

**院 系：计 算 机 学 院**

**实验课程：编译原理**

**实验项目：XLEX生成器**

**指导老师：黄煜廉**

**专 业：计算机科学与技术**

**班 级：2016级 4 班**

**学 生：陈铭海**

**学 号：20162180198**

**华南师范大学教务处**

**XLEX生成器**

**一．实验目的**

设计一个应用软件，以实现将正则表达式-->NFA--->DFA-->DFA最小化-->词法分析程序。

1. **实验内容**

1.要提供一个源程序编辑界面，让用户输入正则表达式（可保存、打开源程序）

2.需要提供窗口以便用户可以查看转换得到的NFA（用状态转换表呈现即可）

3.需要提供窗口以便用户可以查看转换得到的DFA（用状态转换表呈现即可）

4.需要提供窗口以便用户可以查看转换得到的最小化DFA（用状态转换表呈现即可）

5.需要提供窗口以便用户可以查看转换得到的词法分析程序（该分析程序需要用C语言描述）

**三．设计思路**

**1.正则表达式到NFA**

(1). 先让用户输入或选择文件导入正则表达式

(2). 获取正则表达式后先判断是否含有禁止符号(即：~，本实验让~作为空)

(3). 将正则表达式中的连接运算显示使用“.”来表示

(4). 将正则表达式转为逆波兰表达式(即后缀表达式)

(5). 使用汤姆森构造法(Thompson Construction)，将正则表达式构造为NFA

**2.NFA到DFA**

(1). 将获取到的NFA使用子集构造法(Subset Construction)转为DFA

(2). 使用二维数组记录下闭包运算的各个子集(每个元素都是NFA对应的标号)，以便输入状态转移表。其次记录下各个节点的状态

**3.DFA到最小DFA**

(1). 使用 上一步记录下来的各个子集的二位数组和记录下来的状态表，对逐个状态进行比较，判断是否可以进行合并，从而进行最小化DFA。

**4.最小DFA生成词法分析程序**

(1). 使用Switch-Case的形式生成词法分析程序。

**四．程序源代码**

程序的主要源代码如下：（请看文件中的XLexGeneratorDlg.h和 XLexGeneratorDlg.cpp）

1. **用Thompson构造法构造NFA**

// 用Thompson构造法构造NFA

void CXLexGeneratorDlg::ThompsonConstruction()

{

int i, j;

char ch;

int s1, s2;

LStack<int> \*states = new LStack<int>();

states->Clear();

if (strlen(post) < 1)

{

MessageBox(L"No Valid Regular Expression Found!");

exit(1);

}

NFATable->SetValue(0, 0);

i = 1;

j = 0;

ch = post[j];

while (ch != '\0')

{

if (ch == '.')

{

s2 = states->Pop();

int temp1 = states->Pop();

int temp2 = states->Pop();

s1 = states->Pop();

NFATable->InsertEdgeByValue(temp2, temp1, '~');

states->Push(s1);

states->Push(s2);

}

else if (ch == '|')

{

s2 = states->Pop();

int temp1 = states->Pop();

int temp2 = states->Pop();

s1 = states->Pop();

NFATable->InsertVertex(i);

NFATable->InsertVertex(i + 1);

NFATable->InsertEdgeByValue(i, s1, '~');

NFATable->InsertEdgeByValue(i, temp1, '~');

NFATable->InsertEdgeByValue(temp2, i + 1, '~');

NFATable->InsertEdgeByValue(s2, i + 1, '~');

s1 = i;

s2 = i + 1;

states->Push(s1);

states->Push(s2);

i += 2;

}

else if (ch == '\*')

{

s2 = states->Pop();

s1 = states->Pop();

NFATable->InsertVertex(i);

NFATable->InsertVertex(i + 1);

NFATable->InsertEdgeByValue(i, i + 1, '~');

NFATable->InsertEdgeByValue(s2, s1, '~');

NFATable->InsertEdgeByValue(i, s1, '~');

NFATable->InsertEdgeByValue(s2, i + 1, '~');

s1 = i;

s2 = i + 1;

states->Push(s1);

states->Push(s2);

i += 2;

}

else

{

NFATable->InsertVertex(i);

NFATable->InsertVertex(i + 1);

NFATable->InsertEdgeByValue(i, i + 1, ch);

s1 = i;

s2 = i + 1;

states->Push(s1);

states->Push(s2);

i += 2;

}

j++;

ch = post[j];

}

s2 = states->Pop();

s1 = states->Pop();

NFATable->InsertEdgeByValue(0, s1, '~');

if (!states->IsEmpty())

{

MessageBox(L"Some error in your input string!");

exit(1);

}

NFAStatesNumber = s2 + 1;

}

1. **用子集构造法构造DFA**

// 用子集构造法构造DFA

void CXLexGeneratorDlg::SubsetConstruction()

