Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное

образовательное учреждение высшего образования

«Национальный исследовательский университет

«Московский институт электронной техники»

институт системной и программной инженерии

и информационных технологий»

Курс «Теория алгоритмических языков и компиляторов»

Лабораторная работа № 2

Конечные детерминированные автоматы. Преобразование недетерминированного конечного автомата к детерминированному

Выполнил:

Студент группы ПИН-41Д

Джугели Дмитрий Александрович

Москва, 2025

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);).

Код:

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

using namespace std;

map<pair<string, char>, set<string>> nd_table;
map<pair<string, char>, string> d_table;
vector<string> unused_st;
set<string> used_st;
set<char> alphabet;

string make_state(const set<string>& v) {
    string name;
    for (const auto& i : v) {
        name += i;
    }
    sort(name.begin(), name.end());
    for (const auto& i : alphabet) {
        for (const auto& j : v) {
            auto tmp_pair = make_pair(j, i);
            if (nd_table.find(tmp_pair) != nd_table.end()) {
                auto tmp_vec = nd_table[tmp_pair];
                nd_table[make_pair(name, i)].insert(tmp_vec.begin(), tmp_vec.end());
            }
        }
    }
    return name;
}

void print(const map<pair<string, char>, string>& table) {
    for (const auto& [k, v] : table) {
        cout << k.first << "," << k.second << "=" << v << "\n";
    }
}

void determ() {
    while (!unused_st.empty()) {
        string Qcur = unused_st[0];
        for (const auto& i : alphabet) {
            auto tmp_pair = make_pair(Qcur, i);
            if (nd_table.find(tmp_pair) == nd_table.end()) {
                continue;
            }
            auto tmp_vec = nd_table[tmp_pair];
            string state = make_state(tmp_vec);
            if (used_st.find(state) == used_st.end() && !state.empty()) {
                unused_st.push_back(state);
            }
            d_table[make_pair(Qcur, i)] = state;
        }
        used_st.insert(Qcur);
        unused_st.erase(unused_st.begin());
    }
}

bool parse_automate(const string& file_name) {
    ifstream file(file_name);
```

```
    if (!file.is_open()) {
        cerr << "Failed to open file: " << file_name << endl;
        return false;
    }

    string str;
    char c;
    string q, f;

    getline(file, str);
    q = str.substr(0, str.find(','));
    unused_st.push_back(q);
    c = str.substr(str.find(',') + 1, str.find_last_of('=') - str.find(',') - 1)[0];
    alphabet.insert(c);
    f = str.substr(str.find_last_of('=') + 1);
    nd_table[make_pair(q, c)].insert(f);

    while (getline(file, str)) {
        q = str.substr(0, str.find(','));
        c = str.substr(str.find(',') + 1, str.find_last_of('=') - str.find(',') - 1)[0];
        alphabet.insert(c);
        f = str.substr(str.find_last_of('=') + 1);
        nd_table[make_pair(q, c)].insert(f);
    }

    bool is_determ = true;
    for (const auto& [k, v] : nd_table) {
        if (v.size() > 1) {
            is_determ = false;
            break;
        }
    }

    if (is_determ) {
        for (const auto& [k, v] : nd_table) {
            d_table[k] = *v.begin();
        }
    }
    else {
        determ();
    }

    cout << "Automaton is " << (is_determ ? "deterministic" : "non deterministic") << endl;
    cout << "Transition table:" << endl;
    print(d_table);

    return true;
}

int parse_str(const string& str) {
    string Qcur = "q0";
    size_t len = str.size();
    for (size_t i = 0; i < len; i++) {
        auto tmp_pair = make_pair(Qcur, str[i]);
        if (d_table.find(tmp_pair) == d_table.end()) {
            return -1;
        }
        Qcur = d_table[tmp_pair];
    }
    return Qcur.find('f') != string::npos ? 0 : -1;
}

void parsing_loop() {
    string str;
    while (cin >> str) {
```

```
        int ret = parse_str(str);
        cout << (ret < 0 ? "NOT VALID" : "VALID") << "\n";
    }
}

int main() {
    if (parse_automate("C:\\var3_nd.txt")) {
        parsing_loop();
    }
    return 0;
}
```

Тестовые данные: