



湖南大学
HUNAN UNIVERSITY

课程实验报告

课 程 名 称: 编译原理
实验项目名称: DFA 的编程实现
专 业 班 级: 计科 1706 班
姓 名: 李立天
学 号: 201708010806
指 导 教 师: 全哲
完 成 时 间: 2019 年 10 月 13 日

信息科学与工程学院

一、实验目的

通过本次实验，加深对 DFA 及其识别的语言的理解，学习对一般的 DFA 的表达方法与编程实现方法。

二、实验任务

编写一个 C 语言程序，模拟实现 DFA 识别字符串的过程。

三、实验内容

(1) DFA 的输入：

分别输入 DFA 的“字符集”、“状态集”、“开始状态”、“接受状态集”、“状态转换表”等内容，并保存在设定的变量中。

(2) DFA 的存储与读写：

将上述 DFA 的五元组保存在一个文本文件中，扩展名指定为.dfa。请自行设计 DFA 文件的存储格式，并说明其含义。能将保存在内存变量中的 DFA 写入 DFA 文件，也能将 DFA 文件读入内存中。（思考：对稀疏 DFA（转换表中为空的地方较多）或用“或”表达转换的 DFA（如下的测试用例三），如何改进 DFA 转换表的表达。）

(3) DFA 的正确性检查：

- 检查所有集合的元素的唯一性。
- 检查“开始状态”是否唯一，并是否包含在“状态集”中。
- 检查“接受状态集”是否为空，并是否包含在“状态集”中。
- 检查“状态转换表”是否满足 DFA 的要求。
- 检查“状态转换表”并非填满时的处理是否得当...

(4) DFA 的语言集列表显示：

输入待显示的字符串的最大长度 N，输出以上定义的 DFA 的语言集中长度 $\leq N$ 的所有规则字符串。（提示：注意算法中 while 循环的次数）

(5) DFA 的规则字符串判定：

输入（或用字符集随机生成）一个字符串，模拟 DFA 识别字符串的过程判定该字符串是否是规则字符串（属于 DFA 的语言集）。

四、系统设计及实现

1. 键盘输入模块

模块功能：

包括函数 inPutDFA()；在控制台通过键盘输入 DFA 的字符集、状态集、开始状态、接收状态集以及状态转换表，并将它们保存在全局数据结构当中；

实现思路：

使用字符串类型 string 保存字符集；

使用 int 类型保存状态个数，默认状态编号从 0 开始；

使用 int 类型保存 DFA 的开始状态；

使用数组类型 vector<int>保存接收状态集；

使用二维数组 vector<vector<int>>保存状态转换表；

使用哈希表 unordered_map<char, size_t>保存字符到编号的映射；

具体实现（伪）代码：

```
string charSet;           //字符集
int stateNum;             //状态集大小
int startState;           //开始状态
vector<int> acceptStateSet; //接受状态
vector<vector<int>>> trans; //状态转换表
unordered_map<char, size_t> getCharNum; //字符->字符编号
```

2. DFA 文件读写模块

模块功能：

包含读入模块 readDFA(string path)，即从指定文件中读入 DFA；输出模块 writeDFA(string path)，即将存储的 DFA 输出到指定文件；

实现思路：

首先，需要设计一种文本格式用以存储 DFA；其次，使用 fstream 类可以使得文件的输入输出变得简单易行。这里我们采用邻接矩阵来保存状态转换表，当 DFA 稀疏时可以考虑使用邻接链表。

具体实现：

采用规定的格式存储 DFA，例子如下：

```
/*o      # 字符集，o 表示除了前面字符的任意字符
5        # 状态集大小，状态编号从 0 开始
0        # 开始状态
1        # 接受状态集大小
4        # 接受状态集
1 -1 -1  # 状态转换表
-1 2 -1
-1 3 2
4 3 2
-1 -1 -1
```

