Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
**«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Институт космических и информационных технологий
институт
Программная инженерия
кафедра

# ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №1

Конечные автоматы
тема

Преподаватель _____ А. С. Кузнецов
подпись, дата — инициалы, фамилия

Студент КИ23-17/1Б, 032320072 _____ М. А. Мальцев
номер группы, зачетной книжки — подпись, дата — инициалы, фамилия

Красноярск 2025

## 1 Цель

Реализация и исследование детерминированных и недетерминированных конечных автоматов.

## 2 Задание

Вариант – 6.

Для выполнения практической работы необходимо разработать в системе JFLAP конечные автоматы и произвести программную реализацию на языке C++ для следующих автоматов:

1) Построить ДКА, допускающий в алфавите {0, 1} множество всех цепочек, у которых на пятой позиции справа стоит 1.

2) Построить НКА с количеством состояний, не превышающим 3, для языка $\{a^n : n \geq 1\} \cup \{b^m a^k : m, k \geq 0\}$.

## 3 Ход выполнения

Для начала была установлена программа JFLAP, в которой были построены конечные автоматы из условия задания. Каждый КА был сначала протестирован в JFLAP тестовыми цепочками, затем была написана программная реализация на C++, которая также была протестирована на корректность работы теми же самыми тестовыми цепочками.

### 3.1 Построение ДКА

Необходимо было реализовать детерминированный конечный автомат (ДКА), допускающий в алфавите {0, 1} множество всех цепочек, у которых на пятой позиции справа стоит 1. Исходя из формулировки задачи было выдвинуто предположение, что на вход автомату может поступать строка неограниченной длины и необходимо определить, является ли пятый символ с конца единицей. Для решения данной задачи была придумана модель скользящего окна, которая имеет длину в 5 символов и проходит от самого начала строки до её конца (в самом начале окно слева от строки и заполнено нулями, в самом конце оно содержит последние пять символов исследуемой строки). В таком окне

максимально возможное количество различных комбинаций 1 и 0 это 32, и все они могут получиться при сдвиге окна, поэтому в ДКА будет 32 состояния и из каждого будут выходить два перехода (для 0 и 1). На рисунке 1 показана реализация ДКА первого задания в системе JFLAP.

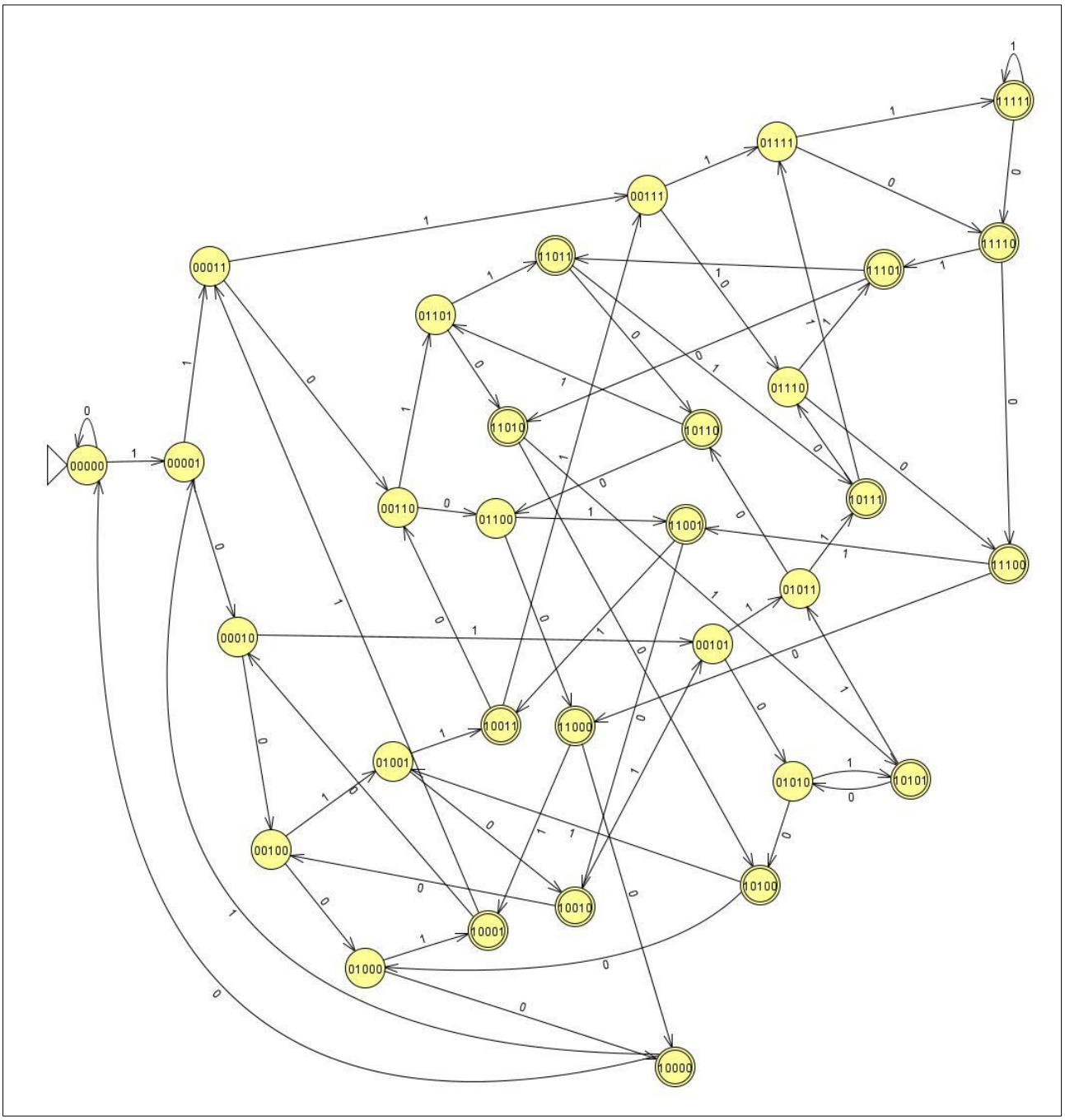



Рисунок 1 – ДКА в JFLAP

Конечный автомат получился довольно объемным благодаря условию задачи. Переписывание 64 функции перехода в таблицу могло занять много времени, поэтому было принято решение перейти к тестам. На рисунке 2 показан

тест ДКА для цепочки «0000110101», пятая цифра справа которого является единицей.

