

1. *GrammarToFA*

2. *DetermineNFAorDFA*

3. *NFAToDFA*

4. *MinimizeDFA*

5. *RunMin – DFAModel*

运行方式

*Windows*环境下，配置*g++*编译器，版本*g++(GCC)12.2.0*

*WORKDIR fa*下，运行*run.bat* 完成编译和打开运行

测试样例

输入过程为：

1

gra.in

```
=====
|  1.read grammar.  |
|-----|
|  2.construct FA.  |
|-----|
|  3.NFA to DFA.    |
|-----|
|  4.Minimize DFA.  |
|-----|
|  5.RUN FA Model.  |
|-----|
|  6.EXIT.          |
|-----|
=====
```

1

```
=====
| Please Input File Name |
=====
```

gra.in

```
=====
| Completed |
=====
```

```

=====
|  FA AS FOLLOW  |
=====

```

$N = (Q, E, M, S, F) = \{$
 $Q = \{\$ A B Z\}$
 $E = \{a b \}$

```

+---+---+---+
| * | a | b |
+---+---+---+
| $ | A | B |
| A | Z | B |
| B | A | Z |
| Z | ZA |   |
+---+---+---+

```

Initial state : \$
 Termination state : Z
 };

```

=====
|  Is NFA  |
=====

```

```
=====
|  DFA AS FOLLOW  |
=====
```

$N = (Q, E, M, S, F) = \{$

$Q = \{ \$ A AZ B Z \}$

$E = \{ a b \}$

```
+-----+-----+-----+
```

```
| * | a | b |
```

```
+-----+-----+-----+
```

```
| $ | A | B |
```

```
| A | Z | B |
```

```
| AZ | AZ | B |
```

```
| B | A | Z |
```

```
| Z | AZ | |
```

```
+-----+-----+-----+
```

Initial state : \$

Termination state : {AZ Z }

}

```
=====
| Split 'AZ' From {AZ Z } |
=====
```

```
=====
| Split '$' From {$ A B } |
=====
```

```
=====
| Split 'A' From {A B } |
=====
```

```
=====
| Min DFA AS FOLLW |
=====
```

$N = (Q, E, M, S, F) = \{$

$Q = \{AZ \$ Z A B \}$

$E = \{a b \}$

```
+---+---+---+
```

```
| * | a | b |
```

```
+---+---+---+
```

```
| AZ | AZ | B |
```

```
| $ | A | B |
```

```
| Z | AZ | |
```

```
| A | Z | B |
```

```
| B | A | Z |
```

```
+---+---+---+
```

Initial state : \$

Termination state : {AZ Z }

}

5

aaaaaaabbbb

5

```
=====
| Please Input Sentence |
=====
```

aaaaaaabbbb

A Z AZ AZ AZ AZ B Z EMPTY!

```
=====
| Illegal sentences |
=====
```

5

abababab

5

```
=====
| Please Input Sentence |
=====
```

abababab

A B A B A B A B

```
=====
| Legal Sentences |
=====
```

1. Grammar To FA

```
1 struct FA
2 {
3     bool is_dfa;
4     std::vector<char> Q;
5     std::vector<char> E;
6     std::map<std::pair<std::string, char>, std::string> M;
7     char S;
8     char F;
9 };
10
11 struct DFA
12 {
13     std::vector<std::string> Q;
14     std::vector<char> E;
15     std::map<std::pair<std::string, char>, std::string> M;
16     char S;
17     std::vector<std::string> F;
18 };
```

NFA 和 *DFA* 结构体，即为五元组存储方式.

```
1 std::ifstream in;
2
3 auto read = [&] () {
4     shows("Please Input File Name");
5     std::string filename;
6     std::cin >> filename;
7     std::cout << "\n";
8     in.open(filename);
9 };
10
11 std::vector<std::string> s;
12 std::string grammar;
13
14 std::set<char> ct;
15 std::set<char> symbol;
16
17 auto getG = [&] () {
18     s.clear();
19     ct.clear();
20
21     while (in >> grammar) {
22         std::cout << grammar << "\n";
23         ct.insert(grammar[0]);
24         s.push_back(grammar);
25     }
26     std::cout << "\n";
27
28     int n = (int) ct.size();
29     n += 1;
30     std::vector<std::vector<std::array<char, 2>>> g(300);
31     for (int i = 0; i < (int) s.size(); i++) {
32         std::string sg = s[i];
33
34         char l = sg[0];
35         if (sg.length() == 5) {
36             char N = sg[3];
37             char T = sg[4];
38             symbol.insert(T);
39             g[N].push_back({l, T});
40         }
41         else if (sg.length() == 4) {
42             char T = sg[3];
43             symbol.insert(T);
44             g['$'].push_back({l, T});
45         }
46     }
47     return g;
48 };
```

