Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе № 1

по курсу «Конструирование компиляторов»

на тему: «Распознавание цепочек регулярного языка»

Вариант № 6

Студент ИУ7-21М _____ Кормановский М. В.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Ступников А. А.
(Подпись, дата) (И. О. Фамилия)

2025 г.

1 Постановка задачи

Цель работы: приобретение практических навыков реализации важнейших элементов лексических анализаторов на примере распознавания цепочек регулярного языка.

Задачи работы:

1) ознакомиться с основными понятиями и определениями, лежащими в основе построения лексических анализаторов;

2) прояснить связь между регулярным множеством, регулярным выражением, праволинейным языком, конечноавтоматным языком и недетерминированным конечно-автоматным языком;

3) разработать, тестировать и отладить программу распознавания цепочек регулярного или праволинейного языка в соответствии с предложенным вариантом грамматики.

2 Задание по варианту

Напишите программу, которая в качестве входа принимает произвольное регулярное выражение, и выполняет следующие преобразования:

1) преобразует регулярное выражение непосредственно в ДКА;

2) по ДКА строит эквивалентный ему КА, имеющий наименьшее возможное количество состояний (воспользоваться алгоритмом, приведенным по адресу: https://clck.ru/3LNWXc);

3) моделирует минимальный КА для входной цепочки из терминалов исходной грамматики.

3 Текст программы

Полный текст программы приводится в Приложении.

4 Набор тестов

Набор тестов представлен в листингах 4.1–4.3.

Листинг 4.1 – Набор тестов

```
PASS tests/fsm.spec.ts
  Finite-state machine
    is built from tree for regular expression "(a*b*)*" (5 ms)
    does accept string "abababababa" for regular expression
      "(a*b*)*" (2 ms)
    is built from tree for regular expression
      "(a|b)*(aa|bb)(a|b)*" (1 ms)
    does not accept string "a" for regular expression
      "(a|b)*(aa|bb)(a|b)*" (1 ms)
    is built from tree for regular expression
      "(a|b)*(aa|bb)(a|b)*" (1 ms)
    does accept string "aabb" for regular expression
      "(a|b)*(aa|bb)(a|b)*" (1 ms)
    is built from tree for regular expression "a+b" (1 ms)
    does not accept string "b" for regular expression "a+b" (1
      ms)
    is built from tree for regular expression "a+b|b+a" (1 ms)
    does accept string "ba" for regular expression "a+b|b+a" (1
      ms)
    is built from tree for regular expression "ə" (1 ms)
    does accept string "" for regular expression "ə" (1 ms)
    is built from tree for regular expression "(ə|a)(b|ə)"
    does accept string "ab" for regular expression "(ə|a)(b|ə)"
    is built from tree for regular expression "ab+ba+" (1 ms)
    does not accept string "" for regular expression "ab+ba+"
      (1 ms)
    is built from tree for regular expression "(a+b)a" (1 ms)
    does not accept string "aaaaaaaaaabaa" for regular
      expression "(a+b)a" (1 ms)

PASS tests/min-fsm.spec.ts
  Finite-state machine minimization
    is performed on finite-state machine tree for regular
      expression (a*b*)* (2 ms)
    produces result that does accept string "abababababa" for
      regular expression "(a*b*)*"
```

Листинг 4.2 – Набор тестов (продолжение)

```
 is performed on finite-state machine tree for regular
    expression (a|b)*(aa|bb)(a|b)* (1 ms)
     produces result that does not accept string "a" for regular
        expression "(a|b)*(aa|bb)(a|b)*" (2 ms)
     is performed on finite-state machine tree for regular
        expression (a|b)*(aa|bb)(a|b)* (1 ms)
     produces result that does accept string "aabb" for regular
        expression "(a|b)*(aa|bb)(a|b)*" (1 ms)
     is performed on finite-state machine tree for regular
        expression a+b
     produces result that does not accept string "b" for regular
        expression "a+b" (1 ms)
     is performed on finite-state machine tree for regular
        expression a+b|b+a (1 ms)
     produces result that does accept string "ba" for regular
        expression "a+b|b+a"
     is performed on finite-state machine tree for regular
        expression э (1 ms)
     produces result that does accept string "" for regular
        expression "э"
     is performed on finite-state machine tree for regular
        expression (э|a)(b|э) (1 ms)
     produces result that does accept string "ab" for regular
        expression "(э|a)(b|э)" (1 ms)
     is performed on finite-state machine tree for regular
        expression ab+ba+ (1 ms)
     produces result that does not accept string "" for regular
        expression "ab+ba+"
     is performed on finite-state machine tree for regular
        expression (a+b)a
     produces result that does not accept string "aaaaaaaaaabaa"
        for regular expression "(a+b)a" (1 ms)

PASS tests/tree.spec.ts
  Syntax tree
     is correct for regular expression "a" (2 ms)
     is correct for regular expression "ab" (1 ms)
     is correct for regular expression "a+" (1 ms)
     is correct for regular expression "a*"
     is correct for regular expression "a|b" (1 ms)
```

