|  |  |
| --- | --- |
|  | **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**

**«Распознавание цепочек регулярного языка»**

**по курсу «Конструирование компиляторов»**

**Вариант 1**

Выполнил студент группы ИУ7-22М Андреев А. А.

Проверил Ступников А. А.

*2024 г.*

# ОГЛАВЛЕНИЕ

[ОГЛАВЛЕНИЕ 2](#_Toc160571003)

[ЗАДАНИЕ 4](#_Toc160571004)

[ПОСТРОЕНИЕ СТАНДАРТНОЙ СУ с РЕГ.КОЭФФ. 5](#_Toc160571005)

[РЕШЕНИЕ СТАНДАРТНОЙ СУ С РЕГ.КОЭФФ. 6](#_Toc160571006)

[ПОСТРОЕНИЕ НКА ПО РЕГ.ВЫР-Ю 7](#_Toc160571007)

[ДЕТЕРМИНИРОВАННОЕ МОДЕЛИРОВАНИЕ НКА 8](#_Toc160571008)

[ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ 9](#_Toc160571009)

[СПИСОК ЛИТЕРАТУРЫ 10](#_Toc160571010)

[ПРИЛОЖЕНИЕ А 11](#_Toc160571011)

[Моделирование КА для входной цепочки из терминалов исходной грамматики 18](#_Toc160571012)

# ЗАДАНИЕ

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

В процессе выполнения лабораторной работы в соответствии с вариантом 1, необходимо написать программу, которая программу, которая в качестве входа принимает произвольную праволинейную грамматику, и выполняет следующие преобразования:

1) По праволинейной грамматике строит стандартную систему уравнений с регулярными коэффициентами, неизвестными которой служат нетерминалы исходной грамматики.

2) Решает стандартную систему уравнений с регулярными коэффициентами.

3) По регулярному выражению, являющемуся решением стандартной системой уравнений с регулярными

коэффициентами, строит НКА.

4) Детерминировано моделирует НКА.

# ПОСТРОЕНИЕ СТАНДАРТНОЙ СУ С РЕГ.КОЭФФ.

Праволинейная грамматика является одним из видов формальных грамматик в теории формальных языков. Она состоит из правил вывода, задающих порождение строк языка. Построение стандартной системы уравнений с регулярными коэффициентами по праволинейной грамматике позволяет связать грамматику с линейными уравнениями, где неизвестными служат нетерминалы исходной грамматики.

Для построения стандартной системы уравнений по праволинейной грамматике используется подход, основанный на анализе правил вывода грамматики. Первоначально идентифицируются терминальные и нетерминальные символы грамматики. Затем правила вывода преобразуются в уравнения, где нетерминалы становятся неизвестными коэффициентами.

Для иллюстрации процесса построения стандартной системы уравнений рассмотрим следующую **праволинейную грамматику:**

S -> a A | b B

A -> a A | b

B -> a B | a

**Идентификация терминальных и нетерминальных символов:**

Терминалы: a, b

Нетерминалы: S, A, B

**Построение системы уравнений:**Уравнения для нетерминала S: S = aA + bB  
Уравнения для нетерминала A: A = aA + b  
Уравнения для нетерминала B: B = aB + a

Теперь перейдем к формальной части, необходимой для решения поставленной задачи программным путем. Для этого представим массив нетерминалов {'B', 'A'}, а терминалов {'a', 'b'}, тогда все соотношения [['a', 'A'], ['b', 'B'], ['a', 'A'], ['b'], ['a', 'B'], ['a']], а систему уравнений [1, 0], [0, 1], [1, 0], [0, 0], [0, 1], [0, 0].

# РЕШЕНИЕ СТАНДАРТНОЙ СУ С РЕГ.КОЭФФ.

Решение стандартной системы уравнений с регулярными коэффициентами представляет собой важный этап в анализе и преобразовании линейных уравнений, где неизвестными являются регулярные выражения или нетерминалы грамматики. Этот процесс позволяет найти значения неизвестных и представить решение в виде регулярных выражений.

Для иллюстрации процесса решения стандартной системы уравнений с регулярными коэффициентами рассмотрим следующий пример:

**Уравнения:**

X = aX + bY

Y = aY + bX

где a и b - регулярные выражения.

