

Аметов К.Л. ПИ-231(2)

Задание: Минимизировать ДКА Мили, заданный таблицей выходов и переходов.

Построить эквивалентный автомат 2 рода. Провести проверку в JFLAP

№ 1

A \ Q	1	2	3	4	5	6	7	8	9
a	4,x	3,y	7,x	5,x	8,x	7,x	2,y	5,x	2,y
b	7,y	4,y	9,x	9,y	7,y	9,y	8,x	9,y	5,x

Построим множества π_1 эквивалентных состояний: это состояния, не различимые никаким однобуквенным воздействием: $\pi_1 = \{A, B, C, D\}$, $Ax = \{1, 4, 5, 6, 8\}$ $Bx = \{9, 7\}$ $Cx = \{3\}$ $Dy = \{2\}$.

Индексами для удобства обозначены реакции на воздействия a, b соответственно.

Класс Axu был расщеплен, процедура была завершена за два шага.

В самом низу картинки представлен эквивалентный гомоморфный нормализованный автомат мили с прежними обозначениями.

A \ Q	1	2	3	4	5	6	7	8	9
a	4x	3y	7x	5x	8x	7x	2y	5x	2y
b	7y	4y	9x	9y	7y	9y	8x	9y	5x

Ax	By	Cx	Dy
1	4	5	6
8	9	7	3
2	A	B	C
B	A	B	A

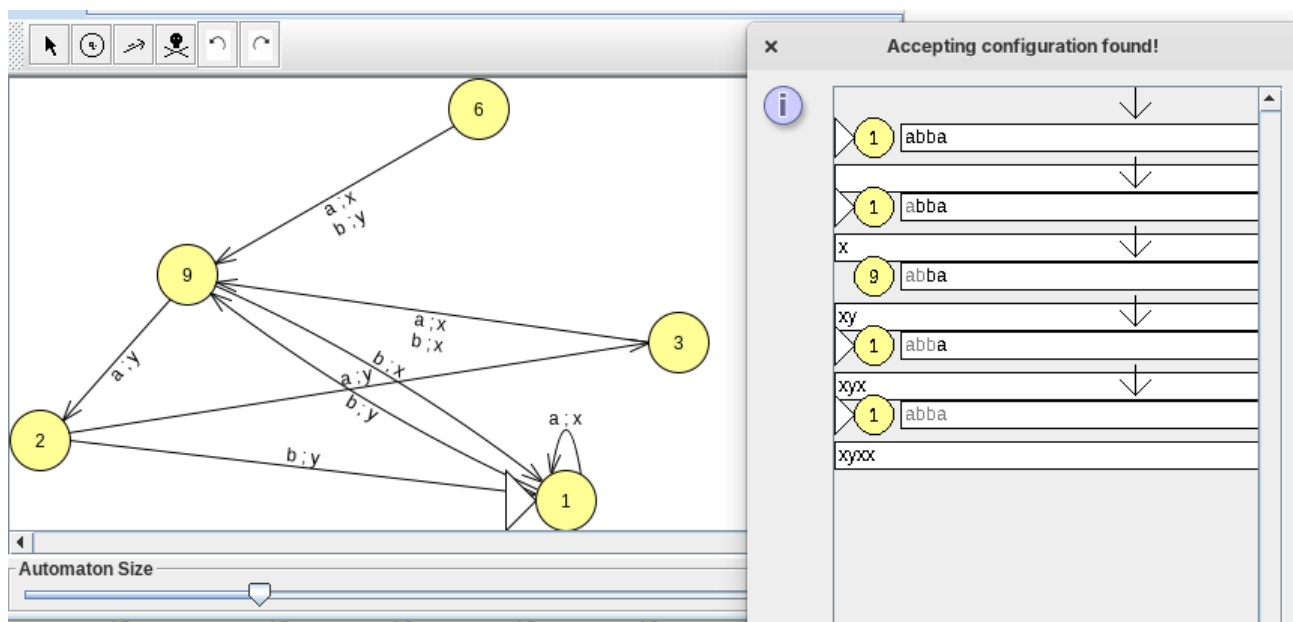
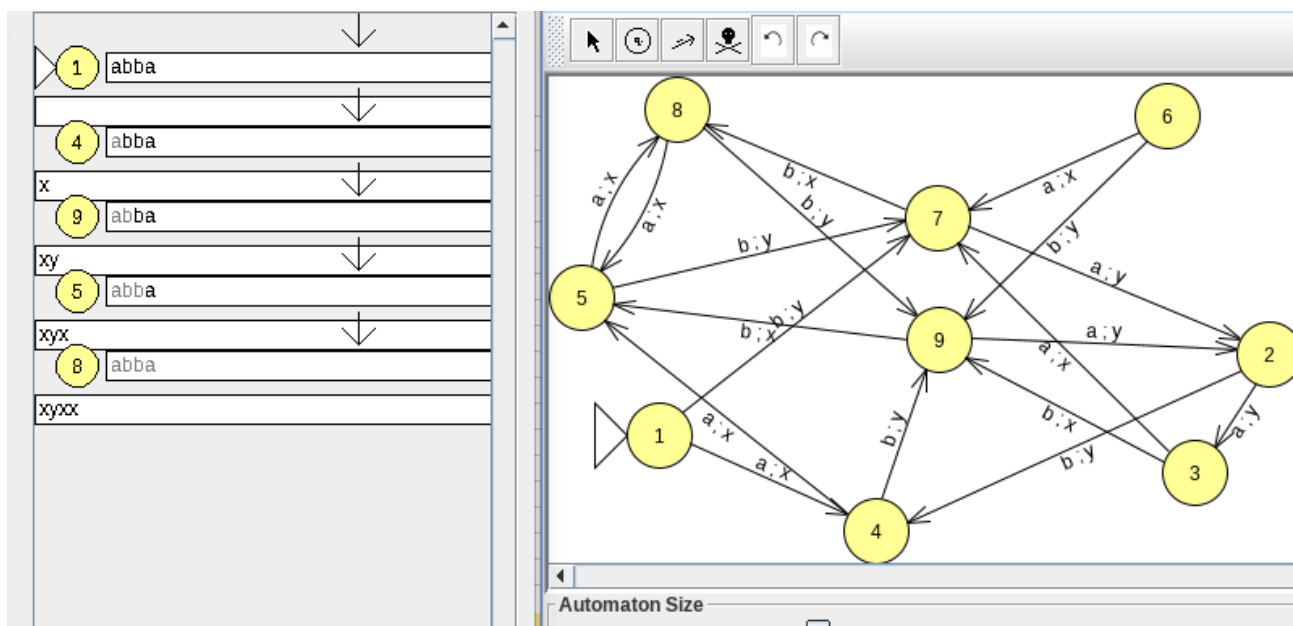
B0	B1	B2	B3	B4
1	4	5	8	9
7	3	2	6	B3
B0	B0	B0	B0	B0
B1	B1	B1	B1	B1

B0	B1	B2	B3	B4
1	9	3	2	6
B0	B3	B1	B2	B1
B1	B0	B1	B0	B1

A \ Q	1	2	3	4	5	6
a	1/x	2/y	9x	3y	9x	9y
b	9/y	1/x	9x	1y	9y	

Убедимся на частном примере, что автоматные преобразования двух машин одинаковы.