3. DFA 正确性检查模块

模块功能：

bool isLegalDFA()，检查通过键盘或文件读入的 DFA 是否合法。

实现思路：

要检查 DFA 是否合法，就是要检查组成 DFA 的五元式是否合法，具体如下：

- 字符集重复元素检查
- 检查初始状态是否属于状态集
- 检查接受态是否为空集
- 检查接受状态集是否为状态集的子集
- 对状态转移表的检查

具体（伪）代码实现

```

bool isLegalDFA()
{
    //字符集重复元素检查
    for i = 0 to charSet.size()
        for j = i + 1 to charSet.size()
            if (charSet[i] == charSet[j]) return false;

    //检查初始状态是否属于状态集
    if (startState < 0 || startState >= stateNum) return false;

    //检查接受态是否为空集
    if (acceptStateSet.empty()) return false;

    //检查接受状态集是否为状态集的子集
    for i = 0 to acceptStateSet.size()
        if (acceptStateSet[i] < 0 || acceptStateSet[i] >= stateNum) return false;

    //对状态转移表的检查
    for i = 0 to stateNum
        for j = 0 to charSet.size()
            if (trans[i][j] < 0) continue;
            if (trans[i][j] < 0 || trans[i][j] >= stateNum) return false;    //不能到达一个不存在的状态

    return true;
}

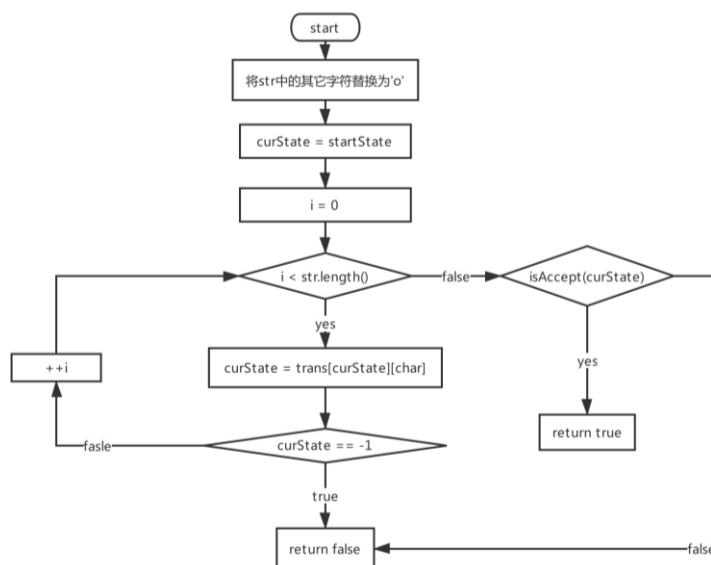
```

4. 字符串判定模块

模块功能:

bool isLegalString(string str), 判断字符串 str 是否能被 DFA 接受;

实现思路:



具体（伪）代码实现：

```
bool isLegalString(string str)
{
    //如果出现了字符集合以外的字符，替换为'o'
    for (size_t i = 0; i < str.length(); ++i)
        if (charSet.find(str[i]) == string::npos)
            str[i] = 'o';

    //状态转换
    int curState = startState;
    for (size_t i = 0; i < str.length(); ++i)
    {
        //如果读入字符不在字符集中，将到达出错状态-1
        if (getCharNum.find(str[i]) == getCharNum.end()) curState = -1;
        else curState = trans[curState][getCharNum[str[i]]];
        if (curState == -1) return false;
    }

    return isAcceptState(curState);
}
```

5. 语言集列表显示模块

模块功能：

void getString(int N)，给定一个最大长度 N，在控制台上打印出长度小于等于 N 的所有能够被 DFA 接受的字符串。

实现方法：

采用深度优先搜索算法，对于长度为 maxLength 时的情况：

初始化字符串为空串，当前状态为初始状态，深度为 0；

对于当前状态的每一条出边进行拓展，更新状态，并通过添加字符来更新字符串；

当深度等于 maxLength 时，判断当前状态是否为可接受状态；

若是，打印字符串；若不是，退出递归函数。

具体的代码实现：

```
/*深搜：找到长度为 maxLength 的可接受字符串*/
void dfs(int curState, int curLength, int maxLength, string& str)
{
    if (curLength == maxLength)
        if (isAcceptState(curState)) cout << (str.empty() ? ("ε") : str) << endl;
        return;

    for (int j = 0; j < charSet.size(); ++j)
        if (trans[curState][j] != -1)
            str += charSet[j];
```

```

        dfs(trans[curState][j], curLength + 1, maxLength, str);
        str.pop_back();
    }

    /*打印出所有长度不超过 N 的可接受字符串*/
    void getString(int N)
    {
        string str = "";
        for (int maxLength = 0; maxLength < N + 1; ++maxLength)
            dfs(startState, 0, maxLength, str);
    }

