中山大学数据科学与计算机学院本科生实验报告

课程名称：编译器构造实验 任课教师：陈炬桦

|  |  |  |  |
| --- | --- | --- | --- |
| 学年度 | 2018-2019 | 学期 | 第二学期 |
| 年级 | 16级 | 专业（方向） | 计算机科学与技术 |
| 学号 | 16337341 | 姓名 | 朱志儒 |
| 电话 | 15989184223 | Email | 739741104@qq.com |
| 开始日期 | 2019/3/22 | 完成日期 | 2019/3/22 |

1. 实验题目

**εNDFA 确定化**

* 1. Description

输入字母表，非确定εNDFA确定化FA状态集，映射集；用造表法。

输出确定ＦＡ状态集，映射集；

* 1. Input

输入字母个数，字母表  
状态个数，状态表（状态名称，开始状态，终止状态：0否1是），空格符分隔；  
映射个数，映射表（起，终，字母），空格符分隔，k表示ε；

* 1. Output

Determine State:  
状态表: 状态名称，开始状态，终止状态，[子集]  
Determine Mapping:  
映射表: 起，终，字母,(顺序号

1. 算法描述（介绍程序模块功能，流程图）

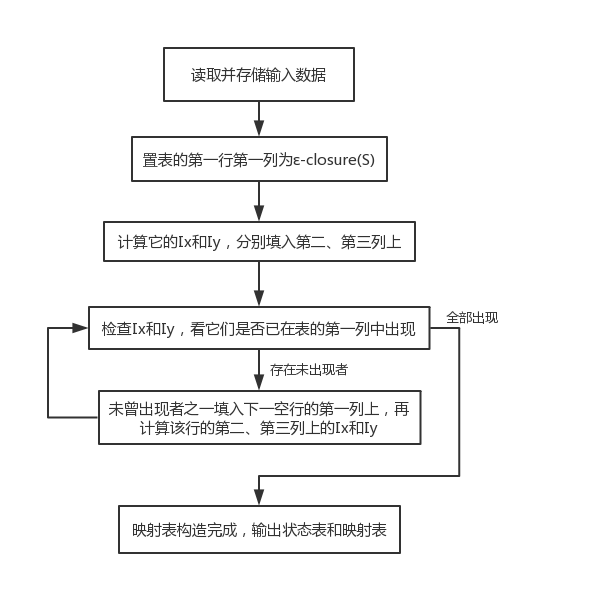
求Ia的方法：

求出所有可从子集I中的任一状态出发，经过一条a连线（跳过a连线前的连线）而到达的状态的集合J，再求出J的-闭包即可得到Ia**。**

造表法：

1. 置表的第一行第一列为，即一列包含M初态集S的ε闭包，然后计算它的Ix和Iy，分别填入第二、第三列上。
2. 检查Ix和Iy，看它们是否已在表的第一列中出现，把未曾出现者之一填入下一空行的第一列上，再计算该行的第二、第三列上的Ix和Iy。
3. 重复第二步，直至表中所有第二、第三列上的元素全部在第一列出现为止。

流程图：



1. 测试数据（2组）

|  |  |  |
| --- | --- | --- |
|  | 输入 | 输出 |
| 第一组 | 2  x y  9  0 1 0  1 0 0  2 0 0  3 0 0  4 0 0  5 0 0  6 0 0  7 0 0  8 0 1  12  0 1 x  0 3 y  0 6 x  1 2 k  2 2 y  2 8 k  3 4 k  4 4 x  4 5 k  5 8 y  6 7 y  7 8 x | Determine State:  0 1 0 [0]  1 0 1 [1268]  2 0 0 [345]  3 0 1 [278]  4 0 0 [45]  5 0 1 [8]  6 0 1 [28]  Determine Mapping:  0 1 x (0  0 2 y (1  1 3 y (2  2 4 x (3  2 5 y (4  3 5 x (5  3 6 y (6  4 4 x (7  4 5 y (8  6 6 y (9 |
| 第二组 | 2  x y  5  0 1 0  1 0 0  2 0 0  3 0 0  4 0 1  9  0 1 x  0 2 y  1 2 x  1 3 x  2 1 y  2 3 y  3 3 x  3 3 y  3 4 x | Determine State:  0 1 0 [0]  1 0 0 [1]  2 0 0 [2]  3 0 0 [23]  4 0 0 [13]  5 0 1 [34]  6 0 1 [234]  7 0 0 [3]  Determine Mapping:  0 1 x (0  0 2 y (1  1 3 x (2  2 4 y (3  3 5 x (4  3 4 y (5  4 6 x (6  4 7 y (7  5 5 x (8  5 7 y (9  6 5 x (10  6 4 y (11  7 5 x (12  7 7 y (13 |

