Київський національний університет імені Тараса Шевченка
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

Задачі аналізу еквівалентності скінченних автоматів

Кваліфікаційна робота бакалавра
зі спеціальності:123 «Комп'ютерна інженерія»
студентки 4 року навчання
Дарини ДИКОЇ

_____

Науковий керівник:
доктор технічних наук, професор
кафедри комп'ютерної інженерії
Сергій ПОГОРІЛИЙ

_____

Рецензент:
кандидат фізико-математичних наук,
асистент
Іван КОЛОМІЄЦЬ

_____

До захисту допускаю:
Завідувач кафедри КІ
к.ф.-м.н., доцент
Юрій БОЙКО

_____

Ухвалено на засіданні кафедри від "____"_____2022 р., протокол № ____

КИЇВ 2022

РЕФЕРАТ

Обсяг роботи 53 сторінки, 25 рисунків, 8 таблиць, 2 додатки, 8 джерел посилань.

СКІНЧЕННІ АВТОМАТИ, ЕКВІВАЛЕНТНІСТЬ, АВТОМАТ МІЛІ, АВТОМАТ МУРА, ДЕТЕРМІНОВАНІ СКІНЧЕННІ АВТОМАТИ, НЕДЕТЕРМІНОВАНІ СКІНЧЕННІ АВТОМАТИ, ε-НЕДЕТЕРМІНОВАНІ СКІНЧЕННІ АВТОМАТИ, СИНТЕЗ СКІНЧЕННИХ АВТОМАТІВ, ПЕРЕВІРКА ЕКВІВАЛЕНТНОСТІ, PYTHON

Об’єктом роботи є дослідження властивості еквівалентності СА на основі методів пошуку еквівалентних автоматів, а саме переходів від автоматів Мілі до автоматів Мура та від НСА і ε-НСА до ДСА. Предметом роботи є реалізація автоматичного пошуку та синтезу еквівалентних СА.

Метою роботи є:

— створення застосунку для синтезу СА Мура еквівалентних до СА Мілі;

— створення застосунку для синтезу ДСА до НСА (в тому числі ε-НСА);

— написання навчально-методичних матеріалів для проведення контролю знань студентів при вивченні курсу «Дискретна математика».

Інструменти розроблення: інтегроване середовище розробки Visual Studio Code, мова програмування Python, веб-додаток Jupyter Notebook для візуалізації СА.

Результат роботи: було проведено аналіз теоретичного матеріалу на тему СА, поняття мінімізації, синтезу та еквівалентності СА, розглянуто алгоритми переходу від одного типу СА до іншого, створено застосунок для синтезу еквівалентних СА різних типів з можливістю автоматичної побудови таблиць переходів та виходів, графоїдів.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ І ПОЗНАЧЕНЬ

СА — скінченний автомат

НСА — недетермінований скінченний автомат

ε-НСА — ε-недетермінований скінченний автомат

ДСА — детермінований скінченний автомат

ЗМІСТ

ВСТУП

Теорія автоматів зародилася і почала активно розвиватися в кінці 50-х та на початку 60-х років 20-го століття. Вважається, що теорія автоматів, що працює зі словами скінченної довжини, бере свій початок в 1956 році після публікації робіт Кліні. Основні поняття та сучасна нотація теорії автоматів остаточно встановилися в кінці 60-х років минулого століття. Становлення сучасної теорії автоматів пов'язане з такими відомими вченими як Н. Хомський, Е.Ф. Мур, Г.Х. Мілі, В.М. Глушков та інші. Спочатку теорія автоматів і, зокрема, теорія СА розглядала їх як динамічні системи, що мають певну кількість станів. Потім було суттєво узагальнено поняття СА і зараз теорію автоматів подають як математичну модель обчислень, за допомогою якої моделюються інші процеси.

СА можна розглядати як пристрій для опрацювання певної дискретної інформації. Однакове опрацювання інформації можливе різними видами СА, які будуть відрізнятись кількістю станів, функціями виходів тощо. Такі автомати називають еквівалентними.

Існує далеко не один вид СА, а також велика кількість алгоритмів переходу від одного виду автомата до іншого. Постає питання, як переконатися, що отриманий після синтезу СА є еквівалентним до заданого? Поняття еквівалентності розглядається в курсі дискретної математики, як важлива властивість двох або більшої кількості автоматів.