输入文件名指定正则文法读取路径，使用c++容器实例 `std::vector<std::vector<std::array<char,2>>>` 存储正则文法，存储方式为带权有向图。

2.Determine NFA or DFA

```
1 auto print = [&] (std::vector<std::vector<std::array<char, 2>>> g) {
2     shows("FA AS FOLLOW");
3     bool is_dfa = true;
4     int n = (int) ct.size();
5     std::cout << "N = {Q, E, M, S, F} = {\n";
6     std::cout << "Q = {"$";
7     for (auto c : ct) {
8         std::cout << " ' ' << c;
9     }
10    std::cout << "}\n";
11
12    // symbol
13    std::cout << "E = {"$";
14    for (auto c : symbol) {
15        std::cout << c << " ' '";
16    }
17    std::cout << "}\n";
18
19    std::map<char, int> mp;
20    std::map<char, int> smx;
21    for (auto c : g) {
22        if (c.empty()) {
23            continue;
24        }
25        mp.clear();
26        int mx = 0;
27        for (auto cc : c) {
28            mp[cc[1]] += 1;
29            smx[cc[1]] = std::max(smx[cc[1]], mp[cc[1]]);
30            if (smx[cc[1]] >= 2) {
31                is_dfa = false;
32            }
33        }
34    }
35    std::vector<int> ssmx;
36    ssmx.push_back(1);
37    for (auto c : symbol) {
38        ssmx.push_back(smx[c]);
39    }
40    std::vector<char> cct(ct.begin(), ct.end());
41    int index = 1;
42    std::vector<char> title(symbol.begin(), symbol.end());
43    std::vector<std::string> ntitle;
44    ntitle.push_back("");
45    for (auto t : title) {
46        std::string cnt = "";
47        cnt += t;
48        ntitle.push_back(cnt);
49    }
50    std::vector<std::vector<std::string>> menu;
51    for (int i = 0; i < 200; i++) {
52        if (g[i].empty()) {
53            continue;
54        }
55        std::vector<std::string> cnt;
56        cnt.clear();
57        std::string fr = "";
58        fr += (char) (i);
59        cnt.push_back(fr);
60        std::map<char, std::vector<char>> mmp;
61        for (auto cc : g[i]) {
62            mmp[cc[1]].push_back(cc[0]);
63        }
64        for (auto cc : title) {
65            std::string cs = "";
66            for (auto ccc : mmp[cc]) {
67                cs += ccc;
68            }
69            cnt.push_back(cs);
70        }
71        menu.push_back(cnt);
72    }
73
74    Draw_Datas(ssmx, menu, ntitle, (int) ntitle.size(), (int) menu.size());
75
76    std::cout << "Initial state : $\n";
77    std::cout << "Termination state : " << s[0][0] << "\n";
78
79    std::cout << "};\n\n";
80
81    std::map<std::pair<std::string, char>, std::string> r;
82
83    int ind = 0;
84
85    for (auto c : menu) {
86        int indx = 0;
87        for (int i = 1; i < (int) c.size(); i++) {
88            r[{c[0], title[indx++]}] = c[i];
89        }
90    }
91
92    FA f;
93    f.is_dfa = is_dfa;
94    f.Q = cct;
95    f.E = title;
96    f.M = r;
97    f.S = '$';
98    f.F = s[0][0];
99
100    return f;
101 };
```

自定义 *print* 函数输出图关键信息，并将图传给*mysql*式表格输出函数，判断是否*DFA*方式为如果穿的值里有一列长度不为1，既有多个选择时，即为*NFA*。

DFA Output

```
1 void Draw_line(vector<int> xmax,int columns) {
2     for (int i = 0; i < columns; i++) {
3         cout << "+-";
4         for (int j = 0; j <= xmax[i]; j++) {
5             cout << "-";
6         }
7     }
8     cout << "+" << endl;
9 }
10
11 void Draw_Datas(vector<int> xmax, vector<vector<string>> Str,vector<string> D,int columns,int row) {
12     Draw_line(xmax,columns);
13     for (int i = 0; i < D.size(); i++) {
14         cout << "| " << setw(xmax[i]) << setiosflags(ios::left) << setfill(' ') << D[i] << " ";
15     }
16     cout << "|" << endl;
17     Draw_line(xmax, columns);
18     for (int i = 0; i < row; i++) {
19         for (int j = 0; j < columns; j++) {
20             cout << "| " << setw(xmax[j]) << setiosflags(ios::left) << setfill(' ');
21             cout << Str[i][j] << " ";
22         }
23         cout << "|" << endl;
24     }
25     Draw_line(xmax, columns);
26 }
```

