

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

ЛАБОРАТОРНАЯ РАБОТА №2
по дисциплине «Теория автоматов и формальных языков»

Выполнил
студент гр. 5130904/30108

Ребдев П.А.

Проверил Тышкевич А.И.

Санкт-Петербург
2026

1. Формулировка задания

Реализовать программу для минимизации конечного автомата с использованием алгоритма k-эквивалентности и проверки эквивалентности исходного и минимального автоматов.

2. Алгоритм минимизации

Алгоритм основан на последовательном построении разбиений состояний по k -эквивалентности:

1. $K^0 = \{S\}$ - все состояния в одном классе
2. K^1 - разбиение по выходной функции (1-эквивалентность)
3. K^2, K^3, \dots - дальнейшее уточнение разбиения по переходной функции
4. Процесс продолжается, пока $K^{(k+1)} \neq K^{(k)}$

3. Программное решение

Для решения задачи была реализована программа на языке программирования с++

main.cpp:

```
#include <iostream>
#include <vector>
#include <queue>
#include <set>
#include <map>

struct Automaton
{
    int statesCount;
    int inputAlphabetSize;
    int outputAlphabetSize;
    int initialState;
    std::vector<std::vector<int>> transition;
    std::vector<std::vector<int>> output;
};

struct MinimalAutomaton
{
    int statesCount;
    int inputAlphabetSize;
    int outputAlphabetSize;
    int initialState;
    std::vector<std::vector<int>> transition;
    std::vector<std::vector<int>> output;
    std::map<int, std::set<int>> equivalenceClasses;
};

bool areEquivalent(const Automaton & a1, const Automaton & a2)
{
    if (a1.inputAlphabetSize != a2.inputAlphabetSize)
    {
        return false;
    }

    int n1 = a1.statesCount;
    int n2 = a2.statesCount;
    int m = a1.inputAlphabetSize;

    std::vector<std::vector<bool>> visited(n1, std::vector<bool>(n2, false));
    std::queue<std::pair<int, int>> q;

    q.push(std::make_pair(a1.initialState, a2.initialState));
    visited[a1.initialState][a2.initialState] = true;

    while (!q.empty())
    {
        auto current = q.front();
        int s1 = current.first;
        int s2 = current.second;
        q.pop();

        for (int a = 0; a < m; ++a)
        {
            if (visited[s1][a] == false)
            {
                for (int i = 0; i < n2; ++i)
                {
                    if (a1.transition[s1][a] == a2.transition[s2][i])
                    {
                        visited[s1][a] = true;
                        q.push(std::make_pair(s1, a2.transition[s2][i]));
                    }
                }
            }
        }
    }

    for (int i = 0; i < n1; ++i)
    {
        for (int j = 0; j < n2; ++j)
        {
            if (visited[i][j] == false)
            {
                return false;
            }
        }
    }

    return true;
}
```

```

int out1 = a1.output[s1][a];
int out2 = a2.output[s2][a];

if (out1 != out2)
{
    return false;
}

int next1 = a1.transition[s1][a];
int next2 = a2.transition[s2][a];

if (!visited[next1][next2])
{
    visited[next1][next2] = true;
    q.push(std::make_pair(next1, next2));
}
}

return true;
}

MinimalAutomaton minimizeAutomaton(const Automaton & automaton)
{
    MinimalAutomaton minimal;
    minimal.inputAlphabetSize = automaton.inputAlphabetSize;
    minimal.outputAlphabetSize = automaton.outputAlphabetSize;

    std::vector<std::set<int>> currentPartition;
    std::vector<std::set<int>> nextPartition;
    std::vector<int> stateToClass(automaton.statesCount);

    std::cout << "\n==== Начало минимизации ===\n";

    std::set<int> allStates;
    for (int i = 0; i < automaton.statesCount; ++i)
    {
        allStates.insert(i);
    }
    currentPartition.push_back(allStates);

    std::cout << "K0: { ";
    for (int state : allStates)
    {
        std::cout << state + 1 << " ";
    }
    std::cout << "}\n";

    int iteration = 0;
    bool changed = true;

    while (changed)
    {
        ++iteration;
        changed = false;

        std::cout << "\n--- Итерация " << iteration << " ---\n";

        std::map<std::string, std::set<int>> signatureToStates;

        for (int i = 0; i < automaton.statesCount; ++i)
        {

