Лабораторная работа №2

Конечные детерминированные автоматы. Преобразование недетерминированного конечного автомата к детерминированному

Теоретическая часть

Конечный автомат представляет собой пятерку

$$A = (Q, T, t, q_0, F)$$

где, Q - конечное множество состояний автомата,

T - непустое конечное множество входных символов, входной алфавит,

$q_0 \in Q$ - *начальное состояние*,

$F \subseteq Q$ - непустое множество заключительных состояний автомата,

t- переходная функция

$$t : Q \times T^* \to R(Q)$$

Такой автомат работает как распознаватель следующим образом (рис.1). На ленте записана входная строка, ограниченная с двух сторон концевыми маркерами. Управляющее устройство находится в начальном состоянии q_0. Входная лента последовательно просматривается слева направо по одному символу. Допустим, что с помощью считывающей головки оказались просмотрены символы $a_0 a_1 \ldots a_{i-1}$ и управляющее устройство находится в состоянии q_{i-1}. Символ a_i допускается, если существует хотя бы одно состояние $t(q, a_i) \in Q$.

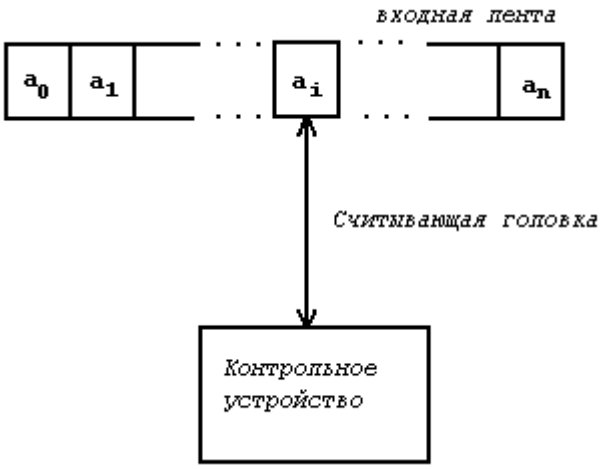

Рис. 1

Если такого состояния не существует, автомат останавливает свою работу. В общем случае таких состояний может быть несколько, и распознаватель будет недетерминированным, т.е. речь в этом случае будет идти о *недетерминированном конечном автомате*.

Строка $x = a_1 a_2 \ldots a_n \in T^+$ допускается автоматом, если существует последовательность состояний $q_0, q_1, \ldots, q_n$, такая, что $q_{i+1} = t(q_i, a_{i+1})$, где $i = 0,1,\ldots,n-1, q_n \in F$ и после прочтения последнего символа a_n строки был встречен

концевой маркер. Предполагается, что начальный маркер читается в состоянии q_0. Этот факт можно записать с помощью функции перехода следующим образом: $t(q_0, x) \in F$. В этом случае функцию перехода, соответствующую одному такту работы распознавателя, можно рассматривать, как частный случай ее общей формы записи для строки *x*, содержащей один символ $x = a_{i+1}$ и состояния q_i. Таким образом, функцию *t* можно определить рекурсивно $t(q, x) = t(t(q, a), y)$, если строка $x = ay$, где $a \in T$ и $y \in T^*$.

Множество всех строк *T(A)*, допускаемых автоматом *A*, может быть записано в виде

$$T(A) = \{x \mid x \in T^*, t(q_0, x) \in F\}$$

Практика показала, что одним из удобных способов задания функции переходов является *граф переходов* или его еще называют *граф состояний*. Граф состояний (V, E) является направленным графом и строится следующим образом:

Каждому состоянию множества *Q* соответствует одна вершина множества *V*.

Все вершины, кроме заключительных, обозначаются на рисунках в виде окружностей.

На начальную вершину, соответствующую начальному состоянию автомата, указывает стрелка.

Заключительная вершина изображается прямоугольником.

Ребро $(B, C) \in E$ существует на графе, если для символа $a \in T$ и состояния $B \in Q$ значение функции $t(B, a)$ не пусто и равно $C \in Q$.

На рис. 2 изображен граф состояний автомата *A*.

