КАФЕДРА

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

должность, уч. степень, звание	подпись, дата	инициалы, фамилия

Отчет о лабораторной работе №5

Синтез конечных автоматов

По дисциплине: Теория вычислительных процессов

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № _____ _____ _____

подпись, дата | инициалы, фамилия

Санкт-Петербург 2024

Цель работы:

Задание:
В данной лабораторной работе требуется:
- Построить конечный автомат, который осуществляет проверку входного слова на допустимость в заданном регулярном выражении используя алгоритм синтеза конечных автоматов;
- Привести в отчете процесс синтеза конечного автомата;
- Создать программу на языке высокого уровня реализующую алгоритм синтеза конечного автомата на основе заданного регулярного выражения.

Требования к программе
- Входные данные
Входными данными является текстовый файл, содержащий регулярное выражение.
- Выходные данные
Выходными данными является текстовый файл, содержащий автоматную матрицу построенного КНА

Выполнение задания:
Вариант:
13) <x<e>f>abc(x|<l|m>)

<xf>: Этот блок представляет собой внешний цикл, который может повторяться 0 или более раз. Внутри него есть цикл **<e>**, который также может повторяться 0 или более раз. Сценарии могут быть следующими:
Могут быть последовательности вроде xeeeef, или, например, xf, или xfxfxeeef, или просто ничего, так как цикл может повториться 0 раз.

abc: Это обязательная последовательность терминальных символов, которая всегда присутствует после внешнего цикла.

(x|<l|m>): В этом блоке либо просто символ x, либо внутренний цикл <l|m>, который подразумевает чередование символов l и m (например, lmlmlm, mllm, или ничего, если повторений нет).

Процесс синтеза КНА

Правила подчинения мест:

1. Начальные места всех термов (букв, символов) многочлена, помещенного в обычные или итерационные скобки, подчинены месту, расположенному непосредственно слева открывающей скобки.
2. Место, расположенное непосредственно справа от закрывающей скобки подчинено конечным местам всех термов многочлена, заключенного в эти скобки, а в случае итерационных скобок, еще и месту расположенному непосредственно слева от соответствующей открывающей скобки.
3. Начальные места всех термов многочлена, заключенного в итерационные скобки, подчинены расположенному непосредственно справа от соответствующей закрывающей скобки.
4. Если место `c` подчинено месту `b` , а место `b` подчинено месту `a` , то место `c` подчинено месту `a` .
5. Каждое место подчинено самому себе.
6. Других случаев подчинения мест в регулярном выражении нет.

Примем следующее правило метки синтезируемого КА:

Каждое состояние Qi, определяемое подмножеством множество **основных** мест регулярных выражений, заданных R1...Rp, отмечается множеством, содержащим все те и только те выражения R1...Rp, конечные места которых подчинены хотябы одному основному месту из числа мест, входящих в подмножество Rp

Некоторое слово альфа в алфавите X регулярных выражений, тогда и только тогда переводит конечный автомат S из начального состояния Q0 в некоторое состояние Qj отмеченное произвольным множеством, содержащее любое из заданных регулярных выражений Ri, когда начальное место РВ Ri связанно с конечным местом этого же РВ словом альфа.

Описание алгоритма синтеза:

1. В заданных исходных РВ все места различаются.
2. Каждому основному месту в РВ R1...Rp в качестве его индекса преписывается натуральное число. При этом всем начальным местам выражения R1...Rp преписывается индекс 0. Все введенные в этом пункте индексы отмечающие основные места, назовем основными индексами.
3. Каждый основной индекс (a) распространяется в качестве не основного индекса на все места, подчиненные месту a, но отличные от него самого. При этом каждое подчиненное место (b) получает некоторое множество не основных индексов.
4. Каждое состояние синтезируемого КА S отмечается подмножеством множества всех основных индексов, которые будем обозначать i1, i2. Пустое множество основных индексов будем обозначать {}. Начальное состояние КА Q0 отмечается 0. Таблица переходов, синтезируемого КА, формируется следующим образом:
1. На пересечении произвольной Xi строки и произвольного qj столбца формируется множество основных индексов, отмечающие это состояние КА в числе которых находятся индексы всех тех и только тех основных мест, которые Xi следуют за предосновными местами, в числе индексов которых имеется хотябы один индекс, отмечающий состояние qj.
2. В противном случае, на пересечении Xi строки и qj столбца пишется пустое состояние {}
5. Каждое подмножество i1...ik, отмечающее состояние КА (q) и соответствующий столбец таблицы переходов отмечается подмножеством множества R1...Rp, конечные места которых, содержит в числе свой индексов, хотябы один свой индекс i1...ik. Так отмеченное состояние является конечным.

