第二章：语法树

一、画语法树（画语法树都是根据特定的句子（符号串）画）

①树的根是文法的开始符号

②每个产生式都能构成一个子树，子树的根是左部的非终结符号，子树的叶子节点是右部的符号

③最后子树的所有叶子节点，从左到右排列正好是上述特定的句子

画的过程实际上就是用各个部分去凑原字符串

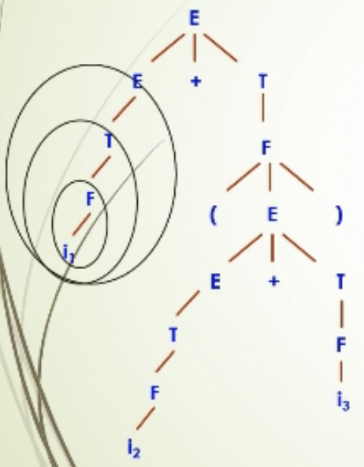
二、确定短语、直接短语、句柄

找短语：分离子树，子树的叶子节点若都是句子的符号，则这些符号都是短语

找直接短语：若短语全都是分离出的子树的子节点，那么短语也是直接短语（叶子节点和子根节点的父子关系，即一定是某个产生式的左部和右部）

找句柄：最左直接短语是句柄

【例】G[E]：E→E+T|T T→T\*F|F F→(E)|i，语法树如下：



i1是句型i+(i+i)相对于F或T或E的短语，i2是句型i+(i+i)相对于F或T或E的短语，i3是句型i+(i+i)相对于F或T的短语；i2+i3是句型i+(i+i)相对于E的短语，(i2+i3)是句型i+(i+i)相对于F的短语，i1+(i2+i3)是i+(i+i)相对于E的短语；总结：i1、i2、i3、i2+i3、(i2+i3)、i1+(i2+i3)是短语

i1、i2、i3是直接短语；i1是句柄

三、文法二义性判断

试图根据特定的句子画出两个不同的语法树，若同一个句型对应两棵不同的语法树，则文法是二义的

第三章：词法分析

一、NFA转DFA（NFA的确定化）

NFA：任一当前状态，输入某一符号后，转到下一个确定的状态

DFA：某一当前状态，输入某个符号后，可能转换到不同的状态

NFA→DFA的本质：将原来NFA的一个或多个后继状态作为DFA的一个状态，确保状态转移的确定性

方法：用矩阵转换表进行转换

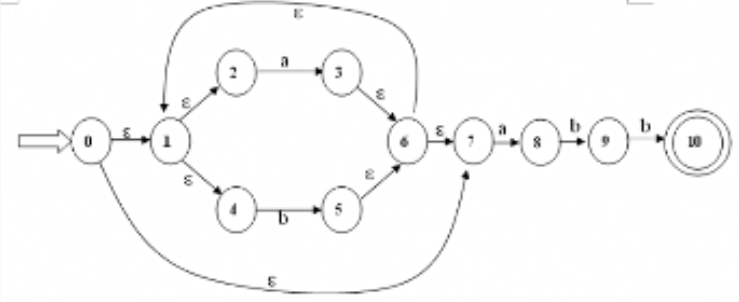
①若当前状态的后继状态是多个，将他们组成后继状态集，此后继状态集是DFA的一个状态