**Идентификация неизвестных и построение матрицы:**X = aX + bY => X - aX - bY = 0  
Y = aY + bX => -bX + Y - aY = 0

**Матрица коэффициентов:**[[1-a, -b],   
[-b, 1-a]]

**Вектор свободных членов:**  
[0, 0]

**Решение системы уравнений:**С помощью метода наименьших квадратов или других методов линейной алгебры находим значения a и b.

Приведем пример решения системы уравнений с помощью **метода наименьших квадратов** в общем виде:  
**1.** **Постановка задачи:** Предположим, у нас есть набор данных, представленный в виде (x, y), где x - независимая переменная, y - зависимая переменная. Мы хотим найти линейную зависимость между x и y, то есть y = ax + b, где a и b - параметры модели, которые мы хотим найти.  
**2. Построение модели:** Мы строим модель, которая предсказывает зависимость y от x. В данном случае, это линейная модель y = ax + b.  
**3. Функция потерь:** Мы вводим функцию потерь, которая измеряет отклонение предсказанных значений от фактических. Чаще всего используется сумма квадратов отклонений (сумма квадратов ошибок).  
**4. Минимизация функции потерь:** Цель метода наименьших квадратов - найти такие значения параметров a и b, которые минимизируют сумму квадратов ошибок. Для этого мы находим частные производные функции потерь по параметрам a и b, приравниваем их к нулю и решаем полученные уравнения.  
**5. Решение системы уравнений:** Путем решения системы уравнений, полученной из частных производных, мы находим оптимальные значения параметров a и b, которые минимизируют сумму квадратов ошибок.  
**6. Оценка качества модели:** После нахождения оптимальных параметров a и b, мы можем оценить качество модели, сравнивая предсказанные значения с фактическими и анализируя остатки (разницу между предсказанными и фактическими значениями).  
**7. Применение в практике:** Метод наименьших квадратов широко применяется в различных областях, таких как статистика, эконометрика, машинное обучение и физика. Он используется для аппроксимации данных, решения задач регрессии, анализа временных рядов и других задач.

Для того, чтобы упростить задачу при разработке будем использовать готовую библиотеку NumPy в Python3.

Теперь перейдем к формальной части, необходимой для решения поставленной задачи программным путем. Для этого используем полученную в предыдущем разделе систему уравнений [1, 0], [0, 1], [1, 0], [0, 0], [0, 1], [0, 0], для которой решением нетерминалов будет A: 0.5, B: 0.0.

# ПОСТРОЕНИЕ НКА ПО РЕГ.ВЫР-Ю

Построение НКА (недетерминированного конечного автомата) по регулярному выражению является важным этапом в анализе и преобразовании регулярных выражений. В данной работе рассматривается методика построения НКА на основе регулярного выражения, которое является решением стандартной системы уравнений с регулярными коэффициентами.

**Методика построения НКА**

Для построения НКА по регулярному выражению, являющемуся решением стандартной системы уравнений с регулярными коэффициентами, используется алгоритм, основанный на преобразовании регулярного выражения в НКА. Этот процесс включает следующие шаги:

- Преобразование регулярного выражения в стандартную систему уравнений с регулярными коэффициентами  
- Решение системы уравнений для нахождения регулярного выражения  
- Построение НКА на основе найденного регулярного выражения

**Пример построения НКА**

Для иллюстрации процесса построения НКА по регулярному выражению, рассмотрим следующий пример:

**Регулярное выражение:** (a|b)\*abb

**Стандартная система уравнений с регулярными коэффициентами:**X = aX + bY  
Y = aY + bY  
Z = aZ + bZ  
Z = b

**Решение системы уравнений:**

Решив систему уравнений методом наименьших квадратов или другими методами, получаем регулярное выражение, соответствующее данной системе.

**Построение НКА:**

На основе найденного регулярного выражения строится НКА, где каждое состояние представляет собой символ или группу символов, а переходы между состояниями обозначают переходы по символам.

