ГУАП

КАФЕДРА № 43

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Дожность		Рогачев С.А
старший преподаватель	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №5

Синтез конечных автоматов

по дисциплине: Теория вычислительных процессов

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4236		Л. Мвале
	подпись, дата	инициалы, фамилия

Санкт-Петербург
2025

1. Цель работы

Освоить алгоритмы синтеза конечных автоматов по регулярным выражениям и разработать программу для автоматического построения конечного автомата Мили для выражения (x|d)<n|m>k(<a>x<b>) с произвольными параметрами n, m, k.

2. Постановка задачи

Для регулярного выражения (x|d)<n|m>k(<a>x<b>) требуется:

– Разработать алгоритм синтеза конечного автомата Мили

– Реализовать программу, принимающую параметры n, m, k

– Автоматически генерировать автоматную матрицу

– Построить конечный автомат тремя способами

– Реализовать проверку входных слов на соответствие регулярному выражению

Формальное описание языка:

$$L = \{ w \mid w = G_1 G_2 \ldots G_k \;\; a\ x\ b \}$$

где каждая $G_i \in \{x^n, d^m\}$

3. Алгоритм синтеза КНА

Шаг 1: Анализ регулярного выражения

- Выделить компоненты: группы (x|d), повторения <n|m>, количество групп k, суффикс (<a>x<b>)

- Определить структуру состояний

Шаг 2: Построение базовых состояний

```
Q = {Qstart, ReadingX_0_1...ReadingX_k-1_n, ReadingD_0_1...ReadingD_k-1_m, SuffixX, SuffixB, Qfinal, Qtrap}
```

Шаг 3: Определение переходов с учетом счетчиков

- Для каждой группы генерируются состояния чтения символов
- Учитывается количество завершенных групп
- После k групп осуществляется переход к обработке суффикса

Шаг 4: Формирование функций переходов и выходов

- $\delta: Q \times \Sigma \rightarrow Q$
- $\lambda: Q \times \Sigma \rightarrow \Delta$

Шаг 5: Генерация автоматной матрицы

4. Процесс синтеза КНА

Этап 1: Инициализация параметров

- Чтение параметров n, m, k из входного файла
- Инициализация структур данных для хранения состояний и переходов

Этап 2: Генерация состояний для групп

- Для каждой группы от 0 до k-1:

  - Генерация состояний ReadingX_group_i для i от 1 до n

- Генерация состояний ReadingD_group_j для j от 1 до m

Этап 3: Добавление служебных состояний

- Qstart - начальное состояние
- SuffixX, SuffixB - состояния обработки суффикса
- Qfinal - конечное состояние
- Qtrap - состояние-ловушка для недопустимых переходов

Этап 4: Формирование переходов между состояниями

- Из Qstart: переход к первому символу первой группы
- Внутри групп: переходы между состояниями чтения символов
- Между группами: после завершения группы переход к началу следующей
- После последней группы: переход к обработке суффикса
- Обработка суффикса: строгая последовательность a→x→b

Этап 5: Назначение выходных сигналов

- Выход 1 только при успешном завершении слова (переход в Qfinal)
- На всех остальных переходах выход 0