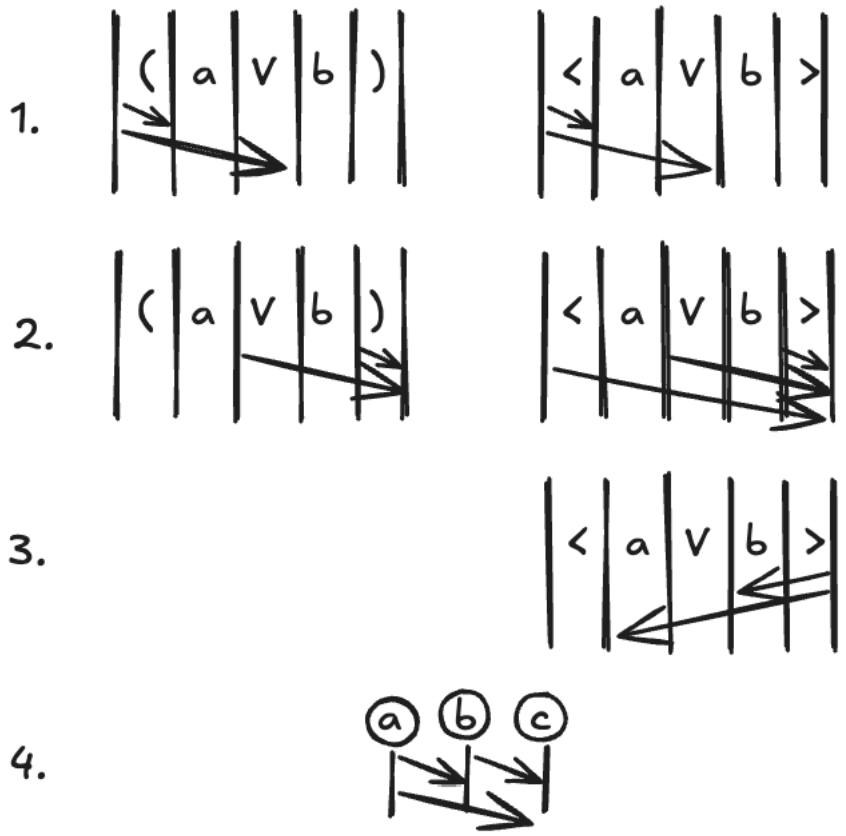


Рисунок 1 – правила синтеза

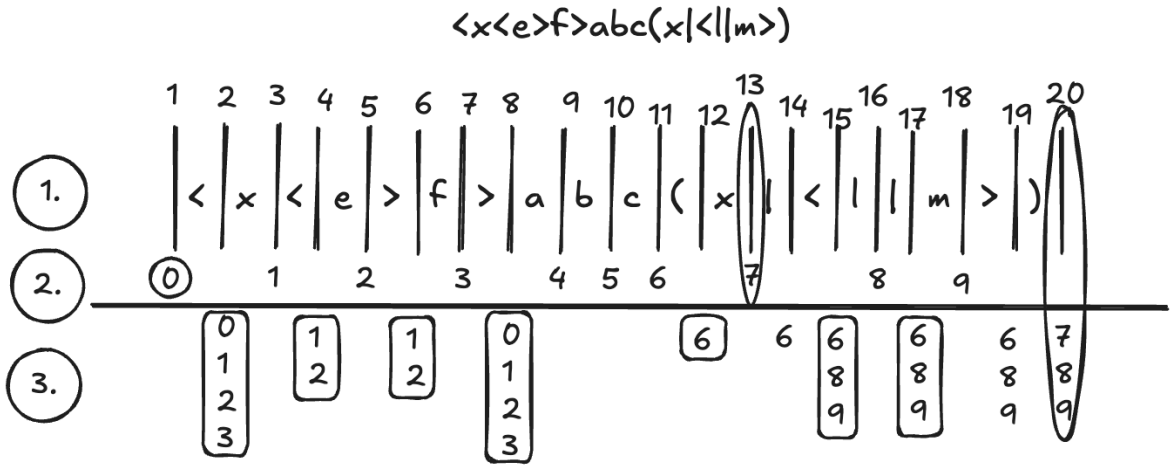


Рисунок 2 – разметка выражения, индексация основных мест, определение по правилам не основных и предосновных мест, определение финальных состояний

<x<e>f>abc(x|<l|m>)

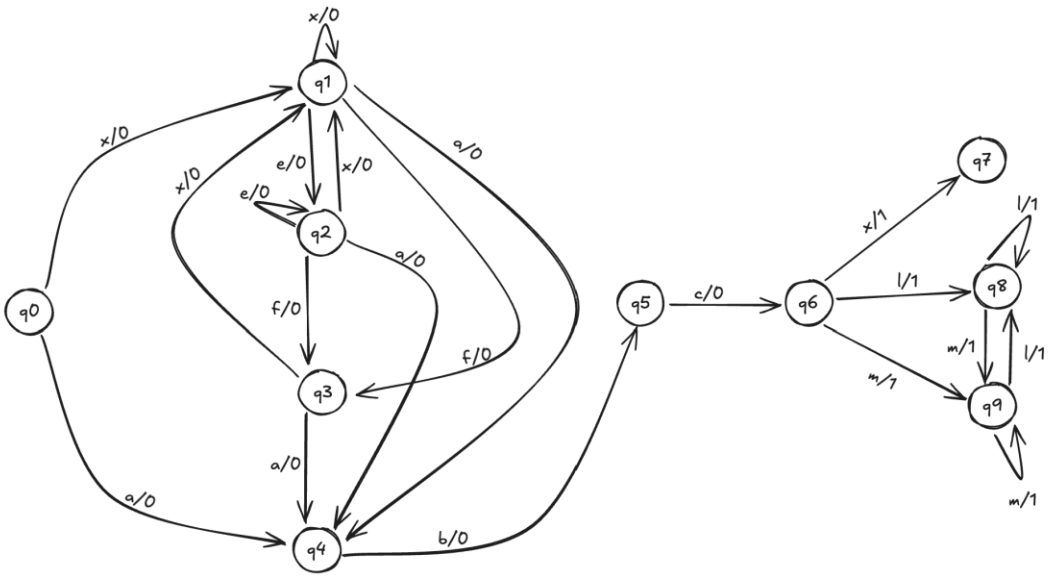
Автоматная матрица:

	q0	q1	q2	q3	q4	q5	q6	q7	q8	q9
X\Q	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
x	1	1	1	1			7			
e		2	2							
f		3	3							
a	4	4	4	4						
b					5					
c						6				
l							8		8	8
m							9		9	9

Таблица переходов:

Q	**x**	**e**	**f**	**a**	**b**	**c**	**l**	**m**
q0	q1			q4				
q1	q1	q2	q3	q4				
q2	q1	q2	q3	q4				
q3	q1			q4				
q4					q5			
q5						q6		
q6	q7						q8	q9
q7								
q8							q8	q9
q9							q8	q9

Граф:

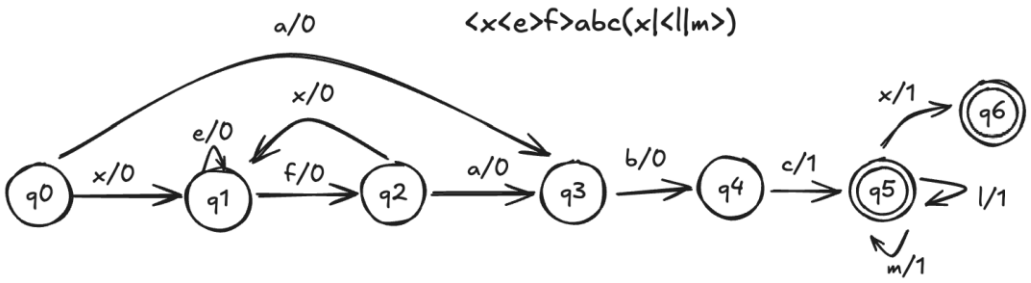


Оптимизация:

1. Объединенная:

q_j / x_i	x	e	f	a	b	c	l	m
q0	q1/0			q3/0				
q1		q1/0	q2/0					
q2	q1/0			q3/0				
q3					q4/0			
q4						q5/1		
q5	q6/1						q5/1	Q5/1
q6								