{

int i, j, k;

DStatesNumber = 0;

DtranNumber = 0;

// 初始化DStates和Dtran两个构造表及AcceptStates数组

DStates = (int\*\*)(new int\*[NFAStatesNumber + 1]);

for (i = 0; i < NFAStatesNumber + 1; i++)

{

DStates[i] = new int[NFAStatesNumber + 1];

}

Dtran = (int\*\*)(new int\*[NFAStatesNumber + 1]);

for (i = 0; i < NFAStatesNumber + 1; i++)

{

Dtran[i] = new int[edgeNumber + 1];

}

for (i = 0; i < NFAStatesNumber + 1; i++)

{

for (j = 0; j < edgeNumber + 1; j++)

{

Dtran[i][j] = -1;

}

}

AcceptStates = new int[NFAStatesNumber + 1];

for (i = 0; i < NFAStatesNumber + 1; i++)

{

AcceptStates[i] = 0;

}

// 调用闭包函数和移动函数构造DStates和Dtran两个表及AcceptStates数组

int \*T = new int[NFAStatesNumber + 1];

int \*temp = new int[NFAStatesNumber + 1];

T[0] = 0;

T[1] = -1;

T = NFATable->Closure(T);

DStates[DStatesNumber] = T;

DStatesNumber++;

k = 0;

while (k < DStatesNumber)

{

for (i = 0; edge[i] != '\0'; i++)

{

temp = NFATable->Closure(NFATable->Move(T, edge[i]));

if (temp[0] != -1)

{

for (j = 0; j < DStatesNumber; j++)

{

if (CompArray(temp, DStates[j]) == 1)

{

Dtran[k][i] = j;

break;

}

}

if (j == DStatesNumber)

{

DStates[DStatesNumber] = temp;

Dtran[k][i] = DStatesNumber;

DStatesNumber++;

}

}

}

k++;

T = DStates[k];

}

DtranNumber = k;

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

{

for (j = 0; DStates[i][j] != -1; j++)

{

if (DStates[i][j] == NFAStatesNumber - 1)

{

AcceptStates[i] = 1;

break;

}

}

}

}

1. **用子集构造法构造DFA**

// 最小化Dtran表，即最小化DFA

int CXLexGeneratorDlg::MinimizeDFAStates(int \*\*Dtran, int \*AcceptStates, int DtranNumber, int edgeNumber)

{

int h, i, j, k, l;

for (i = 0; i < DtranNumber - 1; i++)

{

for (j = i + 1; j < DtranNumber; j++)

{

if (AcceptStates[i] == AcceptStates[j])

{

for (k = 0; k < edgeNumber; k++)

{

if (Dtran[i][k] != Dtran[j][k])

{

break;

}

}

if (k == edgeNumber)

{

for (l = j; l < DtranNumber - 1; l++)

{

for (k = 0; k < edgeNumber; k++)

{

Dtran[l][k] = Dtran[l + 1][k];

}

AcceptStates[l] = AcceptStates[l + 1];

}

for (l = 0; l < DtranNumber - 1; l++)

{

for (k = 0; k < edgeNumber; k++)

{

if (Dtran[l][k] == j)

{

Dtran[l][k] = i;

}

}

}

for (h = j; h < DtranNumber; h++)

{

for (l = 0; l < DtranNumber - 1; l++)

{

for (k = 0; k < edgeNumber; k++)

{

if (Dtran[l][k] == h + 1)

{

Dtran[l][k] = h;

}

}

}}

DtranNumber--;

j--;

}

}

}}

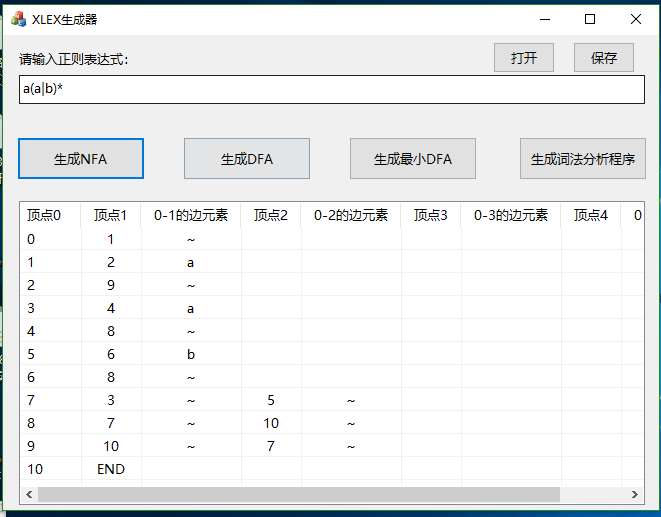
return DtranNumber;

