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

# ЛАБОРАТОРНАЯ РАБОТА №3

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

Дан недетерминированный конечный автомат (НКА) с 6 состояниями, входным алфавитом {0,1}, одним начальным и одним финальным состоянием. Требуется методом подмножеств построить эквивалентный детерминированный конечный автомат (ДКА) – распознаватель языка, задаваемого исходным НКА.

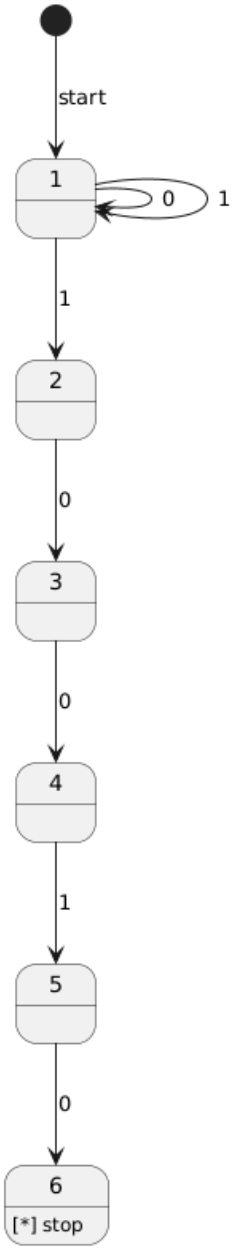
# 2. Исходный недетерминированный автомат

2.1 Формальное описание

- Входной алфавит: A = {0, 1}.
- Множество состояний: V = {1, 2, 3, 4, 5, 6}.
- Множество начальных состояний: H = {1}.
- Множество финальных состояний: F = {6}.
- Функция переходов g: A → P(V × V) задана следующими множествами пар:

$$g(0) = \{(1, 1), (2, 3), (3, 4), (5,6)\},$$
$$g(1) = \{(1, 1), (1, 2), (4, 5)\}.$$

2.2 Графическое представление:



## 3. Алгоритм построения ДКА (метод подмножеств)

1. Начальным состоянием ДКА является множество начальных состояний НКА (ε-замыкание не требуется, так как ε-переходы отсутствуют):

$M_0 = H = \{1\}$.

2. Для каждого уже построенного множества состояний M и каждого входного символа x вычисляется

$$G(M, x) = \{v \mid \exists u \in M: (u, v) \in g(x)\}.$$

Полученное множество либо уже существует (соответствует некоторому состоянию ДКА), либо добавляется как новое состояние.

3. Процесс продолжается, пока не будут обработаны все достижимые множества.

4. Состояние ДКА, соответствующее множеству M, объявляется финальным тогда и только тогда, когда $M \cap F \neq \emptyset$.

# 4. Пошаговое построение

Шаг 1.

M0 = {1}. Обозначим это множество как A. Заносим в очередь.

Шаг 2. Обработка A = {1}.

• По символу 0: из 1 → 1 → {1} = A.

• По символу 1: из 1 → 1 и 1 → 2 → {1, 2}. Множество новое, обозначим B. Помещаем в очередь.

Шаг 3. Обработка B = {1, 2}.

• По 0: из 1 → 1, из 2 → 3 → {1, 3}. Новое множество, обозначим C.

• По 1: из 1 → 1, 1 → 2, из 2 нет переходов по 1 → {1, 2} = B.

Шаг 4. Обработка C = {1, 3}.

• По 0: из 1 → 1, из 3 → 4 → {1, 4}. Новое множество, обозначим D.

• По 1: из 1 → 1, 1 → 2, из 3 нет переходов по 1 → {1, 2} = B.

Шаг 5. Обработка D = {1, 4}.

• По 0: из 1 → 1, из 4 нет переходов по 0 → {1} = A.

• По 1: из 1 → 1, 1 → 2, из 4 → 5 → {1, 2, 5}. Новое множество, обозначим E.

Шаг 6. Обработка E = {1, 2, 5}.

• По 0: из 1 → 1, из 2 → 3, из 5 → 6 → {1, 3, 6}. Новое множество, обозначим F.

• По 1: из 1 → 1, 1 → 2, из 5 нет переходов по 1 → {1, 2} = B.

Шаг 7. Обработка F = {1, 3, 6}.

• По 0: из 1 → 1, из 3 → 4, из 6 нет переходов по 0 → {1, 4} = D.

• По 1: из 1 → 1, 1 → 2, из 3,6 нет переходов по 1 → {1, 2} = B.

Новых множеств не появилось, очередь пуста. Построение завершено.

# 5. Полученные множества и их обозначения

| Состояние ДКА | Подмножество НКА |
|---|---|
| A | {1} |
| B | {1,2} |
| C | {1,3} |
| D | {1,4} |
| E | {1,2,5} |
| F | {1,3,6} |

**Начальное состояние**: A.

**Финальные состояния**: множество, содержащее финальное состояние НКА (6). Таким состоянием является F = {1,3,6}. Следовательно, F – финальное состояние ДКА.

# 6. Построение таблиц переходов и выходов ДКА

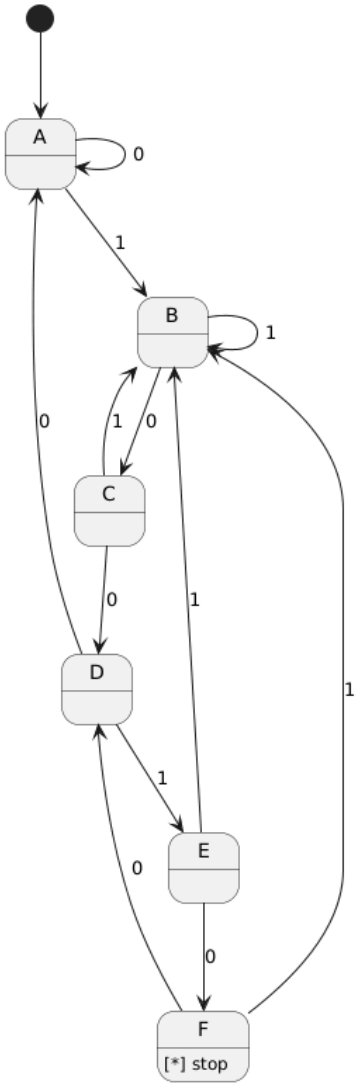
Функция переходов δ определяется по вычисленным значениям G(M, x).

Функция выхода (λ) для автомата-распознавателя: λ = 1, если состояние финальное, иначе 0.

Таблица переходов и выходов:

| Состояние | δ(0)δ(0) | δ(1)δ(1) | λ |
|---|---|---|---|
| A | A | B | 0 |
| B | C | B | 0 |
| C | D | B | 0 |
| D | A | E | 0 |
| E | F | B | 0 |
| F | D | B | 1 |

Диаграмма ДКА:



# 7. Программная реализация алгоритма

Для автоматизации построения ДКА была разработана программа на языке C++. Она реализует метод подмножеств в точном соответствии с алгоритмом, описанным в разделе 3. Программа вводит описание НКА с консоли, строит все достижимые множества состояний, присваивает им имена (A, B, C, …) и выводит таблицу переходов и выходов полученного ДКА.

main.cpp:

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

struct NFA
{
  int numStates;
  int alphabetSize;
  int initialState;
  std::set< int > finalStates;
  std::vector< std::vector< std::pair< int, int > > > transitions;
};

int main()
{
  std::cout << "Введите количество состояний НКА: ";
  int numStates;
  std::cin >> numStates;

  std::cout << "Введите размер входного алфавита: ";
  int alphaSize;
  std::cin >> alphaSize;

  std::cout << "Введите начальное состояние: ";
  int initState;
  std::cin >> initState;

  std::cout << "Введите количество финальных состояний: ";
  int numFinal;
  std::cin >> numFinal;
  std::set< int > finalStates;
  for (int i = 0; i < numFinal; ++i)
  {
    int f;
    std::cin >> f;
    finalStates.insert(f);
  }

  std::vector< std::vector< std::pair< int, int > > > transitions(alphaSize);
  for (int sym = 0; sym < alphaSize; ++sym)
  {
    std::cout << "Для символа " << sym << " введите количество переходов: ";
    int numTrans;
    std::cin >> numTrans;
```

```
  std::cout << "Введите пары (from to):\n";
  for (int j = 0; j < numTrans; ++j)
  {
    int from, to;
    std::cin >> from >> to;
    transitions[sym].push_back(std::make_pair(from, to));
  }
}

std::map< std::set< int >, int > stateToIndex;
std::vector< std::set< int > > indexToState;
std::queue< int > q;

std::set< int > startSet;
startSet.insert(initState);
stateToIndex[startSet] = 0;
indexToState.push_back(startSet);
q.push(0);

while (!q.empty())
{
  int curIdx = q.front();
  q.pop();
  std::set< int > curSet = indexToState[curIdx];

  for (int sym = 0; sym < alphaSize; ++sym)
  {
    std::set< int > nextSet;
    for (int s : curSet)
    {
      for (std::size_t k = 0; k < transitions[sym].size(); ++k)
      {
        if (transitions[sym][k].first == s)
        {
          nextSet.insert(transitions[sym][k].second);
        }
      }
    }
    if (!nextSet.empty())
    {
      std::map< std::set< int >, int >::iterator it = stateToIndex.find(nextSet);
      if (it == stateToIndex.end())
      {
        int newIdx = indexToState.size();
        stateToIndex[nextSet] = newIdx;
        indexToState.push_back(nextSet);
        q.push(newIdx);
      }
    }
  }
}

int dfaStatesCount = indexToState.size();
std::vector< std::vector< int > > dfaTrans(dfaStatesCount,
                                    std::vector< int >(alphaSize, -1));

for (int i = 0; i < dfaStatesCount; ++i)
{
  std::set< int > curSet = indexToState[i];
  for (int sym = 0; sym < alphaSize; ++sym)
  {
    std::set< int > nextSet;
```

```
    for (int s : curSet)
    {
      for (std::size_t k = 0; k < transitions[sym].size(); ++k)
      {
        if (transitions[sym][k].first == s)
        {
          nextSet.insert(transitions[sym][k].second);
        }
      }
    }
    if (!nextSet.empty())
    {
      dfaTrans[i][sym] = stateToIndex[nextSet];
    }
  }
}

std::vector< bool > dfaFinal(dfaStatesCount, false);
for (int i = 0; i < dfaStatesCount; ++i)
{
  for (int s : indexToState[i])
  {
    if (finalStates.count(s))
    {
      dfaFinal[i] = true;
      break;
    }
  }
}

std::vector< char > stateLetters;
for (int i = 0; i < dfaStatesCount; ++i)
{
  stateLetters.push_back('A' + i);
}

std::cout << "\nПостроенный ДКА:\n";
std::cout << "Количество состояний: " << dfaStatesCount << "\n";
std::cout << "Начальное состояние: " << stateLetters[0] << " (множество {";
bool first = true;
for (int s : startSet)
{
  if (!first) std::cout << ",";
  std::cout << s;
  first = false;
}
std::cout << "})\n";

std::cout << "Состояния ДКА и соответствующие множества НКА:\n";
for (int i = 0; i < dfaStatesCount; ++i)
{
  std::cout << stateLetters[i] << ": {";
  first = true;
  for (int s : indexToState[i])
  {
    if (!first) std::cout << ",";
    std::cout << s;
    first = false;
  }
  std::cout << "}";
  if (dfaFinal[i]) std::cout << " (финальное)";
  std::cout << "\n";
```

```
  }

  std::cout << "\nТаблица переходов ДКА:\n";
  std::cout << "Состояние";
  for (int sym = 0; sym < alphaSize; ++sym)
  {
    std::cout << "\t" << sym;
  }
  std::cout << "\tВыход\n";
  for (int i = 0; i < dfaStatesCount; ++i)
  {
    std::cout << stateLetters[i];
    for (int sym = 0; sym < alphaSize; ++sym)
    {
      int next = dfaTrans[i][sym];
      if (next != -1)
      {
        std::cout << "\t" << stateLetters[next];
      }
      else
      {
        std::cout << "\t-";
      }
    }
    std::cout << "\t" << (dfaFinal[i] ? 1 : 0) << "\n";
  }

  return 0;
}
```

Результаты работы программы:

```
Введите количество состояний НКА: 6
Введите размер входного алфавита: 2
Введите начальное состояние: 1
Введите количество финальных состояний: 1
6
Для символа 0 введите количество переходов: 4
Введите пары (from to):
1 1
2 3
3 4
5 6
Для символа 1 введите количество переходов: 3
Введите пары (from to):
1 1
1 2
4 5

Построенный ДКА:
Количество состояний: 6
Начальное состояние: A (множество {1})
Состояния ДКА и соответствующие множества НКА:
A: {1}
B: {1,2}
C: {1,3}
D: {1,4}
E: {1,2,5}
F: {1,3,6} (финальное)

Таблица переходов ДКА:
Состояние   0   1   Выход
A   A   B   0
B   C   B   0
C   D   B   0
D   A   E   0
E   F   B   0
F   D   B   1
```

# 8. Проверка эквивалентности

Для нескольких произвольных слов сравним реакцию исходного НКА и построенного ДКА.

| Слово | ДКА (конечное состояние) | Финальное? | НКА (существует путь в 6?) | Результат |
|---|---|---|---|---|
| ε | A | нет | нет | ✓ |
| 0 | A | нет | нет | ✓ |
| 1 | B | нет | нет | ✓ |
| 10 | C | нет | нет | ✓ |
| 100 | D | нет | нет | ✓ |
| 1001 | E | нет | нет | ✓ |
| 10010 | F | **да** | **да** (1→2→3→4→5→6) | ✓ |
| 100101 | B | нет | нет | ✓ |

Все проверки подтверждают эквивалентность.

# 9. Вывод

В ходе выполнения задания успешно построен детерминированный конечный автомат, эквивалентный заданному недетерминированному автомату с 6 состояниями. Применён метод подмножеств, подробно описан каждый шаг преобразования. Полученный ДКА содержит 6 состояний, одно из которых финальное. Корректность построения проверена на контрольных словах.