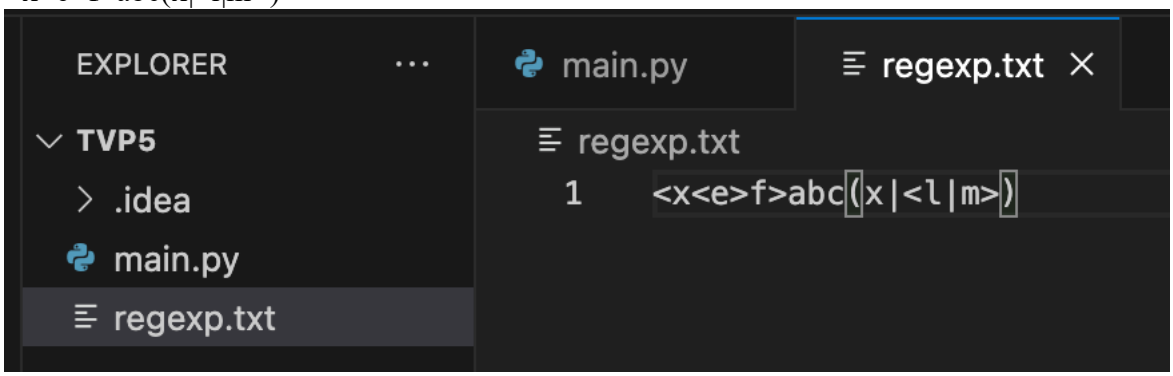
2. Граф



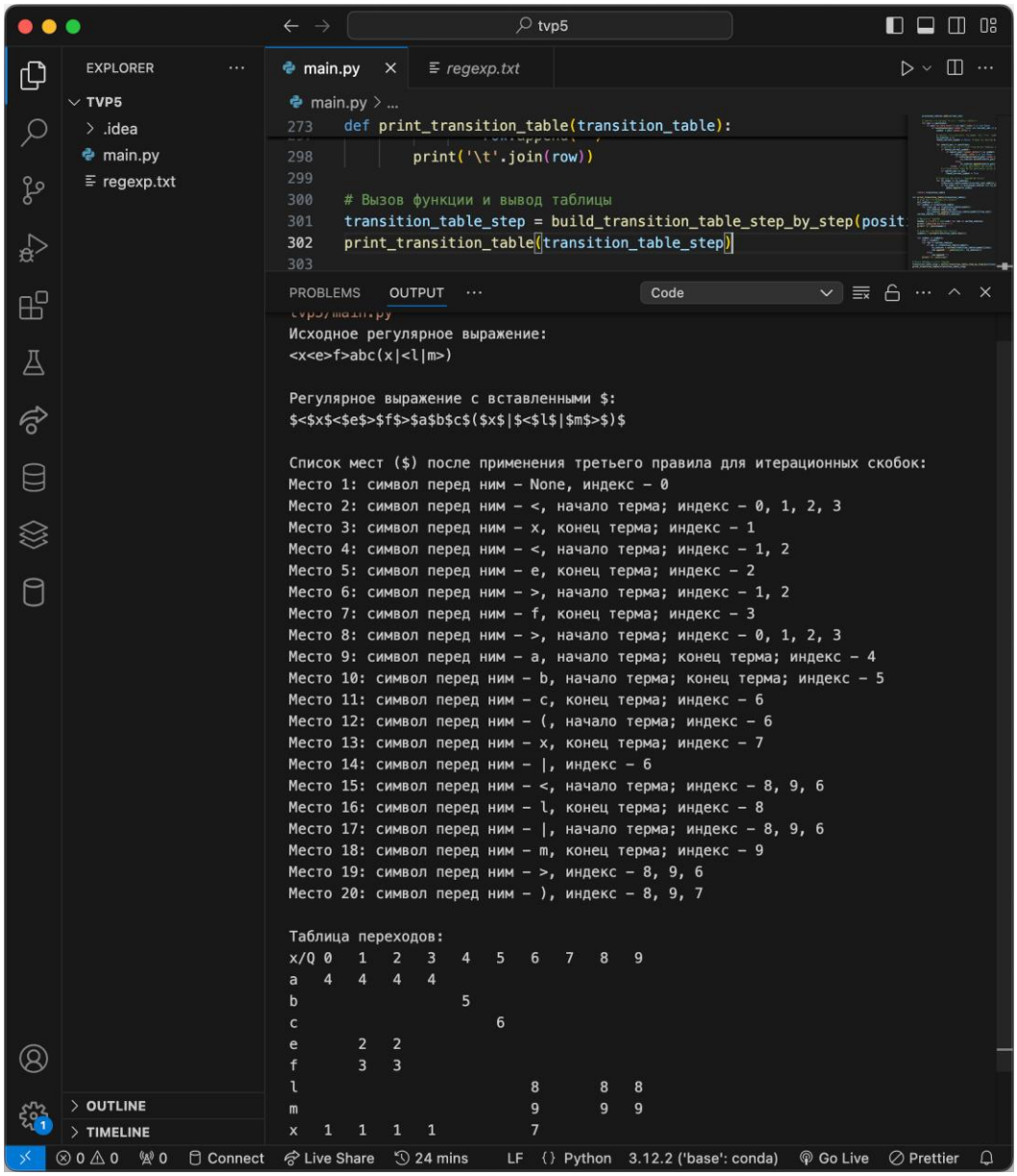
3. Автоматная

q_j / q_i	q0	q1	q2	q3	q4	q5	q6
q0		e/1		a/0			
q1		e/0	f/0				
q2		x/0		a/0			
q3					b/0		
q4						c/1	
q5						l/1 m/1	x/1
q6							

Текстовый файл выходных данных программы:
<x<e>f>abc(x|<l|m>)



Результаты программы:



Работа программы:

Шаг 1: Чтение регулярного выражения из файла

```
with open('regexp.txt', 'r', encoding='utf-8') as f:
    regexp = f.read().strip()

print('Исходное регулярное выражение:')
print(regexp)
```

Что происходит:

1. Открывается файл regexp.txt в режиме чтения с использованием кодировки UTF-8.
2. Содержимое файла считывается, обрезаются пробелы по краям (strip()), и сохраняется в переменной regexp.
3. Исходное регулярное выражение выводится в консоль.

Цель:
Считать и подготовить регулярное выражение для дальнейшей обработки.

Шаг 2: Вставка символа $ между каждым символом

```
new_regexp = '$' + '$'.join(regexp) + '$'

print('\nРегулярное выражение с вставленными $:')
print(new_regexp)
```

Что происходит:

1. Символ $ вставляется между каждым символом регулярного выражения, а также добавляется в начало и конец строки.
   - Например, если regexp = "a(b|c)", то new_regexp = "a(b$|$c$)$".
2. Новый формат регулярного выражения выводится в консоль.

Цель:
Создать структуру, в которой каждый символ можно рассматривать отдельно, а $ служит индикатором границ между символами.

Шаг 3: Создание структуры для хранения информации о $

```
positions = []
dollar_counter = 0

for i, char in enumerate(new_regexp):
    if char == '$':
        dollar_counter += 1
        symbol_before = new_regexp[i - 1] if i > 0 else None
        symbol_after = new_regexp[i + 1] if i + 1 < len(new_regexp) else None

        positions.append({
            'dollar_number': dollar_counter,
            'symbol_before': symbol_before,
            'symbol_after': symbol_after,
            'index': None,
```

```
            'is_term_start': False,
            'is_term_end': False,
        })
```

Что происходит:

1. Программа итерируется по каждому символу new_regexp. Если текущий символ — $, сохраняются следующие данные:
   - dollar_number: порядковый номер $.
   - symbol_before: символ перед $.
   - symbol_after: символ после $.
   - index: изначально None, индекс будет присвоен позже.
   - is_term_start: флаг, указывающий на начало терма (изначально False).
   - is_term_end: флаг, указывающий на конец терма (изначально False).
2. Для каждого $ создается запись в списке positions.

