Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

# Разработка компиляторов

Домашнее задание №2

Вариант 13

Выполнил:

Маликов Глеб Игоревич

Группа № P3324

Преподаватель:

Лаздин Артур Вячеславович

Санкт-Петербург

2025

## Оглавление

## Задание

По заданному регулярному выражению (см. вариант)

- Построить недетерминированный КА;
- По полученному НДА построить ДКА;
- Минимизировать полученный ДКА;
- Для минимального ДКА написать программу-распознаватель предложений языка, порождаемого регулярным выражением.

Продемонстрировать работу распознавателя на различных примерах (не менее трех правильных) предложений.

Использование символов + и ? в регулярных выражениях.

Символ + используется для определения регулярного выражения, повторяющегося один или более раз. В этом смысле p+ = pp*.

Символ ? используется для указания того, что регулярное выражение встречается ноль или один раз, тогда p? = ε|p.

Внимание!

Операции итерации, конкатенации и объединения имеют приоритеты, причем приоритет итерации высший, а объединения – низший. Обычно скобки будут опускаться везде, где их отсутствие не влияет на определение регулярного множества. Регулярное выражение ((a)(b*))|(c) может быть записано следующим образом: ab*|c.

Вариант 13:  a((ab)|(bc))*c

# Выполнение

## Построение НКА

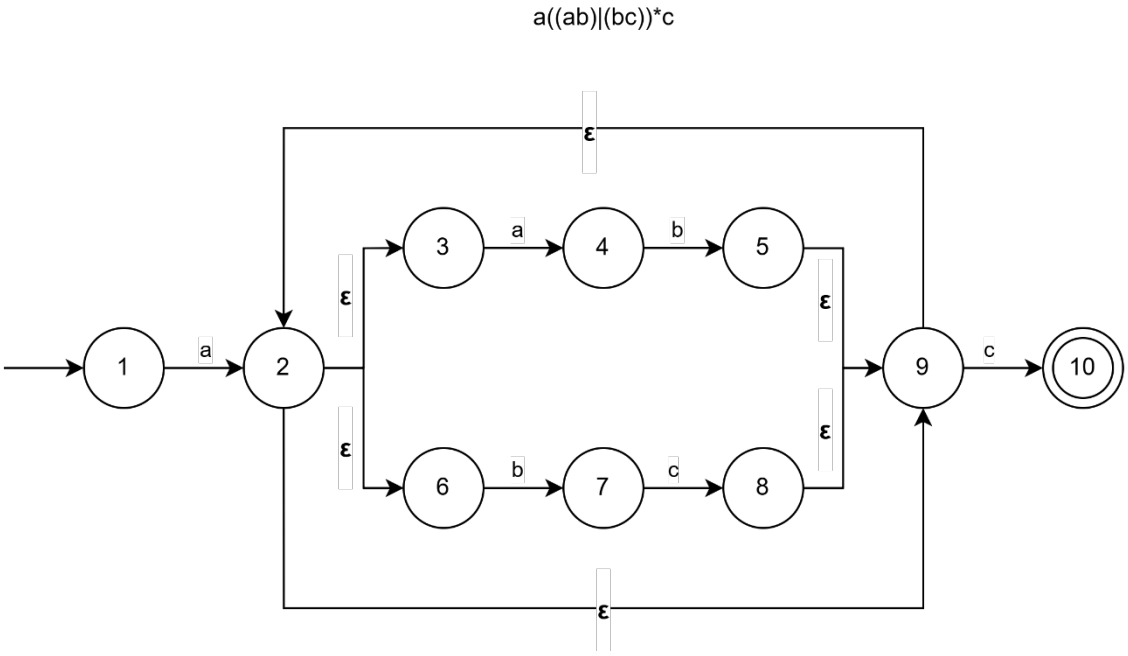


*Рисунок 1 - Выражение a((ab)|(bc))*c в виде НДА*

## Построение ДКА по НДА

Начальное состояние = ε-closure (1)