Листинг 4.3 – Набор тестов (продолжение)

```
    is correct for regular expression "(a|b)c" (1 ms)
    is correct for regular expression "a+|b"
    is correct for regular expression "a+b+" (1 ms)
    is correct for regular expression "a|b*"
```

5 Результаты выполнения программы

Пример работы программы представлен листингах 5.1–5.2. Примеры полученного дерева разбора ДКА и минимального ДКА показаны на рисунках 5.1–5.3.

Листинг 5.1 – Пример работы программы

```
Menu:
0: exit
1: read regular expression
2: build and save syntax tree in .dot format
3: build and save determined finite-state machine in .dot format
4: minimize and save pre-built determined finite-state machine
   in .dot format
5: simulate minimized finite-state machine
Enter command:
1
Enter file path or "-" to read regex from stdin
-
Enter regex, operations supported: (), |, +, *
(a|b)*(aa|bb)(a|b)*
Read regex: "(a|b)*(aa|bb)(a|b)*"
Enter command:
3
Enter file path to save the finite-state machine or press ENTER
   to skip saving

Enter the alphabet or press ENTER to guess it from the regular
   expression

Built finite-state machine
Enter command:
4
Enter file path to save the finite-state machine or press ENTER
```

Листинг 5.2 – Пример работы программы (продолжение)

```
Minimized finite-state machine
Enter command:
5
Enter input string or press ENTER to exit
aabb
Simulation Result
Input Accepted:  true
Steps:
>>> Step #1
Current State: 1
Character just read: <NULL>
State is initial: true; state is final: false
>>> Step #2
Current State: 2
Character just read: a
State is initial: false; state is final: false
>>> Step #3
Current State: 0
Character just read: a
State is initial: false; state is final: true
>>> Step #4
Current State: 0
Character just read: b
State is initial: false; state is final: true
>>> Step #5
Current State: 0
Character just read: b
State is initial: false; state is final: true
Simulation finished
Enter input string or press ENTER to exit

Enter command:
0
```