②包括原来NFA的终态的后继状态集，是DFA的终态

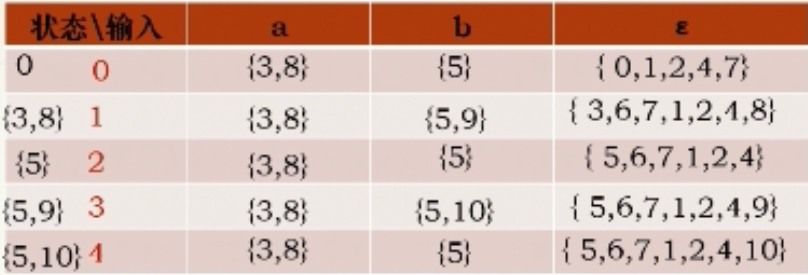
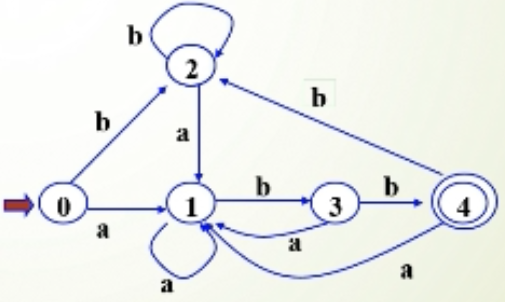
③由空串ε连接的状态，本质上是一个状态，考虑下一个状态时，考虑本质上相同的状态

状态转换表：

|  |  |  |  |
| --- | --- | --- | --- |
| 状态/输入 | 输入1 | 输入2 | ε |
| {起始状态} | A | B |  |
| 出现在A、B中但尚未出现在本列状态集合 |  |  |  |

【例】NFA：

状态转换表： DFA：

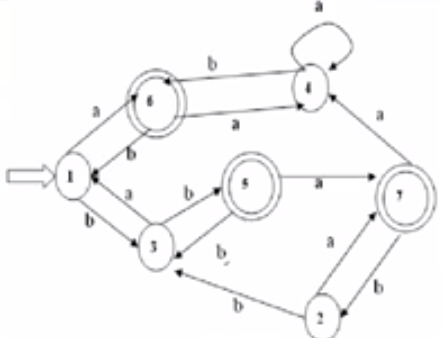
二、DFA的简化

方法：分割法

①首先将状态划分为两个子集：终态子集、非终态子集

②将上述两个子集不断分割，使同一个子集中的所有状态，在输入相同的符号时，跳转到相同的子集

③分割结束后，每个子集可任选一个状态代表子集，构成最简DFA

【例】左图中的DFA

1.划分为非终态集{1,2,3,4}和终态集{5,6,7}

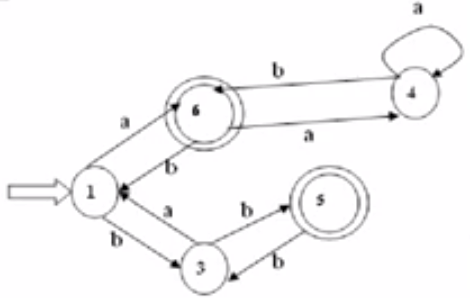
2.输入a，状态1、2离开非终态集，状态3、4未离开，因此划分为{1,2}、{3,4}

3.输入a，状态5离开终态集，状态6、7未离开，因此划分为{5}、{6,7}

4.输入a，状态1、2转移到6、7；输入b，转移到3、4；因此状态1、2等价

5.输入a，状态3离开集合{3,4}，状态4未离开，因此划分为{3}、{4}

6.输入a，状态6、7转移到{4}；输入b，转移到{1,2}；因此状态6、7等价

简化后的DFA：

第四章：LL(1)文法（自顶向下语法分析）

LL(1)是自顶向下的分析方法，充要条件：对每个非终结符A的两个不同产生式A→α，A→β，满足SELECT(A→α)∩SELECT(A→β)＝∅，其中α和β不能同时推出ε

一、给定文法判断该文法是不是LL(1)文法

（一）若文法含有左递归或左公因式，则必然不是LL(1)文法

1.改造左公因子：提取左公因子（例如A→αβ|αγ，则提取为A→α(β|γ)），将(…)用新引入的非终结符替换（A→α(β|γ)变为A→αA’，A’→ β|γ），若提取后仍有左公因子，则重复上述操作直至全部提取