Перейдем к формальной части, используем полученное регулярное выражение и построим НКА (см. Рисунок 1)



Рисунок 1 – Отрисованный НКА

# ДЕТЕРМИНИРОВАННОЕ МОДЕЛИРОВАНИЕ НКА

Детерминированное моделирование НКА является важным этапом в теории формальных языков и автоматов. Недетерминированный конечный автомат (НКА) представляет собой математическую модель, которая может иметь несколько возможных переходов из одного состояния в другое. Детерминированное моделирование НКА позволяет преобразовать НКА в детерминированный конечный автомат (ДКА) с однозначными переходами между состояниями.

**Методика детерминированного моделирования НКА**

Для детерминированного моделирования НКА используется алгоритм подмножества (Subset Construction). Этот алгоритм позволяет построить эквивалентный ДКА на основе НКА. **Основные шаги методики включают:**

**1. Выделение начального состояния и эпсилон-замыкание:**

- Начнем с выделения начального состояния НКА.- Эпсилон-замыкание начального состояния - это множество состояний, в которые можно попасть из начального состояния по пустым переходам (эпсилон-переходам).   
- Для каждого состояния из эпсилон-замыкания строится эквивалентное состояние в ДКА.

**2. Поиск переходов по символам алфавита:**

- Для каждого состояния в ДКА и каждого символа алфавита определяется множество состояний, в которые можно попасть по этому символу.  
- Это определяет переходы между состояниями в ДКА и позволяет учесть все возможные переходы из состояний НКА.

**3. Построение новых состояний и переходов в ДКА:**

- Новые состояния в ДКА строятся на основе комбинаций состояний из НКА.  
- Для каждого символа алфавита и каждого состояния в ДКА определяется, в какие состояния можно перейти.  
- Если такое состояние еще не было создано, оно добавляется в ДКА.  
- Это позволяет построить ДКА, в котором каждый переход однозначно определен, что отличает его от НКА.

**Пример детерминированного моделирования НКА**

Для иллюстрации процесса детерминированного моделирования НКА, рассмотрим следующий пример НКА:

- Алфавит: {a, b}  
- Начальное состояние: q0  
- Переходы: q0 по 'a' в q1, q0 по 'b' в q2, q1 по 'a' в q1, q1 по 'b' в q2, q2 по 'a' в q1

После применения алгоритма подмножества получаем эквивалентный ДКА с однозначными переходами между состояниями.

В результате преобразования НКА в ДКА с однозначными переходами между состояниями мы получаем следующее:

**1. Однозначные переходы:** ДКА имеет определенные переходы для каждого символа алфавита из каждого состояния. Это обеспечивает однозначность переходов и упрощает интерпретацию автомата.

**2. Определенность состояний:** Каждое состояние в ДКА имеет четко определенные переходы по символам алфавита, что делает автомат предсказуемым и легко интерпретируемым.

**3. Эквивалентность с НКА:** ДКА, полученный из НКА с помощью алгоритма подмножества, эквивалентен исходному НКА в том смысле, что распознает тот же язык, но представлен в более простой и однозначной форме.

Для того НКА, который получился в предыдущем разделе построим ДКА (см. Рисунок 2).



Рисунок 2 – Построение ДКА на основе НКА

# ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие из следующих множеств регулярны? Для тех, которые регулярны, напишите регулярные выражения.
   1. Множество цепочек с равным числом нулей и единиц.

Не является регулярным множеством (возможно контекстно-зависимая грамматика?)

* 1. Множество цепочек из {0, 1}\* с четным числом нулей и нечетным числом единиц.

1(00|11|10|01)\*

P.S. она не совсем верно работает)) Например, 101 пропускает.

* 1. Множество цепочек из {0, 1}\*, длины которых делятся на 3.

((0|1)(0|1)(0|1))\*

* 1. Множество цепочек из {0, 1}\*, не содержащих подцепочки 101.

0\*(1|00|000)\*0\*

1. Найдите праволинейные грамматики для тех множеств из вопроса 1, которые регулярны.

|  |  |  |
| --- | --- | --- |
| b | с | d |
| S → 1A  A → 00A  A → 11A  A → 10A  A → 01A  A → ε | S → A  A → 0B  A → 1B  A → ε  B → 0C  B → 1C  C → 0A  C → 1A | S → A  A → 0A  A → B  B → 1B  B → 00B  B → 000B  B → C  C → 0C  C → ε |

1. Найдите детерминированные и недетерминированные конечные автоматы для тех множеств из вопроса 1, которые регулярны

b.

