文法。

\*\*注意：任何二义性文法都不是LL（1）文法

1. 关于LL(1)文法的一些结论
2. 任何LL(1)文法都无二义性
3. 左递归文法必然不是LL(1)文法
4. 存在一种算法，它能判定任意文法是否为LL(1)文法
5. 存在一种算法，它能判定任意两个ll(1)文法是否产生相同的语言
6. 不存在这样的算法，它能判定任意上下文无关语言能否由LL(1)文法产生。
7. 存在非LL(1)语言
8. 预测分析方法（确定的自顶向下分析方法）
9. 定义：又称LL(1)分析法，每一步分析为（栈顶符号，当前符号）—> （栈顶操作，输入符好操作）
10. 分析过程描述：

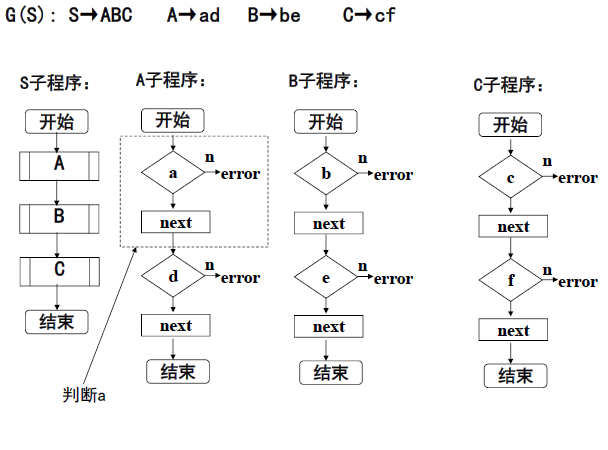
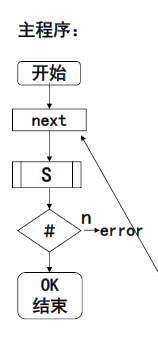




1. LL(1)分析中的一种错误处理方式
2. 发现错误：栈顶的终结符与当前输入符不匹配

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

1. 处理错误：使用递归子程序法
2. 递归子程序法
3. 定义：一种确定的自顶向下的分析方法，又称递归下降分析法
4. 特点
5. 只适用于LL(1)文法
6. 根据文法可直接写出处理子程序
7. 每个非终结符对应一个子程序



1. 基本思想

对S匹配串的分析可以分解为对n个子串的匹配分析。递归子程序中主程序的结构相同，都是读入第一个符号，然后调用开始符对应的子程序。

不论一个非终结符有多少个产生式，一个非终结符只对应一个子程序。、

1. 递归子程序中的接口条件

\*主程序和子程序，子程序和子程序之间的接口条件

1. 主程序读来第一个符号
2. 每个子程序入口时已读入它要分析的第一个符号
3. 每个子程序出口时已读入Follow符号
4. 本章小结
5. 语法分析是分析程序的语法结构
6. 语法分析是编译过程的核心部分
7. 语法分析方法：自底向上、自底向下
8. 对LL(1)文法的判别
9. 预测分析法和递归下降子程序法
10. 自底向上优先分析法
11. 定义

（1）短语

\* 如果Ｓ->\* αＡβ and A->+γ，则称γ是句型αγβ的相对于变量A的短语

1. 直接短语

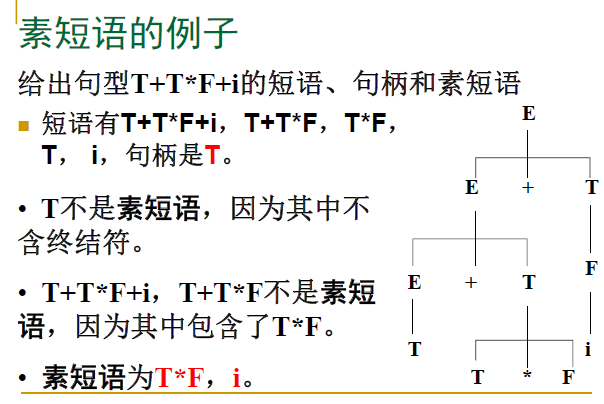
\* 如果Ｓ->\* αＡβ and A->γ，则称γ是句型αγβ的相对于变量A的直接（简单）短语