Застосунок, який необхідно розробити в рамках цієї роботи ставить за мету дослідити еквівалентність автоматів, на основі алгоритму переходу від СА першого роду (автомата Мілі), до автомата другого роду (автомата Мура) та алгоритму пошуку еквівалентного ДСА до заданого НСА (ε-НСА). Еквівалентність двох автоматів досліджується і доводиться шляхом подання на вхід автоматів однакової інформації та порівнянні вихідної інформації на ідентичність.

Розроблений застосунок потрібен для використання як швидкий та зручний спосіб знаходження еквівалентного до СА Мілі СА Мура, а також еквівалентного до НСА (ε-НСА) — ДСА. Результати роботи можуть бути використані, як для дослідження еквівалентності автоматів, так як і для перевірки вміння студентами знаходити еквівалентні автомати.

Як мову програмування було обрано Python, яка має достатньо легкий та зрозумілий синтаксис і необхідну продуктивність виконання заданих задач.

РОЗДІЛ 1. БАЗОВІ ТЕОРЕТИЧНІ ВІДОМОСТІ ЗІ СА

СА як загальна модель дискретних обчислень

Поняття СА

СА — це деяка абстракція, що використовується для опису зміни стану об’єкту, що в свою чергу залежить від вхідної інформації (вхідного слова) та стану об’єкта в цей момент часу. Простими словами можна сказати, що СА реалізує перетворення деякого вхідного слова в вихідне слово.

ДСА — це автомат, в якому при вхідній букві алфавіту може виконатись перехід з одного внутрішнього стану в не більше ніж один інший внутрішній стан. Цей вид автомату задається п’ятьма елементами$\langle$, , , $_0$, $\rangle$ а саме, скінченний набір станів *Q*, скінченний алфавіт вхідних символів *Σ*, функції переходів *δ: Q* × *Σ* → *Q*, початковий стан $_0 \in$, набір допустимих станів .

ДСА починає свою роботу з початкового стану. Далі він виконує переходи в відповідності до своїх функцій переходу $_{+1} = (, _{+1})$, *i = 0, ..., n-1*, де — стани автомату, $_{+1}$ — елемент вхідного рядку, — функція переходу. ДСА допускає деяке слово з елементів алфавіту автомата, тільки якщо остання літера слова належить набору допустимих станів. В іншому випадку таке слово є забороненим або недопустимим. Набір допустимих автоматом слів називають мовою *L(M)*, де *M* — автомат.

ДСА широко використовуються для розробки компіляторів, розробки та перевірки цифрових схем, аналізу текстових масивів та інших задач.

Автомати Мілі та Мура являються ДСА, відмінністю є те що для їх опису використовуються вже шість елементів, а саме$\langle$, , , , , $_0$ $\rangle$ де *Z* — це скінченний алфавіт вихідних символів, а — функції виходів.

СА Мілі — це ДСА першого роду, в якому вихідний сигнал визначається літерою на вході та його поточним станом, *λ: Q* × *Σ* → *Z*.

СА Мура — це ДСА другого роду, в якому вихідний сигнал визначається тільки поточним станом, *λ: Q* → *Z*.

Існує декілька варіантів задання автоматів, а саме табличний спосіб, графічний спосіб та матричний спосіб.

Табличний спосіб полягає в заданні автомата за допомогою таблиці виходів та таблиці переходів, рядки таблиць позначають вхідними літерами, стовпчики — станами автомата. Для позначення СА Мілі можуть використовувати суміщену таблицю автоматів, де на перетині відповідного стовпчика та рядка записують, як і наступний стан автомата, так і вихідну літеру. Для СА Мура використовують позначену таблицю, де на перетинах позначені наступні стани, а кожному стану окремо відповідають вихідні літери.

Графічний спосіб використовує для задання автомата орієнтований граф (графоїд), де вершини графа позначають станами, а орієнтовні дуги позначають, у випадку автомата Мілі — вхідну букву та вихідну букву. В автоматі Мура дугу позначають тільки вхідною літерою, а вихідною буквою позначають вершини графа.

