МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №  43

ОТЧЁТ

ЗАЩИЩЁН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

### ст.пр.                                   Рогачёв С.А.

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

ОТЧЁТ О ЛАБОРАТОРНОЙ РАБОТЕ №5.

Синтез КНА.

по курсу: Теория вычислительных процессов

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4136                                                                                Бобрович Н. С.

                                                                         подпись, дата                      инициалы, фамилия

Санкт-Петербург 2024

1. **Цель работы:**

Изучить синтез КНА.

1. **Постановка задачи:**

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 отчёте процесс синтеза конечного автомата;

Создать программу на языке высокого уровня реализующую алгоритм синтеза конечного автомата на основе заданного регулярного выражения.

**Вариант 3: nm(c|d)n<n|m>**

1. **Алгоритм синтеза КНА:**

Шаг 1: Разбор регулярного выражения

Первым шагом является разбор регулярного выражения на составляющие элементы. Регулярное выражение может содержать символы алфавита, а также специальные операторы, такие как \*, | и () для обозначения повторений, альтернатив и группировок соответственно.

Шаг 2: Построение синтаксического дерева

На основании разбора регулярного выражения строится синтаксическое дерево. Это дерево представляет собой иерархическую структуру, где листья соответствуют символам алфавита, а внутренние узлы — операторам.

Шаг 3: Конструирование базовых автоматов

Каждому листу синтаксического дерева сопоставляется базовый автомат, состоящий из двух состояний: начального и конечного. Переходы между этими состояниями определяются символами алфавита.

Шаг 4: Объединение базовых автоматов

Используя операторы синтаксического дерева, базовые автоматы объединяются в более сложные конструкции. Основные операции объединения включают:

Конкатенация: Автоматы соединяются последовательно, так что конечное состояние первого автомата становится начальным состоянием второго.

Альтернатива: Создаётся новый начальный и конечный состояния, из которых исходят и в которые входят переходы соответствующих автоматов.

Повторение (\*): Из начального состояния нового автомата идут переходы в начальные состояния подавтоматов, а из их конечных состояний возвращаются обратно в начальное состояние нового автомата.

Шаг 5: Оптимизация полученного автомата

После построения основного автомата могут проводиться оптимизации, направленные на уменьшение количества состояний и переходов. Примеры таких оптимизаций:

Удаление недостижимых состояний: Удаляются состояния, в которые невозможно попасть ни из какого другого состояния.

Объединение эквивалентных состояний: Если два состояния ведут себя одинаково относительно всех возможных входных символов, они могут быть объединены.

Шаг 6: Определение начального и конечного состояний

Начальное состояние определяется как то, из которого начинаются все возможные пути в автомате. Конечные состояния — те, в которых автомат заканчивает свою работу и принимает входную последовательность.

1. **Текст программы на языке высокого уровня:**

import sys

# Функция для чтения алфавита из регулярного выражения

def read\_alphabet(regex):

alphabet = []

for i in range(len(regex)): # Проходимся по каждому символу в регулярном выражении

if regex[i] not in alphabet and regex[i] not in ['(', ')', '|', '<', '>']: # Проверяем, является ли текущий символ символом алфавита

alphabet.append(regex[i]) # Добавляем уникальный символ в алфавит

return alphabet

# Функция для вывода матрицы переходов в файл

def print\_output(output\_matrix, conditions):

with open('output.txt', 'w', encoding='utf-8') as f:

original\_stdout = sys.stdout # Сохраняем стандартный вывод

sys.stdout = f # Перенаправляем вывод в файл

print(' ', ' '.join(['q'+str(i+1) for i in range(0, len(conditions))]))

for i in range(len(conditions)):

print('q'+str(i+1), end=' ')

for j in range(len(conditions)):

print(output\_matrix[i][j], end=' ')

print()

sys.stdout = original\_stdout # Восстанавливаем стандартный вывод

# Функция для определения правил подчинения в регулярном выражении

def subordination\_rules(regex, markup, A):

subordination\_dependencies = [[] for \_ in range(len(regex)+1)]

for i in range(len(regex)):

bracket\_counter = 0

if regex[i] == '(' or regex[i] == '<':

subordination\_dependencies[i+1].append(i)

for j in range(i, len(regex)):

if regex[j] == '(' or regex[j] == '<':

bracket\_counter += 1

if regex[j] == ')' or regex[j] == '>':

if bracket\_counter == 1:

break

else:

bracket\_counter -= 1

if regex[j] == '|' and bracket\_counter == 1:

subordination\_dependencies[j+1].append(i)

bracket\_counter = 0

if regex[i] == '(' or regex[i] == '<':

helper = []

if regex[i] == '<':

helper.append(i

for j in range(i, len(regex)):

if regex[j] == '(' or regex[j] == '<':

bracket\_counter += 1

if regex[j] == ')' or regex[j] == '>':