5. Текст программы на языке высокого уровня

```
import os

class MealyMachineGenerator:

    def __init__(self, n, m, k):

        self.n = n
```

```
        self.m = m
        self.k = k
            self.states = ["Qstart", "SuffixX", "SuffixB", "Qfinal", "Qtrap"]  # УДАЛЕН SuffixA
        self.transitions = {}

    def generate_automaton(self):
        self._generate_all_states()
        self._generate_all_transitions()
        return self.transitions

    def _generate_all_states(self):
        # Генерация состояний для всех k групп
        for group in range(self.k):
            # Состояния X-группы: ReadingX_группа_i (i от 1 до n)
            for i in range(1, self.n + 1):
                state_name = f"ReadingX_{group}_{i}"
                self.states.append(state_name)

            # Состояния D-группы: ReadingD_группа_j (j от 1 до m)
            for j in range(1, self.m + 1):
                state_name = f"ReadingD_{group}_{j}"
                self.states.append(state_name)

    def _generate_all_transitions(self):
        # Инициализация всех состояний переходами в ловушку
```

```
        for state in self.states:
            self.transitions[state] = {}
            for symbol in ['x', 'd', 'a', 'b']:
                self.transitions[state][symbol] = ("Qtrap", "0")

        # Переходы из начального состояния
        self.transitions["Qstart"] = {
            'x': ("ReadingX_0_1", "0"),
            'd': ("ReadingD_0_1", "0"),
            'a': ("Qtrap", "0"),
            'b': ("Qtrap", "0")
        }

        # Генерация переходов для всех групп
        for group in range(self.k):
            self._add_x_group_transitions(group)
            self._add_d_group_transitions(group)

        # ИСПРАВЛЕННЫЕ переходы суффикса - ФИКС!
        # После завершения ПОСЛЕДНЕЙ группы, мы читаем 'a' и переходим прямо в SuffixX
        # Затем SuffixX --x--> SuffixB (ожидаем 'x')
        # Затем SuffixB --b--> Qfinal (ожидаем 'b', выход 1)

        self.transitions["SuffixX"] = {
            'x': ("SuffixB", "0"),  # После 'a', ожидаем 'x'
```

```
            'a': ("Qtrap", "0"),
            'd': ("Qtrap", "0"),
            'b': ("Qtrap", "0")
        }
        self.transitions["SuffixB"] = {
            'b': ("Qfinal", "1"),   # После 'x', ожидаем 'b' для завершения
            'x': ("Qtrap", "0"),
            'a': ("Qtrap", "0"),
            'd': ("Qtrap", "0")
        }

    def _add_x_group_transitions(self, group):
        for i in range(1, self.n + 1):
            state_name = f"ReadingX_{group}_{i}"

            if i < self.n:
                # Продолжаем в той же x-группе - читаем следующий 'x'
                self.transitions[state_name]['x'] = (f"ReadingX_{group}_{i+1}", "0")
            else:
                # Завершили x-группу (получили n 'x')
                if group < self.k - 1:
                    # Нужно больше групп - можем начать либо x-группу, либо d-группу
                    self.transitions[state_name]['x'] = (f"ReadingX_{group+1}_1", "0")
                    self.transitions[state_name]['d'] = (f"ReadingD_{group+1}_1", "0")
```

```
            else:
                # Завершили последнюю группу - теперь ожидаем 'a' для начала суффикса
                # После чтения 'a', переходим прямо в SuffixX (ожидая 'x')
                self.transitions[state_name]['a'] = ("SuffixX", "0")

    def _add_d_group_transitions(self, group):
        for j in range(1, self.m + 1):
            state_name = f"ReadingD_{group}_{j}"

            if j < self.m:
                # Продолжаем в той же d-группе - читаем следующий 'd'
                self.transitions[state_name]['d'] = (f"ReadingD_{group}_{j+1}", "0")
            else:
                # Завершили d-группу (получили m 'd')
                if group < self.k - 1:
                    # Нужно больше групп - можем начать либо x-группу, либо d-группу
                    self.transitions[state_name]['x'] = (f"ReadingX_{group+1}_1", "0")
                    self.transitions[state_name]['d'] = (f"ReadingD_{group+1}_1", "0")
                else:
                    # Завершили последнюю группу - теперь ожидаем 'a' для начала суффикса
                    # После чтения 'a', переходим прямо в SuffixX (ожидая 'x')
                    self.transitions[state_name]['a'] = ("SuffixX", "0")
```

```
def validate_word(word, automaton_matrix):
    """Проверяет, принимается ли слово автоматом"""
    current_state = "Qstart"
    output_sequence = []
    path = [current_state]
    transition_log = []

    print(f"\nПроверка слова: '{word}'")
    print(f"Ожидаемый паттерн: k групп из (n 'x' ИЛИ m 'd') затем 'a x b'")

    for i, char in enumerate(word):
        if char not in ['x', 'd', 'a', 'b']:
            print(f"Ошибка: Неверный символ '{char}' на позиции {i}")
            return False, output_sequence, path, transition_log

        if current_state not in automaton_matrix:
            print(f"Ошибка: Неизвестное состояние '{current_state}'")
            return False, output_sequence, path, transition_log

        if char not in automaton_matrix[current_state]:
                print(f"Ошибка: Нет перехода для '{char}' из состояния '{current_state}'")
                print(f"Доступные переходы из {current_state}: {list(automaton_matrix[current_state].keys())}")
            return False, output_sequence, path, transition_log
```

```
        next_state, output = automaton_matrix[current_state][char]
        transition_info = f"{current_state} --{char}--> {next_state}"
        print(f"Шаг {i+1}: {transition_info} (выход: {output})")