$A = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b, c\}, t, q, \{q_3, q_4\})$, где

$t(q_0, a) = q_1$

$t(q_0, b) = q_2$

$t(q_1, b) = q_4$

$t(q_2, c) = q_3$

$t(q_2, a) = q_4$

$t(q_4, c) = q_3$

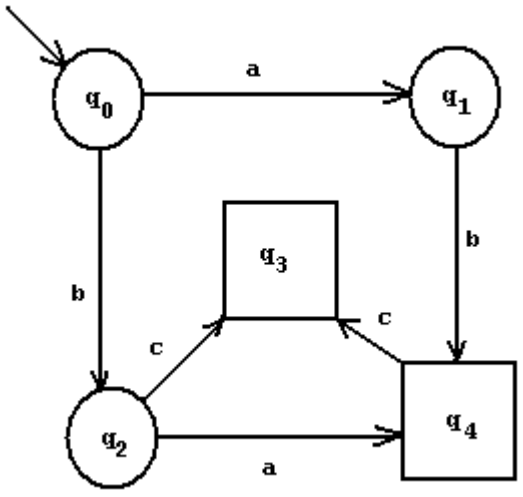

Рис. 2

Рассмотрим еще один пример грамматики, порождающей следующие предложения языка

$$a, a0, a00, \cdots, b0, b00, b00, \ldots$$

Построим для этой грамматики конечный автомат. Граф его состояний приведен на рис. 3.

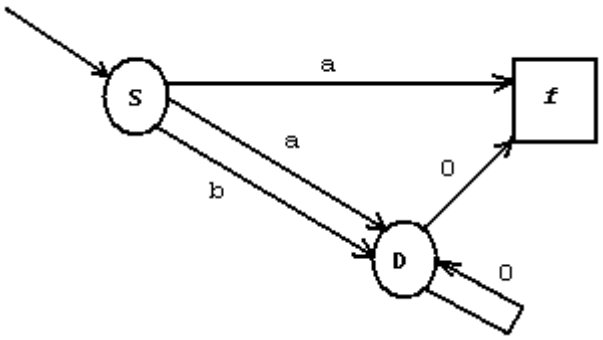

Рис. 3

По графу видно, что конечный автомат недетерминированный, поскольку из состояния *S* выходят два ребра с символом *a*. Функция перехода для символа *a* в состоянии *S* имеет два значения.

$$t(S, a) = \{f, D\}$$

Как известно, недетерминированный автомат достаточно сложно использовать для разбора. Поэтому интересует возможность преобразования недетерминированного автомата в детерминированный. Для конечных автоматов этот вопрос решается положительно. Доказано, что, если *A* - недетерминированный автомат, то существует детерминированный автомат *A'*, такой, что $T(A) = T(A')$. Допустим, что функция переходов $t(q,a)$ имеет множество значений $Q' \subset Q$. Будем считать это множество одним новым состоянием q'. Присоединим это состояние к множеству состояний Q автомата, т.е. $Q = Q \mid q'$. Определим функцию перехода для нового состояния q'.

$$t(q', a) = \bigcup_{i \in Q'} t(i, a) \text{ для всех } a \in T$$

Выполнив данное преобразование для всех функций перехода, имеющих не единственное значение, получим детерминированный автомат. Запишем для приведенного выше примера все функции перехода.

$$t(S, a) = \{f, D\}$$
$$t(S, b) = D$$
$$t(D, 0) = D$$
$$t(D, 0) = \{f\}$$

Введем новое состояние $q' = \{D, f\}$ и перепишем функции перехода.

$$t(S, a) = q'$$
$$t(S, b) = D$$
$$t(D, 0) = q'$$

$t(q',0) = q'$ так как одно состояние из $\{D, f\}$ f - конечное, а $t(D,0) = q'$

Граф состояний полученного детерминированного автомата показан на рис. 4.

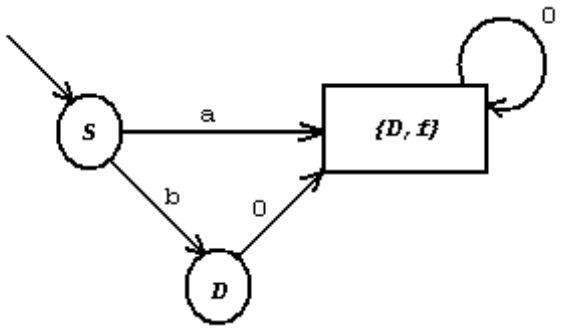

Рис. 4

Следует заметить, что детерминирование автомата не должно изменить множество предложений (или входных строк), которые этот автомат разбирает.

Задание на лабораторную работу

Написать программу, реализующую работу конечного автомата.