}

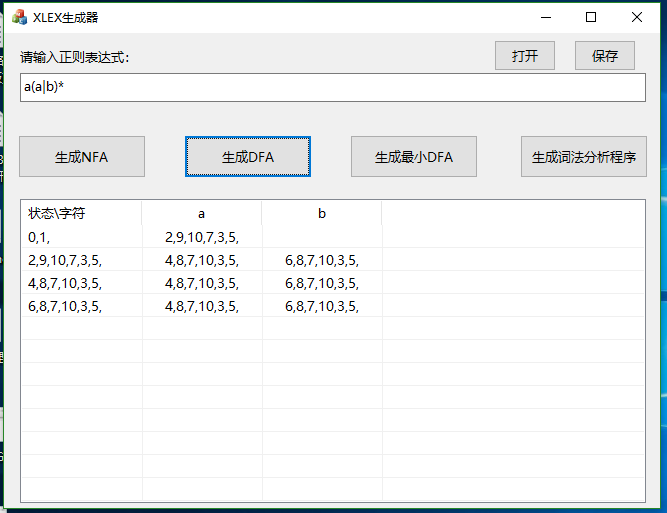
**五．实验结果**

运行程序，截图如下：

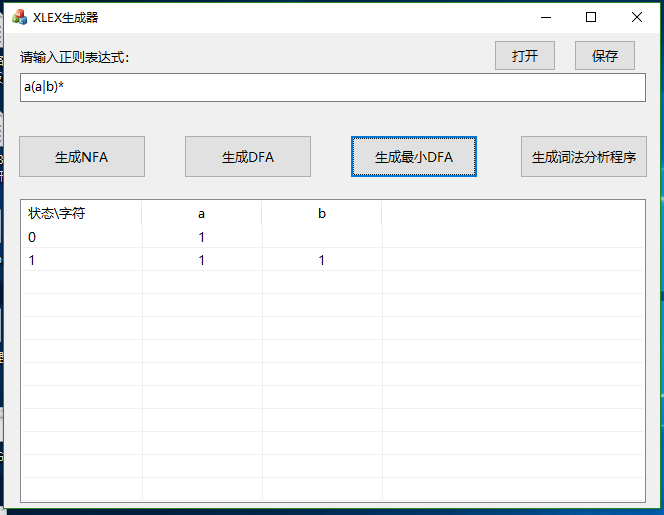
1. 生成NFA：



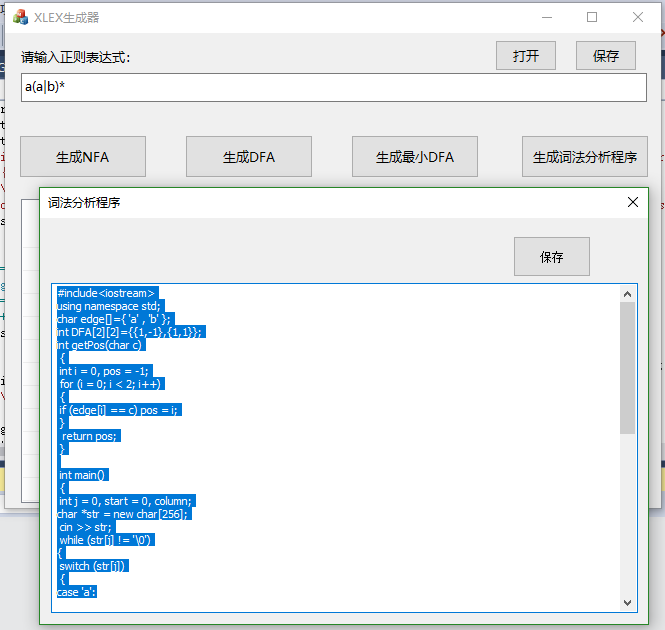
1. 生成DFA



1. 生成最小DFA



1. 生成词法分析程序



**六．实验小结**

本次实验通过对编译原理中词法分析器的实现，更加深刻地了解了词法分析的相关知识，对于正则表达式的理解更进一步

经过测试，对于正则表达式可以很好的生成NFA，DFA，最小DFA，最后生成可以直接运行的词法分析程序cpp。

虽然在理论在知道正则，NFA，DFA之间是怎么转换的，但在编程时却有不少阻碍，本实验的编程技巧和思路大部分是根据博客提供的程序参考的，部分采用了其源程序。由此可知，本人的编程技巧实在是欠缺。

1. **实验小结**

参考资料：

1. :编译原理及实践，机械工业出版社，Kenneth C.Louden著，冯博琴 冯岚等译
2. :黄煜廉老师的ppt
3. :博客:https://blog.csdn.net/johnf\_nash/article/details/22086149，正则表达式转NFA
4. :博客:https://blog.csdn.net/Hungryof/article/details/46822841，NFA到DFA的转换