        transition_log.append(transition_info)
        output_sequence.append(output)
        current_state = next_state
        path.append(current_state)

    # Проверяем, закончили ли мы в конечном состоянии
    is_accepted = (current_state == "Qfinal")
    print(f"Финальное состояние: {current_state}, Принято: {is_accepted}")

    return is_accepted, output_sequence, path, transition_log

def main():
    # Получаем текущую директорию
    current_dir = os.path.dirname(os.path.abspath(__file__))
    input_file_path = os.path.join(current_dir, 'input.txt')

    print(f"Поиск входного файла по пути: {input_file_path}")

    # Проверяем существование файла
    if not os.path.exists(input_file_path):
        print("Входной файл не найден. Создание файла input.txt по умолчанию...")
```

```
        # Создаем файл по умолчанию
        with open(input_file_path, 'w', encoding='utf-8') as f:
            f.write("n=2 m=3 k=2\n")
            f.write("xxdddaxb\n")

            print("Создан файл input.txt по умолчанию с n=2, m=3, k=2 и тестовым словом 'xxdddaxb'")

    # Читаем параметры из файла
    test_word = None
    try:
        with open(input_file_path, 'r', encoding='utf-8') as f:
            lines = [line.strip() for line in f.readlines() if line.strip()]

            if not lines:
                print("Входной файл пуст. Используются значения по умолчанию.")
                n, m, k = 2, 3, 1
            else:
                # Первая строка должна содержать параметры
                params_line = lines[0]
                params = dict(item.split('=') for item in params_line.split())
                n = int(params['n'])
                m = int(params['m'])
                k = int(params['k'])
```

```
            # Вторая строка (если существует) должна содержать тестовое слово
            if len(lines) > 1:
                test_word = lines[1].replace(" ", "")  # Удаляем пробелы из тестового слова

        print(f"Успешно прочитаны параметры: n={n}, m={m}, k={k}")
        print(f"Паттерн: {k} групп из ({n} 'x' ИЛИ {m} 'd') затем 'a x b'")
        if test_word:
            print(f"Тестовое слово: '{test_word}'")
            print(f"Ожидаемая длина: {k} групп + 'a x b' = общая длина: {k * max(n, m) + 3}")
            print(f"Фактическая длина: {len(test_word)}")
        else:
            print("Тестовое слово не предоставлено в input.txt")

    except Exception as e:
        print(f"Ошибка чтения входного файла: {e}")
        print("Используются значения по умолчанию: n=2, m=3, k=1")
        n, m, k = 2, 3, 1

    # Генерируем автомат
    generator = MealyMachineGenerator(n, m, k)
    automaton_matrix = generator.generate_automaton()

    # Записываем выходные данные в файл
```

```
output_file_path = os.path.join(current_dir, 'output.txt')
with open(output_file_path, 'w', encoding='utf-8') as f:
    f.write(f"Автоматная матрица для n={n}, m={m}, k={k}\n")
    f.write(f"Паттерн: {k} групп по ({n} 'x' ИЛИ {m} 'd') затем 'a x b'\n")
    f.write("Состояние | x | d | a | b\n")
    f.write("-" * 80 + "\n")

    # Получаем все состояния в порядке для согласованного вывода
    all_states = ["Qstart"]

    # Добавляем состояния групп в порядке
    for group in range(k):
        for i in range(1, n + 1):
            state_name = f"ReadingX_{group}_{i}"
            if state_name in generator.states:
                all_states.append(state_name)
        for j in range(1, m + 1):
            state_name = f"ReadingD_{group}_{j}"
            if state_name in generator.states:
                all_states.append(state_name)

    # Добавляем состояния суффикса
    all_states.extend(["SuffixX", "SuffixB", "Qfinal", "Qtrap"])

    for state in all_states:
```

```
            if state in automaton_matrix:
                row = [state]
                for symbol in ['x', 'd', 'a', 'b']:
                    if symbol in automaton_matrix[state]:
                        next_state, output = automaton_matrix[state][symbol]
                        row.append(f"{next_state}/{output}")
                    else:
                        row.append("-/-")
                f.write(" | ".join(row) + "\n")

        # Тестируем слово, если предоставлено
        if test_word:
            f.write(f"\nПроверка слова: '{test_word}'\n")
                        is_valid, output_sequence, path, transition_log = validate_word(test_word, automaton_matrix)

            f.write(f"Путь состояний: {' -> '.join(path)}\n")
            f.write("Переходы:\n")
            for i, transition in enumerate(transition_log, 1):
                f.write(f"  Шаг {i}: {transition}\n")
                                f.write(f"Выходная последовательность: {''.join(output_sequence)}\n")
                    f.write(f"Результат: Слово {'ПРИНЯТО' if is_valid else 'ОТВЕРГНУТО'}\n")

            # Показываем примеры допустимых слов
```

```
                f.write(f"\nПримеры допустимых слов для n={n}, m={m}, k={k}:\n")
        examples = []
        if k == 1:
            examples.append(f"{'x'*n}axb")
            examples.append(f"{'d'*m}axb")
        else:
            # Генерируем некоторые примеры комбинаций
            examples.append(f"{'x'*n}{'d'*m}axb")
            examples.append(f"{'d'*m}{'x'*n}axb")
            examples.append(f"{'x'*n}{'x'*n}axb")
            examples.append(f"{'d'*m}{'d'*m}axb")

        for example in examples:
            f.write(f"  - {example}\n")

    print(f"\nВыходные данные записаны в: {output_file_path}")

if __name__ == "__main__":
    main()
```