ε-closure (1) = {1} == State 1

Move ({1}, a) = {2}

ε-closure ({2}) = {2, 3, 6, 9} == State 2

Move ({1}, b) = {}

Move ({1}, c) = {}

State 1 пройден

Move ({2, 3, 6, 9}, a) = {4}

ε-closure ({4}) = {4} == State 3

Move ({2, 3, 6, 9}, b) = {7}

ε-closure ({7}) = {7} == State 4

Move ({2, 3, 6, 9}, c) = {10}

ε-closure ({10}) = {10} == State 5

State 2 пройден

Move ({4}, a) = {}

Move ({4}, b) = {5}

ε-closure ({5}) = {5, 2, 3, 6, 9} == State 6

Move ({4}, c) = {}

State 3 пройден

Move ({7}, a) = {}

Move ({7}, b) = {}

Move ({7}, c) = {8}

ε-closure ({8}) = {8, 2, 3, 6, 9} == State 7

State 4 пройден

Move ({10}, a) = {}

Move ({10}, b) = {}

Move ({10}, c) = {}

State 5 пройден

Move ({5, 2, 3, 6, 9}, a) = {4}

ε-closure ({4}) = {4} == State 3

Move ({5, 2, 3, 6, 9}, b) = {7}

ε-closure ({7}) = {7} == State 4

Move ({5, 2, 3, 6, 9}, c) = {10}

ε-closure ({10}) = {10} == State 5

State 6 пройден

Move ({8, 2, 3, 6, 9}, a) = {4}

ε-closure ({4}) = {4} == State 3

Move ({8, 2, 3, 6, 9}, b) = {7}

ε-closure ({7}) = {7} == State 4

Move ({8, 2, 3, 6, 9}, c) = {10}

ε-closure ({10}) = {10} == State 5

State 7 пройден

| № | Состояние | a | b | c |
|---|---|---|---|---|
| 1 | 1 | 2 | - | - |
| 2 | 2,3,6,9 | 4 | 7 | 10 |
| 3 | 4 | - | 5,2,3,6,9 | - |
| 4 | 7 | - | - | 8,2,3,6,9 |
| **5** | 10 | - | - | - |
| 6 | 5,2,3,6,9 | 4 | 7 | 10 |
| 7 | 8,2,3,6,9 | 4 | 7 | 10 |

*Таблица 1 - Построение ДКА по НДА*

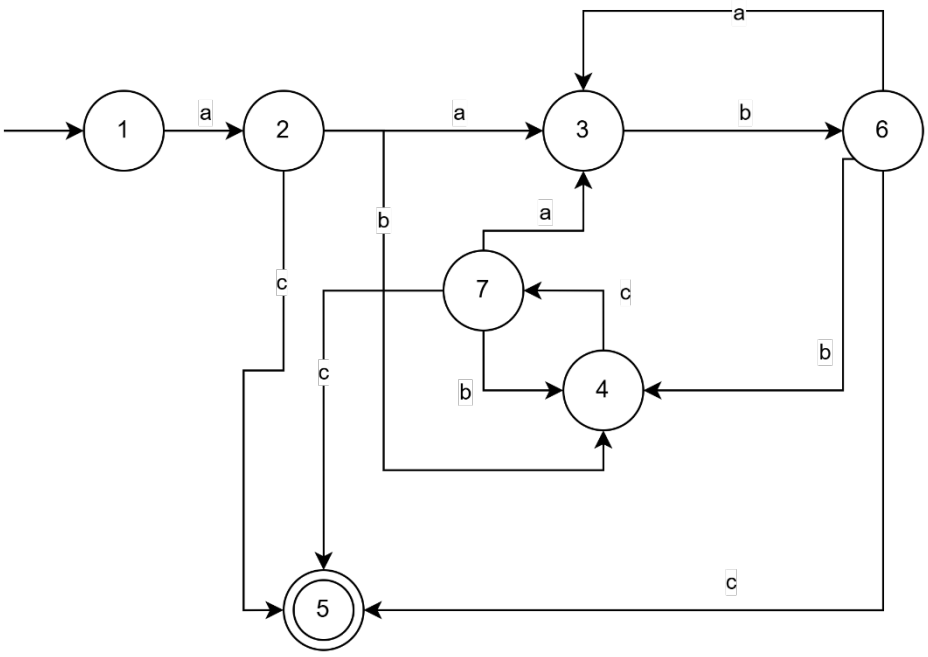


*Рисунок 2 - ДКА выражения a((ab)|(bc))*c*

## Минимизация ДКА

| Состояние\Переход | a | b | c |
|---|---|---|---|
| 1 | 2 | - | - |
| 2 | 3 | 4 | 5 |
| 3 | - | 6 | - |

| | | | |
|---|---|---|---|
| 4 | - | - | 7 |
| 5 (Конечное) | - | - | - |
| 6 | 3 | 4 | 5 |
| 7 | 3 | 4 | 5 |

*Таблица 2 - Состояния и переходы ДКА*

Разделим состояния на группы:

