Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних інформаційних систем

Алгебро-автоматичні методи проектування програмного забезпечення

Лабораторна робота 2

“Реалізація алгоритму побудови автомата Мура за автоматом Мілі(еквівалентного)”

Виконали студенти 1-го курсу

Групи ПЗС-1

Богатько Олександр Геннадійович

Юзина Сергій Сергійович

Полосенко Павло Олегович

2023

Мета: Реалізувати алгоритм побудови автомата Мура за автоматом Мілі(еквівалентного)

Псевдокод

```
def convert_to_moore(self):
    # Ініціалізація змінних
    # Отримання перехідних станів
    # Отримання нового початкового стану (якщо необхідно)
    # Обробка перехідних станів і заповнення таблиці переходів
    # Заповнення таблиці виходів
    # Створення нового Мур автомата
    # Виведення нового Мур автомата
    print(moore_from_mealy)
```

Код програми:

https://github.com/sbohatko/MastersLabs/blob/main/algebra-automatic-software-design-methods/lab2/lab2.py

Результат виконання програми:

```
alexbogatko@MacBook-Pro-Alex ~/D/M/a/lab2 (main)> python3 lab2.py

Mealy Machine
States ['a', 'b']
Transitions {'a': {'x': ('b', '0'), 'y': ('a', '1')}, 'b': {'x': ('a', '0'), 'y': ('b', '1')}}
Inital State a
Inital Alphabet ['x', 'y']
Output Alphabet['0', '1']

Moore Machine
States = ['b0', 'b1', 'a1', 'a0']
Input Alphabet = ['x', 'y']
Output Alphabet = ['0', '1']
Transitions = {'b0': {'x': 'a0', 'y': 'b1'}, 'b1': {'x': 'a0', 'y': 'b1'}, 'a1': {'x': 'b0', 'y': 'a1'}, 'a0': {'x': 'b0', '
y': 'a1'}}
Initial State = {'a0': '0', 'b1': '1', 'b0': '0', 'a1': '1'}
Output Table = a0
```

Пояснення

Цей код реалізує алгоритм побудови автомата Мура за автоматом Мілі(еквівалентного)

Розглянемо найважливіші частини коду алгоритма побудови автомата Мура:

1. **Ініціалізація змінних**

```
moore_transitions = {}
temp_list = []
moore_output_table = {}
moore_initial_state = self.initial_state
```

moore_transitions: Це словник, в якому ключами є стани, а значеннями є словники, що містять переходи для кожної букви вхідного алфавіту.
temp_list: Це тимчасовий список, до якого будуть додаватися переходи для подальшого аналізу.
moore_output_table: Це словник, в якому ключами є стани, а значеннями - вихідні символи для кожного стану.
moore_initial_state: Це початковий стан нового Мур автомата. В початку він встановлюється таким самим, як у вхідного Мілі автомата.

2. **Отримання перехідних станів**

```
for x in self.transitions.keys():
	for a in self.input_alphabet:
		temp_list.append(self.transitions[x][a])
```

В цьому блоку коду ми проходимо по всіх станах у вхідному Мілі автоматі і для кожного стану та вхідного символу додаєте перехід до **temp_list**.

3. **Обробка перехідних станів та заповнення таблиці переходів**

```
for x in temp_list:
	for y in temp_list:
		if x[0] == y[0] and x[1] != y[1]:
			if x not in temp_list_2 and y not in temp_list_2:
				temp_list_2.append(x)
				temp_list_2.append(y)
```

У цьому блоку ми шукаємо пари переходів, де стан однаковий, а вихідні символи різні. Якщо такі пари знайдені, то вони додаються до temp_list_2.

4. **Визначення нового початкового стану (за необхідності)**

```


temp_list_3 = []
for x in temp_list_2:
	if x[0] not in temp_list_3:
		temp_list_3.append(x[0])

if self.initial_state in temp_list_3:
	moore_initial_state = self.initial_state + self.output_alphabet[0]
```

Тут ми перевіряємо, чи входить поточний початковий стан до списку **temp_list_3**, який містить стани з різними вихідними символами. Якщо так, то до початкового стану додається перший символ вихідного алфавіту.