6. Текстовый файл выходных данных программы

input.txt:

n=2 m=3 k=3

Xxxxdddaxb

output.txt:

Автоматная матрица для n=2, m=3, k=3

Паттерн: 3 групп по (2 'x' ИЛИ 3 'd') затем 'a x b'

Состояние | x | d | a | b

Qstart | ReadingX_0_1/0 | ReadingD_0_1/0 | Qtrap/0 | Qtrap/0

ReadingX_0_1 | ReadingX_0_2/0 | Qtrap/0 | Qtrap/0 | Qtrap/0

ReadingX_0_2 | ReadingX_1_1/0 | ReadingD_1_1/0 | Qtrap/0 | Qtrap/0

ReadingD_0_1 | Qtrap/0 | ReadingD_0_2/0 | Qtrap/0 | Qtrap/0

ReadingD_0_2 | Qtrap/0 | ReadingD_0_3/0 | Qtrap/0 | Qtrap/0

ReadingD_0_3 | ReadingX_1_1/0 | ReadingD_1_1/0 | Qtrap/0 | Qtrap/0

ReadingX_1_1 | ReadingX_1_2/0 | Qtrap/0 | Qtrap/0 | Qtrap/0

ReadingX_1_2 | ReadingX_2_1/0 | ReadingD_2_1/0 | Qtrap/0 | Qtrap/0

ReadingD_1_1 | Qtrap/0 | ReadingD_1_2/0 | Qtrap/0 | Qtrap/0

ReadingD_1_2 | Qtrap/0 | ReadingD_1_3/0 | Qtrap/0 | Qtrap/0

ReadingD_1_3 | ReadingX_2_1/0 | ReadingD_2_1/0 | Qtrap/0 | Qtrap/0

ReadingX_2_1 | ReadingX_2_2/0 | Qtrap/0 | Qtrap/0 | Qtrap/0

ReadingX_2_2 | Qtrap/0 | Qtrap/0 | SuffixX/0 | Qtrap/0

ReadingD_2_1 | Qtrap/0 | ReadingD_2_2/0 | Qtrap/0 | Qtrap/0

ReadingD_2_2 | Qtrap/0 | ReadingD_2_3/0 | Qtrap/0 | Qtrap/0

ReadingD_2_3 | Qtrap/0 | Qtrap/0 | SuffixX/0 | Qtrap/0

SuffixX | SuffixB/0 | Qtrap/0 | Qtrap/0 | Qtrap/0

SuffixB | Qtrap/0 | Qtrap/0 | Qtrap/0 | Qfinal/1

Qfinal | Qtrap/0 | Qtrap/0 | Qtrap/0 | Qtrap/0

Qtrap | Qtrap/0 | Qtrap/0 | Qtrap/0 | Qtrap/0

Проверка слова: 'xxxxdddaxb'