实现表格的格式化输出，借鉴*mysql*命令行数据库输出格式完成。

```

1  auto print_dfa = [&] (DFA d) {
2
3      std::cout << "N = {Q, E, M, S, F} = {\n";
4      std::cout << "Q = {";
5      for (auto c : d.Q) {
6          std::cout << c << ' ';
7      }
8      std::cout << "}\n";
9      std::cout << "E = {";
10     for (auto c : d.E) {
11         std::cout << c << ' ';
12     }
13     std::cout << "}\n";
14
15     std::vector<std::vector<std::string>> Str;
16     std::vector<std::string> cnt;
17
18     for (auto c : d.Q) {
19         cnt.clear();
20         cnt.push_back(c);
21         for (auto cc : d.E) {
22             cnt.push_back(d.M[{c, cc}]);
23         }
24         Str.push_back(cnt);
25     }
26
27     std::vector<int> xmax((int) d.E.size() + 1, 1);
28
29     int columns = (int) d.E.size() + 1;
30     int row = (int) Str.size();
31
32     for (int i = 0; i < row; i++) {
33         for (int j = 0; j < (int) Str[i].size(); j++) {
34             xmax[j] = std::max(xmax[j], (int) Str[i][j].size());
35         }
36     }
37
38     std::vector<string> D;
39     D.push_back("");
40     for (auto c : d.E) {
41         D.push_back(tos(c));
42     }
43
44     Draw_Datas(xmax, Str, D, columns, row);
45
46     std::cout << "Initial state : " << d.S << "\n";
47     std::cout << "Termination state : {";
48     for (auto c : d.F) {
49         std::cout << c << ' ';
50     }
51     std::cout << "}\n}\n";
52
53     std::cout << std::endl;
54 };

```

实现 *Print - DFA* 函数，作用为格式化输出DFA。

3.NFA To DFA

```
1 auto tos = [&] (char x) {
2     std::string ccccc = "";
3     ccccc += x;
4     return ccccc;
5 };
6
7 std::map<std::string, std::vector<std::string>> dfa_s;
8 std::map<std::pair<std::string, char>, std::string> bce;
9
10 auto sol = [&] () {
11     dfa_s.clear();
12     bce.clear();
13     std::string beg = "";
14     beg += re.S;
15     std::map<std::string, bool> is_checked;
16
17     std::queue<std::string> cnt;
18
19     cnt.push(beg);
20
21     while (!cnt.empty()) {
22         auto q = cnt.front();
23         cnt.pop();
24         if (q.empty()) {
25             continue;
26         }
27
28         std::vector<std::string> cnt_dfa_s;
29
30         for (auto c : re.E) {
31             std::string ne = "";
32             for (auto cc : q) {
33                 ne += re.M[{tos(cc), c}];
34             }
35             ne.erase(unique(ne.begin(), ne.end()), ne.end());
36             std::sort(ne.begin(), ne.end());
37             if (!is_checked[ne]) {
38                 cnt.push(ne);
39                 is_checked[ne] = true;
40             }
41             bce[{q, c}] = ne;
42             cnt_dfa_s.push_back(ne);
43         }
44
45         dfa_s[q] = cnt_dfa_s;
46     }
47
48     DFA dfa;
49     dfa.E = re.E;
50     dfa.M = bce;
51     dfa.S = re.S;
52     for (auto c : bce) {
53         dfa.Q.push_back(c.first.first);
54     }
55     dfa.Q.erase(unique(dfa.Q.begin(), dfa.Q.end()), dfa.Q.end());
56
57     for (auto c : dfa.Q) {
58         if (c.find(re.F) != -1) {
59             dfa.F.push_back(c);
60         }
61     }
62     dfa.F.erase(unique(dfa.F.begin(), dfa.F.end()), dfa.F.end());
63     return dfa;
64 };
```

实现 *sol* 函数，实现 *NFA* 到 *DFA* 的转换，具体办法为不断将新的状态添加到队列末尾，直至无新状态的产生，即为 *DFA*，返回值为 *DFA* 五元组。

