设计说明

采用了栈的设计，对输入符号串进行处理.

**数据结构说明**

char L[200]="abcd#EAB"; ///列判断依据

int del[10]={0,2,2,2,1,2,1};//0-6号文法每个文法长度

char head[20]={'S','E','E','A','A','B','B'};

stack<int>con; ///状态栈

stack<char>cmp; ///符号栈

char cod[300]="0";///初始状态栈对应输出数组

int cindex = 0;

char sti[300]="#";///初始符号栈对应输出数组

int sindex = 0;

**思路**

计算LR（0）项目集 I 的闭包 CLOSURE(I)的算法：

function CLOSURE(I){

J:=I;

repeat for J 中的每个项目A->.B和产生式 B->γ 

do 若B->.γ不在 J 中，则加B->.γ到J中

until 上一次循环不再有新项目加到J中

return J };

初态 I0 = CLOSURE({S’->.S})

LR（0）FSM 的状态转移函数:

GO (I,X) = CLOSURE(J)

其中，I为LR(0) FSM 的状态（闭包的项目集），X 为文法符号，J={ A->αX.β|->α.Xβ∈I}

从 LR（0）FSM 的初态出发，应用上述转移函数，可逐步构造出完整的 LR（0）FSM

LR(0)计算 LR（0）FSM 的所有状态的集合:

设文法 G[S] 的增广文法为 G’[S’], 则 LR（0）FSM 的所有状态的集合 C 可由如下算法计算：

C:= { CLOSURE ({S’->.S}) }

Repeat

For C 中每一项目集 I 和每一文法符号X Do

if GO(I,X) 非空且不属于C

Then 把 GO(I,X) 放入C中

Until C 不再增大

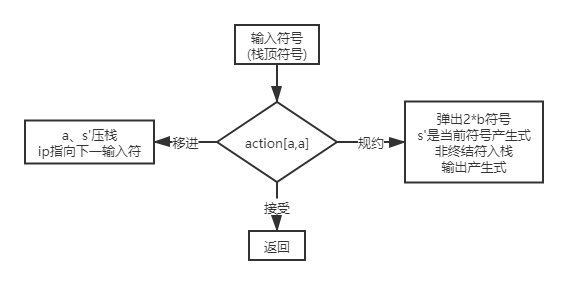
/\* 因此FSM中包含了状态集C和状态转换函数GO(I,X) \*/

分析表构造算法：

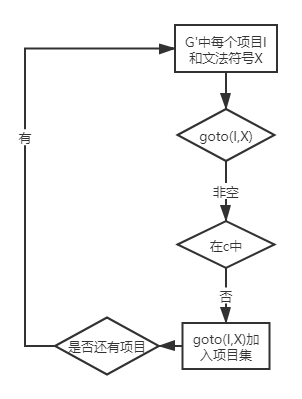
假定C={I0, I1,…，In},令状态Ik对应的LR(0)分析表的栈顶状态为k；令含有项目S’->.S的状态为I0, 因此0为初态。ACTION表项和GOTO表项可按如下方法构造：

* 若项目A->α.aβ属于Ik且GO (Ik,a)= Ij(图中), a 为终结符，则ACTION[k,a] 为“把状态j和符号a**移进**栈”，简记为“sj”;
* 若项目A->α.属于Ik,则对任何终结符a, 置ACTION[k, a]为“用产生式A->α进行**归约**”，简记为“rj”;其中，假定A->α为文法G的第j个产生式；
* 若项目S’->S.属于Ik, 则置ACTION[k,#]为“**接受**”，简记为“acc”;
* 若GO(Ik,A)=Ij,A为非终结符,则置GOTO(k,A)=j;
* 分析表中凡不能用上述规则填入信息的空白格均置上“**出错**标志”。

算法实现流程图：



LR0驱动程序流程图

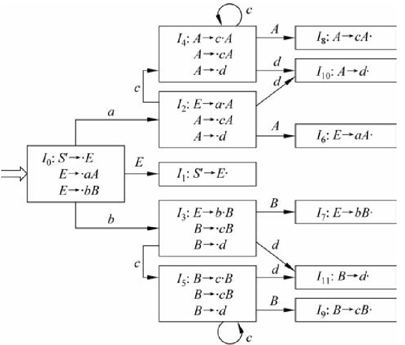


LR0项目集的构造

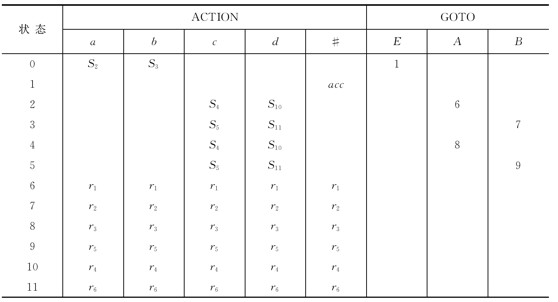
对于文法G[S]：

1. S->E (1)E->aA (2) E->bB (3) A->cA (4) A->d (5) B->cB (6)B->d

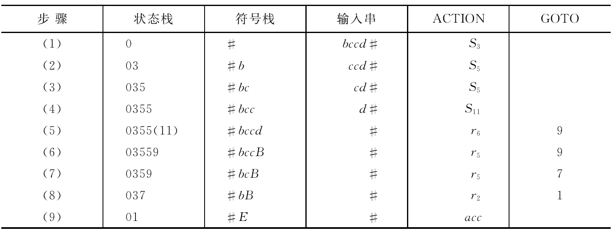
构造出它的DFA:



而后可根据上图得出LR0分析表：

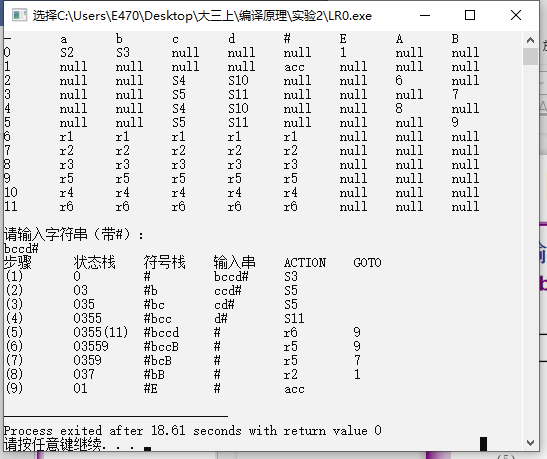


根据分析表对输入串”bccd#”给出分析过程：



**执行结果**

屏幕输出



output.txt