Входные данные:

1. текстовый файл, описывающий граф переходов конечного автомата. Файл представляет собой набор строк. В каждой строке задаётся **одно** правило перехода в следующем виде:

   q<N>,<C>=<q|f><N>

   Здесь символ q обозначает состояние автомата, f – конечное состояние автомата, <N> - произвольное число, обозначающее номер состояния, <C> - **один** символ.

   Пример: q12,g=f0 – запись означает, что если автомат находится в состоянии №12 и читает с ленты символ ‘g’, то он перейдёт в конечное состояние с номером 0

   Дополнительные условия
   Количество строк в файле (возможных переходов) – неограниченное количество
   Начальное состояние автомата (с которого начинается его работа) – q0
   Строки в файле не обязаны быть отсортированы по какому-либо критерию
   Состояния автомата необязательно нумеруются последовательно

2. строка символов, которую нужно проанализировать с помощью построенного автомата и дать заключение о возможности (или невозможности) разбора этой строки с помощью данного автомата

   Пример: строки *ab, abc, ba* допускаются автоматом, граф переходов которого изображён на рис. 2. Строки *b, ak, bad* этим автоматом не допускаются.

Выходные данные:

1. Заключение о детерминированности или недетерминированности заданного автомата.
2. В случае недетерминированного автомата вывести переходы для соответствующего ему детерминированного автомата (в виде, соответствующем входному файлу).
3. Заключение о возможности (невозможности) разбора автоматом введённой строки символов

Задание минимум (на 3**):** считать из файла автомат (файл не содержит синтаксических ошибок, заданный с его помощью автомат детерминирован), дать заключение о возможности (невозможности) разбора этим автоматом введённой строки.

Задание максимум (на 5. В принципе, совершенству нет предела, но тем не менее**)**: считать из файла автомат (файл может содержать синтаксические ошибки, заданный с его помощью автомат недетерминирован, возможно, граф переходов содержит висячие вершины), вывести информацию об автомате (детерминирован/нет, всё, что угодно), детерминировать автомат, вывести таблицу переходов для нового автомата, разобрать входную строку и дать заключение о возможности/невозможности её разбора.

Весьма неплохо было бы при написании программы использовать возможности **объектно-ориентированного** программирования. В качестве дополнения, например, можно графически представить граф переходов для автомата до и после проведения операции детерминирования.

При возникновении неоднозначности в оценке, может быть предложено в качестве дополнительного задания сделать файл для автомата, разбирающего какую-то строку (например, if (a>b) exit(1);).

Литература

При подготовка данной лабораторной работы были использованы лекции Тимофеева П.А. по курсу «Теория алгоритмических языков и компиляторов».

Приложения

1. Пример входного файла для автомата, разбирающего строку for ([abc] = [0-9]; [abc] [<|>] [0-9]; [abc]++)

Здесь [abc] – строка произвольной длины, состоящая из символов a, b, c, A, B, C

[0-9] – строка произвольной длины, состоящая из символов цифр

[<|>] – или символ “<”, или символ “>”

q0, =q0
q0,f=q1
q1,o=q2
q2,r=q3
q3, =q3
q3,(=q4
q4, =q4
q4,i=q5
q5,n=q6
q6,t=q7
q7, =q7
q7,a=q8
q7,b=q8
q7,c=q8
q7,A=q8
q7,B=q8
q7,C=q8
q8,a=q8
q8,b=q8
q8,c=q8
q8,A=q8
q8,B=q8
q8,C=q8
q8,==q10
q8, =q9
q9, =q9
q9,==q10
q10, =q10
q10,0=q11
q10,1=q11
q10,2=q11
q10,3=q11
q10,4=q11
q10,5=q11
q10,6=q11
q10,7=q11
q10,8=q11
q10,9=q11
q11,0=q11
q11,1=q11
q11,2=q11
q11,3=q11
q11,4=q11
q11,5=q11
q11,6=q11
q11,7=q11
q11,8=q11
q11,9=q11
q11,;=q13
q11, =q12
q12, =q12
q12,;=q13
q13, =q13
q13,a=q14
q13,b=q14
q13,c=q14
q13,A=q14
q13,B=q14
q13,C=q14
q14,a=q14
q14,b=q14
q14,c=q14
q14,A=q14
q14,B=q14
q14,C=q14
q14, =q15
q14,<=q16
q14,>=q16
q15, =q15
q15,<=q16
q15,>=q16
q16, =q16
q16,0=q17
q16,1=q17
q16,2=q17
q16,3=q17
q16,4=q17
q16,5=q17
q16,6=q17
q16,7=q17
q16,8=q17
q16,9=q17
q17,0=q17
q17,1=q17
q17,2=q17
q17,3=q17
q17,4=q17
q17,5=q17
q17,6=q17
q17,7=q17
q17,8=q17
q17,9=q17
q17, =q18
q17,;=q19
q18, =q18
q18,;=q19
q19, =q19
q19,a=q20
q19,b=q20
q19,c=q20
q19,A=q20
q19,B=q20
q19,C=q20
q20,a=q20
q20,b=q20
q20,c=q20
q20,A=q20
q20,B=q20
q20,C=q20
q20, =q21
q20,+=q22
q21, =q21
q21,+=q22
q22,+=q23
q23, =q23
q23,)=f0