2.改造左递归：分析左递归文法定义符号串的特点，分析并改造

【例】A→Aα1|Aα2|Aα3|…|Aαn|β1|β2|β3|…|βn

改造如下：A→A(α1,α2,α3|…|αn)|(β1|β2|β3|…|βn)，令α=α1,α2,α3|…|αn，β=β1|β2|β3|…|βn

则有：A→Aα|β，分析并写成正规式：r=βα\*（即β后面可以连接任意个α）

改写为：A→ΒA’ A’→ε|αA’（此时为右递归，消除了左递归），展开α和β即可

（二）不含有左递归和左公因式，也不一定是LL(1)文法，仍需继续判断

首先分析哪些符号能推出ε；计算每个非终结符号的First()集、所有产生式右部的First()集；对于能推出的非终结符号，还需要计算Follow()集；最后计算同一个非终结符号对应产生式的Select()集，并求其交集是否为空，若为空，则是LL(1)文法

【例】G[A]：A→aA’ A’→ABe|ε B→dB’ B’→ε|bB’

1.First()集：符号串最开始的非终结符号组成的集合，对于非终结符号，展开即可

First(aA’)={a} First(A)=First(aA’)={a}

First(ABe)=First(A)={a} First(ε)={ε} First(A’)={a,ε}

First(dB’)={d} First(B)={d}

First(bB’)={b} First(ε)={ε} First(B’)={b,ε}

2.Follow()集：后面跟随的第一个非终结符；仅需要对能推出ε的非终结符号求Follow()集，开始符号的Follow()集必含有“#”

A’仅出现在A之后，A为开始符号，因此A’的Follow()集中必然含有“#”，A的后面跟随的符号只有B，因此Follow(A’)中也包含First(B)，因此Follow(A’)=Follow(A)={#}∪First(B)={#,d}

B’仅出现在B→dB’和B’→bB’中，因此Follow(B’)=Follow(B)∪Follow(B’)，而在本题中Follow(B’)实质上也就是Follow(B)，因此Follow(B’)=Follow(B)={e}

3.Select()集：右部的First()集，若能推出ε，则结果应为(First()-{ε})∪Follow()；对每个产生式求Select()集，对于产生式有多个右部的，还需要求交集

Select(A→aA’)={a} Select(B→dB’)={d}

Select(A’→ABe’)={a} Select(A’→ε)=Follow(A’)={d,#}

Select(A’→ABe’)∩Select(A’→ε)=∅

同理，Select(B’→bB’)∩Select(B’→ε)={b}∩{e}=∅

由上述分析可知：改造后的文法是LL(1)文法

二、LL(1)文法的语法分析实现方法

（一）递归子程序法（递归下降分析法）

用产生式的右部检验时，右部出现一个非终结符执行一个对应的子程序，出现终结符号直接进行检验判断

实现思想：对于每个非终结符，根据产生式右部编写一个分析子程序（递归子程序Parse()），遇到非终结符时，调用相应的子程序；分析遇到终结符，直接MatchToken()判断输入的单词符号是否和该终结符号相同，相同则匹配成功，继续分析下一个单词符号，不同则进行错误处理。

优点：程序编写简单直观

缺点：递归调用，执行效率不高；深度的递归调用会影响语法分析的效率，速度慢，占空间多

【例】文法G[S]：S→AaS|BbS|d A→a B→ε|c

1.首先判断是否为LL(1)文法，求产生式的Select()集

SELECT(S→AaS)={a} SELECT(S→BbS)={b,c} SELECT(S→d)={d}

SELECT(A→a)={a}

SELECT(B→ε)={b} SELECT(B→c)={c}

因为SELECT(S→AaS)∩SELECT(S→BbS)∩SELECT(S→d)={a}∩{c,b}∩{d}=∅，

SELECT(B→ε)∩SELECT(B→c)={b}∩{c}=∅，所以该文法是LL(1)文法

2.编写递归子程序：该文法有3个非终结符，编写3个子程序

void ParseS() {

switch(lookahead) {

case a:

ParseA();

MatchToken(a);

ParseS();

break;

case b,c:

ParseB();

MatchToken(b);

ParseS();

break;

case d:

MatchToken(d);

break;

default:

print("syntax is error\n")

exit(0)