5. **Обробка переходів та виходів**

```
for x in temp_list_2:
      for a in self.input_alphabet:
            if self.transitions[x[0]][a][0] in temp_list_3:
            # Опрацювання переходу, де наступний стан входить в temp_list_3
            else:
            # Опрацювання іншого випадку
```

Ми розглядаєте кожен перехід в **temp_list_2** і для кожного символу вхідного алфавіту перевіряєте, чи наступний стан входить до **temp_list_3**. Якщо так, то обробляєте його одним способом, в іншому випадку - іншим.

6. **Створення, та виведення нового Мур автомата**

```
moore_from_mealy = Moore(
moore_states,
self.input_alphabet,
self.output_alphabet,
moore_transitions,
moore_output_table,
moore_initial_state
)
print(moore_from_mealy)
```

Тут створюється і виводиться новий об'єкт Мур автомата з отриманими параметрами.

Другий алгоритм перетворення автомата Мілі в Мура

```
def convert_to_moore(mealy_machine):
    moore_machine = {
        "states": set(),
        "input_symbols": mealy_machine["input_symbols"],
        "output_symbols": mealy_machine["output_symbols"],
        "transitions": {},
        "state_output": {},
        "initial_state": mealy_machine["initial_state"]
    }
    moore_machine["states"].add(mealy_machine["initial_state"])
    moore_machine["transitions"][mealy_machine["initial_state"]] =
moore_machine["transitions"].get(mealy_machine["initial_state"], {})
    for alphab in mealy_machine["input_symbols"]:
        moore_machine["transitions"][mealy_machine["initial_state"]][alphab] =
mealy_machine["initial_state"] + alphab
    for state in mealy_machine["states"]:
        for input_symbol in mealy_machine["transitions"][state].keys():
            state1 = mealy_machine["transitions"][state][input_symbol][0]
```

```
            output_symbol =
mealy_machine["transitions"][state][input_symbol][1]
            new_state = state + input_symbol
            moore_machine["states"].add(new_state)
            moore_machine["state_output"][new_state] = output_symbol
            moore_machine["transitions"][new_state] = {}
            for alphab in moore_machine["input_symbols"]:
                moore_machine["transitions"][new_state][alphab] = state1 +
alphab
    return moore_machine
mealy_machine = {
 "states": ['a1', 'a2', 'a0'],
 "input_symbols": ['b', 'a'],
 "output_symbols": ['1', '0', '2'],
 "transitions": {'a1': {'b': ('a2', '1'), 'a': ('a1', '0')}, 'a2': {'b':
('a1', '0'), 'a': ('a0', '2')}, 'a0': {'b': ('a0', '2'), 'a': ('a2', '1')}},
 "initial_state": "a1"
}
moore_machine = convert_to_moore(mealy_machine)
print("Moore Machine")
print("States", moore_machine["states"])
print("Input Alphabet =", moore_machine["input_symbols"])
print("Output Alphabet =", moore_machine["output_symbols"])
print("Transitions", moore_machine["transitions"])
print("State Output =", moore_machine["state_output"])
print("Initial State =", moore_machine["initial_state"])
print(moore_machine)
```

Можливий і інший шлях побудови автомата $\mathcal{B}$. Поставимо у відповідність кожному стану a автомата $\mathcal{A}$ множину $\{\overline{a}\} \cup \{(a,x)|x \in \in X\}$. Об'єднання всіх таких множин візьмемо за множину станів автомата $\mathcal{B}$, а функції $f_{\mathcal{B}}$ і $g_{\mathcal{B}}$ переходів і виходів цього автомата задамо за допомогою рівностей:

$$f_{\mathcal{B}}(\overline{a},x) = (a,x),\ f_{\mathcal{B}}((a,x),x') = (f_{\mathcal{A}}(a,x),x');$$
$$g_{\mathcal{B}}(\overline{a},x) = g_{\mathcal{A}}(a,x),\ g_{\mathcal{B}}((a,x),x') = g_{\mathcal{A}}(f_{\mathcal{A}}(a,x),x').$$