1. 句柄

\*最左直接短语叫做句柄，是自底向上语法分析中当前时刻需要归约的符号串

1. 素短语

\*素短语是一个短语，它至少含有一个终结符，而且除他之外不含有其他素短语。

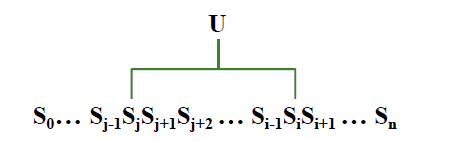


1. 概述

从推导的角度，分析过程中每次归约的都是最左边的简单短语；从语法树角度分析，以输入符号为树的叶子结点，从根节点向上构造语法树。

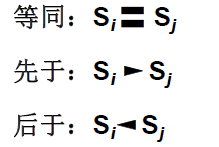
三、自底向上分析过程的实现

1. 移入-归约方法
2. 使用栈来存放归约得到的符号
3. 过程：在分析的过程中，识别程序不断地移入符号。移入的符号暂时存放在一个栈中。一旦在已经移入的（和归约得到的）符号串中包含了一个句柄时，将这个句柄归约成为相应的非终结符号。归约后将句柄从符号栈弹出，并将相应的非终结符号压入栈内，然后再检查栈内符号串是否形成新的句柄，若有则再进行规约，否则移入符号，一直到读入输入串的终止符为止。若栈中仅有终止符和开始符号，表示分析成功。
4. 归约中的动作
5. 移入
6. 归约
7. 接受：到达末尾后
8. 错误处理
9. 简单优先分析法
10. 基本思想：察看句型中相邻的两个符号。通过两个符号的关系判定出前一个符号是句柄的尾。然后，反向找出句柄的头。这样我们就找到了一个句柄。

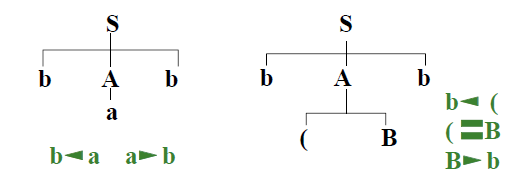


1. 优先关系

\*存在先后约束



\*例：

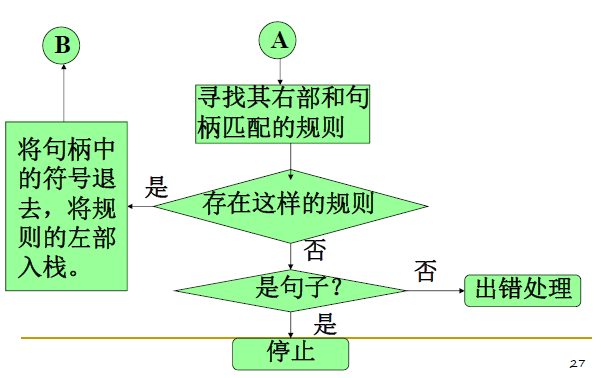
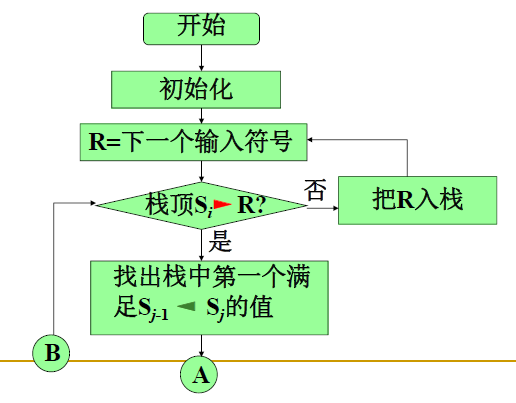