Цель:
Подготовить структуру данных для анализа и обработки всех мест, где находится $.

Шаг 4: Определение начальных и конечных мест термов

```
for i, pos in enumerate(positions):
    symbol_before = pos['symbol_before']

    if symbol_before is not None and symbol_before.isalpha():
        pos['is_term_end'] = True

    if i + 1 < len(positions):
        next_symbol_before = positions[i + 1]['symbol_before']
        if next_symbol_before is not None and next_symbol_before.isalpha():
            pos['is_term_start'] = True
```

Что происходит:

1. Определяются начальные и конечные места термов:
   - Если перед $ находится буква, это конец терма (is_term_end = True).
   - Если за $ находится буква, это начало терма (is_term_start = True).

Пример:
Для выражения a(b$|$c$)$:
- $ перед a — начало терма.
- $ после a — конец терма.

Цель:
Отметить, где начинается и заканчивается каждый терм в регулярном выражении.

Шаг 5: Присвоение индексов местам после термов

```
index_counter = 0

for idx, pos in enumerate(positions):
    if idx == 0:
        pos['index'] = index_counter
        index_counter += 1
    else:
        symbol_before = pos['symbol_before']
        if symbol_before is not None and symbol_before.isalpha():
            pos['index'] = index_counter
            index_counter += 1
```

Что происходит:

1. Присваиваются индексы местам, следующим за окончаниями термов.
2. Первый $ всегда получает индекс 0.
3. Если перед $ находится буква, текущему $ присваивается новый индекс, который увеличивается на единицу.

Цель:
Установить индексы для всех значимых мест регулярного выражения.

Шаг 6: Обработка вложенных конструкций (например, скобок)

```
def apply_initial_index_rule(positions):
    stack = []
    for i, pos in enumerate(positions):
        symbol_before = pos['symbol_before']

        if symbol_before in opening_brackets:
            closing_bracket = opening_brackets[symbol_before]
            parent_index = positions[i - 1]['index'] if i > 0 else None
            stack.append({'closing_bracket': closing_bracket, 'parent_index': parent_index})

        elif symbol_before in closing_brackets:
            if stack and stack[-1]['closing_bracket'] == symbol_before:
                stack.pop()

        if (pos['is_term_start'] or pos['symbol_after'] in ('<', '(')) and pos['index'] is None:
            if stack and stack[-1]['parent_index'] is not None:
                pos['index'] = stack[-1]['parent_index']
```

Что происходит:

1. Стек используется для отслеживания вложенности скобок.
2. Для каждого $, связанного с началом или концом терма, назначается индекс родительской скобки, если он существует.

Цель:
Обеспечить корректную обработку вложенных конструкций, таких как a(b|c) или a<(b)>.

Шаги 7–8: Применение правил для закрывающих и итерационных скобок

1. Обрабатываются сложные конструкции:
- Закрывающие скобки корректно связываются с открывающими.
- Индексы обновляются для мест внутри итерационных скобок < >.

Цель:
Обеспечить поддержку всех типов скобок и сложных структур регулярного выражения.

Шаг 9: Построение таблицы переходов

```
def build_transition_table_step_by_step(positions):
    transition_table = defaultdict(lambda: defaultdict(set))
    processed_indices = set()
    queue = [0]

    while queue:
        current_idx = queue.pop(0)
        processed_indices.add(current_idx)

        for pos in positions:
            if pos['is_term_start'] and pos['index'] is not None:
                symbol = pos['symbol_after']
                to_indices = []
                found_current_symbol = False

                for search_pos in positions:
                    if found_current_symbol:
                        if search_pos['symbol_before'] == symbol:
                            if search_pos['index'] is not None:
                                to_indices.append(search_pos['index'])
                            break
                    if search_pos == pos:
                        found_current_symbol = True

                for to_index in to_indices:
                    transition_table[symbol][current_idx].add(to_index)
                    if to_index not in processed_indices and to_index not in queue:
                        queue.append(to_index)
    return transition_table
```

Что происходит:
1. Создается таблица переходов:
- from_index: начальный индекс.
- symbol: символ перехода.
- to_index: целевой индекс.
2. Индексы обрабатываются в порядке очереди, пока все переходы не будут учтены.

Цель:
Создать таблицу переходов для конечного автомата, который описывает регулярное выражение.

