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 教学助理（TA）：

|  |  |  |  |
| --- | --- | --- | --- |
| 学年度 | 2018-2019 | 学期 | 第二学期 |
| 年级 | 2016 | 专业（方向） | 计算机科学与技术 |
| 学号 | 16337052 | 姓名 | 杜尔鑫 |
| 电话 | 15626281204 | Email | duerx@mail2.sysu.edu.cn |
| 开始日期 | 2019-03-22 | 完成日期 | 2019-03-22 |

1. 实验题目

**1.1 Description**

输入字母表，非确定εNDFA 确定化ＦＡ状态集，映射集；用造表法算法。  
输出确定ＦＡ状态集，映射集；

此题需要提交实验报告；“实验报告用“学号+姓名+42”

**1.2 Input**

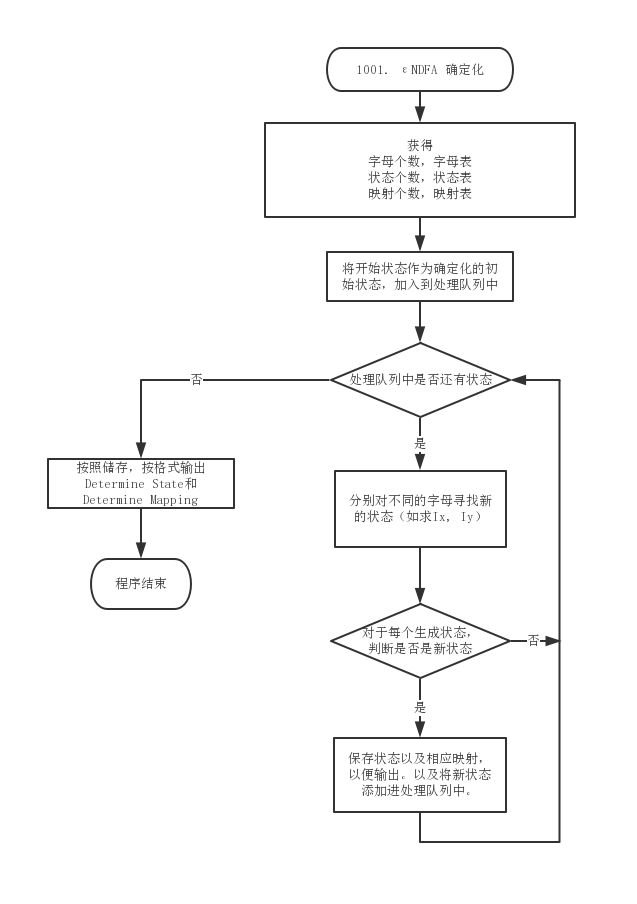
输入字母个数，字母表  
状态个数，状态表（状态名称，开始状态，终止状态：0否1是），空格符分隔；  
映射个数，映射表（起，终，字母），空格符分隔，k表示ε；  
【参考作业3.2<1>】

**1.3 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

2. 算法描述(介绍程序模块功能；流程图)

1. 输入字母个数，字母表；状态个数，状态表；映射个数，映射表；
2. 将开始状态作为确定化的初始状态。并初始化一个处理队列，处理所有未遍历的状态；
3. 判断处理队列中是否存在未遍历状态。如果有，进入4)，否则进入7)；
4. 对于每个状态，分别对于每个字母寻找新的状态；
5. 对于每个生成状态，判断是为新状态。如果是，进入6)，否则返回3)；
6. 保存新状态，以及相应映射，以便后续输出。并将新状态添加到处理队列中。然后返回3)；
7. 按照储存，按照格式输出Determine State和Determine Mapping。



3. 测试数据(2组)

**3.1 Standard Example**

**3.1.1 Input**

|  |
| --- |
| 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 |

**3.1.2 Output**

|  |
| --- |
| 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 |

**3.2 Addition Test（改编自1000）**

**3.2.1 Input**

|  |
| --- |
| 2  x y  6  0 1 0  1 0 0  2 0 0  3 0 0  4 0 0  5 0 1  12  0 1 x  0 2 y  1 2 x  1 3 x  2 1 y  2 3 y  2 5 y  3 3 x  3 3 y  3 4 x  3 5 x  4 5 k |

**3.2.2 Output**

|  |
| --- |
| Determine State:  0 1 0 [0]  1 0 0 [1]  2 0 0 [2]  3 0 0 [23]  4 0 1 [135]  5 0 1 [345]  6 0 1 [2345]  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 |