2. Пример работающей программы (операция детерминирования не производится)

```
#include <string>
#include <vector>
#include <algorithm>
#include <iostream>
#include <fstream>

typedef unsigned char uchar;

using namespace std;

class format_error: public runtime_error {
public:
  format_error(const char* msg): runtime_error(msg){}
};

class StateReader{ //class for reading states from file
public:
  struct ElementarySwitch{ // one switch for automat
    int initialState;      // number of current state
    uchar letter;          // reading symbol
    bool isTerminalState;  // is next state terminal?
    int nextState;         // number of next state
    ElementarySwitch():
      initialState(-1),
      letter(0),
      isTerminalState(false),
      nextState(-1)
      {}

    // next 2 operators - for sorting states array
    friend bool operator > (const ElementarySwitch& el, const ElementarySwitch& er){
      if (el.initialState > er.initialState) return true;
      if (el.initialState == er.initialState){
        if (el.letter > er.letter) return true;
        if (el.letter == er.letter){
          if (el.isTerminalState != er.isTerminalState) return er.isTerminalState;
          if (el.nextState > er.nextState) return true;
        }
      }
      return false;
    }

    friend bool operator < (const ElementarySwitch& el, const ElementarySwitch& er){
      return !operator>(el, er);
    }
  };

  typedef vector<ElementarySwitch> StatesSwitchArray;

  StateReader(const char* filename);
  ~StateReader() {stateFile.close();}

protected:
  StatesSwitchArray statemachineStates; // array of switches for automat
```

```
private:
  ifstream stateFile; // stream for reading file
};

StateReader::StateReader(const char* filename):stateFile(filename){
  if(!stateFile.is_open()) throw runtime_error("Invalid states file"); // can't open file

  string tmpStr;
  while(getline(stateFile, tmpStr)){
    if (tmpStr.size() == 0) continue; // skip empty string
    // several check for input file format
    if (tmpStr[0] != 'q' && tmpStr[0] != 'Q')
      throw format_error("Line must begin with 'q' letter");
    string::size_type commaPos = tmpStr.find(',');
    if (commaPos == string::npos)
      throw format_error("There is no comma");
    string stateNumber = tmpStr.substr(1, commaPos - 1);
    ElementarySwitch tmpSw; // prepare next elementh in array
    tmpSw.initialState = atoi(stateNumber.c_str());
    if (tmpSw.initialState == 0 && stateNumber[0] != '0')
      throw format_error("State number must contains digits only");
    tmpSw.letter = tmpStr[commaPos + 1];
    if (tmpStr[commaPos + 2] != '=')
      throw format_error("Expected '=' sign");
    switch (tmpStr[commaPos + 3]){
      case 'f':
      case 'F':
        tmpSw.isTerminalState = true;
        break;
      case 'q':
      case 'Q':
        tmpSw.isTerminalState = false;
        break;
      default:
        throw format_error("Next state must begin with 'q' or 'f' letter");
    }
    stateNumber = tmpStr.substr(commaPos + 4);
    tmpSw.nextState = atoi(stateNumber.c_str());
    if (tmpSw.nextState == 0 && stateNumber[0] != '0')
      throw format_error("State number must contains digits only");

    statemachineStates.push_back(tmpSw); // add one switch to array of switches
  }
}

class StateMachine: public StateReader{
public:
  StateMachine(const char* filename);
  bool isDeterministic() const {return deterministic;}
  bool hasHangs() const {return hangs;} // means that graph contains isolated node(s)
  const StateReader::StatesSwitchArray& GetSwitches() const {return statemachineStates;} // just for
printing
  bool isExpressionCorrect(const string& expression, int& errorPos);

protected:
  void SortStates();
  bool _isDeterministic();
```

```
  bool _hasHangs();
  int _findNextIndex(int curState, uchar sym);

private:
  bool deterministic;
  bool hangs;
};

StateMachine::StateMachine(const char* filename):
  StateReader(filename),
  deterministic(true),
  hangs(false)
{
  SortStates();
  //some little check of machine