синтез конченого автомата по регулярному выражению происходит так:
1. на вход поступает выражение - <x<e>f>abc(x|<l|m>)
2. и оно разбивается на каждый отдельный символ подряд - то есть
$<$x$<$e$>$f$>$abc$($x$|$<l|m>$)$
$ - обозначает место в выражении
3. далее нужно прописать индексы
под первым $ ставится 0 - тк это самое начало выражения
далее ставит индексы по порядку на каждый знак $, который стоит именно после буквы, то есть получается:
под третьим $ - 1
под пятым $ - 2
под седьмым $ - 3
под девятым $ - 4
под десятым $ - 5
под одиннадцатым $ - 6
под тринадцатым $ - 7
под шестнадцатым $ - 8
под восемнадцатым $ - 9
4. после того как мы подписали все индексы, надо подписать значения (уже подчинения) под остальными знаками $
подчинения работают по правилам:
1 правило:
если выражение типа: $($a$+$b$)$
то значение (или значения), которое под первым $ переходит в значение (или значения) под вторым $
и так же значение (или значения), которое под первым $ переходит в значение (или значения) под четвертым $
если выражение типа: $<$a$+$b$>$
то значение (или значения), которое под первым $ переходит в значение (или значения) под вторым $
и так же значение (или значения), которое под первым $ переходит в значение (или значения) под четвертым $
2 правило:
если выражение типа: $($a$+$b$)$
то значения (или значения) под третьим $ переходит в значение (или значения) под шестым $
и так же значение (или значения), которое под пятым $ переходит в значение (или значения) под шестым $
если выражение типа: $<$a$+$b$>$
то значения (или значения) под первым $ переходит в значение (или значения) под шестым $
и так же значения (или значения) под третьим $ переходит в значение (или значения) под шестым $
и так же значение (или значения), которое под пятым $ переходит в значение (или значения) под шестым $
3 правило:
если выражение типа: $<$a$+$b$>$

то значения (или значения) под шестым $ переходит в значение (или значения) под четвертым $
и так же значения (или значения) под шестым $ переходит в значение (или значения) под вторым $
4 правило:
если значения (или значения) переходят из первого $ во второй, а значения из второго $ переходят в третий $, то это значит, что значения из первого $ должны переходить в третий $
и так, опираясь на все эти правила, в оставшиеся $ для нашего выражения - $<$x$<$e$>$f$>$abc$($x$|$<l|m>$)$ вписываются такие значения:
под вторым $ - 0, 1, 2, 3, 4
под четвертым $ - 0, 1, 2, 3, 4
под шестым $ - 0, 1, 2, 3, 4
под восьмым $ - 0, 1, 2, 3, 4
под двенадцатым $ - 6
под пятнадцатым $ - 6, 8, 9
под семнадцатым $ - 6, 8, 9
под девятнадцатым $ - 6, 8, 9
под двадцатым $ - 6, 7, 8, 9
5. теперь выбираем те значения $, которые находятся перед буквами $<$x$<$e$>$f$>$abc$($x$|$<l|m>$)$ - смотреть будем на них
у нас остается:
под вторым $ - 0, 1, 2, 3, 4
под четвертым $ - 0, 1, 2, 3, 4
под шестым $ - 0, 1, 2, 3, 4
под восьмым $ - 0, 1, 2, 3, 4
под двенадцатым $ - 6
под пятнадцатым $ - 6, 8, 9
под семнадцатым $ - 6, 8, 9
и так же смотрим на основные индексы, которые получили ранее:
под третьим $ - 1
под пятым $ - 2
под седьмым $ - 3
под девятым $ - 4
под десятым $ - 5
под одиннадцатым $ - 6
под тринадцатым $ - 7
под шестнадцатым $ - 8
под восемнадцатым $ - 9
6. теперь составляем автоматную матрицу для выражения - $<$x$<$e$>$f$>$abc$($x$|$<l|m>$)$
строки — это буквы: x, e, f, a, b, c, l, m - каждую отдельную букву записать в свою строку.
столбцы — это "состояния", туда мы записываем значения. в первым столбец записываем 0. тк это первое значение у индекса.
после этого начинаем заполнять таблицу. заполняется она так.
сопоставляем столбец и строку с нашим выражением - $<$x$<$e$>$f$>$abc$($x$|$<l|m>$)$
у нас первая строка x и первый столбец 0. смотрим на выражение и ищем там x, смотрим есть ли в $ перед ним значение 0 - если есть, то ставим в таблицу значение $,

который после этого $ (то есть 1). 1 — это новое “состояние” - создаем для нее новый столбец.
и так со всеми буквами нужно пройтись по всем цифрам и подобавлять новые столбцы.