1. 程序清单
2. #include <iostream>
3. #include <string>
4. #include <vector>
5. #include <queue>
6. #include <algorithm>
7. **using** **namespace** std;
9. **int** n, state\_row, mapping\_row, state[100][3], ccount = 0;
10. **char** mapping[100][3];
11. string start, endd;
12. **char** str[100];
14. vector<string> dfastate;
15. vector<string> ffstate;
16. queue<string> rstate;
17. string dmapping[100][3];
19. string eclosure(string sstate) {
20. string oldstate = sstate, newstate = sstate;
21. **for** (**int** i = 0; i < sstate.size(); ++i) {
22. **for** (**int** j = 0; j < mapping\_row; ++j)
23. **if** (mapping[j][0] == sstate[i] && mapping[j][2] == 'k' && newstate.find(mapping[j][1]) == string::npos)
24. newstate += mapping[j][1];
25. }
26. **while** (newstate != oldstate) {
27. oldstate = newstate;
28. newstate = eclosure(newstate);
29. }
30. **return** newstate;
31. }
33. **void** createnewstate(string newstate) {
34. dfastate.push\_back(newstate);
35. **for** (**int** k = 0; k < n; ++k) {
36. string newstringstate;
37. **for** (**int** i = 0; i < newstate.size(); ++i) {
38. **for** (**int** j = 0; j < mapping\_row; ++j)
39. **if** (mapping[j][0] == newstate[i] && mapping[j][2] == str[k] && newstringstate.find(mapping[j][1]) == string::npos)
40. newstringstate += mapping[j][1];
41. }
42. newstringstate = eclosure(newstringstate);
43. **if** (!newstringstate.empty()) {
44. dmapping[ccount][0] = newstate;
45. dmapping[ccount][1] = newstringstate;
46. dmapping[ccount][2] = str[k];
47. ccount++;
48. **if** (find(dfastate.begin(), dfastate.end(), newstringstate) == dfastate.end() && find(ffstate.begin(), ffstate.end(), newstringstate) == ffstate.end()) {
49. ffstate.push\_back(newstringstate);
50. rstate.push(newstringstate);
51. }
52. }
53. }
54. }


58. **bool** is\_endstate(string lstate) {
59. **for** (**int** i = 0; i < lstate.size(); ++i) {
60. **if** (endd.find(lstate[i]) != string::npos)
61. **return** **true**;
62. }
63. **return** **false**;
64. }
66. **int** main() {
67. cin >> n;
68. **for** (**int** i = 0; i < n; ++i)
69. cin >> str[i];
70. cin >> state\_row;
71. **for** (**int** i = 0; i < state\_row; ++i) {
72. **for** (**int** j = 0; j < 3; ++j) {
73. cin >> state[i][j];
74. **if** (j == 1 && state[i][j] == 1)
75. start += to\_string(state[i][0]);
76. **if** (j == 2 && state[i][j] == 1)
77. endd += to\_string(state[i][0]);
78. }
79. }
80. cin >> mapping\_row;
81. **for** (**int** i = 0; i < mapping\_row; ++i)
82. **for** (**int** j = 0; j < 3; ++j)
83. cin >> mapping[i][j];
85. createnewstate(eclosure(start));
86. **while** (!rstate.empty()) {
87. createnewstate(rstate.front());
88. rstate.pop();
89. }
90. vector<string> endstate;
91. **for** (**int** i = 0; i < dfastate.size(); ++i) {
92. **if** (is\_endstate(dfastate[i]))
93. endstate.push\_back(dfastate[i]);
94. }
95. cout << "Determine State:" << endl;
96. **for** (**int** i = 0; i < dfastate.size(); ++i) {
97. string output = dfastate[i];
98. sort(output.begin(), output.end());
99. **if** (dfastate[i] == start)
100. cout << i << ' ' << 1 << ' ' << 0;
101. **else** **if** (find(endstate.begin(), endstate.end(), dfastate[i]) != endstate.end())
102. cout << i << ' ' << 0 << ' ' << 1;
103. **else**
104. cout << i << ' ' << 0 << ' ' << 0;
105. cout << ' ' << '[' << output << ']' << endl;
106. }
107. cout << "Determine Mapping:" << endl;
108. **for** (**int** i = 0; i < ccount; ++i) {
109. auto pos1 = distance(dfastate.begin(), find(dfastate.begin(), dfastate.end(), dmapping[i][0]));
110. auto pos2 = distance(dfastate.begin(), find(dfastate.begin(), dfastate.end(), dmapping[i][1]));
111. cout << pos1 << ' ' << pos2 << ' ' << dmapping[i][2] << " (" << i << endl;
112. }
113. **return** 0;
114. }