Складність алгоритму

Складність виконання функції **convert_to_moore** може бути аналізована за кількістю операцій, які вона виконує у залежності від розміру вхідних даних. Основні операції у вашому коді включають ітерації через списки та словники, виконання умовних перевірок та операції над рядками.

Давайте розглянемо окремі етапи вашого коду та їх потенційні впливи на складність:

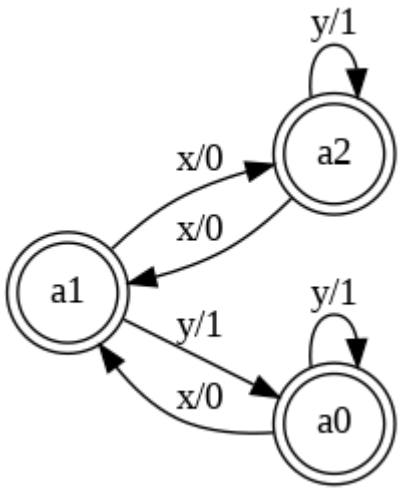
1. **Отримання перехідних станів:**

- Проводяться дві вкладені ітерації через **temp_list**, кожна з яких має розмір **O(n^2**), де **n** - кількість переходів. Отже, цей етап має складність **O(n^2)**.

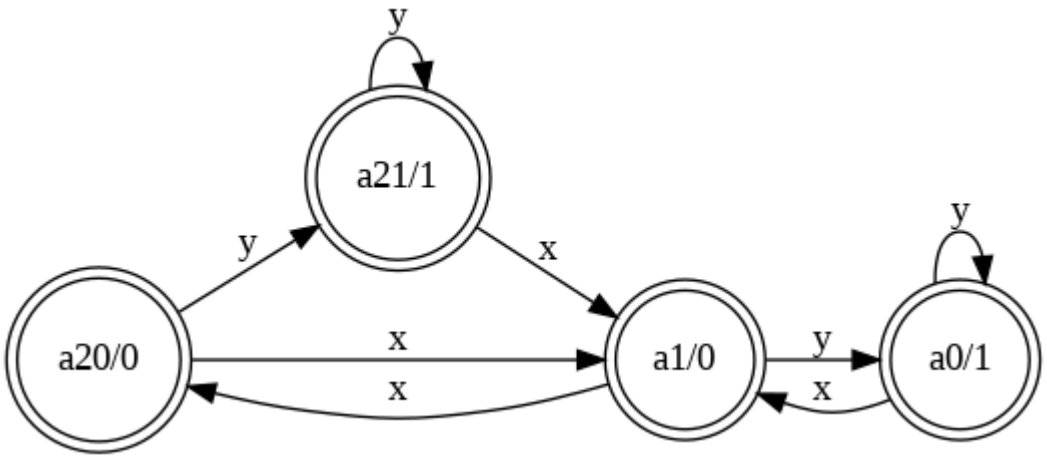
2. **Обробка перехідних станів і заповнення таблиці переходів:**
   - Є ще одна подвійна ітерація через **temp_list_2** з потенційною складністю **O(n^2)**.
   - Додавання/оновлення елементів в словнику **moore_transitions** не займає багато часу, оскільки це операція середньої складності **O(1)**.
   - Всього цей етап має потенційну складність **O(n^2)**.
3. **Заповнення таблиці виходів:**
   - Подібно до попереднього пункту, цей етап має потенційну складність **O(n^2)**.
4. **Створення нового Мур автомата:**
   - Цей етап включає кілька операцій, які мають фіксовану часову складність, не залежну від кількості переходів.
5. **Виведення нового Мур автомата:**
   - Операції виведення даних зазвичай мають часову складність, яка залежить від кількості символів у вихідному рядку.