Матричний спосіб використовує суміщення матриць переходів та матриць виходів. Для позначення використовують цифри 1 та 0. В матрицях переходів 1, якщо даний перехід існує, 0 — не існує. В матрицях виходів 1, якщо значення вихідної букви відповідає дійсності, 0 — не відповідає. Частіше за все для задання використовується матриця з'єднань. Стовпчики та рядки відповідають станам автомата, починаючи з першого стану, на перетині ставиться відповідна літера вхідного алфавіту, що викликає перехід з одного стану в інший, а в дужках ставиться буква вихідного алфавіту, що виникає на виході автомата при переході. Якщо не існує літери вхідного алфавіту, що переводить автомат з одного стану в інший в матрицю записується 0.

Важливо також надати визначення цілком визначених та часткових СА. Цілком визначеними СА називають ті автомати, в яких відображення виходів та переходів визначені для всіх пар (q, x), де q — стан, x — вхідна літера. Автомати в яких деякі відображення є невизначними називаються частковими, місця де відповідні переходи та виходи невизначені в таблицях позначаються прочерками або нулями.

Крім ДСА, також існують НСА. НСА називають абстрактний автомат, в якому в деяких станах при вхідній букві алфавіту є можливість переходу в більше ніж один стан. Іншими словами можна стверджувати, що НСА має властивість знаходитись в декількох станах одночасно. Недетермінований автомат задається п’ятіркою ,⟨, , 0, , де Q —⟩скінченний набір станів,
Σ — скінченний алфавіт вхідних символів, $\Delta: Q \times \Sigma \to P(Q)$ — відношення переходів, 0 ∈ — початковий стан, F — набір допустимих станів. $P(Q)$ в функціях переходів зазначає множину підмножин Q.

Робота НСА починається з початкового стану q_0 . Переходи відбувається у відповідності до відношення переходу, $_{+1} = (, _{+1}), i = 0, \ldots, n-1$, де
— стани автомату, $_{+1}$ — елемент вхідного рядку, — функція переходу. Якщо при закінченні роботи автомат опиняється в одному з допустимих станів, то таке вхідне слово можна вважати допустимим, в протилежному випадку — забороненим або недопустимим.

НСА з ε-переходами або також ε-НСА, є розширенням НСА, що дозволяє автомату мати порожній рядок як вхідну літеру, такий перехід і називається ε-переходом. ε-НСА дає можливість моделювати автомати, в яких поточний стан невідомий, тобто при вхідній послідовності невідомо в якому стані буде знаходитись автомат в наступний момент часу. ε-перехід дає можливість перевести автомат в два стани одночасно.

Єдиною відмінністю ДСА від НСА є те, що функція переходів для детермінованого автомата є функцією, а для недетермінованого — відношенням.

Поняття еквівалентності СА

Саме поняття еквівалентності автоматів можна пояснити твердженням, якщо кожному стану автомата M1 відповідає хоча б один еквівалентний стан автомата M2 та кожному стану автомата M2 відповідає як мінімум один еквівалентний йому стан автомата M1 — два автомата є еквівалентними один

до одного. Якщо хоча б одному стані немає еквівалентного стану в іншому автоматі, автомати розглядаються як нееквівалентні.

Іншими словами можна стверджувати, що два абстрактних автомати M1 та M2 з однаковими вхідним *X* і вихідним *Y* алфавітами називаються еквівалентними, якщо індукують одне й те саме відображення *f* множини *Σ(X)* у *Σ(Y)*. Тобто автомати являються еквівалентними, якщо їх вихідні слова є ідентичними при подачі однакових вхідних слів незалежно від поточного стану автомата.

Побудова мінімального СА, еквівалентного до заданого

Два автомати, що мають однаковий алфавіт вхідних літер та алфавіт вихідних літер називаються еквівалентними. Одним із прикладів побудови еквівалентного автомату до заданого є синтез еквівалентного автомата першого роду, автомата Мілі, до автомата другого роду, автомата Мура. Зв'язок між цими двома видами автоматів можна показати через їх функції виходів, так як функції переходів в них є однаковими. Задано функцію виходів автомата другого роду $y(t) = \lambda(q(t), x(t))$. Якщо підставити до цієї функції зсунуту функцію переходів $q(t) = \delta(q(t-1), x(t))$, отримується рівняння
$y(t) = \lambda(\delta(q(t-1), x(t)), x(t)) = \lambda`(q(t-1), x(t))$, що визначає функцію виходів автомата першого роду.