if bracket\_counter == 1:

subordination\_dependencies[j+1] = helper

break

else:

bracket\_counter -= 1

if (regex[j] in A and regex[j+1] not in A) or (regex[j] == '>' and i != 0) and bracket\_counter == 1:

helper.append(j+1)

bracket\_counter = 0

if regex[i] == '>':

place = i+1

for j in range(i, -1, -1):

if regex[j] == '>':

bracket\_counter += 1

if regex[j] == '<':

if bracket\_counter == 1:

break

else:

bracket\_counter -= 1

if (regex[j] in A or regex[j] == '<') and regex[j-1] not in A and bracket\_counter == 1:

subordination\_dependencies[j].append(place)

for i in range(len(subordination\_dependencies)):

for j in range(len(subordination\_dependencies)):

if i in subordination\_dependencies[j]:

markup[j].extend(x for x in markup[i] if x not in markup[j])

return markup # Возвращаем обновленную разметку

def main():

with open('input.txt', 'r', encoding='utf-8') as f:

regex = f.readline().strip()

A = read\_alphabet(regex)

markup = [[] for \_ in range(len(regex)+1)]

pre\_primary\_places = []

counter = 1

for i in range(len(regex)+1):

if i == 0:

markup[i].append(0)

continue

if regex[i-1] in A:

markup[i].append(counter)

pre\_primary\_places.append(i-1)

counter += 1

markup = subordination\_rules(regex, markup, A) # Применяем правила подчинения к разметке

markup = subordination\_rules(regex, markup, A) # Повторяем применение правил подчинения

conditions = [[0]]

i = 0

table = {a:[] for a in A}

while i < len(conditions):

for a in A:

flag = False

adding = []

for c in conditions[i]:

for p in pre\_primary\_places:

if c in markup[p] and regex[p] == a:

adding.extend(markup[p+1])

flag = True

if not flag:

table[a].append(None)

else:

table[a].append(adding)

if adding not in conditions:

conditions.append(adding)

i += 1

print(table)

exit\_simbols = markup[-1]

is\_condition\_in\_regex = [] # Список для хранения информации о наличии условия в регулярном выражении

for i in range(len(conditions)):

help = False

for c in conditions[i]:

if c in exit\_simbols:

help = True

if help:

is\_condition\_in\_regex.append(1)

else:

is\_condition\_in\_regex.append(0)

array\_table = list(table.values()) # Преобразуем таблицу переходов в список значений

output\_matrix = [[' ' for \_ in range(len(conditions))] for \_ in range(len(conditions))] # Создаем пустую матрицу переходов

keys\_list = list(table.keys())

for key in keys\_list:

# Индексируем элементы в таблице

for i, element in enumerate(table[key]):

try:

# Ищем индекс элемента в условиях

index = conditions.index(element)

except ValueError:

continue

output\_matrix[i][index] = key + '/' + str(is\_condition\_in\_regex[index])

for row in output\_matrix:

print(row)

print\_output(output\_matrix, conditions)

if \_\_name\_\_ == '\_\_main\_\_':

main()

1. **Текстовый файл выходных данных программы:**

q1 q2 q3 q4 q5 q6 q7 q8 q9

q1 n/0

q2 m/0

q3 c/0 d/0

q4 n/1 k/0

q5 n/1 k/0

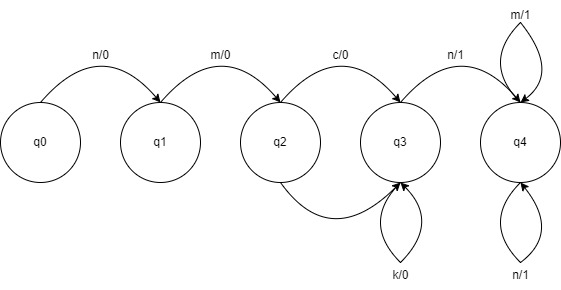
q6 n/1 m/1

q7 n/1 k/0

q8 n/1 m/1

q9 n/1 m/1

1. **Конечный автомат заданный тремя способами:**
2. Граф переходов:



1. Матрица переходов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X\Q | n | m | c | d | k |
| q0 | 0 |  |  |  |  |
| q1 |  | 0 |  |  |  |
| q2 |  |  | 0 | 0 |  |
| q3 | 1 |  |  |  | 0 |
| q4 | 1 | 1 |  |  |  |

3. Автоматная матрица:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Q\Q | q0 | q1 | q2 | q3 | q4 |
| q0 |  | n/0 |  |  |  |
| q1 |  |  | m/0 |  |  |
| q2 |  |  |  | c/0, d/0 |  |
| q3 |  |  |  | k/0 | n/1 |
| q4 |  |  |  |  | n/1, m/1 |

1. **Вывод:**

Изучил синтез КНА.