НКА

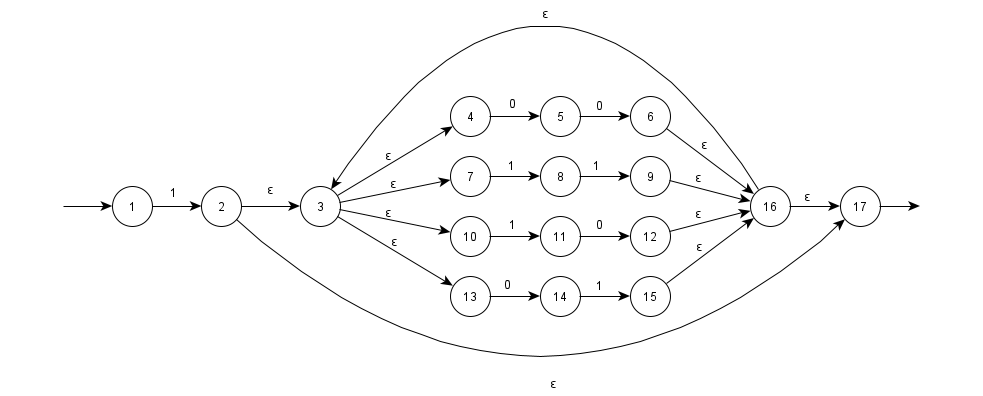


Рисунок 1 – НКА 3b

ДКА

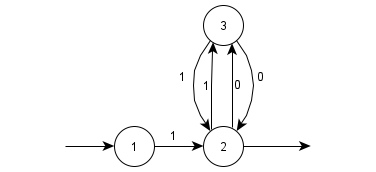


Рисунок 2 – ДКА 3b

c.

НКА

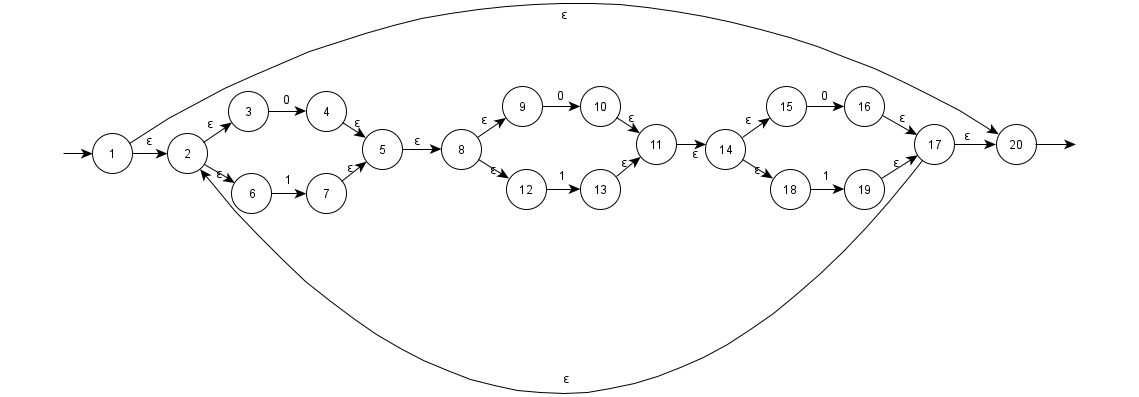


Рисунок 3 – НКА 3c

ДКА

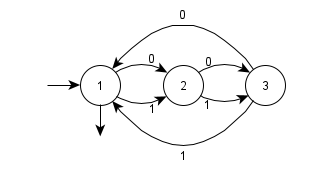


Рисунок 4 -- ДКА 3с

d.