}

}

void ParseA() {

if(lookahead==a) {

MatchToken(a);

}

else {

print("syntax is error\n");

exit(0);

}

}

void ParseB() {

if(lookahead==c) {

MatchToken(c);

}

else if(lookahead==b) {}

else {

print("syntax is error\n");

exit(0);

}

}

（二）预测分析表法（预测分析法）

构造预测分析表和先入后出栈，预测分析程序执行入栈、替换、检验，消去匹配成功的终结符，进行检验判断

构造分析表M是一个二维表，表中的内容可用M[A,a]表示，A是非终结符，a为终结符或句子结束符#，M[A,a]是一条关于A的产生式，表明当用非终结符A向下推导，面临输入符a时，应采取的候选产生式

行填写所有非终结符；列填写所有非终结符及“#”；表格内部填写当前行的非终结符，扫描到列的终结符，应该选用的产生式，如果没有对应的产生式则不填

【例】G：E→E+T|T T→T\*F|F F→i|(E)

1.首先判断该文法不是LL(1)文法，需要改写为LL(1)文法，改写后，得：

E→TE’ E’→ε|+TE’

T→FT’ T’→ε|\*FT’

F→i|(E)

2.求每个产生式的SELECT()，判断改写后的文法是否为LL(1)文法

判断哪些符号能推出ε

First(F)=First(i)∪First(()={i,(}

First(T’)=First(ε)∪First(\*)={\*,ε}

First(T)=First(FT’)=First(F)={i,(}

First(E’)=First(ε)∪First(+)={+,ε}

First(E)=First(TF’)=First(T)={i,(}

只有E’和T’能推出ε，因此只需要求E’和T’的Follow()集

Follow(E’)=Follow(E)={#,)} E出现在两处：E是开始符号，产生式右部(E)