1. 优先矩阵

\*将矩阵作为关系的表示形式

\*冲突：当优先矩阵中出现值不唯一的元素时，文法不适合使用优先识别技术来识别句型。

1. 简单优先文法
2. 定义：某个文法满足字符表中的任意两个符号之间至多只有一种优先关系成立且任何两个规则式的右部不相同，则称该文法为简单优先文法。
3. 分析流程图



1. 例子



1. 局限性

\* 只适用简单优先文法

1. 算符优先分析法
2. 定义：在终结符号之间建立优先关系
3. 基本思想

对于算术表达式，只需要按照运算符之间的优先关系，就可以确定运算的顺序。不

需要考虑操作数就可以对表达式进行分析。

3、算符文法

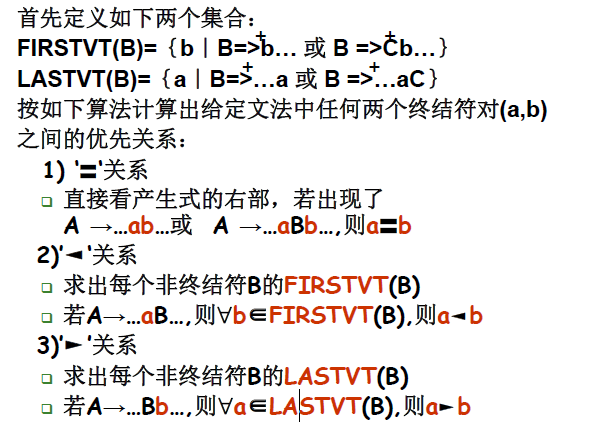
1. 定义：文法中没有形状如：A=...BC...的规则（BC为非终结符），则称该文法为算符文法
2. 性质

性质1：对于算符文法，不存在包含有相邻两个非终结符号的句型。

性质2：如果bA出现在句型中，其中b为终结符号，A为非终结符号，那么包含b的短 语也必然包含A。

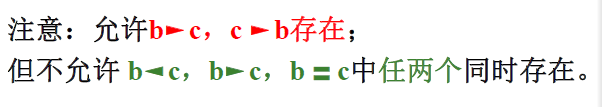
性质3：如果Ab出现在句型中，其中b为终结符号，A为非终结符号，那么包含b的短语也必然包含A。

1. 算符优先关系比较方法
2. 求firstvt集和lastvt集



1. 构造算符优先关系表
2. 算符优先文法
3. 定义：设有一个不含空串的算符文法G，如果其任意两个终结符号之间，算符文法关系最多只有一种关系成立，那么该文法G称为算符优先文法。

\*\*



1. 算符优先技术的说明

\*分析过程不考虑非终结符号

\*编译程序不考虑具体符号的名字，只考虑它们的意义

\*需要有处理素短语的语义处理子程序

\*处理过程中可以使用同一个符号来表示归约得到的非终结符号

1. 算符优先技术的实现
2. 当栈顶终结符的优先级“<”，“=”栈外的终结符的优先级时，移进
3. 栈顶终结符的优先级“>”栈外的终结符优先级时，表明找到了素短语的尾，再往前找头并进行规约。
4. 例



1. 算符优先分析法的局限性

一般语言的文法很难满足算符优先文法的条件，仅在表达式局部中应用; 由于忽略非终结符 在归约过程中的作用，可能 导致对错误的句子得到正确的归约。

1. 与简单优先技术的比较



1. 改进

\*提出了LR分析法，适应范围广，分析速度快，能准确、即时的指出出错位置

1. LR分析方法

\*通过求句柄逐步归约

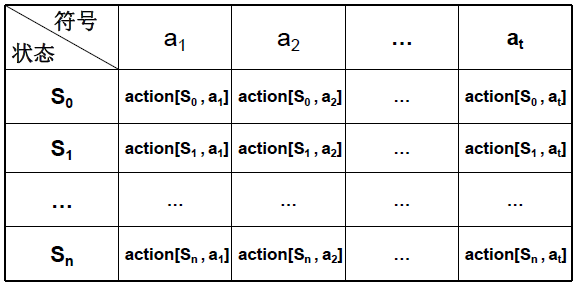
一、LR分析概述

1、LR(k)：根据当前符号栈中的符号串，向右顺序查看输入串的k个符号就可以唯一确定分析的动作是移进还是归约以及用哪个产生式归约。LR：从左向右扫描，自底向下归约