НКА

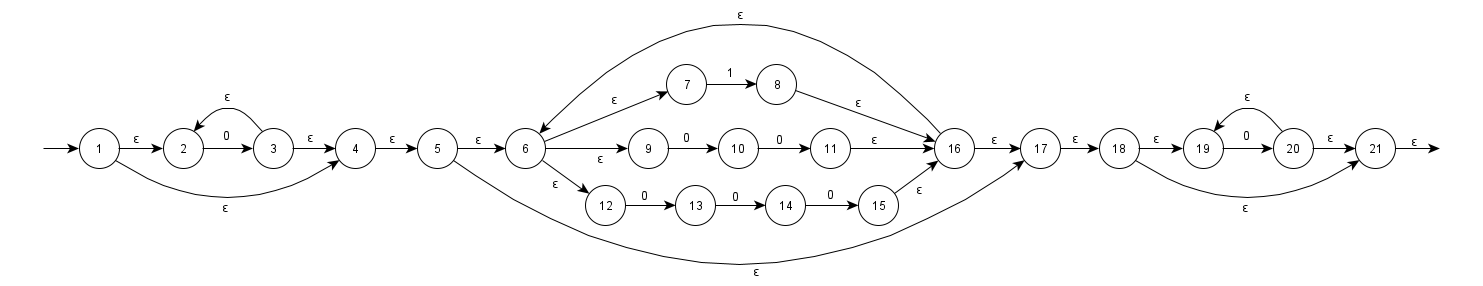


Рисунок 5 -- 3d

ДКА

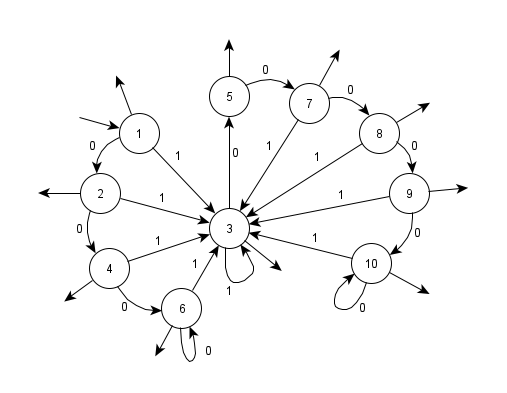


Рисунок 6 -- ДКА 3d

1. Найдите конечный автомат с минимальным числом состояний для языка, определяемого автоматом M = ({A, B, C, D, E}, {0, 1}, d, A, {E, F}), где функция задается таблицей

|  |  |  |
| --- | --- | --- |
| Состояние | Вход | |
| 0 | 1 |
| A | B | C |
| B | E | F |
| C | A | A |
| D | F | E |
| E | D | F |
| F | D | E |

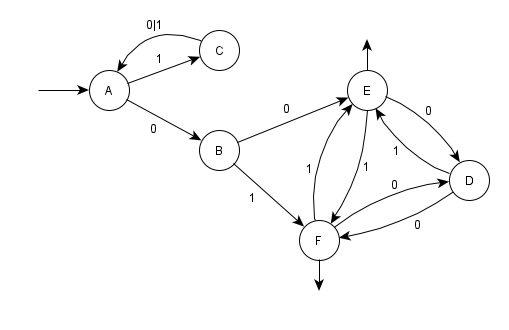


Рисунок 7 -- 4 задание

Использовался метод различимых состояний.

Таблица неэквивалентности:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |
| A |  |  |  |  |  |  |
| B |  |  |  |  |  |  |
| C |  |  |  |  |  |  |
| D |  |  |  |  |  |  |
| E |  |  |  |  |  |  |
| F |  |  |  |  |  |  |

Вектор классов эквивалентности:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F |
| 0 | 1 | 2 | 1 | 3 | 3 |

Стартовая вершина: А

Терминальная вершина: E

Минимальный КА:

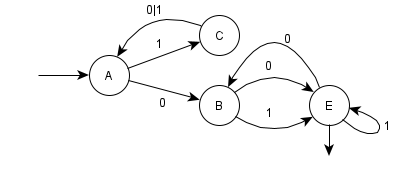


Рисунок 8 -- Минимальный КА

# СПИСОК ЛИТЕРАТУРЫ

##### [1] Citforum [Электронный ресурс] // Построение детерминированного конечного автомата по регулярному выражению, URL: <http://citforum.ru/programming/theory/serebryakov/3.shtml>