Путь состояний: Qstart -> ReadingX_0_1 -> ReadingX_0_2 -> ReadingX_1_1 -> ReadingX_1_2 -> ReadingD_2_1 -> ReadingD_2_2 -> ReadingD_2_3 -> SuffixX -> SuffixB -> Qfinal

Переходы:

Шаг 1: Qstart --x--> ReadingX_0_1

Шаг 2: ReadingX_0_1 --x--> ReadingX_0_2

Шаг 3: ReadingX_0_2 --x--> ReadingX_1_1

Шаг 4: ReadingX_1_1 --x--> ReadingX_1_2

Шаг 5: ReadingX_1_2 --d--> ReadingD_2_1

Шаг 6: ReadingD_2_1 --d--> ReadingD_2_2

Шаг 7: ReadingD_2_2 --d--> ReadingD_2_3

Шаг 8: ReadingD_2_3 --a--> SuffixX

Шаг 9: SuffixX --x--> SuffixB

Шаг 10: SuffixB --b--> Qfinal

Выходная последовательность: 0000000001

Результат: Слово ПРИНЯТО

```
Поиск входного файла по пути: e:\Theory-of-computational-processes\lab5\input.txt
Успешно прочитаны параметры: n=2, m=3, k=3
Паттерн: 3 групп из (2 'x' ИЛИ 3 'd') затем 'a x b'
Тестовое слово: 'xxxxdddaxb'
Ожидаемая длина: 3 групп + 'a x b' = общая длина: 12
Фактическая длина: 10

Проверка слова: 'xxxxdddaxb'
Ожидаемый паттерн: k групп из (n 'x' ИЛИ m 'd') затем 'a x b'
Шаг 1: Qstart --x--> ReadingX_0_1 (выход: 0)
Шаг 2: ReadingX_0_1 --x--> ReadingX_0_2 (выход: 0)
Шаг 3: ReadingX_0_2 --x--> ReadingX_1_1 (выход: 0)
Шаг 4: ReadingX_1_1 --x--> ReadingX_1_2 (выход: 0)
Шаг 5: ReadingX_1_2 --d--> ReadingD_2_1 (выход: 0)
Шаг 6: ReadingD_2_1 --d--> ReadingD_2_2 (выход: 0)
Шаг 7: ReadingD_2_2 --d--> ReadingD_2_3 (выход: 0)
Шаг 8: ReadingD_2_3 --a--> SuffixX (выход: 0)
Шаг 9: SuffixX --x--> SuffixB (выход: 0)
Шаг 10: SuffixB --b--> Qfinal (выход: 1)
Финальное состояние: Qfinal, Принято: True
```

7. Конечный автомат заданный тремя способами

Исправленный общий автомат Мили для (x|d)<n|m>k(<a>x<b>)

Формальное определение

```
M = (Q, Σ, Δ, δ, λ, q₀)

Где:

Q = {Qstart, ReadingX, ReadingD, SuffixA, SuffixX, SuffixB,
Qfinal, Qtrap} × Counter

Σ = {x, d, a, b}

Δ = {0, 1}

q₀ = (Qstart, 0, 0, 0)  # (состояние, количество x, количество
d, завершённые группы)
```

7.1 Метод 1: Матрица переходов (δ)

Текущее состояние	x_count	d_count	groups	Вход x	Вход d	Вход a	Вход b
Qstart	0	0	0	ReadingX(1,0,0)	ReadingD(0,1,0)	Qtrap	Qtrap
ReadingX	i < n	0	g	ReadingX(i+1,0,g)	Qtrap	Qtrap	Qtrap
ReadingX	i = n	0	g < k-1	ReadingX(1,0,g+1)	ReadingD(0,1,g+1)	Qtrap	Qtrap
ReadingX	i = n	0	g = k-1	Qtrap	Qtrap	SuffixA	Qtrap
ReadingD	0	j < m	g	Qtrap	ReadingD(0,j+1,g)	Qtrap	Qtrap