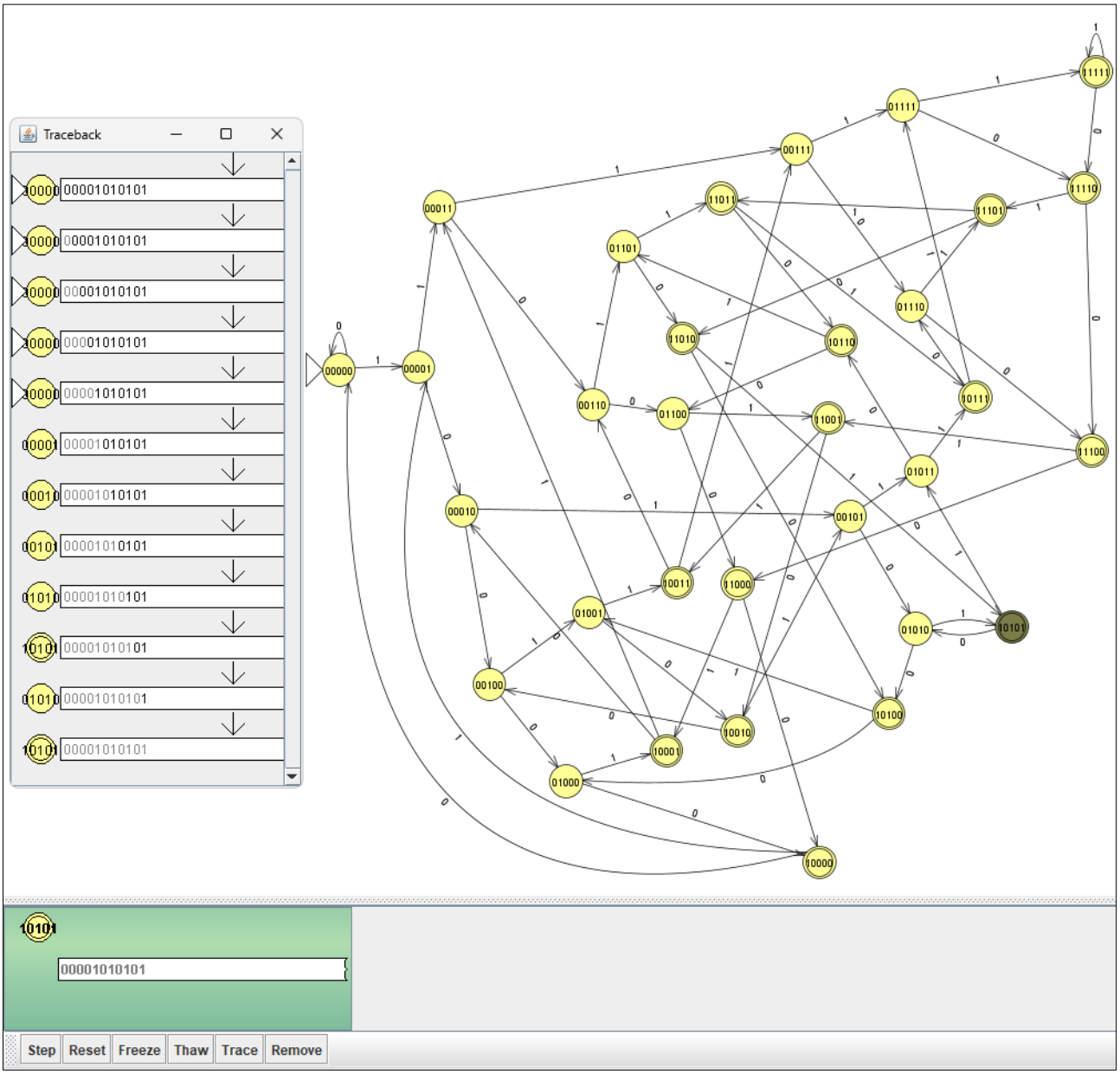



Рисунок 2 – Тест для цепочки «00001010101»

Также была проверка на отклонение строки, которая не удовлетворяет условию. На рисунке 3 показан тест ДКА для цепочки «0100001».