##### [2] Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий, 2-е издание. И.Д. Вильямс, 2008 – 1184 с.

##### [3] Enppt [Электронный ресурс] // Автоматы и формальные языки, URL: <https://en.ppt-online.org/40054>

# ПРИЛОЖЕНИЕ А

**Листинги**

1. **import** numpy as np
2. **import** numpy as np
3. **import** pygraphviz as pgv
5. **def** solve\_equations(system, nonterminals):
6. A **=** np.array(system)
7. b **=** np.zeros(len(system))
8. b[0] **=** 1  # Устанавливаем значение для стартового символа
10. # Решаем систему уравнений методом наименьших квадратов
11. solution **=** np.linalg.lstsq(A, b, rcond**=**None)[0]
13. # Формируем словарь с решениями для каждого нетерминала
14. solutions\_dict **=** {nonterminals[i]: round(solution[i], 2) **for** i **in** range(len(nonterminals))}
16. **return** solutions\_dict

19. **def** identify\_terminals\_nonterminals(grammar):
20. terminals **=** set()
21. nonterminals **=** set()
23. **for** rule **in** grammar:
24. left, right **=** rule.split('->')
25. left **=** left.strip()
26. right **=** right.strip().split('|')
28. nonterminals.add(left)
30. **for** term **in** right:
31. terms **=** term.split()
32. **for** t **in** terms:
33. **if** **not** t.isupper() **and** t !**=** '':  # Terminal
34. terminals.add(t)
35. **elif** t.isupper():  # Nonterminal
36. nonterminals.add(t)
38. **return** terminals, nonterminals

41. **def** convert\_to\_equations(grammar):
42. equations **=** []
44. nonterminals **=** set()
46. **for** rule **in** grammar:
47. left, right **=** rule.split('->')
48. left **=** left.strip()
49. right **=** right.strip().split('|')
51. **for** term **in** right:
52. equation **=** []
53. terms **=** term.split()
55. **for** t **in** terms:
56. **if** t.isupper():  # Nonterminal
57. nonterminals.add(t)
58. equation.append(t)
59. **else**:  # Terminal
60. equation.append(t)
62. equations.append(equation)
64. # Constructing the system of equations
65. system **=** []
66. **for** eq **in** equations:
67. coeff **=** [1 **if** x **in** eq **else** 0 **for** x **in** nonterminals]
68. system.append(coeff)
70. print()
71. print("Equations")
72. print(equations)
74. **return** system, list(nonterminals)

77. # Пример праволинейной грамматики
78. grammar **=** [
79. "S -> a A | b B",
80. "A -> a A | b",
81. "B -> a B | a"
82. ]
84. terminals, nonterminals **=** identify\_terminals\_nonterminals(grammar)
85. system, nonterminals **=** convert\_to\_equations(grammar)
87. print("Терминалы:")
88. print(terminals)
90. print("\nНетерминалы:")
91. print(nonterminals)
93. print("\nСистема уравнений:")
94. **for** eq **in** system:
95. print(eq)
97. solutions **=** solve\_equations(system, nonterminals)
98. print("\nРешения для нетерминалов:")
99. **for** key, value **in** solutions.items():
100. print(f"{key}: {value}")