4. 程序清单

|  |
| --- |
| #include <iostream>  #include <queue>  #include <stdlib.h>  #include <string>  #include <sstream>  using namespace std;  struct  {      int num;      char c[100];  } alpha = {0};  struct  {      int num;      int single\_state[100];      bool isStart[100];      bool isEnd[100];  } state = {0};  struct  {      int num;      int start[100];      int end[100];      char c[100];  } map = {0};  struct  {      int num;      int res\_state[100];      bool isStart[100];      bool isEnd[100];      string subSet[100];  } newState = {0};  struct  {      int num;      int start[100];      int end[100];      char c[100];  } newMap = {0};  struct ns  {      int num;      int sub[100];      bool isStart;      bool isEnd;  };  queue<ns> temp;  struct  {      int num;      string left[100];      string right[100];      char c[100];  } haha = {0};  queue<int> c\_kong;  void input();  void output();  void init();  string intToStr(int a);  string arrToStr(ns teres);  bool isExist(ns a);  bool isExistStart( ns a );  bool isExistEnd(ns a);  void sort(ns a);  void init() {      newState.num = 0;      ns first;      first.num = 1;      first.isStart = state.isStart[0];      first.isEnd = state.isEnd[0];      first.sub[0] = state.single\_state[0];      temp.push(first);      newState.res\_state[0] = state.single\_state[0];      newState.isStart[0] = state.isStart[0];      newState.isEnd[0] = state.isEnd[0];      newState.subSet[0] = intToStr( state.single\_state[0] );      newState.num++;  }  void input() {      cin >> alpha.num;      for (int i = 0; i < alpha.num; i++) {          cin >> alpha.c[i];      }      cin >> state.num;      for (int i = 0; i < state.num; i++) {          cin >> state.single\_state[i] >> state.isStart[i] >> state.isEnd[i];      }      cin >> map.num;      for (int i = 0; i < map.num; i++) {          cin >> map.start[i] >> map.end[i] >> map.c[i];      }  }  void output() {      cout << "Determine State:" << endl;      for (int i = 0; i < newState.num; i++) {          cout << newState.res\_state[i] << " " << newState.isStart[i] << " " << newState.isEnd[i] << " [" << newState.subSet[i] << "]" << endl;      }      cout << "Determine Mapping:" << endl;      for (int i = 0; i < newMap.num; i++) {          cout << newMap.start[i] << " " << newMap.end[i] << " " << newMap.c[i] << " (" << i << endl;      }  }  bool isExistStart( ns a ) {      for (int i = 0; i < a.num; i++) {          for (int j = 0; j < state.num; j++ ) {              if (state.isStart[j] == 1 && a.sub[i] == state.single\_state[j] ) return true;          }      }      return false;  }  bool isExistEnd(ns a) {      for (int i = 0; i < a.num; i++) {          for (int j = 0; j < state.num; j++ ) {              if (state.isEnd[j] == 1 && a.sub[i] == state.single\_state[j] ) return true;          }      }      return false;  }  bool isExist(ns a) {      if (a.num == 0) return 1;      for (int i = 0; i < newState.num; i++) {          if (newState.subSet[i] == arrToStr(a) ) {              return 1;          }      }      return 0;  }  string intToStr(int n) {      string res = "";      res = res + char(n + '0');      return res;  }  string arrToStr( ns a) {      string res = "";      for (int i = 0; i < a.num; ++i)          res = res + intToStr(a.sub[i]);      return res;  }  int main() {      input();      init();      while (1) {          ns te = temp.front(); temp.pop();          for (int i = 0; i < state.num; i++ ) {              ns teres;              teres.num = 0;              for (int j = 0; j < te.num ; j++) {                  for (int z = 0; z < map.num ; z++) {                      if ( alpha.c[i] == map.c[z] && te.sub[j] == map.start[z] ) {                          bool is = 1;                          for (int q = 0; q < teres.num; q++) {                              if (map.end[z] == teres.sub[q] ) is = 0;                          }                          if (is) {                              teres.sub[teres.num ] = map.end[z];                              teres.num++;                              c\_kong.push( map.end[z] );                              while (1) {                                  int temp = c\_kong.front(); c\_kong.pop();                                  for (int r = 0; r < map.num; r++ ) {                                      if (temp == map.start[r] && map.c[r] == 'k' ) {                                          bool iss = 1;                                          for (int qq = 0; qq < teres.num; qq++) {                                              if (map.end[r ] == teres.sub[qq] ) is = 0;                                          }                                          if (iss) {                                              teres.sub[teres.num ] = map.end[r];                                              teres.num++;                                              c\_kong.push(map.end[r]);                                          }                                      }                                  }                                  if (c\_kong.empty()) break;                              }                          }                      }                  }              }              int tempq;              for (int ii = 0; ii < teres.num - 1; ii++ ) {                  for (int jj = ii + 1; jj < teres.num; jj++ ) {                      if (teres.sub[jj] < teres.sub[ii] ) {                          tempq = teres.sub[ii];                          teres.sub[ii] = teres.sub[jj];                          teres.sub[jj] = tempq;                      }                  }              }              if (!isExist(teres)) {                  temp.push(teres);                  newState.res\_state[ newState.num ] = newState.num;                  newState.subSet[newState.num] = arrToStr(teres);                  if ( isExistStart(teres) ) {                      newState.isStart[newState.num] = 1;                  }                  else {                      newState.isStart[newState.num] = 0;                  }                  if ( isExistEnd(teres)) {                      newState.isEnd[newState.num] = 1;                  }                  else {                      newState.isEnd[newState.num] = 0;                  }                  newState.num++;              }              if (teres.num != 0) {                  haha.left[haha.num ] = arrToStr(te);                  haha.right[haha.num] = arrToStr(teres);                  haha.c[haha.num] = alpha.c[i];                  haha.num++;              }          }          if (temp.empty()) break;      }      for (int i = 0; i < haha.num; i++) {          int k;          for (k = 0; k < newState.num && newState.subSet[k] != haha.left[i]; k++ ) ;          newMap.start[newMap.num] = newState.res\_state[k];          int p;          for (p = 0; p < newState.num && newState.subSet[p] != haha.right[i]; p++ );          newMap.end[newMap.num ] = newState.res\_state[p];          newMap.c[newMap.num] = haha.c[i];          newMap.num++;      }      output();      return 0;  } |