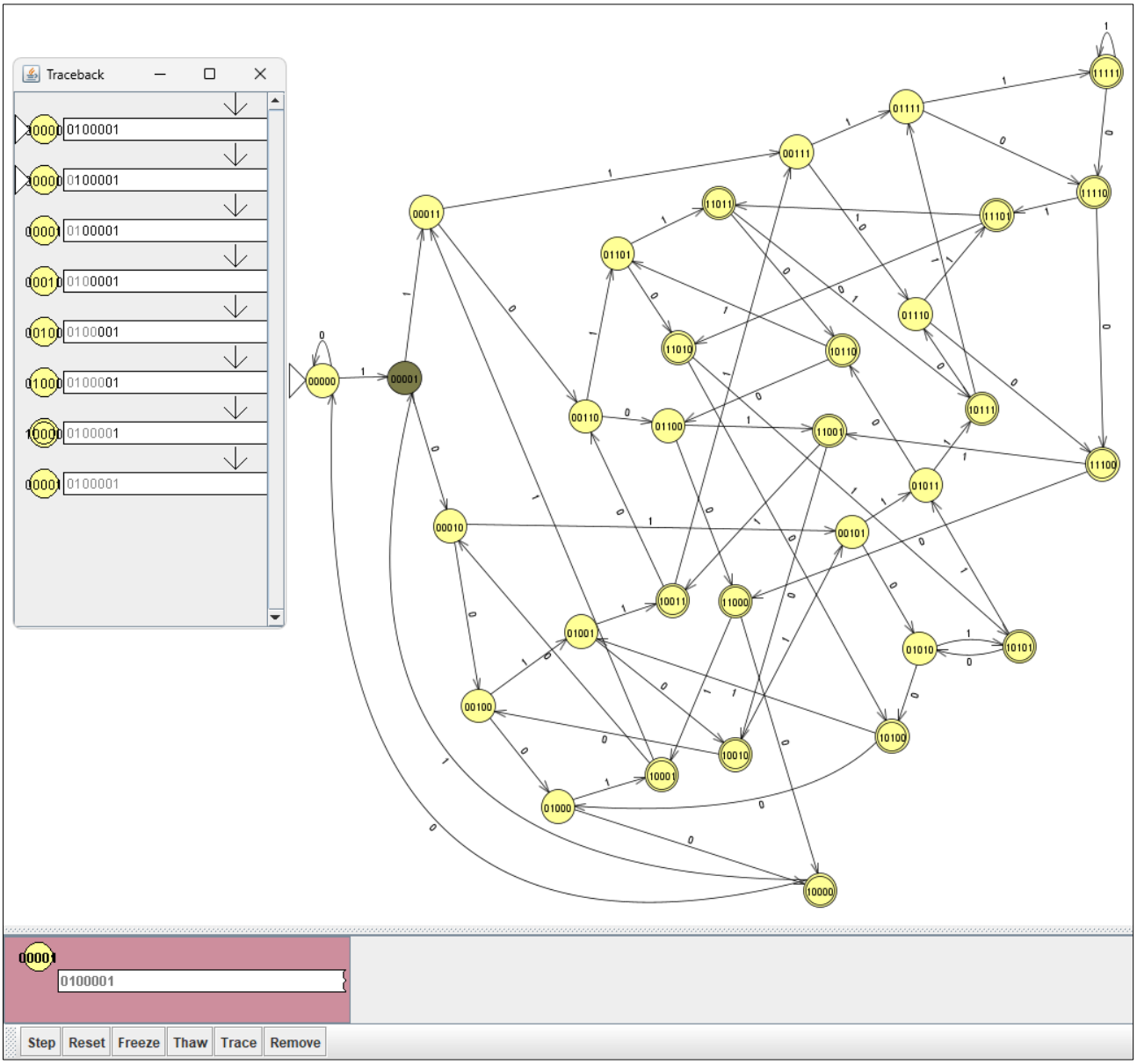
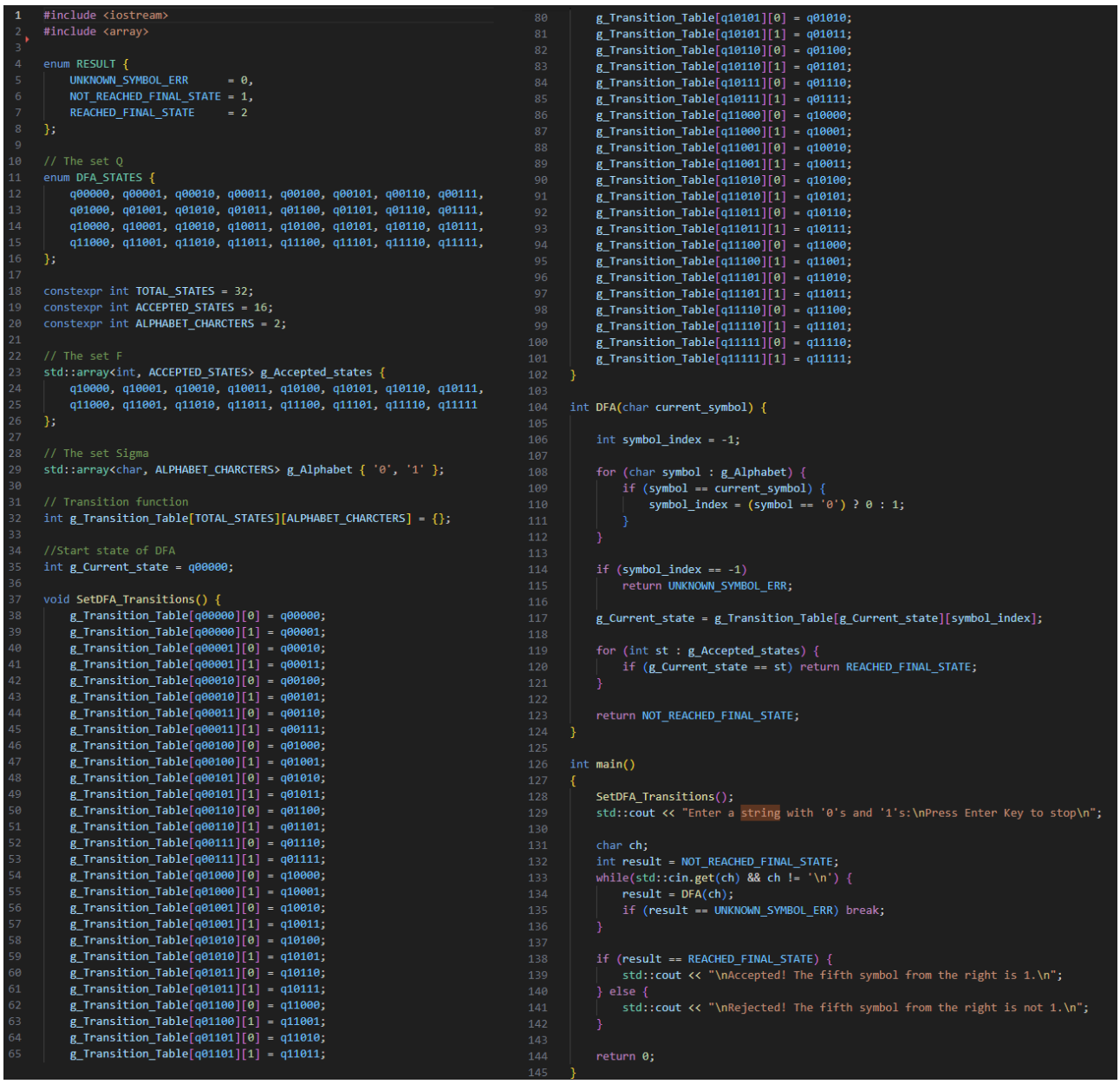



Рисунок 3 – Тест для цепочки «0100001»

Не смотря на свой размер, ДКА хорошо прошла все тесты. Метод минимизации ДКА (алгоритм заполнения таблицы) не дал результатов, поэтому данный вид является её основным.