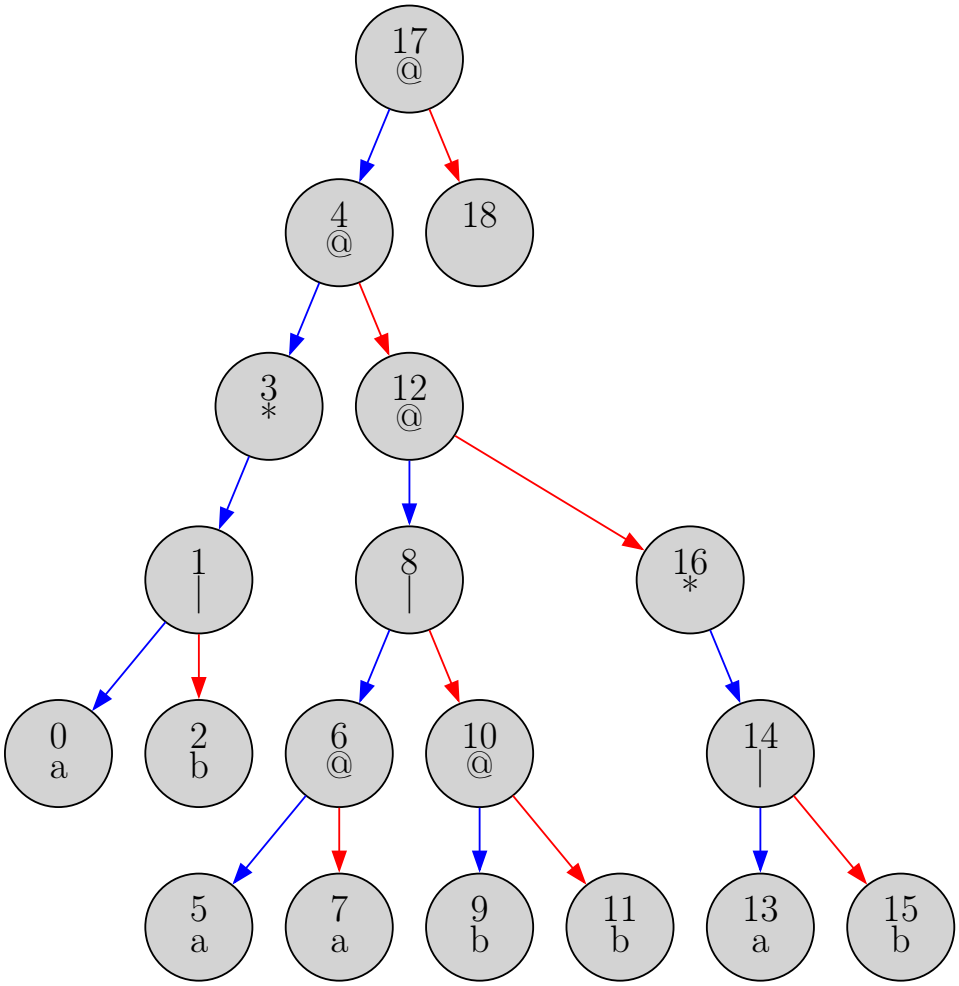

Рисунок 5.1 – Пример дерева разбора (@ обозначена конкатенация)

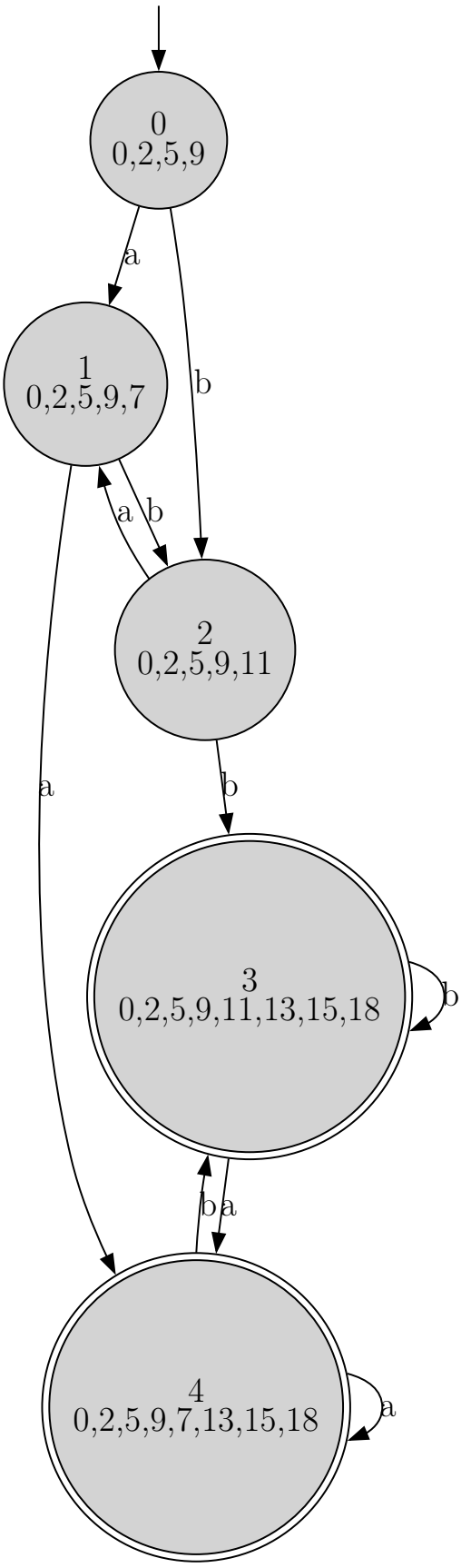

Рисунок 5.2 – Пример ДКА

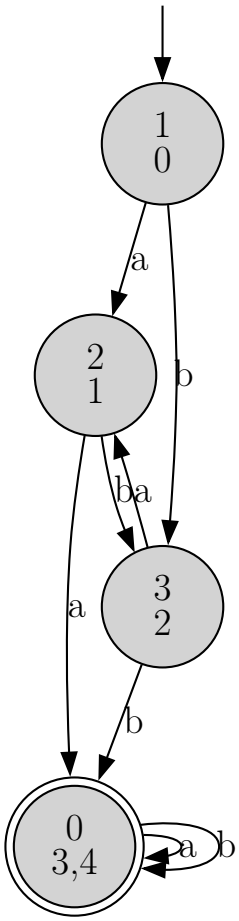

Рисунок 5.3 – Пример минимального ДКА

6 Ответы на контрольные вопросы

Ответы на контрольные вопросы приводятся в Приложении.

7 Выводы

Цель работы достингута: приобретены практические навыкои реализации важнейших элементов лексических анализаторов на примере распознавания цепочек регулярного языка.

Выполнены все задачи работы:

1) ознакомиться с основными понятиями и определениями, лежащими в основе построения лексических анализаторов;

2) прояснить связь между регулярным множеством, регулярным выражением, праволинейным языком, конечноавтоматным языком и недетерминированным конечно-автоматным языком;

3) разработать, тестировать и отладить программу распознавания цепочек регулярного или праволинейного языка в соответствии с предложенным вариантом грамматики.