ReadingD	0	j = m	g < k-1	ReadingX(1,0,g+1)	ReadingD(0,1,g+1)	Qtrap	Qtrap
ReadingD	0	j = m	g = k-1	Qtrap	Qtrap	SuffixA	Qtrap
SuffixA	-	-	-	Qtrap	Qtrap	SuffixX	Qtrap
SuffixX	-	-	-	Qtrap	Qtrap	Qtrap	SuffixB
SuffixB	-	-	-	Qtrap	Qtrap	Qtrap	Qfinal
Qfinal	-	-	-	-	-	-	-
Qtrap	-	-	-	Qtrap	Qtrap	Qtrap	Qtrap

Исправленная матрица выходов (λ)

Текущее состояние	Вход x	Вход d	Вход a	Вход b
Qstart	0	0	0	0
ReadingX	0	0	0	0
ReadingD	0	0	0	0
SuffixA	0	0	0	0
SuffixX	0	0	0	0
SuffixB	0	0	0	1
Qfinal	-	-	-	-
Qtrap	0	0	0	0

7.2 Метод 2 Автоматная матрица (таблица)

M = [m_ij]

где:

X_ij = δ(q_i, x_j) - следующее состояние

U_ij = λ(q_i, x_j) - выходной сигнал

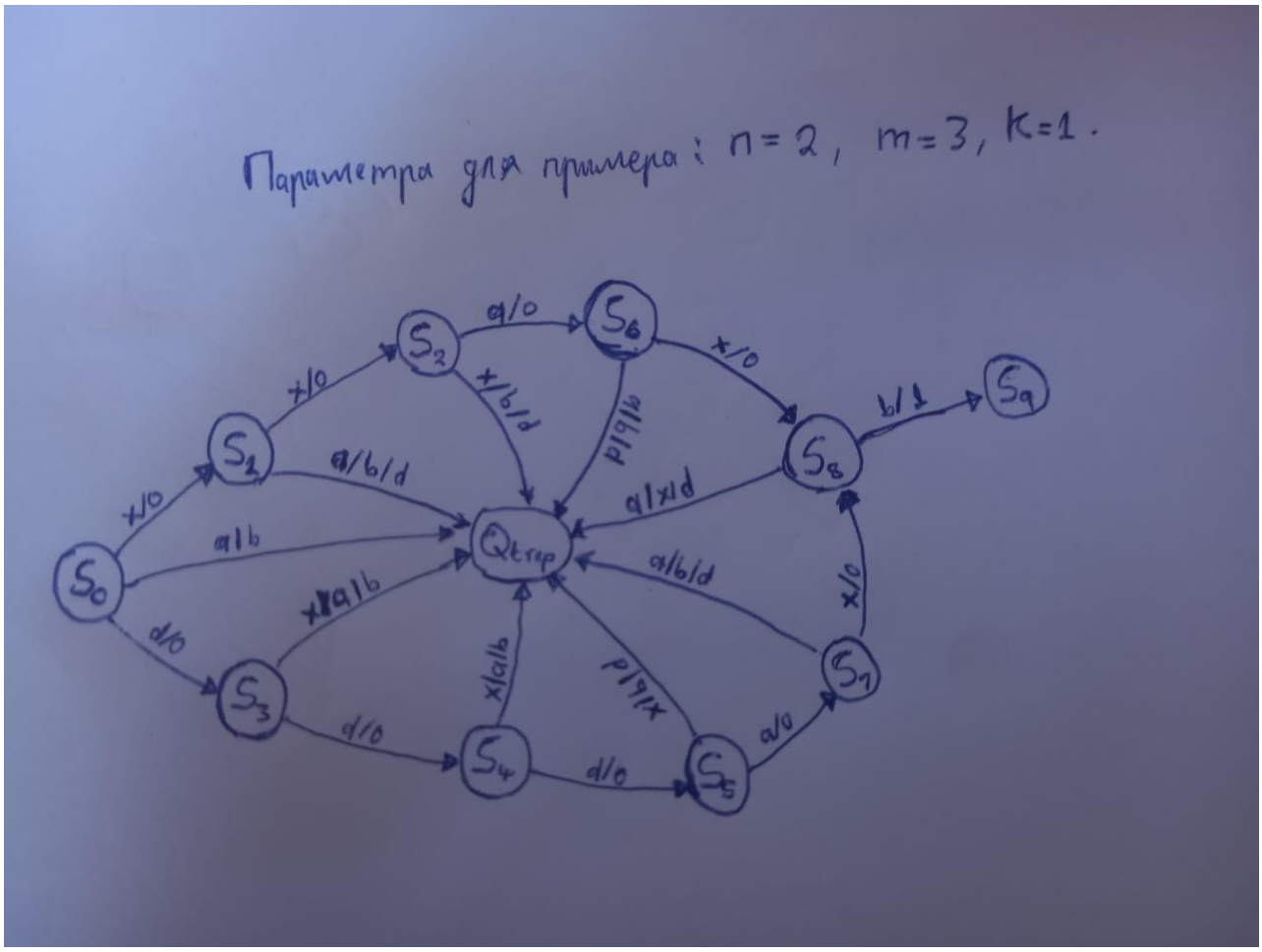
m_ij = X_ij/U_ij

Текущее состояние	**Вход x**	**Вход d**	**Вход a**	**Вход b**
(Qstart,0,0,0)	(ReadingX,1,0,0)/0	(ReadingD,0,1,0)/0	Qtrap/0	Qtrap/0
(ReadingX,i,0,g) i<n	(ReadingX,i+1,0,g)/0	Qtrap/0	Qtrap/0	Qtrap/0
(ReadingX,n,0,g) g<k-1	(ReadingX,1,0,g+1)/0	(ReadingD,0,1,g+1)/0	Qtrap/0	Qtrap/0
(ReadingX,n,0,k-1)	Qtrap/0	Qtrap/0	(SuffixA,-,-,-)/0	Qtrap/0
(ReadingD,0,j,g) j<m	Qtrap/0	(ReadingD,0,j+1,g)/0	Qtrap/0	Qtrap/0
(ReadingD,0,m,g) g<k-1	(ReadingX,1,0,g+1)/0	(ReadingD,0,1,g+1)/0	Qtrap/0	Qtrap/0
(ReadingD,0,m,k-1)	Qtrap/0	Qtrap/0	(SuffixA,-,-,-)/0	Qtrap/0
(SuffixA,-,-,-)	Qtrap/0	Qtrap/0	(SuffixX,-,-,-)/0	Qtrap/0
(SuffixX,-,-,-)	Qtrap/0	Qtrap/0	Qtrap/0	(SuffixB,-,-,-)/0
(SuffixB,-,-,-)	Qtrap/0	Qtrap/0	Qtrap/0	(Qfinal,-,-,-)/1
(Qfinal,-,-,-)	-	-	-	-
Qtrap	Qtrap/0	Qtrap/0	Qtrap/0	Qtrap/0

7.3 Метод 3: Ориентированный граф (диаграмма переходов)