Дальше был написан код на C++ для реализации данного ДКА, он показан на рисунке 4.



```cpp
#include <iostream>
#include <array>

enum RESULT {
    UNKNOWN_SYMBOL_ERR      = 0,
    NOT_REACHED_FINAL_STATE = 1,
    REACHED_FINAL_STATE     = 2
};

// The set Q
enum DFA_STATES {
    q00000, q00001, q00010, q00011, q00100, q00101, q00110, q00111,
    q01000, q01001, q01010, q01011, q01100, q01101, q01110, q01111,
    q10000, q10001, q10010, q10011, q10100, q10101, q10110, q10111,
    q11000, q11001, q11010, q11011, q11100, q11101, q11110, q11111,
};

constexpr int TOTAL_STATES = 32;
constexpr int ACCEPTED_STATES = 16;
constexpr int ALPHABET_CHARCTERS = 2;

// The set F
std::array<int, ACCEPTED_STATES> g_Accepted_states {
    q10000, q10001, q10010, q10011, q10100, q10101, q10110, q10111,
    q11000, q11001, q11010, q11011, q11100, q11101, q11110, q11111
};

// The set Sigma
std::array<char, ALPHABET_CHARCTERS> g_Alphabet { '0', '1' };

// Transition function
int g_Transition_Table[TOTAL_STATES][ALPHABET_CHARCTERS] = {};

//Start state of DFA
int g_Current_state = q00000;

void SetDFA_Transitions() {
    g_Transition_Table[q00000][0] = q00000;
    g_Transition_Table[q00000][1] = q00001;
    g_Transition_Table[q00001][0] = q00010;
    g_Transition_Table[q00001][1] = q00011;
    g_Transition_Table[q00010][0] = q00100;
    g_Transition_Table[q00010][1] = q00101;
    g_Transition_Table[q00011][0] = q00110;
    g_Transition_Table[q00011][1] = q00111;
    g_Transition_Table[q00100][0] = q01000;
    g_Transition_Table[q00100][1] = q01001;
    g_Transition_Table[q00101][0] = q01010;
    g_Transition_Table[q00101][1] = q01011;
    g_Transition_Table[q00110][0] = q01100;
    g_Transition_Table[q00110][1] = q01101;
    g_Transition_Table[q00111][0] = q01110;
    g_Transition_Table[q00111][1] = q01111;
    g_Transition_Table[q01000][0] = q10000;
    g_Transition_Table[q01000][1] = q10001;
    g_Transition_Table[q01001][0] = q10010;
    g_Transition_Table[q01001][1] = q10011;
    g_Transition_Table[q01010][0] = q10100;
    g_Transition_Table[q01010][1] = q10101;
    g_Transition_Table[q01011][0] = q10110;
    g_Transition_Table[q01011][1] = q10111;
    g_Transition_Table[q01100][0] = q11000;
    g_Transition_Table[q01100][1] = q11001;
    g_Transition_Table[q01101][0] = q11010;
    g_Transition_Table[q01101][1] = q11011;
    g_Transition_Table[q10101][0] = q01010;
    g_Transition_Table[q10101][1] = q01011;
    g_Transition_Table[q10110][0] = q01100;
    g_Transition_Table[q10110][1] = q01101;
    g_Transition_Table[q10111][0] = q01110;
    g_Transition_Table[q10111][1] = q01111;
    g_Transition_Table[q11000][0] = q10000;
    g_Transition_Table[q11000][1] = q10001;
    g_Transition_Table[q11001][0] = q10010;
    g_Transition_Table[q11001][1] = q10011;
    g_Transition_Table[q11010][0] = q10100;
    g_Transition_Table[q11010][1] = q10101;
    g_Transition_Table[q11011][0] = q10110;
    g_Transition_Table[q11011][1] = q10111;
    g_Transition_Table[q11100][0] = q11000;
    g_Transition_Table[q11100][1] = q11001;
    g_Transition_Table[q11101][0] = q11010;
    g_Transition_Table[q11101][1] = q11011;
    g_Transition_Table[q11110][0] = q11100;
    g_Transition_Table[q11110][1] = q11101;
    g_Transition_Table[q11111][0] = q11110;
    g_Transition_Table[q11111][1] = q11111;
}

int DFA(char current_symbol) {

    int symbol_index = -1;

    for (char symbol : g_Alphabet) {
        if (symbol == current_symbol) {
            symbol_index = (symbol == '0') ? 0 : 1;
        }
    }

    if (symbol_index == -1)
        return UNKNOWN_SYMBOL_ERR;

    g_Current_state = g_Transition_Table[g_Current_state][symbol_index];

    for (int st : g_Accepted_states) {
        if (g_Current_state == st) return REACHED_FINAL_STATE;
    }

    return NOT_REACHED_FINAL_STATE;
}

int main()
{
    SetDFA_Transitions();
    std::cout << "Enter a string with '0's and '1's:\nPress Enter Key to stop\n";

    char ch;
    int result = NOT_REACHED_FINAL_STATE;
    while(std::cin.get(ch) && ch != '\n') {
        result = DFA(ch);
        if (result == UNKNOWN_SYMBOL_ERR) break;
    }

    if (result == REACHED_FINAL_STATE) {
        std::cout << "\nAccepted! The fifth symbol from the right is 1.\n";
    } else {
        std::cout << "\nRejected! The fifth symbol from the right is not 1.\n";
    }

    return 0;
}
```

Рисунок 4 – Код для ДКА на C++

После компиляции, сборки и запуска программы были повторно проведены тесты, показанные на рисунке 5.

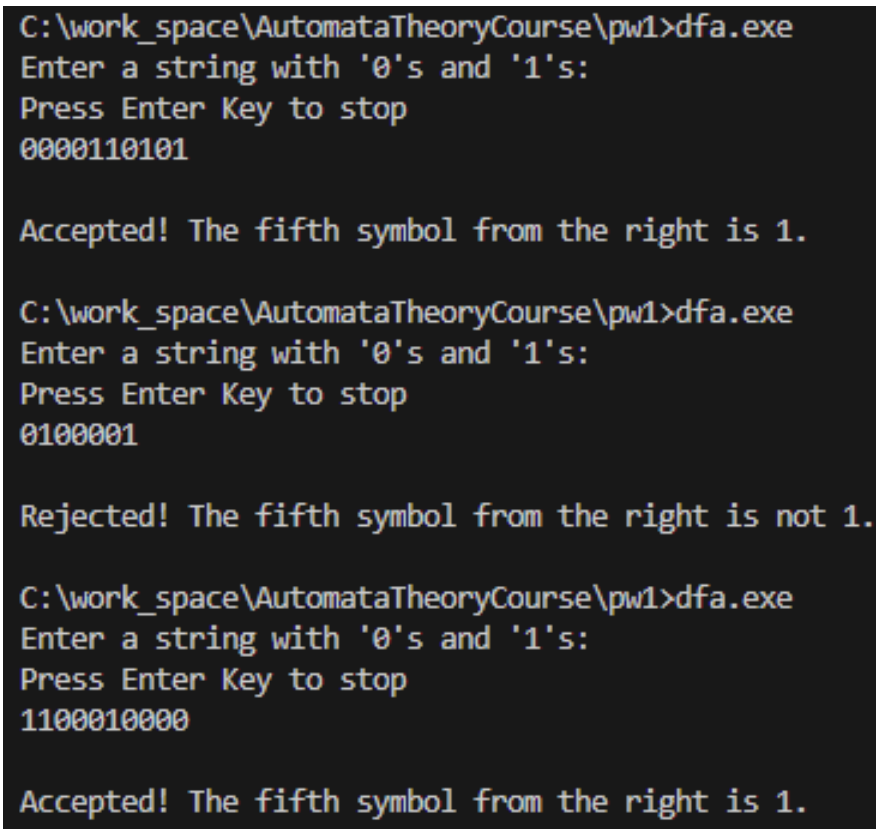



Рисунок 5 – Тесты для ДКА на C++

### 3.2 Построение НКА

Также необходимо было реализовать недетерминированный конечный автомат (НКА), допускающий все строки из алфавита $\{a^n : n \geq 1\} \cup \{b^m a^k : m, k \geq 0\}$. Необходимо было создать не более трёх состояний. Для реализации НКА был принят во внимание тот факт, что ДКА является недетерминированным, например, в том случае, если из какого-либо состояния автомата есть несколько переходов с одним и тем же входным символом, а также то, что цепочка считается допустимой в том случае, если есть хотя бы одно допускающее состояние из всех «параллельно активных». Также анализ алфавита показал, что принимается либо любое количество символа «a», либо любое количество символа «b», либо любое количество «b» и после этого любое количество «a», либо пустая цепочка (при $m=k=0$ $b^m a^k$ равно пустой строке ε). На рисунке 6 показана реализация ДКА первого задания в системе JFLAP.