P0 = {1,2,3,4,6,7} не конечные

Q0 = {5} конечные

| Состояния | a | b | c |
|---|---|---|---|
| P0 = {1,2,3,4,6,7} | {P0,P0,-,-,P0,P0} | {-,P0,P0,-,P0,P0} | {-,P1,-,P0,P1,P1} |
| Q0 = {5} | {-} | {-} | {-} |

*Таблица 3 - Первый шаг минимизации ДКА*

P1 -> {P0,-,-} = {1}

Q1 -> {P0,P0,P1} = {2,6,7}

R1 -> {-,P0,-} = {3}

S1 -> {-,-,P0} = {4}

T1 -> {-,-,-} = {5}

| Состояния | a | b | c |
|---|---|---|---|
| P1 = {1} | {Q1} | {-} | {-} |
| Q1 = {2,6,7} | {R1,R1,R1} | {S1,S1,S1} | {T1,T1,T1} |
| R1 = {3} | {-} | {Q1} | {-} |
| S1 = {4} | {-} | {-} | {Q1} |
| T1 = {5} | {-} | {-} | {-} |

*Таблица 4 - Второй шаг минимизации ДКА*

P2 -> {Q1,-,-} = {1}

Q2 -> {R1,S1,T1} = {2,6,7}

R2 -> {-,Q1,-} = {3}

S2 -> {-,-,Q1} = {4}

T2 -> {-,-,-} = {5}

Группы получились одинаковые со вторым шагом — значит минимальный ДКА получен.

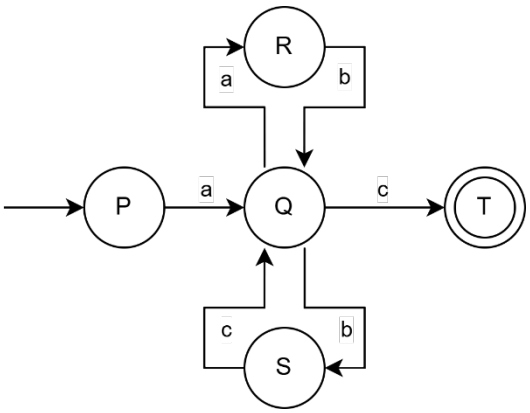


*Рисунок 3 - Минимизированный ДКА выражения a((ab)|(bc))*c*

## Программа распознаватель

```
import graphviz
from transitions import MachineError
from transitions.extensions import GraphMachine

states = [
    'P',
    'Q',
    'R',
    'S',
    'T',
]

transitions_list = [
    {'trigger': 'a', 'source': 'P', 'dest': 'Q'},

    {'trigger': 'a', 'source': 'Q', 'dest': 'R'},
    {'trigger': 'b', 'source': 'Q', 'dest': 'S'},
    {'trigger': 'c', 'source': 'Q', 'dest': 'T'},

    {'trigger': 'b', 'source': 'R', 'dest': 'Q'},
    {'trigger': 'c', 'source': 'S', 'dest': 'Q'},
]


# Simulate regex a((ab)|(bc))*c
class DFA(GraphMachine):
    def __init__(self):
        super(DFA, self).__init__(states=states,
transitions=transitions_list, initial='S', auto_transitions=False,
graph_engine='graphviz')

    def accepts(self, input_string):
        """
        Attempt to consume the input_string symbol by symbol.
```

```
        Return True if we end in W, False otherwise.
        """
        # Start state
        self.set_state('P')

        for symbol in input_string:
            if symbol not in ['a', 'b', 'c']:
                return False
            try:
                self.trigger(symbol)
            except MachineError:
                return False

        # End state
        return self.state == 'T'


if __name__ == "__main__":
    dfa = DFA()
    dfa.get_graph().draw('dfa_transitions.png', prog='dot')
```

## Заключение

В ходе выполнения лабораторной работы был успешно выполнен по заданному регулярному выражению a((ab)|(bc))∗c недетерминированный конечный автомат (НКА), который описывает структуру и логику регулярного выражения. Затем было выполнено преобразование НКА в детерминированный конечный автомат (ДКА) с использованием метода ε-замыканий и переходов между состояниями. После этого проведена минимизация ДКА путём группировки эквивалентных состояний, сохраняя функциональность автомата. На основе минимизированного ДКА была реализована программа-распознаватель на языке Python с использованием библиотеки transitions. Программа была протестирована на нескольких примерах правильных предложений.

Таким образом, в ходе выполнения лабораторной работы были изучены и реализованы основные этапы работы с конечными автоматами: построение НКА, преобразование в ДКА, минимизация ДКА, а также создание программы-распознавателя. Данный опыт позволил глубже понять принципы работы конечных автоматов и их применение в обработке регулярных языков.