Ще одним прикладом може виступати побудова еквівалентного ДСА до заданого НСА. Так як недетермінований автомат включає в себе безліч детермінованих автоматів, будь-яка мова, що обробляється детермінованим автоматом буде також оброблятися недетермінованим. Можна стверджувати, що для кожного НСА існує еквівалентний до нього ДСА, що буде допускати таку ж мову.

Синтез СА Мура, еквівалентного до заданого СА Мілі

Алгоритм синтезу

Нехай автомат Мілі заданий у вигляді суміщеної таблиці переходів-виходів.

Таблиця 1

	0	1	2	3
1	1/1	0/2	1/1	0/1
2	2/2	2/2	3/1	1/2

В таблиці множина $S = \{_0 , _1 , _2 , _3 \}$ є множиною станів автомата Мілі. Множина $X = \{_1, _2\}$ є вхідним алфавітом та множина $Y = \{_1, _2\}$ є вихідним алфавітом.

Першим кроком потрібно позначити однакові пари стан/буква вихідного алфавіту в таблиці переходів виходів. Нехай ідентичні пари позначаються , множина значень яких потім буде множиною станів еквівалентного автомата Мура.

Таблиця 2

	0	1	2	3
1	1/1 3	0/2 2	1/1 3	0/1 1
2	2/2 5	2/2 5	3/1 6	1/2 4

Далі записуються множини станів автомата Мура, що відповідають станам автоматам Мілі, множини еквівалентних станів.

$_0$ => $\{_0, _1, _2\}$;

$_1$ => $\{_3, _4\}$;

$_2$ => $\{_5\}$;

$_3$ => $\{_6\}$.

Останнім кроком є побудова позначеної таблиці автомата Мура. Вихідні сигнали записуються наступним способом: в поміченій таблиці автомата Мілі шукається потрібна позначка зі станом автомату Мура, а в позначену таблицю

записується відповідна вихідна буква. Для запису переходів розглядаються множини еквівалентних станів, в стовпчики станів в позначеній таблиці Мура записуються позначки з таблиці автомата Мілі, що відповідають еквівалентному стану автомата Мілі.

Таблиця 3

	—	1	2	1	2	2	1
	0	1	2	3	4	5	6
1	3	3	3	2	2	3	1
2	5	5	5	5	5	6	4

Побудовано позначену таблицю переходів автомату Мура еквівалентного до автомату Мілі. В таблиці множина $\{_0, _1, _2, _3, _4, _5, _6\}$ є множиною станів автомата Мура, множина $\{_1, _2\}$ є вхідним алфавітом, а множина $\{_1, _2\}$, в свою чергу, вихідним алфавітом.

Приклад синтезу СА Мура еквівалентного до СА Мілі

Заданий автомат Мілі поданий у графічний спосіб.