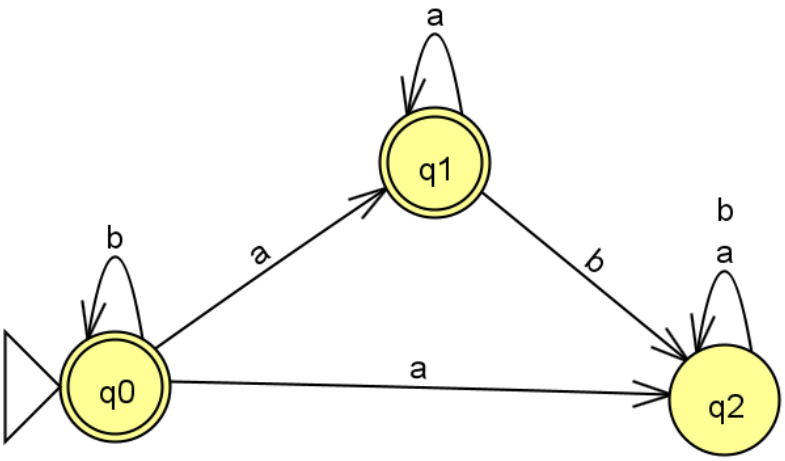



Рисунок 6 - НКА в JFLAP.

Таблица функций переходов данного НКА представлена на таблице 1.

Таблица 2 – Функция переходов для НКА

| **Состояние** | **a** | **b** |
|---|---|---|
| q0 | q1, q1 | q0 |
| q1 | q1 | q2 |
| q2 | q2 | q2 |

Дальше необходимо было провести все самые главные тесты для этой цепочки, они показаны на рисунке 7, 8, 9, 10, 11.

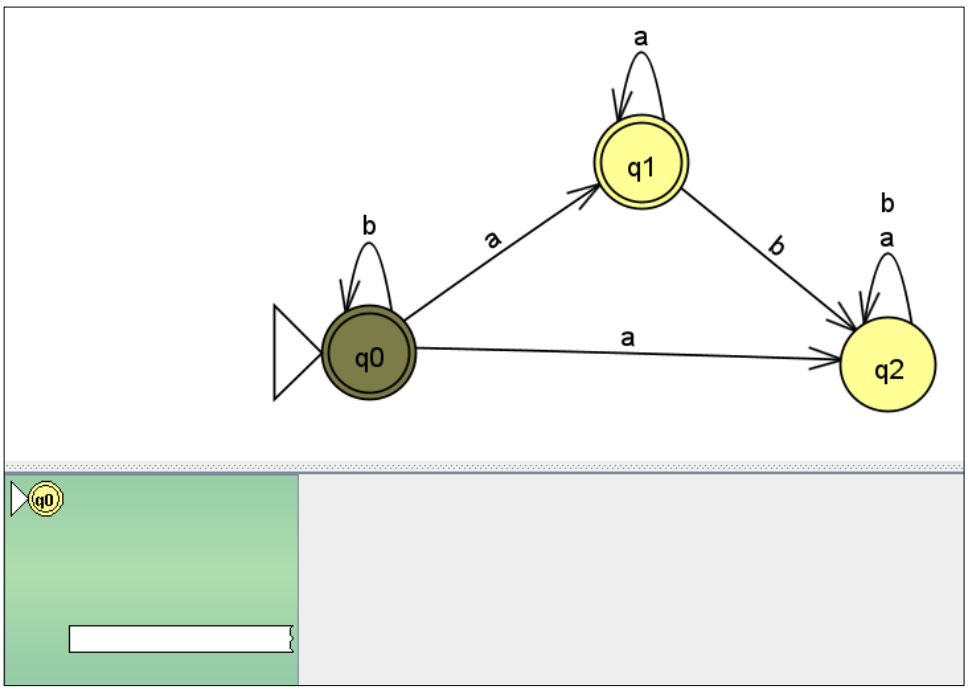



Рисунок 7 – Тест для пустой цепочки (ε).

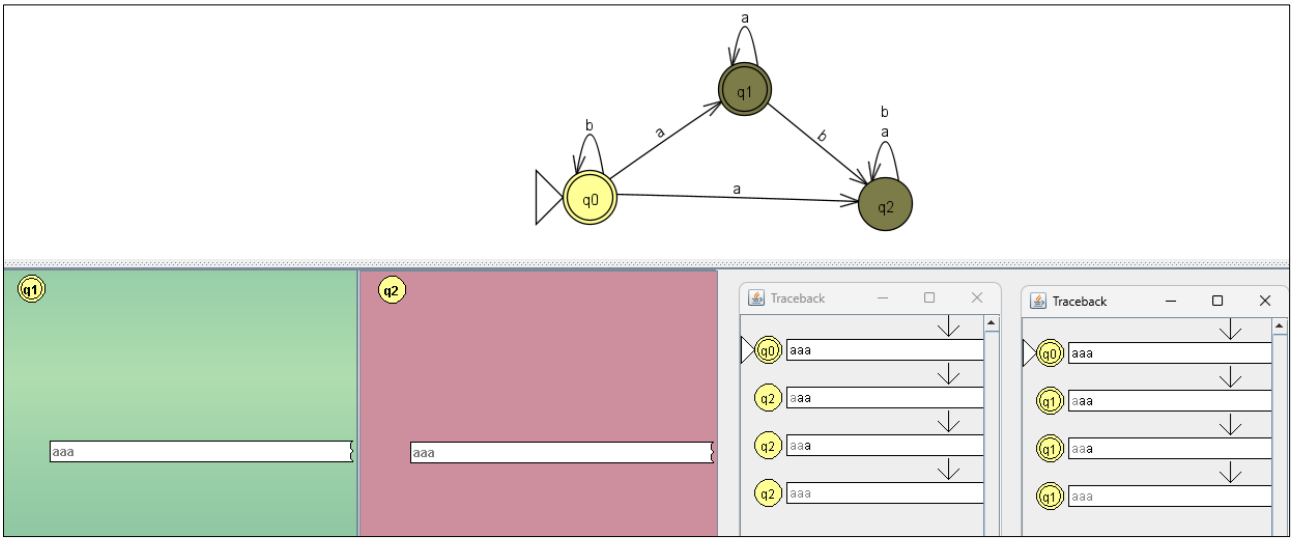



Рисунок 8 – Тест для цепочки «aaa»