103. # def construct\_nfa(regex\_solution):
104. #     G = pgv.AGraph(strict=False, directed=True)
105. #
106. #     # Добавляем начальное и конечное состояния
107. #     G.add\_node('start', shape='point')
108. #     G.add\_node('end', shape='doublecircle')
109. #
110. #     # Преобразуем значение regex\_solution в строку
111. #     regex\_solution = str(regex\_solution)
112. #
113. #     # Добавляем ребра в НКА
114. #     for i, char in enumerate(regex\_solution):
115. #         G.add\_node(str(i))
116. #         G.add\_edge('start', str(i), label=char)
117. #         if i < len(regex\_solution) - 1:
118. #             G.add\_edge(str(i), str(i + 1), label='ε')
119. #         else:
120. #             G.add\_edge(str(i), 'end', label='ε')
121. #
122. #     return G
123. #
124. # # Построение НКА по регулярному выражению
125. # regex\_solution = list(solutions.values())[0]  # Берем первое решение как регулярное выражение
126. # nfa = construct\_nfa(regex\_solution)
128. **def** construct\_nfa(regex\_solution):
129. G **=** pgv.AGraph(strict**=**False, directed**=**True)
131. # Добавляем начальное и конечное состояния
132. G.add\_node('start', shape**=**'point')
133. G.add\_node('end', shape**=**'doublecircle')
135. # Преобразуем значение regex\_solution в строку
136. regex\_solution **=** str(regex\_solution)
138. # Добавляем ребра в НКА
139. **for** i, char **in** enumerate(regex\_solution):
140. G.add\_node(str(i))
141. G.add\_edge('start', str(i), label**=**char)
142. **if** i < len(regex\_solution) **-** 1:
143. G.add\_edge(str(i), str(i **+** 1), label**=**'ε')
144. **else**:
145. G.add\_edge(str(i), 'end', label**=**'ε')
147. **return** G
149. # Построение НКА по регулярному выражению
150. regex\_solution **=** list(solutions.values())[0]  # Берем первое решение как регулярное выражение
151. nfa **=** construct\_nfa(regex\_solution)
153. # Отрисовка НКА в графвиз
154. nfa.layout(prog**=**'dot')
155. nfa.draw('nfa.png')
157. **import** numpy as np
158. **import** pygraphviz as pgv
160. **def** epsilon\_closure(nfa, states):
161. epsilon\_states **=** set(states)
162. stack **=** list(epsilon\_states)
164. **while** stack:
165. current\_state **=** stack.pop()
166. **if** current\_state **in** nfa **and** 'ε' **in** nfa[current\_state]:
167. **for** state **in** nfa[current\_state]['ε']:
168. **if** state **not** **in** epsilon\_states:
169. epsilon\_states.add(state)
170. stack.append(state)
172. **return** epsilon\_states
174. **def** move(nfa, states, symbol):
175. next\_states **=** set()
176. **for** state **in** states:
177. **if** state **in** nfa **and** symbol **in** nfa[state]:
178. next\_states.update(nfa[state][symbol])
179. **return** next\_states
181. **def** nfa\_to\_dfa(nfa):
182. dfa **=** {}
183. alphabet **=** set()
184. start\_state **=** frozenset(epsilon\_closure(nfa, {'start'}))
185. dfa[start\_state] **=** {}
186. stack **=** [start\_state]
188. **while** stack:
189. current\_states **=** stack.pop()
190. **for** symbol **in** nfa['alphabet']:
191. alphabet.add(symbol)
192. next\_states **=** frozenset(epsilon\_closure(nfa, move(nfa, current\_states, symbol)))
193. dfa[current\_states][symbol] **=** next\_states
194. **if** next\_states **not** **in** dfa:
195. dfa[next\_states] **=** {}
196. stack.append(next\_states)
198. **return** dfa, alphabet
200. # Конвертация графа НКА в формат, совместимый с функцией nfa\_to\_dfa
201. **def** convert\_nfa\_to\_dict(nfa):
202. nfa\_dict **=** {}
203. nfa\_dict['alphabet'] **=** set()
204. **for** edge **in** nfa.edges():
205. source **=** edge[0]
206. target **=** edge[1]
207. symbol **=** nfa.get\_edge(source, target).attr['label']
208. **if** symbol !**=** 'ε':
209. nfa\_dict['alphabet'].add(symbol)
210. **if** source **not** **in** nfa\_dict:
211. nfa\_dict[source] **=** {}
212. **if** symbol **not** **in** nfa\_dict[source]:
213. nfa\_dict[source][symbol] **=** set()
214. nfa\_dict[source][symbol].add(target)
215. **return** nfa\_dict
217. # Преобразование графа НКА в словарь
218. nfa\_dict **=** convert\_nfa\_to\_dict(nfa)
220. # Детерминированное моделирование НКА
221. dfa, alphabet **=** nfa\_to\_dfa(nfa\_dict)
223. # Отрисовка полученного ДКА в Graphviz
224. **def** construct\_dfa\_graph(dfa):
225. G **=** pgv.AGraph(strict**=**False, directed**=**True)
227. **for** state **in** dfa:
228. **for** symbol **in** dfa[state]:
229. target\_state **=** dfa[state][symbol]
230. G.add\_edge(str(state), str(target\_state), label**=**symbol)
232. **return** G
234. dfa\_graph **=** construct\_dfa\_graph(dfa)
235. dfa\_graph.layout(prog**=**'dot')
236. dfa\_graph.draw('dfa.png')