Листинг:

```
from collections import defaultdict

# Шаг 1: Чтение регулярного выражения из файла regexp.txt
with open('regexp.txt', 'r', encoding='utf-8') as f:
    regexp = f.read().strip()

print('Исходное регулярное выражение:')
print(regexp)

# Шаг 2: Вставка символа '$' между каждым символом, включая начало и конец
new_regexp = '$' + '$'.join(regexp) + '$'

print('\nРегулярное выражение с вставленными $:')
print(new_regexp)

# Шаг 3: Создание структуры для хранения информации о местах '$'
positions = []
dollar_counter = 0

for i, char in enumerate(new_regexp):
    if char == '$':
        dollar_counter += 1
        if i == 0:
            symbol_before = None  # Перед первым символом нет символа
        else:
            symbol_before = new_regexp[i - 1]

        # Определяем символ после текущего `$`
        if i + 1 < len(new_regexp):
            symbol_after = new_regexp[i + 1]
        else:
            symbol_after = None

        positions.append({
            'dollar_number': dollar_counter,
            'symbol_before': symbol_before,
            'symbol_after': symbol_after,
            'index': None,  # Индекс будет назначен позже
            'is_term_start': False,  # Флаг начала терма
            'is_term_end': False,  # Флаг конца терма
        })

# Шаг 4: Определение начальных и конечных мест термов
opening_brackets = {'(': ')', '<': '>'}
closing_brackets = {')': '(', '>': '<'}
```

```
for i, pos in enumerate(positions):
    symbol_before = pos['symbol_before']

    # Если символ перед местом является буквой, то это конец терма
    if symbol_before is not None and symbol_before.isalpha():
        pos['is_term_end'] = True

    # Если следующий символ перед местом является буквой, то это начало терма
    if i + 1 < len(positions):
        next_symbol_before = positions[i + 1]['symbol_before']
        if next_symbol_before is not None and next_symbol_before.isalpha():
            pos['is_term_start'] = True

# Шаг 5: Присвоение порядковых индексов основным местам (после термов)
index_counter = 0

for idx, pos in enumerate(positions):
    # Первому месту присваиваем индекс 0
    if idx == 0:
        pos['index'] = index_counter
        index_counter += 1
    else:
        symbol_before = pos['symbol_before']
        if symbol_before is not None and symbol_before.isalpha():
            pos['index'] = index_counter
            index_counter += 1

# Шаг 6: Функция для присвоения неосновных индексов начальным местам внутри
скобок
def apply_initial_index_rule(positions):
    stack = []
    for i, pos in enumerate(positions):
        symbol_before = pos['symbol_before']

        # Обработка открывающих скобок
        if symbol_before in opening_brackets:
            closing_bracket = opening_brackets[symbol_before]
            # Получаем индекс непосредственно перед открывающей скобкой
            parent_index = positions[i - 1]['index'] if i > 0 else None

            # Добавляем в стек даже если индекс None, чтобы правильно отслеживать
вложенность
            stack.append({'closing_bracket': closing_bracket, 'parent_index': parent_index})

        # Обработка закрывающих скобок
        elif symbol_before in closing_brackets:
            if stack and stack[-1]['closing_bracket'] == symbol_before:
                stack.pop()
            else:
```

```
            print(f"Несоответствующая закрывающая скобка {symbol_before} на позиции
{i}")

        # Присвоение индексов начальным местам внутри скобок
        if (pos['is_term_start'] or pos['symbol_after'] in ('<', '(')) and pos['index'] is None and
pos['symbol_before'] != '>':
            if stack and stack[-1]['parent_index'] is not None:
                pos['index'] = stack[-1]['parent_index']
            # Если индекс в стеке None, оставляем индекс None для текущего места

# Вызов функции для шага 6
apply_initial_index_rule(positions)

# Шаг 7: Применение правила для закрывающих скобок с учетом вложенности
for i, pos in enumerate(positions):
    symbol_before = pos['symbol_before']

    # Если это закрывающая скобка
    if symbol_before in closing_brackets:
        opening_bracket = closing_brackets[symbol_before]

        # Инициализируем уровень вложенности
        nesting_level = 1
        end_indices = []
        j = i - 1

        while j >= 0:
            current_symbol = positions[j]['symbol_before']

            # Если встречаем такую же закрывающую скобку, увеличиваем уровень
вложенности
            if current_symbol == symbol_before:
                nesting_level += 1
            # Если встречаем соответствующую открывающую скобку
            elif current_symbol == opening_bracket:
                nesting_level -= 1
                if nesting_level == 0:
                    # Если это итерационная скобка, добавляем индекс места перед
открывающей скобкой
                    if opening_bracket == '<' and positions[j - 1]['index'] is not None:
                        end_indices.append(positions[j - 1]['index'])
                    break
            else:
                # Если это конец терма внутри скобок, добавляем его индекс
                if positions[j]['is_term_end'] and positions[j]['index'] is not None:
                    end_indices.append(positions[j]['index'])
            j -= 1

        # Присваиваем индексы текущему месту после закрывающей скобки
        if end_indices:
            # Убираем дублирование индексов
```

```
            end_indices = list(set(end_indices))
            # Если индекс уже есть и это не список, превращаем его в список
            if pos['index'] is not None and not isinstance(pos['index'], list):
                pos['index'] = [pos['index']]
            # Если индекс это список, расширяем его
            if isinstance(pos['index'], list):
                pos['index'].extend(end_indices)
                pos['index'] = list(set(pos['index']))  # Убираем дубликаты
            else:
                pos['index'] = end_indices

# Повторное применение функции для начальных мест после шага 7
apply_initial_index_rule(positions)

# Шаг 8: Применение третьего правила для итерационных скобок
def apply_third_rule(positions):
    i = 0
    while i < len(positions):
        pos = positions[i]
        symbol_before = pos['symbol_before']

        # Если обнаружили открывающую итерационную скобку '<'
        if symbol_before == '<':