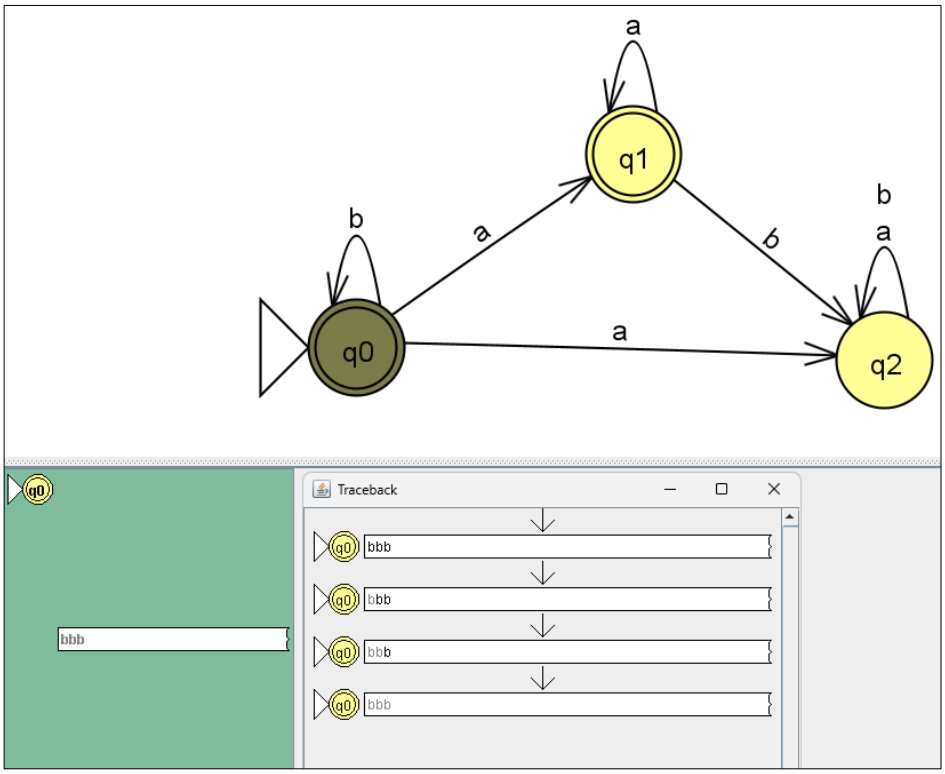



Рисунок 9 – Тест для цепочки «bbb»

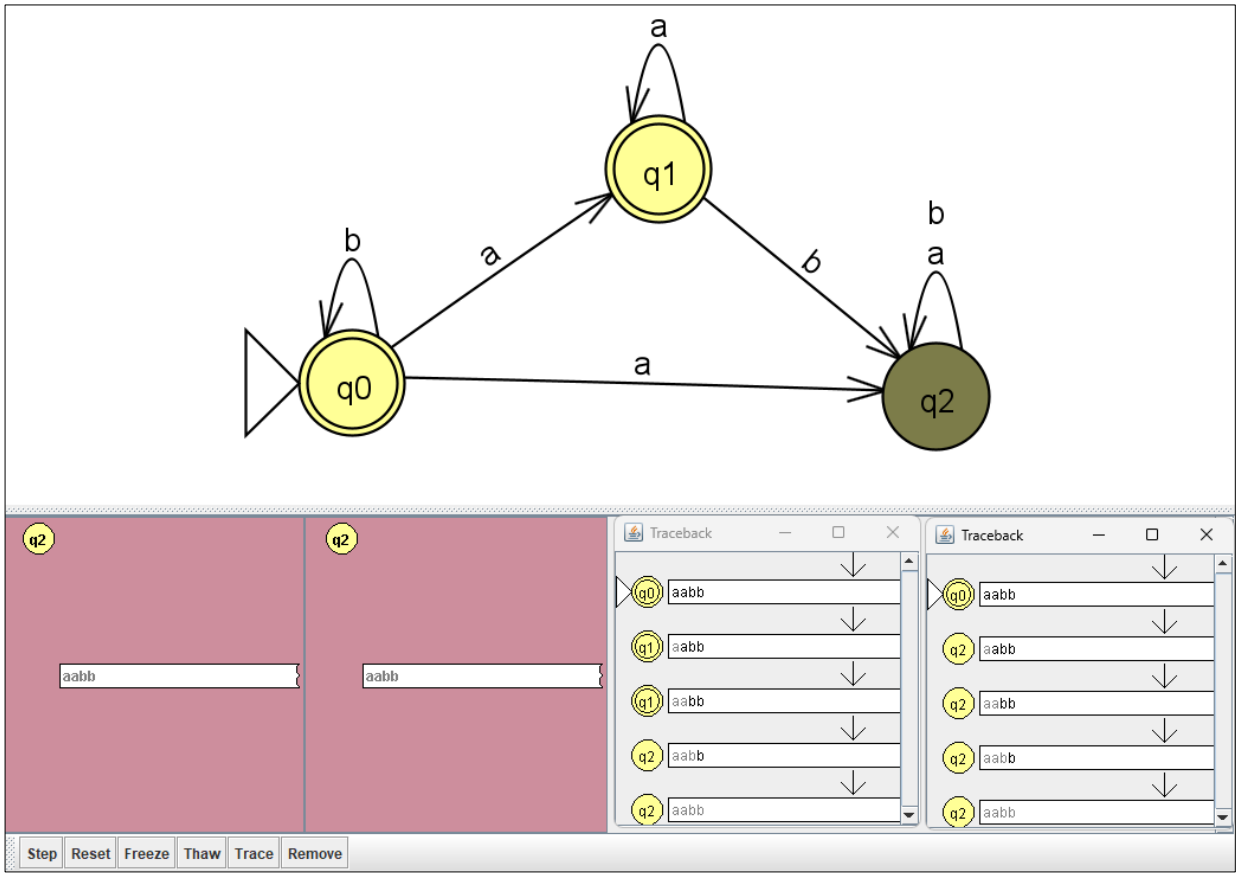



Рисунок 10 – Тест для цепочки «aabb»