Follow(T’)= Follow(T)=((First(E’)-{ε})∪Follow(E)={+,),#}

求每个产生式的SELECT()集：

Select(E→TE’)=First(TE’)=First(T)={i,(}

Select(E→ε)=Follow(E’)={),#}

Select(E→+TE’)=First(+TE’)={+}

Select(T→FT’)=First(FT’)=First(F)={i,(}

Select(T’→ε)=Follow(T’)={+,),#}

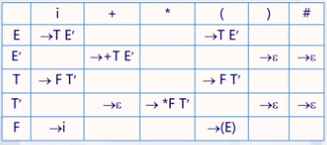
Select(T’→\*FT’)=First(FT’)={\*}

Select(F→i)=First(i)={i}

Select(F→(E))=First((E))={(}

相同的左部产生式的SELECT()集的交集为空，因此修改后的文法是LL(1)文法

3.构造预测分析表



第五章：LR(0)分析法（自底向上语法分析）

一、构造项目集并画出由项目集构成的DFA图

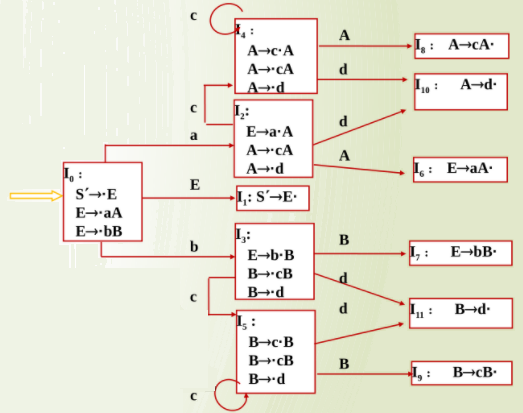
1.文法中的每个产生式的右部的某个位置加一个“·”构成项目

【例】给出拓广文法G’：(0)S’→E (1)E→aA (2)E→bB (3)A→cA (4)A→d (5)B→cB (6)B→d

所有项目：1.S’→·E 2.S’→E· 3.E→·aA 4.E→a·A 5.E→aA· 6.E→·bB 7.E→b·B

8.E→bB· 9.A→·cA 10.A→c·A 11.A→cA· 12.A→·d 13.A→d· 14.B→·cB 15.B→c·B

16.B→cB· 17.B→·d 18.B→d·

2.用CLOSURE()和GOTO()两个函数构造项目集

CLOSURE()实现已知项目的核，如何进行项目拓展

GOTO()实现已知一个项目集，如何推导其他项目集的核

第一个项目集：初态项目集I0，它的核是S’→·E

因为**“·”后面是非终结符号**，所以这个核**可以拓广**，拓广的项目是由E作为左部的产生式构成的“·”在右部最左侧的符号

拓广的结果：初态项目集I0，核为S’→·E，E→·aA E→·bB

把项目集I0中的每个项目的“·”向右移动一个位置，得到的就是新的项目集的核

I1 GOTO(I0,E)：核为S’→E·，无法继续拓展

I2 GOTO(I0,a)：核为E→a·A，A→·cA A→·d

I3 GOTO(I0,b)：核为E→b·B，B→·cB B→·d

用有向线段表示，箭头上方写拓展时“·”跨越的符号

二、构造LR(0)分析表

ACTION表：表示当前状态下所面临的输入符应做的动作是移进、归约、接收还是出错，列标号只包含终结符和“#”

GOTO转换表：表示在当前状态下面临非终结符应转向的下一个状态

当前状态的编号=DFA图中项目集的编号

1.移进操作：若项目集Ik中某一项目**A→α·aβ**，且转换函数GOTO(Ik,a)=Ij，置ACTION[k,a]=Sj

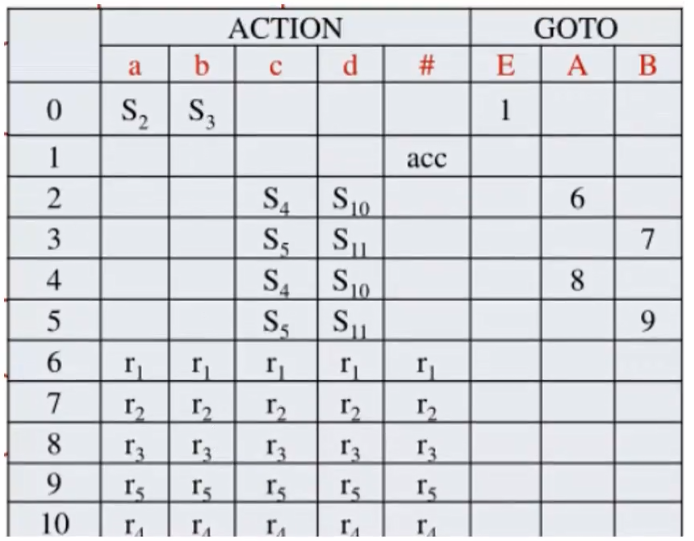
2.归约动作：若项目集Ik中某一项目**A→α·**，置ACTION表k行所有列[k,a]=rj，其中r表示归约动作，j是归约用的表达式编号

3.GOTO表：若项目集Ik中某一项目**S→α·Aβ**，GOTO(Ik,A)=Ij，置GOTO[k,a]=j

4.接受动作：若项目集Ik中某一项目**S’→S·**，置ACTION[k,#]为“acc”表示接受

5.除上述4种情况外，其余格子为空

结果如下：



11 r6 r6 r6 r6 r6

三、SLR(1)分析

给定一个LR(0)项目集I，I={X→α·bβ,A→γ·,B→δ·}，其中α,β,γ,δ是文法符号串，b∈VT，若满足{b}∩FOLLOW(A)∩FOLLOW(B)=∅，则按照如下决策进行操作：

1.若a=b，则根据X→α·bβ操作，执行移进

2.若a∈FOLLOW(A)，则根据项目A→γ·，用产生式A→γ进行归约

3.若a∈FOLLOW(B)，则根据项目B→δ·，用产生式B→δ进行归约

4.其他情况，报错

【例】