$\Gamma = \langle Q, \Sigma, \Delta, \delta, \lambda \rangle$

Диаграмма переходов:



Пояснение к диаграмме:

● **Верхняя ветка**: Обработка последовательности "xxaxb"

● **Нижняя ветка**: Обработка последовательности "dddaxb"

● **Объединение**: Обе ветки сходятся в состоянии S8 для обработки финального "xb"

● **Финальное состояние:** S9 достигается только при корректном завершении слова

8. Выводы

В ходе лабораторной работы был успешно разработан алгоритм синтеза конечных автоматов и создана программа для автоматического построения конечного автомата Мили по регулярному выражению (x|d)<n|m>k(<a>x<b>).

Основные результаты:

1. Разработан универсальный алгоритм синтеза, работающий для любых значений n, m, k
2. Создана программа на Python, автоматически генерирующая автоматную матрицу
3. Реализована возможность настройки параметров через входной файл
4. Реализована функция проверки входных слов на соответствие регулярному выражению
5. Автомат корректно обрабатывает все допустимые комбинации групп и отвергает недопустимые

Преимущества реализованного решения:

- Гибкость: поддержка произвольных параметров n, m, k
- Автоматизация: автоматическая генерация состояний и переходов
- Визуализация: понятное представление автомата тремя способами
- Практичность: возможность тестирования произвольных входных слов

Программа демонстрирует практическое применение теории синтеза конечных автоматов и может быть расширена для поддержки более сложных регулярных выражений. Полученные

навыки могут быть применены в задачах компиляции, лексического анализа и проектирования цифровых устройств.