```

```

    std::string signature;

    for (int j = 0; j < automaton.inputAlphabetSize; ++j)
    {
        signature += std::to_string(automaton.output[i][j]);
    }

    for (const auto & classSet : currentPartition)
    {
        for (int j = 0; j < automaton.inputAlphabetSize; ++j)
        {
            int nextState = automaton.transition[i][j];
            if (classSet.find(nextState) != classSet.end())
            {
                for (int k = 0; k < currentPartition.size(); ++k)
                {
                    if (currentPartition[k] == classSet)
                    {
                        signature += std::to_string(k);
                        break;
                    }
                }
                break;
            }
        }
    }

    signatureToStates[signature].insert(i);
}

nextPartition.clear();
for (const auto & entry : signatureToStates)
{
    nextPartition.push_back(entry.second);
}

std::cout << "K" << iteration << ": ";
for (size_t i = 0; i < nextPartition.size(); ++i)
{
    std::cout << "{ ";
    for (int state : nextPartition[i])
    {
        std::cout << state + 1 << " ";
    }
    std::cout << "}" ;
}
std::cout << "\n";

if (nextPartition.size() != currentPartition.size())
{
    changed = true;
}
else
{
    for (size_t i = 0; i < nextPartition.size(); ++i)
    {
        if (nextPartition[i] != currentPartition[i])
        {
            changed = true;
            break;
        }
    }
}

```

```

}

currentPartition = nextPartition;
}

std::cout << "\nМинимизация завершена. Конечное разбиение:\n";
for (size_t i = 0; i < currentPartition.size(); ++i)
{
    std::cout << "Класс " << char('A' + i) << ": { ";
    for (int state : currentPartition[i])
    {
        std::cout << state + 1 << " ";
        stateToClass[state] = i;
    }
    std::cout << " }\n";
    minimal.equivalenceClasses[i] = currentPartition[i];
}

minimal.statesCount = currentPartition.size();

for (size_t i = 0; i < currentPartition.size(); ++i)
{
    if (currentPartition[i].find(automaton.initialState) != currentPartition[i].end())
    {
        minimal.initialState = i;
        break;
    }
}

minimal.transition.resize(minimal.statesCount,
                         std::vector<int>(minimal.inputAlphabetSize));
minimal.output.resize(minimal.statesCount,
                      std::vector<int>(minimal.outputAlphabetSize));

std::cout << "\n==== Минимальный автомат ====\n";
std::cout << "Состояний: " << minimal.statesCount << "\n";
std::cout << "Начальное состояние: " << char('A' + minimal.initialState) << "\n\n";

std::cout << "Таблица выходов ( $\lambda_{\text{min}}$ ):\n";
std::cout << " a b\n";
for (int i = 0; i < minimal.statesCount; ++i)
{
    int representative = *(currentPartition[i].begin());
    minimal.output[i][0] = automaton.output[representative][0];
    minimal.output[i][1] = automaton.output[representative][1];

    std::cout << char('A' + i) << " "
           << minimal.output[i][0] << " "
           << minimal.output[i][1] << "\n";
}

std::cout << "\nТаблица переходов ( $\delta_{\text{min}}$ ):\n";
std::cout << " a b\n";
for (int i = 0; i < minimal.statesCount; ++i)
{
    int representative = *(currentPartition[i].begin());

    for (int j = 0; j < minimal.inputAlphabetSize; ++j)
    {
        int nextState = automaton.transition[representative][j];
        for (int k = 0; k < minimal.statesCount; ++k)
        {