```

6.主程序的流程



五、样例测试

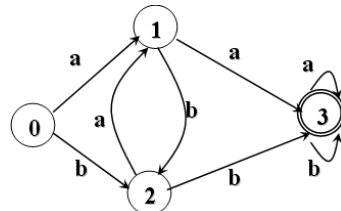
样例 1: 采用键盘输入的形式写入 DFA:

D:\大三上\编译原理\第一次实验\DFA\x64\Debug\DFA.exe

```

请输入字符集(σ代表除了前面字符的任意字符): ab
请输入状态集的大小(状态默认从0开始): 4
请输入开始状态: 0
请输入接受状态的个数: 1
请输入接受状态集: 3
请输入状态转换表:
1 2
3 2
1 3
3 3
该DFA是合法的
请输入字符串的最大长度N: 4
aa
bb
aaa
aab
abb
baa
bba
bbb
aaaa
aaab
aaba
aabb
abaa
abba
abbb
baaa
baab
babb
bbba
bbab
bbba
bbbb
请输入待检查的字符串:
ababb
状态转移情况: 0->1->2->1->2->3, 可接受
ababab
状态转移情况: 0->1->2->1->2->1->2, 不可接受
aacbbba
状态转移情况: 0->1->3->-1, 不可接受
Z
请按任意键继续. . .

```



D:\大三上\编译原理\第一次实验\DFA\out.dfa - Notepad++

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具



out. dfa

```

1 ab
2 4
3 1
4 3
5 1 2
6 3 2
7 1 3
8 3 3
9

```

结果分析:

通过运行截图可以看到, 我们首先通过键盘输入了如图所示的 DFA;

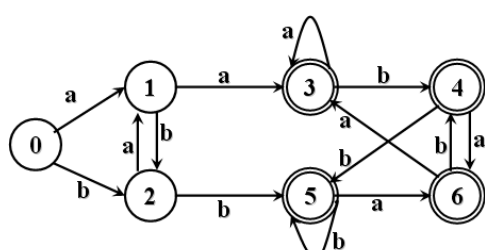
随后判断该 DFA 的合法性, 判断该 DFA 的输入是符合要求的;

输入语言的最大长度为 4, 将打印出所有长度小于等于 4 且包含 “aa” 或 “bb” 的字符串;

最后输入了 3 个待验证的字符串, 分别显示了它们在 DFA 中的状态转移情况和是否被接受的结果;

最后将 DFA 写入到文件 “out.dfa” 当中; 实验结果与事实相符。

样例 2: 采用文件读入的形式得到 DFA:



D:\大三上\编译原理\第一次实验\DFA\2.dfa

文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N)

1. dfa 2. dfa

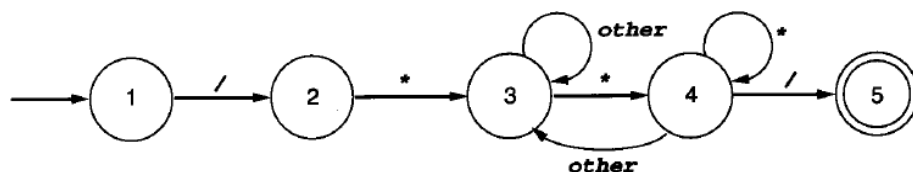
1	ab
2	7
3	0
4	4
5	3 4 5 6
6	1 2
7	3 2
8	1 5
9	3 4
10	6 5
11	6 5
12	3 4

D:\大三上\编译原理\第一次实验\DFA\x64\Debug\DFA.exe

该DFA是合法的
请输入字符串的最大长度N: 4
aa
bb
aaa
aab
abb
baa
bba
bbb
aaaa
aaab
aaba
aabb
abaa
abba
abbb
baaa
baab
babb
bbaa
bbab
bbba
bbbb
请输入待检查的字符串:
aabb
状态转移情况: 0->1->3->4->5, 可接受
bbaac
状态转移情况: 0->2->5->6->3->-1, 不可接受
ababab
状态转移情况: 0->1->2->1->2->1->2, 不可接受
Z
请按任意键继续. . .