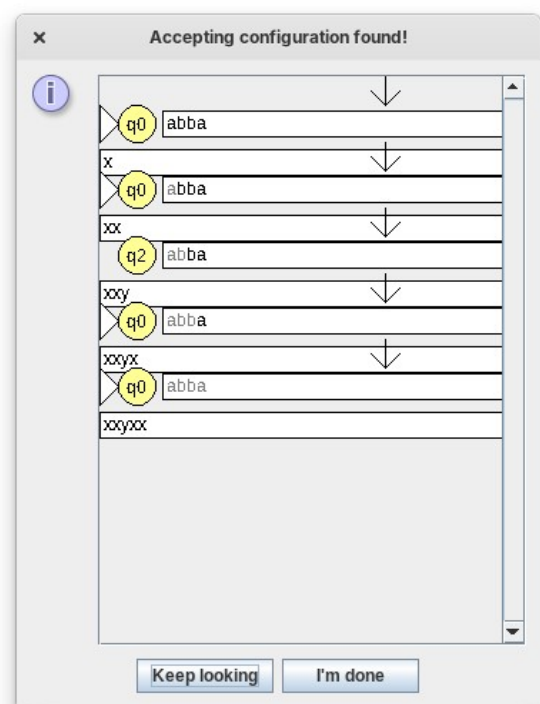
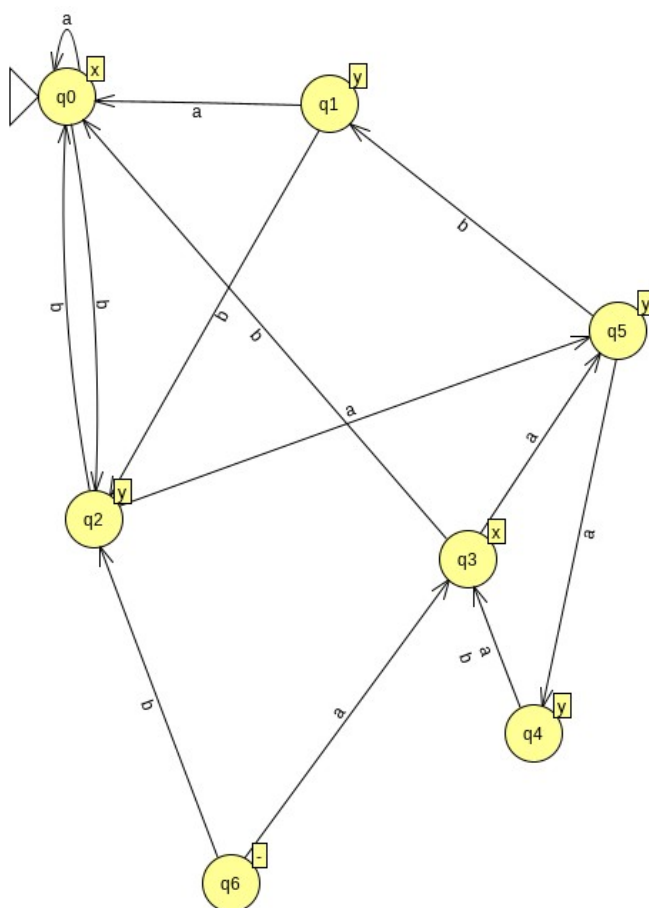
Выберем в исходном автомате в качестве начального состояния 4 и подадим на его вход цепочку "abba". Так же проверим нормализованный автомат.



Перейдем к автомату 2 рода(Мура), построю его и покажу в табличном виде

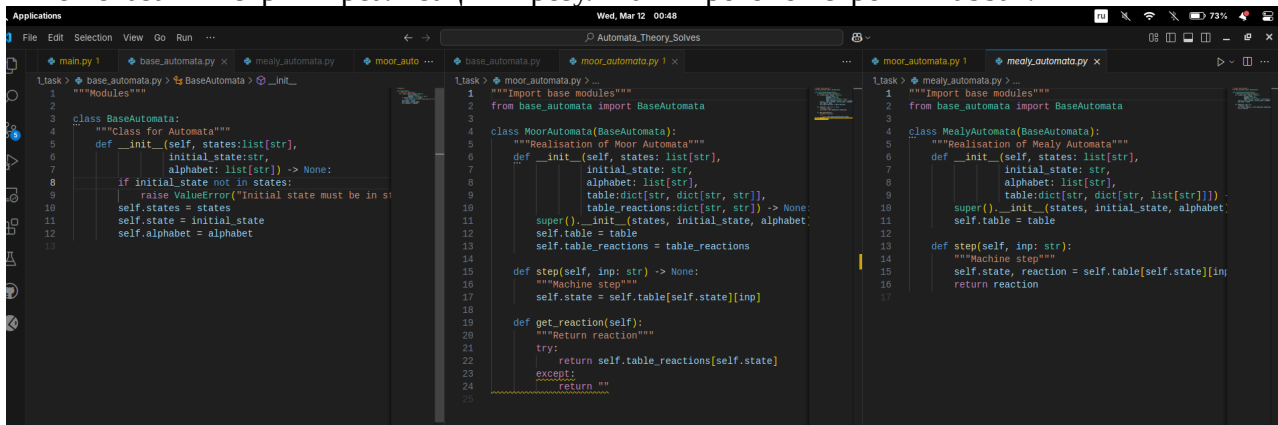
	A	B	C	D	E	F	G	H
A/Q	1	1	9	3	2	6		
a	1/x	1/x	2/y	9x	3y	9x		
b	9/y	9/y	1/x	9x	1y	9y		
Состояния	Реак при дост	Реал						
1	1/x 1/y	A0/x A1/y						
9	9/y 9x	A2/y A3/x						
3	3/y	A4/y						
2	2/y	A5/y						
6	-	A6/-						
	A0	A1	A2	A3	A4	A5	A6	
A/Q	x	y	y	x	y	y	-	
a	A0	A0	A5	A5	A3	A4	A3	
b	A2	A2	A0	A0	A3	A1	A2	