```

```

        if (currentPartition[k].find(nextState) != currentPartition[k].end())
        {
            minimal.transition[i][j] = k;
            break;
        }
    }

    std::cout << char('A' + i) << " "
        << char('A' + minimal.transition[i][0]) << " "
        << char('A' + minimal.transition[i][1]) << "\n";
}

return minimal;
}

int main()
{
    std::cout << "Программа минимизации конечных автоматов\n";
    std::cout << "=====\\n\\n";

    Automaton original;

    original.statesCount = 6;
    original.inputAlphabetSize = 2;
    original.outputAlphabetSize = 2;
    original.initialState = 0;

    original.transition = {
        {1, 5},
        {1, 2},
        {0, 3},
        {4, 2},
        {1, 5},
        {0, 3}
    };

    original.output = {
        {1, 0},
        {0, 1},
        {0, 1},
        {1, 0},
        {0, 1},
        {0, 1}
    };

    std::cout << "==== Исходный автомат ===\\n";
    std::cout << "Состояний: " << original.statesCount << "\\n";
    std::cout << "Входных символов: " << original.inputAlphabetSize << "\\n";
    std::cout << "Выходных символов: " << original.outputAlphabetSize << "\\n";
    std::cout << "Начальное состояние: " << original.initialState + 1 << "\\n\\n";

    std::cout << "Таблица переходов ( $\delta$ ):\\n";
    std::cout << " a b\\n";
    for (int i = 0; i < original.statesCount; ++i)
    {
        std::cout << i + 1 << " "
            << original.transition[i][0] + 1 << " "
            << original.transition[i][1] + 1 << "\\n";
    }

    std::cout << "\\nТаблица выходов ( $\lambda$ ):\\n";
}

```

```

std::cout << " a b\n";
for (int i = 0; i < original.statesCount; ++i)
{
    std::cout << i + 1 << " "
        << original.output[i][0] << " "
        << original.output[i][1] << "\n";
}

MinimalAutomaton minimal = minimizeAutomaton(original);

Automaton minimalConverted;
minimalConverted.statesCount = minimal.statesCount;
minimalConverted.inputAlphabetSize = minimal.inputAlphabetSize;
minimalConverted.outputAlphabetSize = minimal.outputAlphabetSize;
minimalConverted.initialState = minimal.initialState;
minimalConverted.transition = minimal.transition;
minimalConverted.output = minimal.output;

std::cout << "\n==== Проверка эквивалентности ====\n";
std::cout << "Проверяем эквивалентность исходного и минимального автоматаов...\n";

if (areEquivalent(original, minimalConverted))
{
    std::cout << "Автоматы эквивалентны!\n";
}
else
{
    std::cout << "Автоматы не эквивалентны!\n";
}

std::cout << "\n==== Прямое произведение для проверки ====\n";

std::vector<std::vector<bool> > visited(original.statesCount,
                                             std::vector<bool>(minimal.statesCount, false));
std::queue<std::pair<int, int> > q;

q.push(std::make_pair(original.initialState, minimal.initialState));
visited[original.initialState][minimal.initialState] = true;

int step = 0;
while (!q.empty())
{
    auto current = q.front();
    int s1 = current.first;
    int s2 = current.second;
    q.pop();
    ++step;

    std::cout << "Шар " << step << ": состояние("
        << s1 + 1 << ", " << char('A' + s2) << ")\n";

    for (int a = 0; a < original.inputAlphabetSize; ++a)
    {
        int out1 = original.output[s1][a];
        int out2 = minimal.output[s2][a];

        std::cout << " Вход " << a << ": выходы " << out1 << " и " << out2;
        if (out1 != out2)
        {
            std::cout << " -> Различаются!\n";
        }
        else
    }
}

