NFA转化为DFA的原理和算法思路

1. NFA转化为DFA的基本原理——子集构造法

设NFA Mn=(Q,T,,q0,F0),构造一个等效DFA Md=(Qd,T,,q0d,Fd)，其中：

Qd是Q的幂集，Qd中元素为[q1,q2,…,qn]，{q1,q2,…,qn}Q;

q0d=[q0];

Fd的每个状态包含F0中的一个状态；

定义为：

([q1,q2,…,qn],a)=[p1,p2,…,pn]

当且仅当：

({q1,q2,…,qn},a)= {p1,p2,…,pn}

1. 算法思路

先读取输入，确定起始状态和终止状态，存储状态转移函数；生成一个DFA状态表示NFA中的起始状态，并将该状态放入队列中，并标记为已访问。随后每次取出队列的队首元素x，设状态x输入0、1时，分别生成状态y、z，若y、z未被访问过，则生成一个新状态表示它，并将该状态放入队列，并标记为已访问。

1. 伪代码
2. 参考数据结构和函数接口

#define nfaStateNum 10 //nfa的状态数量，假设为10

int nfaDeduce0[nfaStateNum][nfaStateNum] //二维数组，其中nfaDeduce[i]是一个一维数组，存储在NFA中通过状态qi输入0能到达的状态

int nfaDeduce1[nfaStateNum][nfaStateNum] //二维数组，其中nfaDeduce[i]是一个一维数组，存储在NFA中通过状态qi输入1能到达的状态

int dfaStateNum //dfa的状态数量，初始为0

struct dfaStateNode{

bool isStart,isFinal

int nfaState[nfaStateNum] //存储一个dfa状态中包含的nfa状态，以-1结尾

}dfaState[2^ nfaStateNum]

int dfaDeduce0[2^ nfaStateNum] //存储DFA中qi输入0时能到达的状态

int dfaDeduce1[2^ nfaStateNum] //存储DFA中qi输入1时能到达的状态

queue<dfaState> q; //存储DFA状态的队列

void getStatus(String str,int[] state) //从字符串中提取其包含的状态，通过数组state返回

int findStatus(int[] state) //判断一个包含多个NFA状态的数组是否在DFA状态中出现过，若无则返回-1，否则返回其下标

void deduce(int[] fromState,int[] toState,int input) // fromState数组输入input时推倒出toState数组，通过toState返回

1. 输入以及预处理

input(“\t\t0\t1\n”) //读入第一行

string str

while(input(str)!=EOF) //每次读一个字符串，遇到\t或\n结束，此处读每行中的第一个字符串

int state[nfaStateNum]，toState[nfaStateNum] //包含多个NFA状态的数组

getStatus(str,state)

if(str[0] equal ‘(‘) //如果读到小括号，说明此行为起始或终止状态

if(str[1] equal ‘s’)

//生成一个DFA状态表示NFA中的起始状态

dfaState[dfaStateNum].isStart=1

mamcpy(dfaState[dfaStateNum].nfaState,state,sizeof(state))

else

//生成一个DFA状态表示NFA中的终止状态

dfaState[dfaStateNum].isFinal=1

mamcpy(dfaState[dfaStateNum].nfaState,state,sizeof(state))

else

//生成一个DFA状态表示NFA中的状态

mamcpy(dfaState[dfaStateNum].nfaState,state,sizeof(state))

//状态加入队列

q.push(dfaState[dfaStateNum++])

input(str) //读入每行中的第二的字符串

getStatus(str,toState)

memcpy(nfaDeduce0[state[0]],stateTo,sizeof(toState)) //存储状态转移函数

input(str) //读入每行中的第三的字符串

getStatus(str,toState)

memcpy(nfaDeduce1[state[0]],stateTo,sizeof(toState)) //存储状态转移函数

1. 循环处理队首元素

dfaStateNode curState=q.front()

q.pop()

int toState[nfaStateNum]

deduce(curState.nfaState,toState,0)

if(findStatus (toState) equal -1) //若未访问过

//生成一个DFA状态

mamcpy(dfaState[dfaStateNum].nfaState, toState,sizeof(toState))

q.push(dfaState[dfaStateNum])

//存储DFA状态转移表

dfaDeduce0[curState.id]= dfaStateNum++

else

dfaDeduce0[curState.id]= findStatus (toState)

deduce(curState.nfaState,toState,1)

if(findStatus (toState) equal -1) //若未访问过

//生成一个DFA状态

mamcpy(dfaState[dfaStateNum].nfaState, toState,sizeof(toState))

//存储DFA状态转移表

dfaDeduce1[curState.id]= dfaStateNum++

q.push(dfaState[dfaStateNum++])

else

dfaDeduce1[curState.id]= findStatus (toState)

1. 输出

略