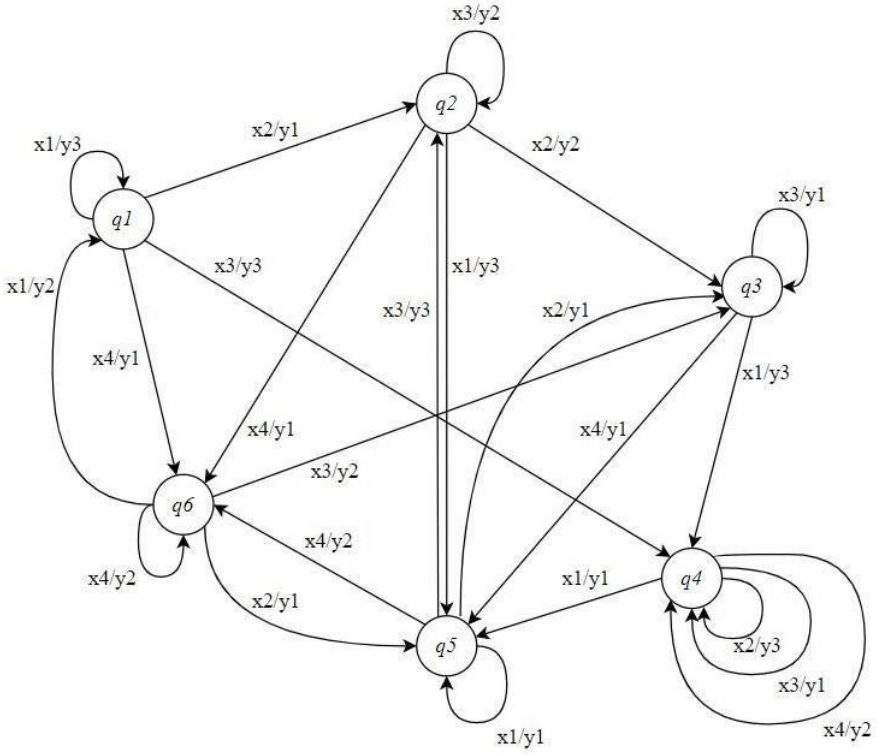

Рис. 1. Графічне зображення СА Мілі

Першим кроком будується суміщена таблиця переходів-виходів автомата Мілі.

Таблиця 4

	1	2	3	4	5	6
1	1/3	5/3	4/3	5/1	5/1	1/2
2	2/1	3/2	3/2	4/3	3/1	5/1
3	4/3	2/2	3/1	4/1	2/3	3/2
4	6/1	6/1	5/1	4/2	6/2	6/2

Множина $Q = \{1, 2, 3, 4, 5, 6\}$ — множина станів автомата Мілі.

Множина $X = \{1, 2, 3, 4\}$ — вхідний алфавіт.

Множина $Y = \{1, 2, 3\}$ — вихідний алфавіт.

Було позначено однакові пари в суміщеній таблиці — .

Таблиця 5

	1	2	3	4	5	6
1	1/3 1	5/3 2	4/3 3	5/1 4	5/1 4	1/2 5
2	2/1 6	3/2 7	3/2 7	4/3 3	3/1 8	5/1 4
3	4/3 3	2/2 9	3/1 8	4/1 10	2/3 11	3/2 7
4	6/1 12	6/1 12	5/1 4	4/2 13	6/2 14	6/2 14

Множини еквівалентних станів:

1 => {0, 1, 5};

2 => {6, 9, 11};

3 => {7, 8};

4 => {3, 10, 13};

5 => {2, 4};

6 => {12, 14}.

Позначена таблиця Мура:

Таблиця 6

	—	3	3	3	1	2	1	2	1	2	1	3	1	2	2
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	1	4	4	4	1	2	3	3	2	4	2	5	4	5
2	6	6	8	3	8	6	7	7	7	7	3	7	4	3	4
3	3	3	11	10	11	3	9	8	8	9	10	9	7	10	7
4	12	12	14	13	14	12	12	4	4	12	13	12	14	13	14

Множина $S = \{_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _{10}, _{11}, _{12}, _{13}, _{14}\}$ — множина станів автомата Мура.

Множина $X = \{_1, _2, _3, _4\}$ — вхідний алфавіт. Множина $Y = \{_1, _2, _3\}$ — вихідний алфавіт.

Перевірка еквівалентності вхідними словами:

На автомат Мілі подано слово $_{22431424312}$, на виході автомата було отримано слово $_{12133212233}$.

На автомат Мура подано слово $_{22431424312}$, на виході автомата було отримано слово $_{12133212233}$.

Заданий автомат Мілі та синтезований автомат Мура є еквівалентними.

Особливості задання та функціонування НСА

Теорема про детермінізацію НСА та відповідний алгоритм

Теорема 1. Для кожного недетермінованого автомата існує еквівалентний детермінований автомат, який допускає ту ж саму мову [1].

Алгоритм перетворення починає свою роботу з початкового стану НСА. Проте якщо даний автомат є ε-НСА першим кроком буде позбавлення від ε-переходів в автоматі. Для позбавлення від ε-переходів в автоматі існують такі типи перетворення:

— всі стани НСА, крім початкового, в які входять тільки ε-дуги прибираються;

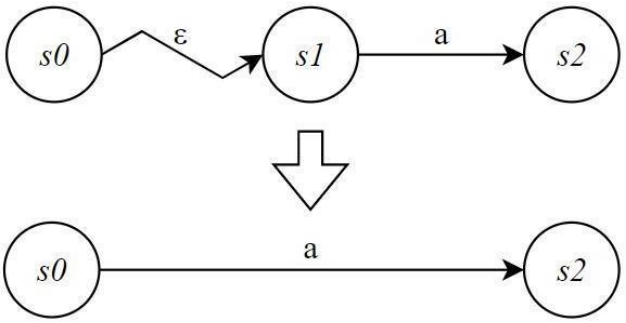

Рис. 2. Позбавлення від ε-переходу

— перехід між двома довільними станами автомату, наприклад $_0$ та $_2$ буде існувати тільки в двох випадках: або якщо така дуга вже існує, або якщо існує такий стан $_1$, що існує ε-перехід від $_0$ до $_1$ та перехід від $_1$ до $_2$. Стан $_1$ може не належати до множини станів НСА, якщо не існує будь-яких переходів до або з $_1$ (Рис. 2);