结果分析: 样例 2 中的 DFA 与样例 1 中的 DFA 是等价的, 在这里我们采用文件读入的方式读取 2.dfa; 其它部分与样例 1 没有差别; 实验结果与实际符合的很好。

样例 3: 采用文件读入的方式获取 DFA



```
D:\大三上\编译原理\第一次实验\DFA\3.dfa - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R)
1. dfa x 2. dfa x 3. dfa x
1 /*o
2 5
3 0
4 1
5 4
6 1 -1 -1
7 -1 2 -1
8 -1 3 2
9 4 3 2
10 -1 -1 -1
11
```

```
D:\大三上\编译原理\第一次实验\DFA\x64\Debug\DFA.exe
该DFA是合法的
请输入字符串的最大长度N: 8
**/
*** /
*o* /
**** /
**o* /
*o** /
*oo* /
**** /
***o* /
**o** /
**oo* /
*o*** /
*oo*o* /
*ooo* /
***** /
***o*o* /
***o** /
***oo* /
**o*** /
**oo*o* /
**oo** /
**ooo* /
*o**** /
*o**o* /
*oo*o* /
*ooo* /
*oooo* /
请输入待检查的字符串:
/*comment*/
状态转移情况: 0->1->2->2->2->2->2->2->2->2->3->4, 可接受
**/**
状态转移情况: 0->1->2->3->4->-1, 不可接受
/*
状态转移情况: 0->1->2->-1, 不可接受
Z
请按任意键继续. . .
```

结果分析:

该 DFA 匹配的字符串特征为: 以 “/*” 为开头, 且以 “*/” 为结尾。

如图所示, 输入最大长度为 8, 显示出所有可接受的字符串, 其中 o 代表除了 / 和 * 之外的所有可能字符; 输入三组字符串, 程序能够正确地判断其是否能被 DFA 接受, 符合事实。

六、实验心得

1. 实验的第一部分是了解并使用 TINY 语言和 TINY 语言的编译器。这部分实验遇到了很多阻力，在消除了强制类型转换，系统环境不兼容等问题之后，成功得到了 TINY 编译器、词法分析器和语法分析器。此外，初步掌握了 TINY 语言的体系（包含编译器和虚拟机），了解到 flex 词法工具和 bison 语法工具的使用，为后续实验的进行奠定了基础。

2. 实验的第二部分是通过 C++ 程序模拟 DFA 的状态转换过程。通过自己编写代码，我对 DFA 的表达方法有了更深一层的认识，对算法的实现更加清晰明了。此外还锻炼了自己的代码编写能力，了解到编译器实际上是一个十分庞大的工程。路漫漫其修远兮，我需要更加努力学习，希望自己最终也能实现一个简单的编译器吧。

3. 对于 DFA 的编程实现，还存在一些可以改进的地方：

- ① 当 DFA 输入时，可以通过对-1 的计数来实现 DFA 的稀疏性判定，如果判定为稀疏 DFA 则采用邻接链表的形式存储，这样可以显著地降低程序的空间复杂度，这对大型编译器来说是很有考虑的必要的；
- ② 编写的程序限定了状态集的开始编号（默认从 0 开始），而实际上状态的编号是可以任意的。为了程序的通用性，我们可以使用一个 unordered_map 即哈希表来保存<状态编号，标号>的映射，这样就可以实现任意的状态编号了；
- ③ 在“获取长度小于等于 N 的语言集”算法中，在源程序中实际上调用了 N+1 次深度优先搜索算法（DFS）。我们可以将其改进为只调用一次 DFS，只需要在结点深度小于 N 时都判断一下当前状态是否是接受状态即可。