Проверю автомат мура



P.S

Так же была выполнена программная реализация автомата Мура и Мили, построение эквивалентного автомату Мили автомата мура и минимизация автомата на ЯП Python. Ниже показанны скрины реализации и результаты прогонов строки “abba”.



```
1 task > base_automata.py > BaseAutomata > __init__
2
3 class BaseAutomata:
4     """Class for Automata"""
5     def __init__(self, states:list[str],
6                 initial_state:str,
7                 alphabet: list[str]) -> None:
8         if initial_state not in states:
9             raise ValueError("Initial state must be in s
10         self.states = states
11         self.state = initial_state
12         self.alphabet = alphabet
13
1 task > moor_automata.py > ...
1
2 """Import base modules"""
3 from base_automata import BaseAutomata
4
5 class MoorAutomata(BaseAutomata):
6     """Realisation of Moor Automata"""
7     def __init__(self, states: list[str],
8                 initial_state: str,
9                 alphabet: list[str],
10                 table:dict[str, dict[str, str]],
11                 table_reactions:dict[str, str]) -> None:
12         super().__init__(states, initial_state, alphabet
13         self.table = table
14         self.table_reactions = table_reactions
15
16 def step(self, inp: str) -> None:
17     """Machine step"""
18     self.state = self.table[self.state][inp]
19
20 def get_reaction(self):
21     """Return reaction"""
22     try:
23         return self.table_reactions[self.state]
24     except:
25         return ""
26
1 task > mealy_automata.py > ...
1
2 """Import base modules"""
3 from base_automata import BaseAutomata
4
5 class MealyAutomata(BaseAutomata):
6     """Realisation of Mealy Automata"""
7     def __init__(self, states: list[str],
8                 initial_state: str,
9                 alphabet: list[str],
10                 table:dict[str, dict[str, list[str]]]) -
11         super().__init__(states, initial_state, alphabet
12         self.table = table
13
14 def step(self, inp: str):
15     """Machine step"""
16     self.state, reaction = self.table[self.state][inp]
17     return reaction
```

Реализация построения эквивалентного автомату Мили, автомата Мура

```

1_task > transformation.py > mealy_to_moor
1  """Modules"""
2  from collections import defaultdict
3  from moor_automata import MoorAutomata
4  from mealy_automata import MealyAutomata
5
6  def mealy_to_moor(automata: MealyAutomata) -> MoorAutomata:
7      """Transforms a Mealy automaton into a Moore automaton."""
8      state_reactions = defaultdict(set)
9      for state in automata.states:
10         for symbol in automata.alphabet:
11             next_state, reaction = automata.table[state][symbol]
12             state_reactions[next_state].add(reaction)
13
14     new_states = {}
15     reaction_table = {}
16     old_to_new_state = {}
17     state_counter = 0
18
19     for state in automata.states:
20         if not state_reactions[state]:
21             state_reactions[state].add("") # Для состояния без реакций
22
23         for reaction in state_reactions[state]:
24             new_state = f"A{state_counter}"
25             new_states[(state, reaction)] = new_state
26             reaction_table[new_state] = reaction
27             old_to_new_state[state] = new_state
28             state_counter += 1
29
30     transition_table = defaultdict(dict)
31     for new_state, old_state in new_states.items():
32         # print(1, new_state, old_state)
33         for symbol in automata.alphabet:
34             next_old_state, reaction = automata.table[old_state][symbol]
35             transition_table[new_state][symbol] = new_states[(next_old_state, reaction)]
36
37     return MoorAutomata(
38         states=transition_table.keys(),
39         initial_state=old_to_new_state[automata.state],
40         alphabet=automata.alphabet,
41         table=transition_table,
42         table_reactions=reaction_table
43     )
44