```

```
{  
    std::cout << " -> Совпадают.\n";  
}  
  
int next1 = original.transition[s1][a];  
int next2 = minimal.transition[s2][a];  
  
if (!visited[next1][next2])  
{  
    visited[next1][next2] = true;  
    q.push(std::make_pair(next1, next2));  
    std::cout << " Добавлено состояние ("  
        << next1 + 1 << ", " << char('A' + next2) << ")\n";  
}  
}  
  
return 0;  
}
```

4. Протокол выполнения

```
Программа минимизации конечных автоматов
=====
==== Исходный автомат ===
Состояний: 6
Входных символов: 2
Выходных символов: 2
Начальное состояние: 1

Таблица переходов ( $\delta$ ):
  a   b
1  2   6
2  2   3
3  1   4
4  5   3
5  2   6
6  1   4

Таблица выходов ( $\lambda$ ):
  a   b
1  1   0
2  0   1
3  0   1
4  1   0
5  0   1
6  0   1

==== Начало минимизации ===
K0: { 1 2 3 4 5 6 }

--- Итерация 1 ---
K1: { 2 3 5 6 } { 1 4 }

--- Итерация 2 ---
K2: { 2 5 } { 3 6 } { 1 4 }

--- Итерация 3 ---
K3: { 2 5 } { 3 6 } { 1 4 }

Минимизация завершена. Конечное разбиение:
Класс A: { 2 5 }
Класс B: { 3 6 }
Класс C: { 1 4 }

==== Минимальный автомат ===
Состояний: 3
Начальное состояние: С

Таблица выходов ( $\lambda_{min}$ ):
  a   b
A  0   1
B  0   1
C  1   0

Таблица переходов ( $\delta_{min}$ ):
  a   b
A  A   B
B  C   C
C  A   B

==== Проверка эквивалентности ===
Проверяем эквивалентность исходного и минимального автоматов...
Автоматы эквивалентны!

==== Прямое произведение для проверки ===
Шаг 1: состояние (1, C)
Вход 0: выходы 1 и 1 -> Совпадают.
Добавлено состояние (2, A)
Вход 1: выходы 0 и 0 -> Совпадают.
Добавлено состояние (6, B)
Шаг 2: состояние (2, A)
Вход 0: выходы 0 и 0 -> Совпадают.
Вход 1: выходы 1 и 1 -> Совпадают.
Добавлено состояние (3, B)
Шаг 3: состояние (6, B)
Вход 0: выходы 0 и 0 -> Совпадают.
Вход 1: выходы 1 и 1 -> Совпадают.
Добавлено состояние (4, C)
Шаг 4: состояние (3, B)
Вход 0: выходы 0 и 0 -> Совпадают.
Вход 1: выходы 1 и 1 -> Совпадают.
Шаг 5: состояние (4, C)
Вход 0: выходы 1 и 1 -> Совпадают.
Добавлено состояние (5, A)
Вход 1: выходы 0 и 0 -> Совпадают.
Шаг 6: состояние (5, A)
Вход 0: выходы 0 и 0 -> Совпадают.
Вход 1: выходы 1 и 1 -> Совпадают.
```

5. Интерпретация результата

1. Минимизация выполнена успешно: Автомат из 6 состояний минимизирован до 3 состояний
 - Классы эквивалентности:
 - Класс А: состояния {1, 4}
 - Класс В: состояния {2, 5}
 - Класс С: состояния {3, 6}
2. Минимальный автомат соответствует ожидаемому результату из примеров
3. Проверка эквивалентности: Исходный и минимальный автоматы эквивалентны
4. Прямое произведение подтверждает эквивалентность для всех достижимых состояний

6. Выводы

1. Реализован алгоритм минимизации конечных автоматов на основе k -эквивалентности
2. Программа корректно находит классы эквивалентности состояний
3. Построенный минимальный автомат сохраняет функциональность исходного
4. Проверка через прямое произведение подтверждает корректность минимизации
5. Результаты соответствуют примерам из учебных материалов
6. Программа может быть использована для минимизации произвольных конечных автоматов