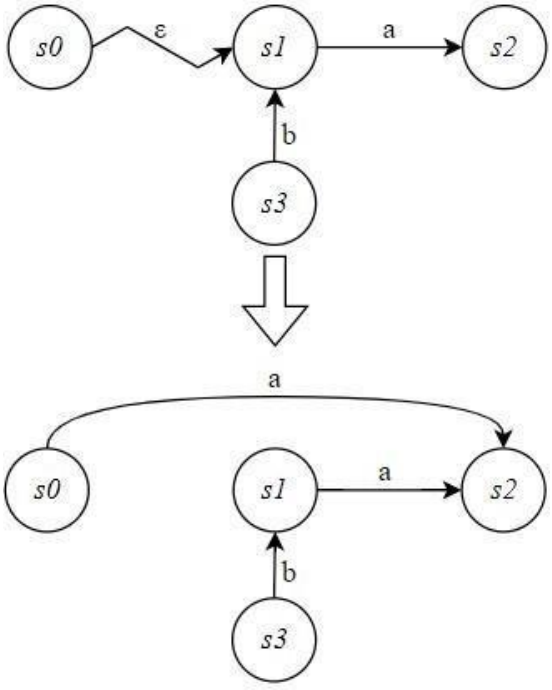

Рис. 3. Позбавлення від ε-переходу

— стан автомату може стати кінцевим, наприклад $_0$, якщо існує ε-перехід з $_0$ в кінцевий стан автомату $_1$ та не існує дуг, що сполучають стан $_1$ з іншими станами автомата.

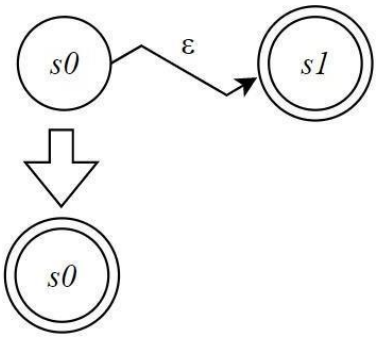

Рис. 4. Позбавлення від ε-переходу

Головна ідея алгоритму переходу полягає в побудові дуг для кожного нового стану та кожного переходу. Для кожного стану будується дуга з в множину таких станів, що існує перехід з елементу множини в цей стан, де — елемент множини станів ДСА, а — елемент множини станів НСА.

Наприклад наступний НСА:

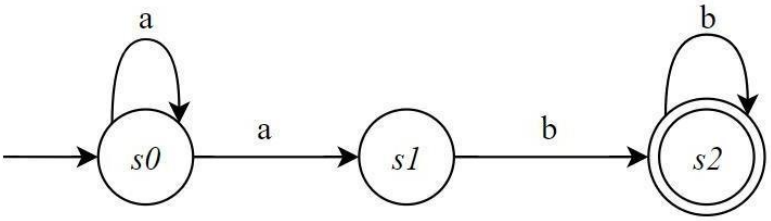

Рис. 5. Графічне зображення НСА

Нехай стан $_0$ є початковим станом НСА, а $_0$ — початковим станом нового ДСА, першим етапом переходу буде побудова дуг зі вхідним сигналом *a* зі стану $_0$ у всі стани, що існує *a* перехід від $_0$ в ці стани. Так як існують *a* переходи з $_0$ в $_0$ та $_1$, буде побудована дуга зі вхідним сигналом *a* зі стану $_0$ в стан $_1$ = {$_0$, $_1$}.

Потім розглядається стан $_1$ = { $_0$, $_1$ }, будуть побудовані дуги, що існують переходи від s_0 або s_1 в ці стани. Так як існує a перехід від $_0$ в $_0$,

в ДСА буде існувати дуга зі стану $_1$ в стан $_1$, також будується *b* перехід з $_1$ в $_2$ = {$_2$}, так як в НСА існує *b* перехід з $_1$ в $_2$.