2、LR分析优缺点

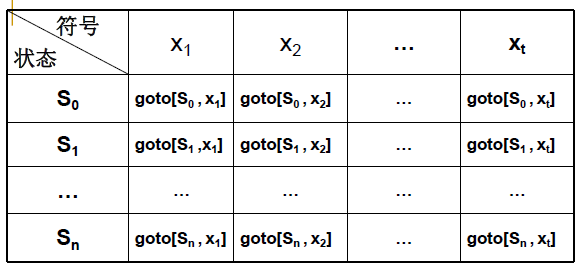
1. 适用于大多数上下文无关文法
2. 效率高
3. 报错及时
4. 工作量大
5. 可自动生成（YACC）
6. 逻辑结构（LR分析器组成）
7. 总控程序：驱动
8. 分析表/分析函数（ACTION & GOTO表）
9. 分析栈：符号栈和状态栈
10. 分析表组成

（1）动作表：移进（S）、归约（r）、接收（acc）、出错（error）



Si：状态栈顶 aj：输入符号

1. 状态转换表



Si：状态栈顶 xj：应转入的下一状态

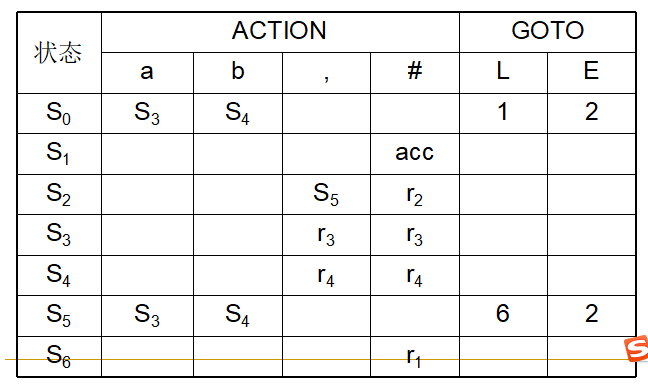
1. 不同种类的分析表
2. SLR分析表（简单LR分析表）
3. LR分析表（规范LR分析表）
4. LALR分析表（超前LR分析表）— YACC

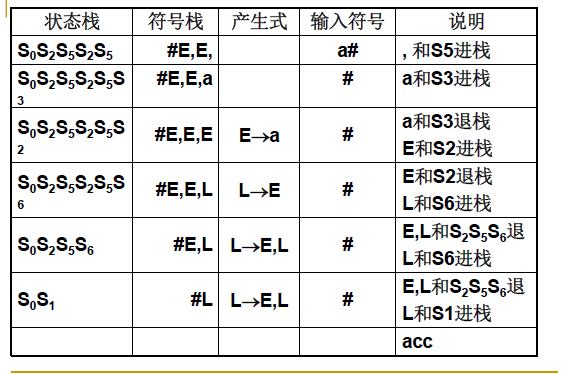
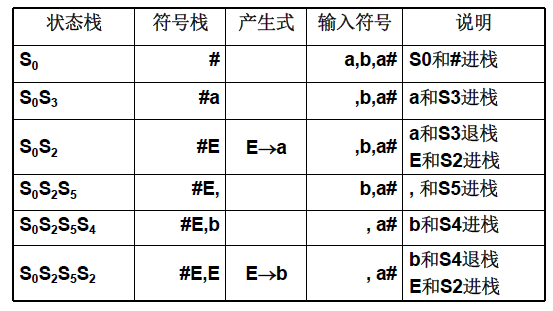
\*\*一个SLR文法必定是LALR文法和LR文法