```

Реализация минимизации

```

1  """Modules"""
2  from collections import defaultdict
3  from copy import deepcopy
4  from mealy_automata import MealyAutomata
5  from moor_automata import MoorAutomata
6  from transformation import mealy_to_moor
7
8  def pi_minimization(automata_: MealyAutomata):
9      """Pi-minimization"""
10     automata = deepcopy(automata_)
11     # Находим таблицу реакций и переходов, а так же все уникальные столбцы реакций
12     table_trans = {s:{a_n:i[0] for a_n, i in a.items()} for s,a in automata.table.items()}
13     table_react = {s:{a_n:i[1] for a_n, i in a.items()} for s,a in automata.table.items()}
14     unikal_reactions = list(set(tuple(j for _,j in a.items()) for _, a in table_react.items()))
15
16     # Разделяем состояния по реакциям
17     reactions = defaultdict(dict)
18     reactions_ = {}
19     for s, a in deepcopy(table_react).items():
20         try:
21             ind = unikal_reactions.index(tuple(react for _,react in a.items()))
22             reactions[ind][s] = a
23             reactions_[s] = ind
24         except:
25             continue
26
27     # Начнем расщепление и минимизацию
28     ind_char = 65
29     while True:
30         temp_reactions = defaultdict(dict)
31         temp_reactions_ = {}
32         for _, group in reactions.items():
33             # Теперь разделяем по переходам
34             for s, a in group.items():
35                 for inp in a:
36                     group[s][inp] = reactions[table_trans[s][inp]]
37             unikal_reactions = list(set(tuple(j for _,j in a.items()) for _, a in group.items()))
38
39             for s, a in group.items():
40                 try:
41                     ind = unikal_reactions.index(tuple(react for _,react in a.items()))
42                     temp_reactions[f"{chr(ind_char)}{ind}"][s] = a
43                     temp_reactions_[s] = f"{chr(ind_char)}{ind}"
44                 except:
45                     continue
46             ind_char+=1
47             if len(temp_reactions.keys()) == len(reactions.keys()):
48                 break
49             reactions = temp_reactions
50             reactions_ = temp_reactions_
51
52     # Возвращаем первоначальные состояния и убираем эквивалентные
53     dt = {}
54     dt_ = {}
55     for s, a in reactions.items():
56         k = list(a.keys())[0]
57         dt[k] = a[k]
58         dt_[s] = k
59     for s,a in dt.items():
60         for inp,tran in a.items():
61             dt[s][inp] = [dt_[tran],table_react[s][inp]]
62     automata.states = list(dt.keys())
63     automata.state = automata.states[0]
64     automata.table = dt
65     return automata

```

Прогон строки “abba” на исходном, нормализованном и эквивалентном автомате второго рода.

```

67 table = {
68     "1": {"a": ["4", "x"], "b": ["7", "y"]},
69     "2": {"a": ["3", "y"], "b": ["4", "y"]},
70     "3": {"a": ["7", "x"], "b": ["9", "x"]},
71     "4": {"a": ["5", "x"], "b": ["9", "y"]},
72     "5": {"a": ["8", "x"], "b": ["7", "y"]},
73     "6": {"a": ["7", "x"], "b": ["9", "y"]},
74     "7": {"a": ["2", "y"], "b": ["8", "x"]},
75     "8": {"a": ["5", "x"], "b": ["9", "y"]},
76     "9": {"a": ["2", "y"], "b": ["5", "x"]}
77 }
78
79
80 # Список состояний
81 states = list(table.keys())
82
83 # Начальное состояние
84 initial_state = "8"
85
86 # Алфавит входных символов
87 alphabet = ["a", "b"]
88
89 automata = MealyAutomata(states, initial_state, alphabet, table)
90 print("Результат работы над строкой 'abba' автомата Мили "+" ".join([automata.step(i) for i in "abba"]))
91
92 automata_minim = pi_minimization(automata)
93 print("Результат работы над строкой 'abba' нормализованного автомата Мили "+" ".join([automata.step(i) for i in "abba"]))
94
95 res = ""
96 moor = mealy_to_moor(automata)
97 for i in "abba":
98     moor.step(i)
99     res += moor.get_reaction()
100 print(f"Результат работы над строкой 'abba' автомата Мура {res}")