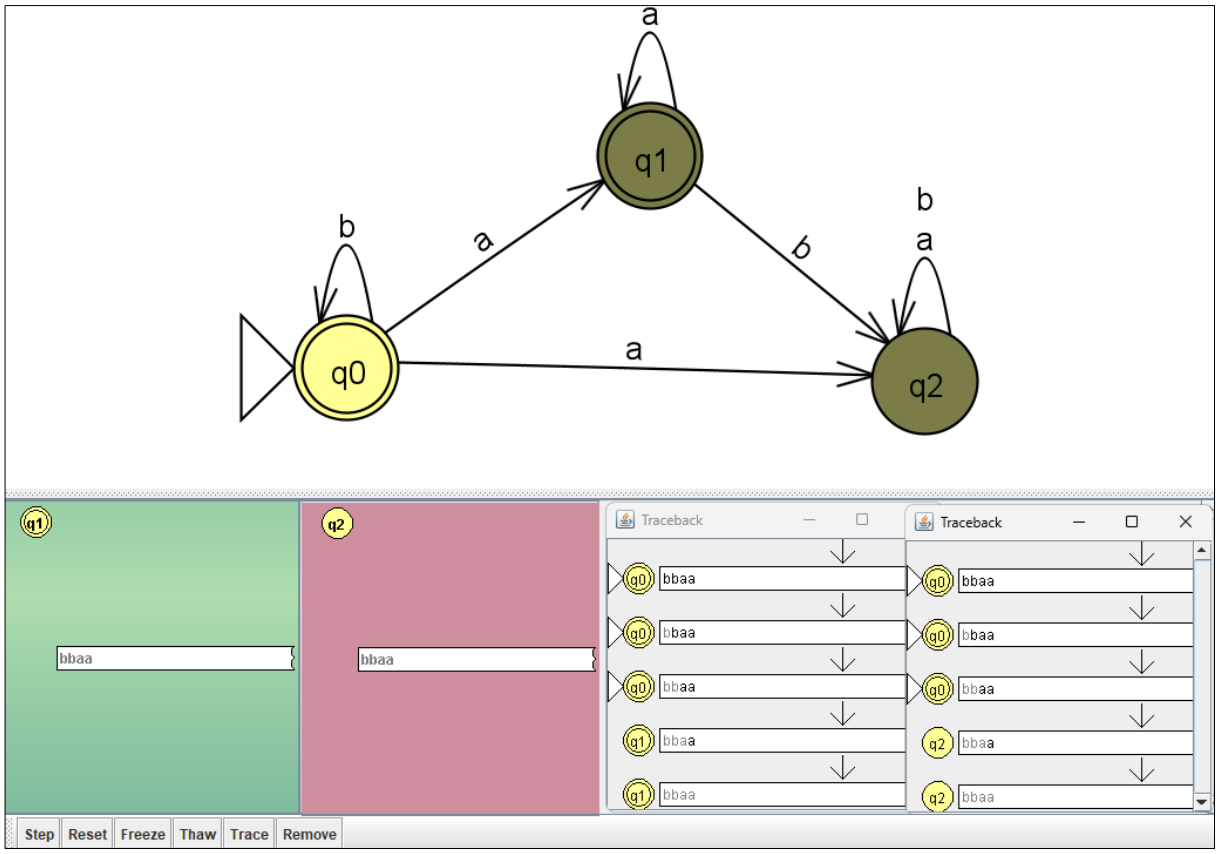
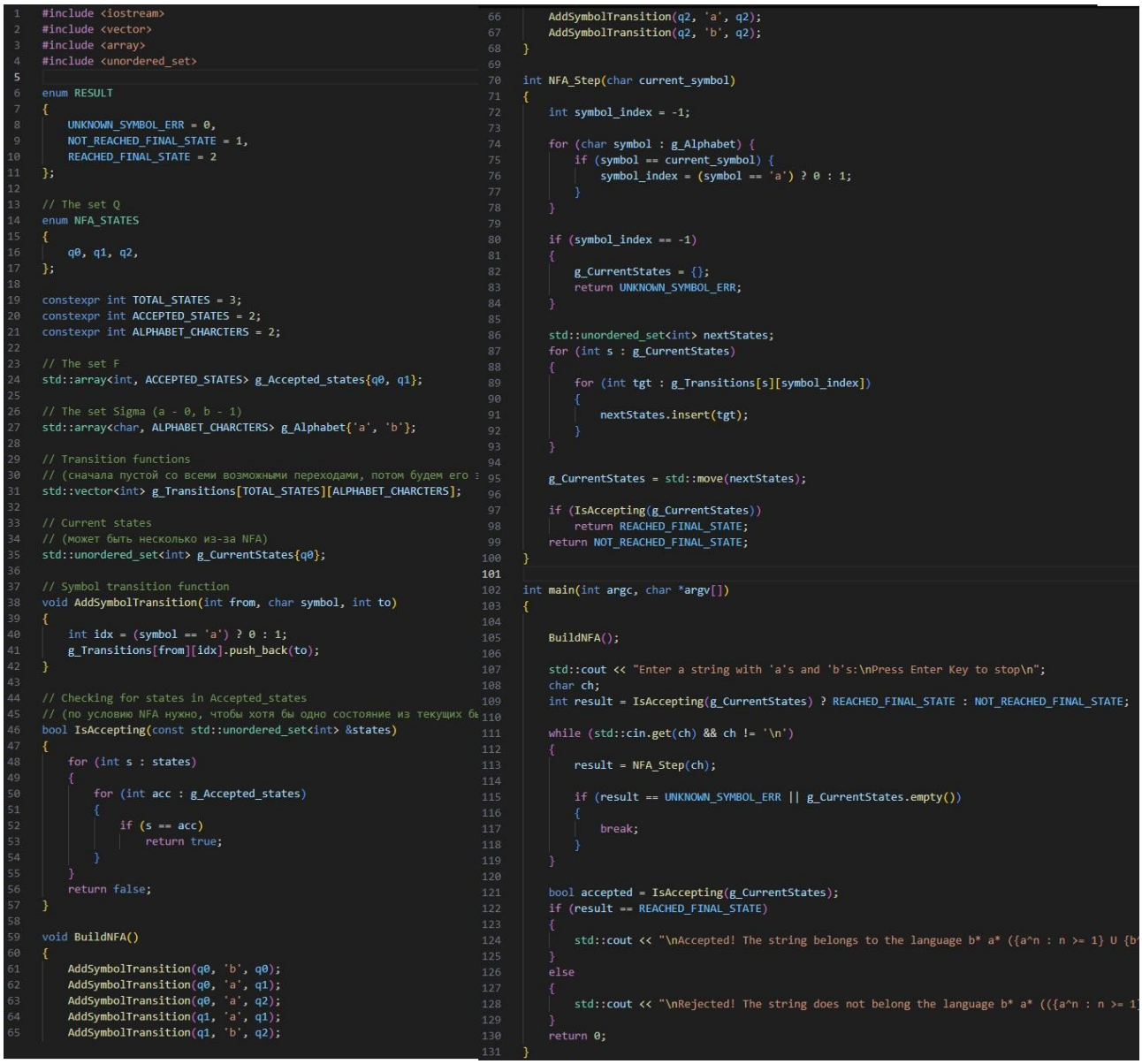



Рисунок 11 – Тест для цепочки «bbaa»