Отже, загальна складність **convert_to_moore** може бути оцінена як **O(n^2)**, де **n** - кількість переходів у вхідному Мілі автоматі. Важливо зауважити, що це оцінка враховує лише потенційну складність та може варіюватися в залежності від реальних даних та вхідних параметрів.

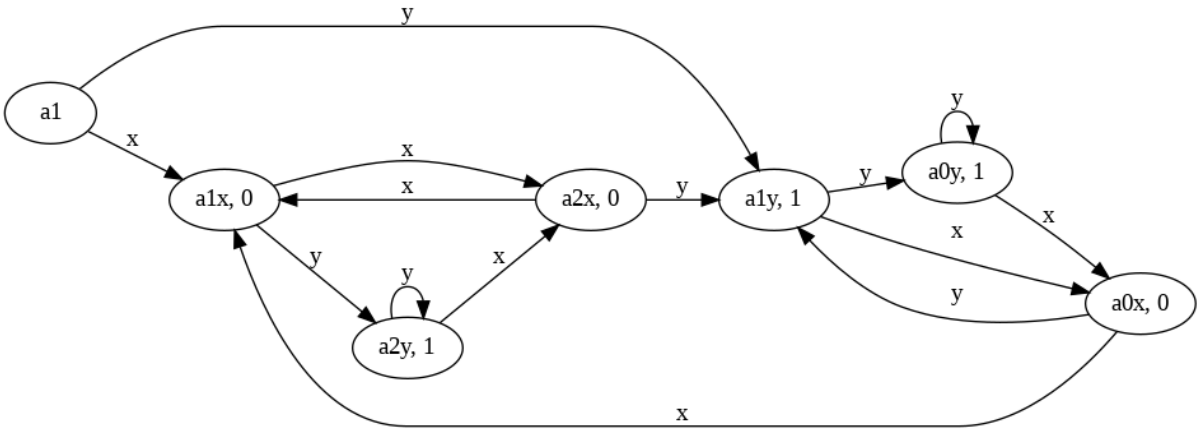
Приклад 1:
Мілі:



Мура:

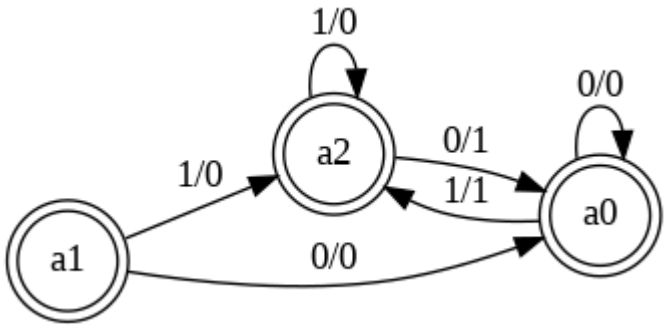


Мура 2-й спосіб:

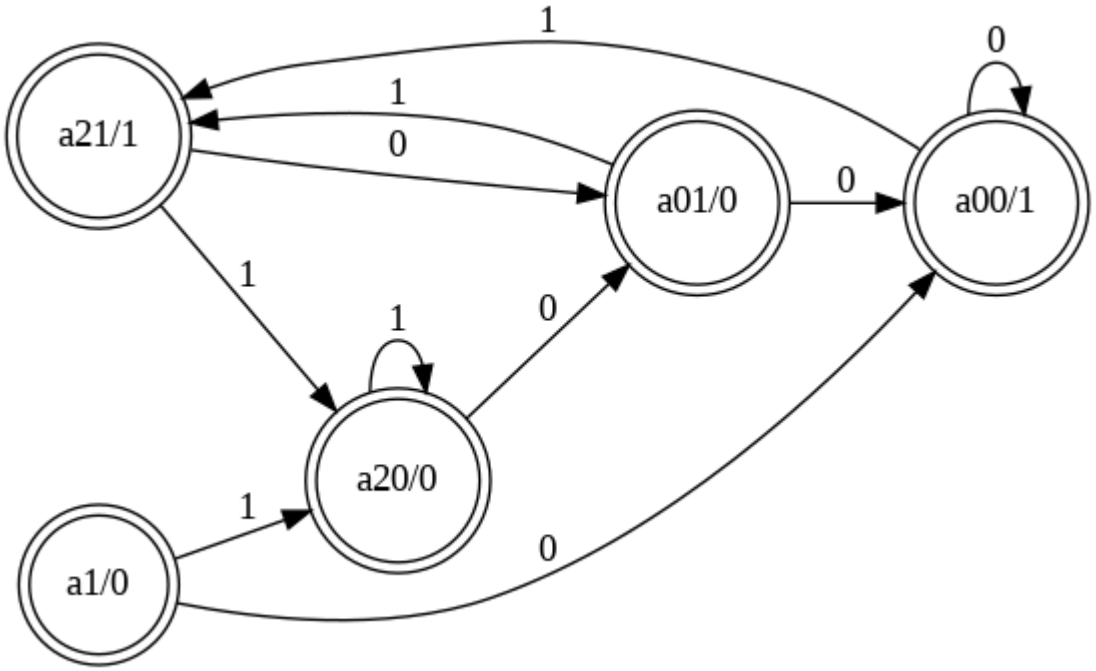


Приклад 2:

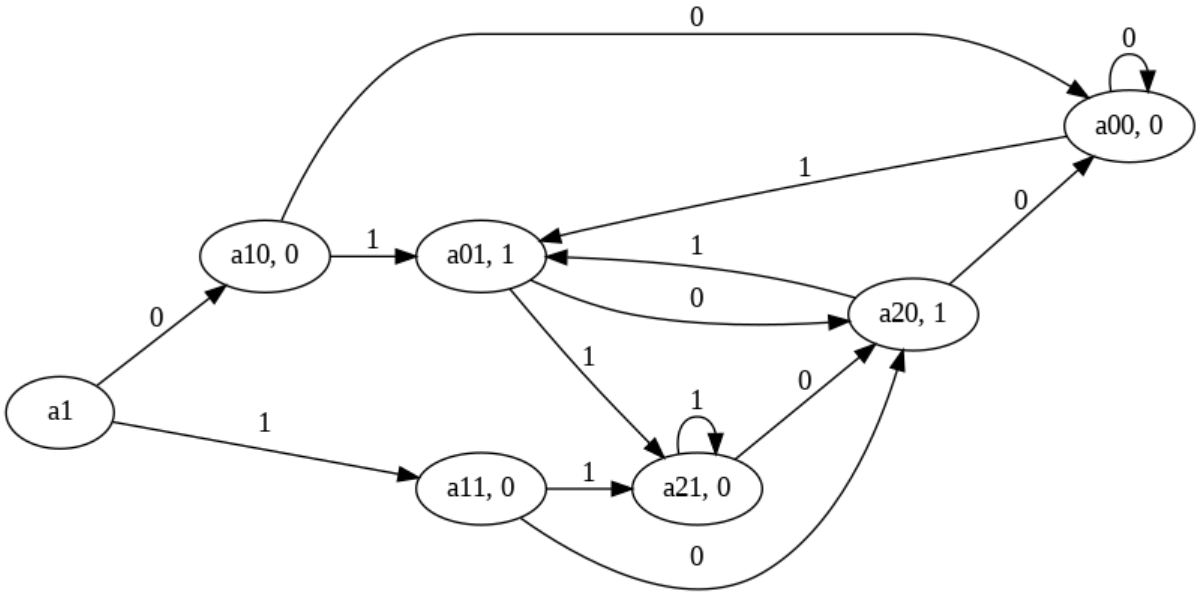
Мілі:



Мура:

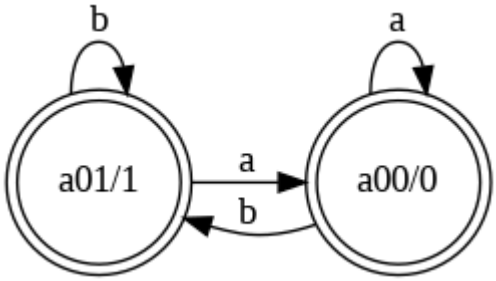


Мура 2-й спосіб:

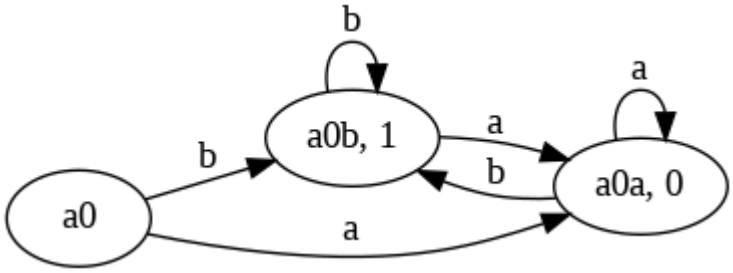


Приклад 3:
Мілі:

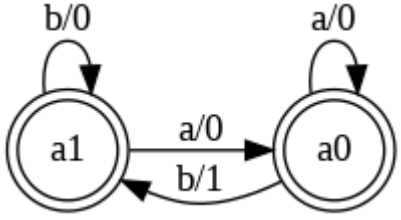
Мура:



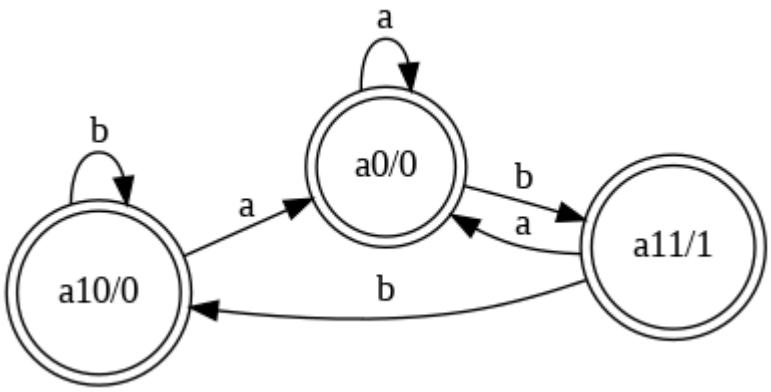
Мура 2-й спосіб:



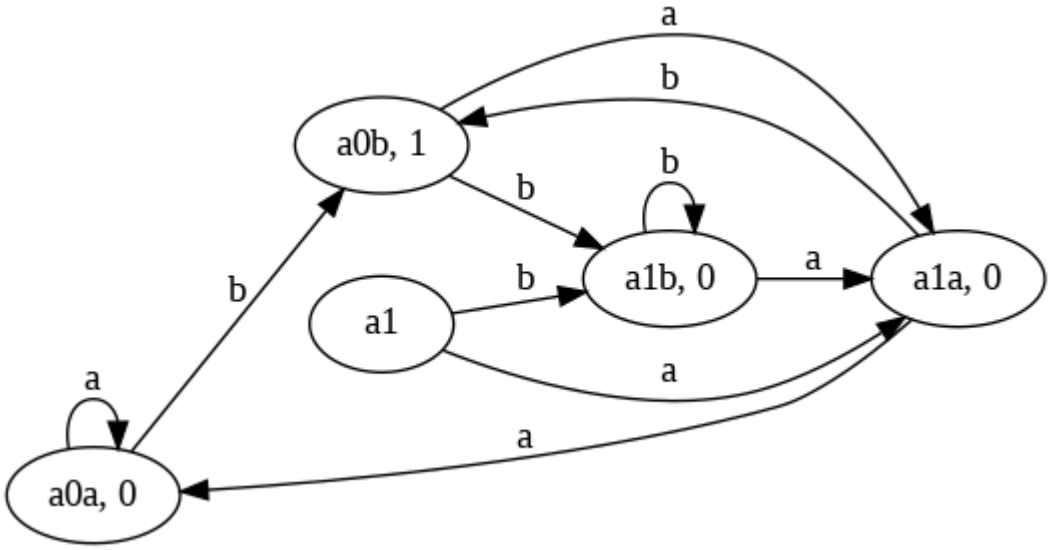
Приклад 4:
Мілі:



Мура:

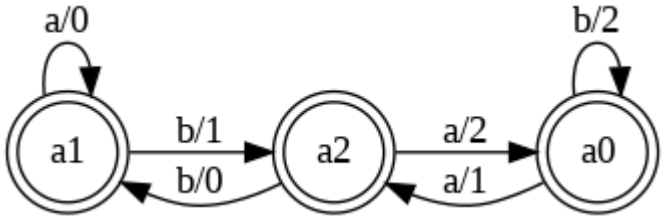


Мура 2-й спосіб:

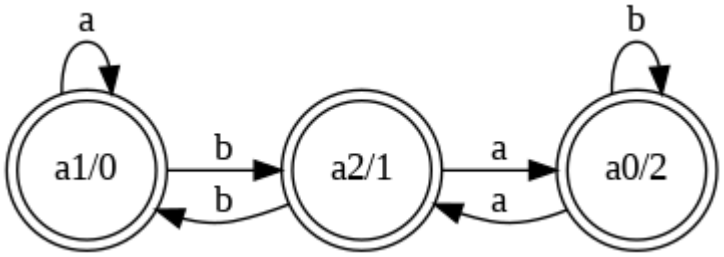


Приклад 5:

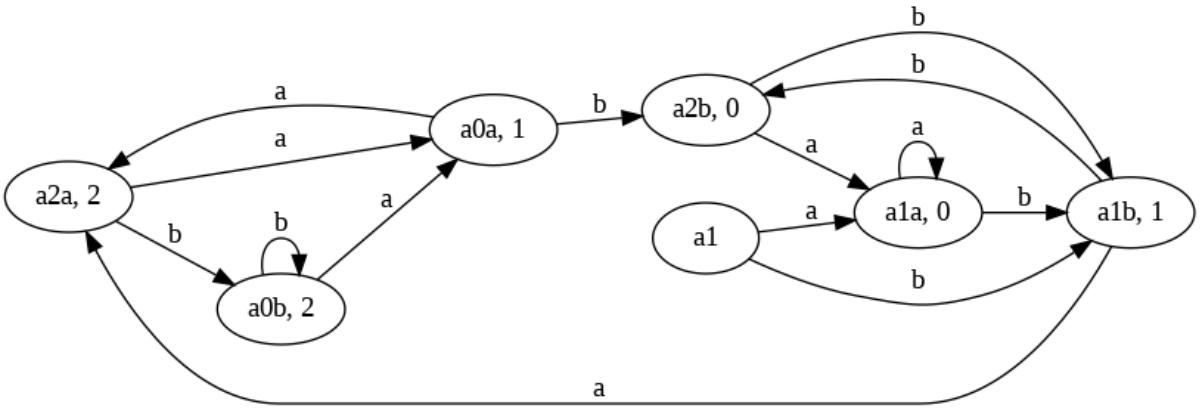
Мілі:



Мура:



Мура 2-й спосіб:



Висновок: Алгоритм надає нам засіб для перетворення автомата Мілі на автомат Мура, враховуючи всі можливі стани. З цієї роботи ми навчилися основам роботи з Автоматом Мілі і Автоматом Мура та перетворення першого в другий.