            # Инициализируем стек для учета вложенности
            stack = [i]
            j = i + 1
            # Словарь для хранения уровней вложенности и соответствующих начальных мест термов
            nested_term_start_positions = {len(stack): []}

            # Проверяем, является ли позиция после открывающей скобки началом терма и не является концом терма
            if pos['is_term_start'] and not pos['is_term_end']:
                nested_term_start_positions[len(stack)].append(i)

            while j < len(positions):
                current_symbol = positions[j]['symbol_before']

                if current_symbol == '<':
                    # Новая открывающая скобка, увеличиваем вложенность
                    stack.append(j)
                    nested_term_start_positions[len(stack)] = []
                elif current_symbol == '>':
                    # Закрывающая скобка, уменьшаем уровень вложенности
                    last_opening = stack.pop()
                    # Получаем индекс места после закрывающей скобки
                    index_after_closing = positions[j]['index']

                    # Назначаем индекс только начальным термам текущего уровня вложенности
                    for idx in nested_term_start_positions[len(stack) + 1]:
```

```
                positions[idx]['index'] = index_after_closing

            # Удаляем записи для уровня вложенности, который завершен
            del nested_term_start_positions[len(stack) + 1]

            # Если стек пуст, выходим из цикла, так как нашли соответствующую
закрывающую скобку
            if not stack:
                break
        # Если это начальное место терма внутри текущего уровня вложенности,
которое не является концом терма
        if positions[j]['is_term_start'] and not positions[j]['is_term_end'] and stack and
positions[j]['symbol_before'] != '>':
            # Добавляем позицию в список начальных мест термов для текущего
уровня вложенности
            nested_term_start_positions[len(stack)].append(j)
        j += 1

    if stack:
        print(f"Несоответствующая открывающая скобка '<' на позиции {i}")
        i += 1
        continue

    # Обновляем счетчик i, чтобы продолжить поиск
    i = j + 1
  else:
    i += 1

# Вызов функции для шага 8
apply_third_rule(positions)

# Вывод итогового списка мест с индексами и метками начала и конца термов
print('\nСписок мест ($) после применения третьего правила для итерационных
скобок:')
for pos in positions:
  start_end = ''
  if pos['is_term_start']:
    start_end += 'начало терма; '
  if pos['is_term_end']:
    start_end += 'конец терма; '
  index_value = pos['index']
  if isinstance(index_value, list):
    index_value = ', '.join(map(str, index_value))
  print(
    f"Место {pos['dollar_number']}: символ перед ним - {pos['symbol_before']},
{start_end}индекс - {index_value}")

# Шаг 9: Построение таблицы переходов согласно алгоритму
def build_transition_table_step_by_step(positions):
  transition_table = defaultdict(lambda: defaultdict(set))
  processed_indices = set()
```

```
    queue = [0]  # Начинаем с индекса 0

    while queue:
        current_idx = queue.pop(0)
        processed_indices.add(current_idx)

        # Находим все начала термов с текущим индексом
        for pos in positions:
            if pos['is_term_start'] and pos['index'] is not None:
                if isinstance(pos['index'], list) and current_idx in pos['index'] or pos['index'] ==
current_idx:
                    symbol = pos['symbol_after']

                    # Находим все возможные `to_index` для этого `symbol_after` после
текущего `from_index`
                    to_indices = []
                    found_current_symbol = False  # Флаг для поиска после первой позиции

                    for search_pos in positions:
                        # Находим первое соответствие после текущего индекса
                        if found_current_symbol:
                            if search_pos['symbol_before'] == symbol:
                                if search_pos['index'] is not None:
                                    if isinstance(search_pos['index'], list):
                                        to_indices.extend(search_pos['index'])
                                    else:
                                        to_indices.append(search_pos['index'])
                                break  # После нахождения переходим к следующему символу
                        # Устанавливаем флаг после нахождения начального места терма
                        if search_pos == pos:
                            found_current_symbol = True

                    # Добавляем переходы в таблицу переходов
                    for to_index in to_indices:
                        transition_table[symbol][current_idx].add(to_index)
                        if to_index not in processed_indices and to_index not in queue:
                            queue.append(to_index)

    return transition_table

def print_transition_table(transition_table):
    # Собираем все уникальные индексы
    all_indices = set()
    for symbol in transition_table:
        for from_idx in transition_table[symbol]:
            all_indices.add(from_idx)
            all_indices.update(transition_table[symbol][from_idx])
    sorted_indices = sorted(all_indices)

    # Заголовок таблицы
    header = ['x/Q'] + [str(idx) for idx in sorted_indices]
```

```
    print('\nТаблица переходов:')
    print('\t'.join(header))

    # Собираем все уникальные символы
    symbols = sorted(transition_table.keys())

    for symbol in symbols:
        row = [symbol]
        for idx in sorted_indices:
            if idx in transition_table[symbol]:
                to_indices = sorted(transition_table[symbol][idx])
                row.append(','.join(map(str, to_indices)))
            else:
                row.append('')
        print('\t'.join(row))

# Вызов функции и вывод таблицы
transition_table_step = build_transition_table_step_by_step(positions)
print_transition_table(transition_table_step)
```

Вывод:

В данной лабораторной работе был реализован алгоритм синтеза конечного недетерминированного автомата (КНА) на основе заданного регулярного выражения. Входным данным служит текстовый файл с регулярным выражением, а выходные данные представлены в виде автоматной матрицы КНА, сохраненной в текстовом файле.