Все тесты были пройдены успешно. На рисунке 12 показан код на C++ для реализации данного НКА.



```cpp
#include <iostream>
#include <vector>
#include <array>
#include <unordered_set>

enum RESULT
{
    UNKNOWN_SYMBOL_ERR = 0,
    NOT_REACHED_FINAL_STATE = 1,
    REACHED_FINAL_STATE = 2
};

// The set Q
enum NFA_STATES
{
    q0, q1, q2,
};

constexpr int TOTAL_STATES = 3;
constexpr int ACCEPTED_STATES = 2;
constexpr int ALPHABET_CHARCTERS = 2;

// The set F
std::array<int, ACCEPTED_STATES> g_Accepted_states{q0, q1};

// The set Sigma (a - 0, b - 1)
std::array<char, ALPHABET_CHARCTERS> g_Alphabet{'a', 'b'};

// Transition functions
// (сначала пустой со всеми возможными переходами, потом будем его з
std::vector<int> g_Transitions[TOTAL_STATES][ALPHABET_CHARCTERS];

// Current states
// (может быть несколько из-за NFA)
std::unordered_set<int> g_CurrentStates{q0};

// Symbol transition function
void AddSymbolTransition(int from, char symbol, int to)
{
    int idx = (symbol == 'a') ? 0 : 1;
    g_Transitions[from][idx].push_back(to);
}

// Checking for states in Accepted_states
// (по условию NFA нужно, чтобы хотя бы одно состояние из текущих бы
bool IsAccepting(const std::unordered_set<int> &states)
{
    for (int s : states)
    {
        for (int acc : g_Accepted_states)
        {
            if (s == acc)
                return true;
        }
    }
    return false;
}

void BuildNFA()
{
    AddSymbolTransition(q0, 'b', q0);
    AddSymbolTransition(q0, 'a', q1);
    AddSymbolTransition(q0, 'a', q2);
    AddSymbolTransition(q1, 'a', q1);
    AddSymbolTransition(q1, 'b', q2);
    AddSymbolTransition(q2, 'a', q2);
    AddSymbolTransition(q2, 'b', q2);
}

int NFA_Step(char current_symbol)
{
    int symbol_index = -1;

    for (char symbol : g_Alphabet) {
        if (symbol == current_symbol) {
            symbol_index = (symbol == 'a') ? 0 : 1;
        }
    }

    if (symbol_index == -1)
    {
        g_CurrentStates = {};
        return UNKNOWN_SYMBOL_ERR;
    }

    std::unordered_set<int> nextStates;
    for (int s : g_CurrentStates)
    {
        for (int tgt : g_Transitions[s][symbol_index])
        {
            nextStates.insert(tgt);
        }
    }

    g_CurrentStates = std::move(nextStates);

    if (IsAccepting(g_CurrentStates))
        return REACHED_FINAL_STATE;
    return NOT_REACHED_FINAL_STATE;
}

int main(int argc, char *argv[])
{

    BuildNFA();

    std::cout << "Enter a string with 'a's and 'b's:\nPress Enter Key to stop\n";
    char ch;
    int result = IsAccepting(g_CurrentStates) ? REACHED_FINAL_STATE : NOT_REACHED_FINAL_STATE;

    while (std::cin.get(ch) && ch != '\n')
    {
        result = NFA_Step(ch);

        if (result == UNKNOWN_SYMBOL_ERR || g_CurrentStates.empty())
        {
            break;
        }
    }

    bool accepted = IsAccepting(g_CurrentStates);
    if (result == REACHED_FINAL_STATE)
    {
        std::cout << "\nAccepted! The string belongs to the language b* a* ({a^n : n >= 1} U {b
    }
    else
    {
        std::cout << "\nRejected! The string does not belong the language b* a* (({a^n : n >= 1
    }
    return 0;
}
```

Рисунок 12 – Код для НКА на C++

После компиляции, сборки и запуска программы были повторно проведены тесты, показанные на рисунке 13.

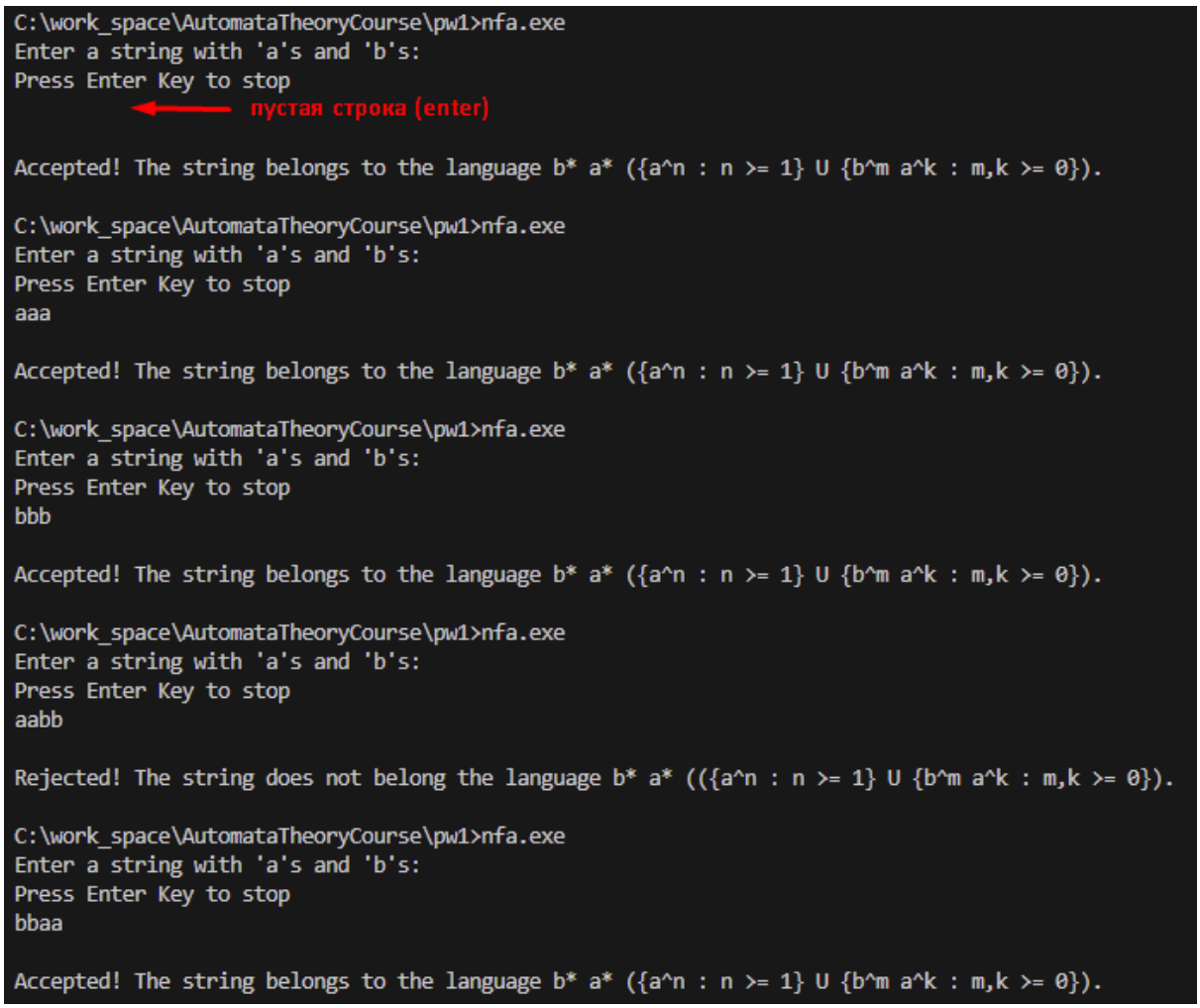



Рисунок 13 – Тесты для НКА на C++

## 4 Выводы

В ходе данной практической работы был изучен материал о детерминированных и недетерминированных конечных автоматах, были выполнены все задания, построены ДКА и НКА в системе JFLAP и реализован программный код на C++.