4. *Minimize DFA*

```

1  auto min = [&] (DFA d) {
2      std::queue<std::vector<std::string>> state;
3
4      state.push(d.F);
5      std::vector<std::string> nonter;
6
7      for (auto c : d.Q) {
8          bool is_ter = false;
9          for (auto cc : d.F) {
10             if (cc == c) {
11                 is_ter = true;
12                 break;
13             }
14         }
15         if (!is_ter) {
16             nonter.push_back(c);
17         }
18     }
19     nonter.erase(unique(nonter.begin(), nonter.end()), nonter.end());
20
21     std::map<std::string, std::string> f;
22
23     for (auto c : d.Q) {
24         f[c] = c;
25     }
26
27     std::function<std::string(std::string)> get = [&] (std::string gs) {
28         return gs == f[gs] ? gs : f[gs] = get(f[gs]);
29     };
30
31     for (int i = 1; i < (int) d.F.size(); i++) {
32         f[get(d.F[i])] = get(d.F[0]);
33     }
34
35     for (int i = 1; i < (int) d.F.size(); i++) {
36         f[get(nonter[i])] = get(nonter[0]);
37     }
38
39     state.push(nonter);
40
41     std::map<std::vector<std::string>, bool> is_vised;
42
43     std::vector<std::vector<std::string>> ans;
44
45     ans.clear();
46
47     while (!state.empty()) {
48         auto q = state.front();
49         state.pop();
50         if (is_vised[q]) {
51             continue;
52         }
53         else {
54             is_vised[q] = true;
55         }
56         bool is_ok = true;
57         for (int i = 1; i < (int) q.size(); i++) {
58             for (auto c : d.E) {
59                 if (get(d.M[{q[0], c}]) != get(d.M[{q[i], c}])) {
60                     is_ok = false;
61                     break;
62                 }
63             }
64         }
65         if (!is_ok) {
66             std::vector<std::string> l;
67             l.push_back(q[0]);
68             std::vector<std::string> r;
69             for (int i = 1; i < (int) q.size(); i++) {
70                 r.push_back(q[i]);
71             }
72             std::string rm = "Split '\" + q[0] + "\" From {'";
73             for (auto c : q) {
74                 rm += c;
75                 rm += ' ';
76             }
77             rm += "}";
78             shows(rm);
79             state.push(r);
80
81             auto scnt = state;
82
83             f.clear();
84             for (auto c : d.Q) {
85                 f[c] = c;
86             }
87             while (!scnt.empty()) {
88                 auto cq = scnt.front();
89                 scnt.pop();
90                 for (int i = 1; i < (int) cq.size(); i++) {
91                     f[get(cq[i])] = get(cq[0]);
92                 }
93             }
94             ans.push_back(l);
95         }
96         else {
97             ans.push_back(q);
98         }
99     }
100
101     std::vector<std::string> nans;
102
103     for (auto c : ans) {
104         if ((int) c.size() >= 2) {
105             for (int i = 1; i <= (int) c.size(); i++) {

```

```

106     for (auto &cc : d.M) {
107         if (cc.second == c[i]) {
108             cc.second = c[0];
109         }
110     }
111     for (auto cc : d.M) {
112         if (cc.first.first == c[i]) {
113             continue;
114         }
115         else {
116             nans.push_back(c[0]);
117         }
118     }
119 }
120 }
121 else {
122     nans.push_back(c[0]);
123 }
124 }
125
126 std::vector<std::vector<std::string>> Str;
127 std::vector<std::string> cnt;
128
129 for (auto c : nans) {
130     cnt.clear();
131     cnt.push_back(c);
132     for (auto cc : d.E) {
133         cnt.push_back(d.M[{c, cc}]);
134     }
135     Str.push_back(cnt);
136 }
137
138 std::vector<int> xmax((int) d.E.size() + 1, 1);
139
140 int columns = (int) d.E.size() + 1;
141 int row = (int) Str.size();
142
143 for (int i = 0; i < row; i++) {
144     for (int j = 0; j < (int) Str[i].size(); j++) {
145         xmax[j] = std::max(xmax[j], (int) Str[i][j].size());
146     }
147 }
148
149 std::vector<string> D;
150 D.push_back("");
151 for (auto c : d.E) {
152     D.push_back(tos(c));
153 }
154
155 DFA mindfa;
156 mindfa.E = d.E;
157 mindfa.Q = nans;
158 mindfa.M = d.M;
159 mindfa.S = d.S;
160 for (auto c : Str) {
161     auto be = c[0];
162     if (be.find(re.F) != -1) {
163         mindfa.F.push_back(be);
164     }
165 }
166
167 return mindfa;
168 };

```

min 函数，实现 *DFA* 的最小化，具体方法为将属性集合不断拆分为不可再分，再将相同状态的集合校区，即得到最小化的 *DFA*。

5.RunMin – DFAModel

```

1 auto run = [&] (DFA d, std::string sen) {
2     bool ok = true;
3     std::string cnt = tos(d.S);
4     for (int i = 0; i < sen.length(); i++) {
5         std::cout << d.M[{cnt, sen[i]}] << ' ';
6         cnt = d.M[{cnt, sen[i]}];
7         if (cnt.empty()) {
8             std::cout << "EMPTY!";
9             ok = false;
10            break;
11        }
12    }
13    std::cout << "\n\n";
14    return ok;
15 };

```

run 函数，实现句子在 *DFA* 上的运行，判断是否可以产生这个句子，并返回真值。