```

PROBLEMS 47 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE COMMENTS

```

Результат работы над строкой 'abba' автомата Мили хухх
Результат работы над строкой 'abba' нормализованного автомата Мили хухх
Результат работы над строкой 'abba' автомата Мура хухх
© (.venv) kemran@kemran:~/Automata_Theory_Solves$

```

Кроме того был написан перевод машины из пайтон в файл jflap файл


```

1  """Modules"""
2  import xml.etree.ElementTree as ET
3  from copy import deepcopy
4  from moor_automata import MoorAutomata
5
6  def convert_moor_to_jflap(automata: MoorAutomata, filename: str) -> None:
7      """Convert Moor automata to jflap file"""
8      automata_ = deepcopy(automata)
9      states = automata_.table_reactions
10     transitions = automata_.table
11     initial_state = automata.state
12     root = ET.Element("structure")
13     ET.SubElement(root, "type").text = "moore"
14     automaton = ET.SubElement(root, "automaton")
15
16     state_map = {state: str(i) for i, state in enumerate(states.keys())}
17
18     for state, output in states.items():
19         state_elem = ET.SubElement(automaton, "state", id=state_map[state], name=state)
20         # ET.SubElement(state_elem, "x").text = str(100 + int(state[1]) * 50)
21         # ET.SubElement(state_elem, "y").text = str(100 + int(state[1]) * 50)
22         ET.SubElement(state_elem, "output").text = output
23         if state == initial_state:
24             ET.SubElement(state_elem, "initial")
25
26     for from_state, paths in transitions.items():
27         for symbol, to_state in paths.items():
28             trans_elem = ET.SubElement(automaton, "transition")
29             ET.SubElement(trans_elem, "from").text = state_map[from_state]
30             ET.SubElement(trans_elem, "to").text = state_map[to_state]
31             ET.SubElement(trans_elem, "read").text = symbol
32             ET.SubElement(trans_elem, "transout").text = states[to_state]
33
34     tree = ET.ElementTree(root)
35     tree.write(f"{filename}.jff", encoding="utf-8", xml_declaration=True)
36

```



```

1  """Modules"""
2  import xml.etree.ElementTree as ET
3  from copy import deepcopy
4  from mealy_automata import MealyAutomata
5
6
7  def convert_mealy_to_jflap(automata: MealyAutomata, filename="mealy_machine.jff") -> None:
8      """Convert Mealy automata to jflap file"""
9      automata_ = deepcopy(automata)
10     table = automata_.table
11     start_state = automata_.state
12     structure = ET.Element("structure")
13     etype = ET.SubElement(structure, "type")
14     etype.text = "mealy"
15     automaton = ET.SubElement(structure, "automaton")
16
17     # Создание состояний
18     states = {}
19     for state in table.keys():
20         state_elem = ET.SubElement(automaton, "state", id=state, name=state)
21         if state == start_state:
22             ET.SubElement(state_elem, "initial")
23         states[state] = state_elem
24
25     # Создание переходов
26     for from_state, transitions in table.items():
27         for symbol, (to_state, output) in transitions.items():
28             transition = ET.SubElement(automaton, "transition")
29             ET.SubElement(transition, "from").text = from_state
30             ET.SubElement(transition, "to").text = to_state
31             ET.SubElement(transition, "read").text = symbol
32             ET.SubElement(transition, "transout").text = output
33
34     tree = ET.ElementTree(structure)
35     tree.write(f"{filename}.jff", encoding="utf-8", xml_declaration=True)

```

Вывод: выполнена минимизация автомата Мили, построен эквивалентный нормализованный автомат Мура; автоматные преобразования для тестовой четырех-символьной цепочки совпадают.