1. LR分析过程

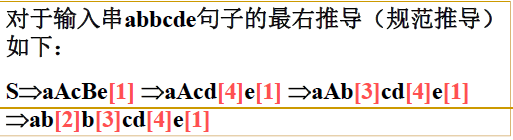
设文法为G[L] （假定已存在LR分析表）

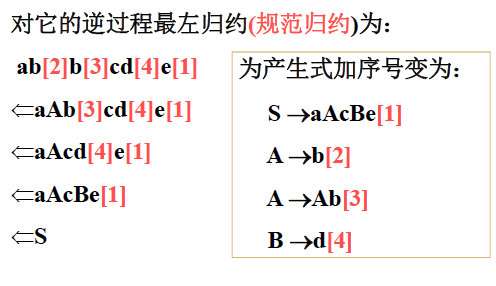
1. L ->E,L(2) L ->E(3) E ->a(4) E ->b



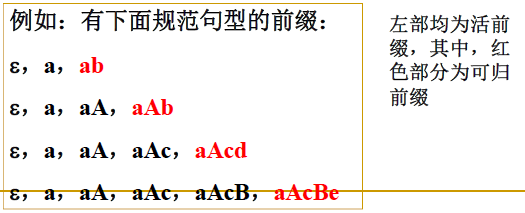


1. 活前缀与可归前缀





把每次采取归约动作前的符号串部分称为可归前缀。LR分析的关键就是识别何时到达可归前缀。



二、LR（0）分析

1. SLR(1)分析
2. LR(1)分析
3. LALR(1)分析
4. 语法制导的语义计算

程序是正确的：语法是正确的&语义是正确的

\*词法分析——>语法分析——>静态语义检查和翻译

1. 语义分析的概念
2. 基本任务：类型的确定、类型的检查、确认含义、其他语义检查
3. 语法制导翻译是目前最常用的语义分析技术：语法树建立与遍历同时完成
4. 属性文法

1、定义：一个属性文法是一个三元组

A=(G, V, F)

G: 一个上下文无关文法

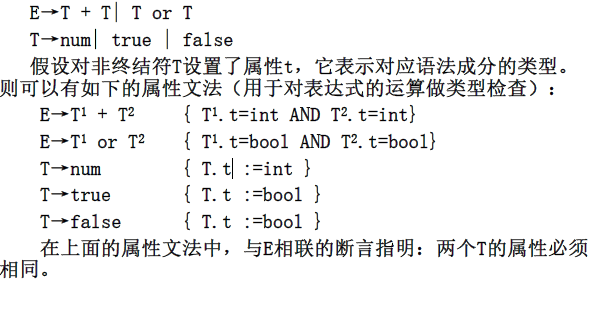
V: 一个属性的有限集合

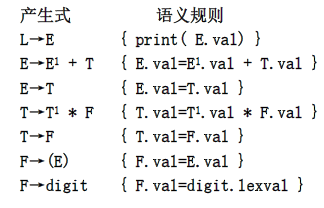
F: 关于属性的断言或谓词的有限集合。

每个属性与文法的某个非终结符或终结符相联。

每个断言与文法的产生式相联。

\*例：

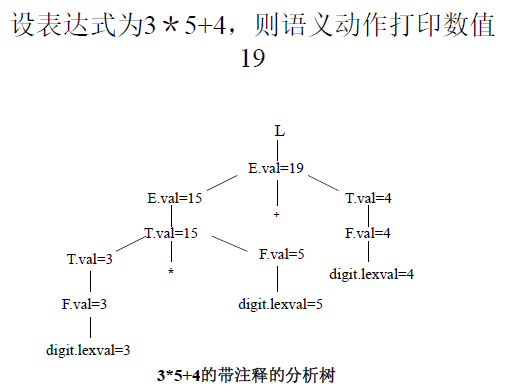




1. 分类：继承属性和综合属性

\*继承属性是 从上而下传递信息。 综合属性自下而上传递信息。 终结符号只有综合属性 非终结符号既有综合属性也可有继承属性

1. 综合属性：左部属性的值的计算来自于右部的非终结符



1. 继承属性
2. 语法制导翻译过程
3. 概念

\*对单词符号串进行分析，构造语法分析树，根据需要构造属性依赖图，遍历语法树，在语法树的各结点处按语义规则进行计算