  if (statemachineStates.size() == 0)
    throw runtime_error("Automat is empty");
  if (statemachineStates[0].initialState != 0)
    throw runtime_error("There is no initial state");
  size_t ln = statemachineStates.size();
  bool hasFinalState = false;
  for (size_t i = 0; i < ln; i++)
    if (hasFinalState = statemachineStates[i].isTerminalState)
      break;
  if (!hasFinalState)
    throw runtime_error("There is no final state");

  deterministic = _isDeterministic(); // check if automat is deterministic
  hangs = _hasHangs();                // check if may be hangs
}

bool StateMachine::_isDeterministic(){
  size_t ln = statemachineStates.size(); // count of elements in array
  bool isDet = true;
  for (size_t i = 1; i < ln; i++)
    if (statemachineStates[i-1].initialState == statemachineStates[i].initialState &&
        statemachineStates[i-1].letter == statemachineStates[i].letter &&
        (statemachineStates[i-1].isTerminalState != statemachineStates[i].isTerminalState ||
         statemachineStates[i-1].nextState != statemachineStates[i].nextState))
    {
      isDet = false;
      break;
    };

  return isDet;
}

bool StateMachine::_hasHangs(){
  size_t ln = statemachineStates.size();
  bool isHangs = false;
  for (size_t i = 0; i < ln; i++){
    if (!statemachineStates[i].isTerminalState){
      bool found = false;
      // very bad algorithm to search in _SORTED_ array. I was laziness to do better :->
      for (size_t j = 0; j < ln; j++){
        if (statemachineStates[i].nextState == statemachineStates[j].initialState){
          found = true;
```

```
        break;
       }
      }
      if (!found){
       isHangs = true;
       break;
      }
    }
  }
  return isHangs;
}

void StateMachine::SortStates(){
  sort(statemachineStates.begin(), statemachineStates.end()); // common sorting algorithm from
<algorithm>
}

int StateMachine::_findNextIndex(int curState, uchar sym){
  int found = -1;
  size_t ln = statemachineStates.size();

  // very bad algorithm to search in _SORTED_ array
  for (size_t j = 0; j < ln; j++){
    if (statemachineStates[j].initialState == curState &&
        statemachineStates[j].letter == sym){
      found = j;
      break;
    }
  }
  return found;
}

bool StateMachine::isExpressionCorrect(const string& expression, int& errorPos){
  if ( !deterministic || hangs)
    throw runtime_error("This automat cannot check expression");
  // emulate automat's task
  int currentState = 0;
  size_t strLen = expression.size();
  for (int i = 0; i < strLen; i++){
    int idx = _findNextIndex(currentState, expression[i]);
    if (idx < 0){
      errorPos = i;
      return false;
    }
    if (statemachineStates[idx].isTerminalState){
      if (i == strLen - 1) return true;
      errorPos = i + 1;
      return false;
    }
    currentState = statemachineStates[idx].nextState;
  }
  errorPos = strLen;
  return false;
}

int _tmain(int argc, _TCHAR* argv[]) {
  try{ // try to create object of StateMachine
```

```
    StateMachine sr("states.txt");
    StateReader::StatesSwitchArray::const_iterator it; // another way to access to array's elements
    for (it = sr.GetSwitches().begin(); it != sr.GetSwitches().end(); it++)
      cout << "q" << it->initialState << "," << it->letter << "=" << (it->isTerminalState ? "f" : "q") << it-
>nextState << endl;
    cout << "There are" << (sr.hasHangs() ? "" : "n't") << " hangs" << endl;
    cout << "Automat is" << (sr.isDeterministic() ? "" : "n't") << " deterministic" << endl;

    string testExpr;
    cout << "Please, enter expression to check ";
    cin >> testExpr;
    // for test with related "states.txt" file
    // testExpr = "  for( int abAccc= 943 ; a<478; bbc++ )";
    // cout << testExpr << endl;
    int err;
    bool res = sr.isExpressionCorrect(testExpr, err);
    if (res) cout << "Expression is correct!" << endl;
    else cout << "Incorrect expression. Error position: " << err << endl;
  }
  catch(const exception& err){
    cerr << err.what() << endl;
  }
  return 0;
}

// Mark for such realization will be 4
```