Останнім розглядається стан $_2$ = {$_2$}, так як існує тільки один a перехід з $_2$ в $_2$ в НСА, також буде існувати єдиний перехід з $_2$ в $_2$ в ДСА. Процес відбувається, поки є можливість створювати нові стани ДСА.

Останнім пунктом є перетворення всіх новостворених станів ДСА в яких міститься елемент множини допустимих станів НСА в допустимі стани ДСА. В заданому НСА є тільки один кінцевий допустимий стан $_2$, і в знайденому ДСА буде тільки один допустимий стан $_2$, так як тільки ця множина станів включає в себе елемент множини кінцевих станів НСА.

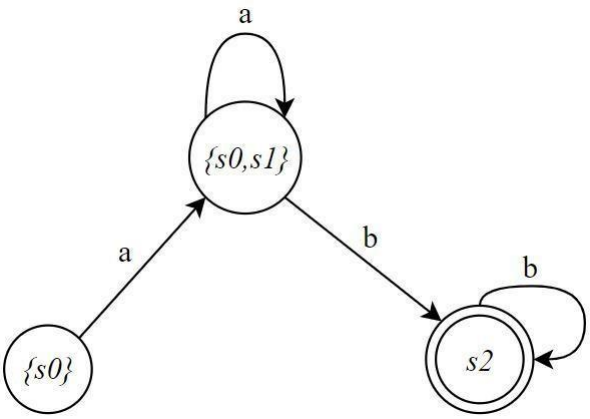

Рис. 6. Графічне зображення ДСА

Приклад детермінізації НСА

Нехай НСА заданий таблицею переходів та графоїдом:

Таблиця 7

	0	*1*
→ 0	1, 3	3
1	2	1, 2
2	3	0
* 3	—	0

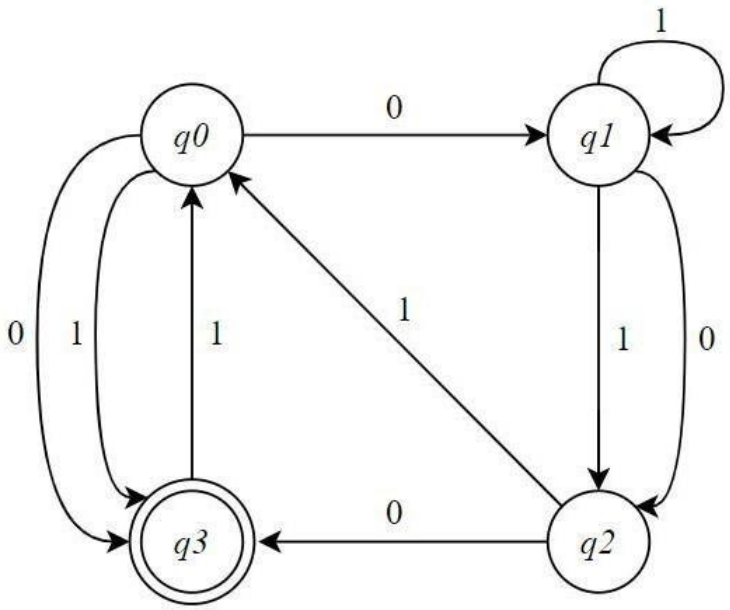

Рис. 7. Графічне зображення НСА

Робота по детермінізації НСА починається з початкового стану, нехай в даному автоматі початковий стан $_0$, тобто шляхом перекодування стан ДСА буде $_0 = \{_0\}$.

Далі розглядаються переходи зі станів по всім вхідним сигналам, тобто по 0 та 1. Зі стану $_0$ в даному автоматі існують переходи по вхідному сигналу 0 в стани $_1$ та $_3$: $_1 = \{_1, _3\}$. По 1 тільки в стан $_3$, який є кінцевим, що означає що новий стан теж буде кінцевим в ДСА: $s_2 = \{_3\}$.

Наступним станом, що розглядається є стан $_1$. Стан $_1$ включає в себе множину станів $\{_1, _3\}$, тому переходи будуть розглядатися з обох станів. По 0 зі стану $_1$ є переходи до стану $_2$, а зі стану $_3$ таких переходів не існує, тому $_3 = \{_2\}$. По 1 за стану $_1$ існують дуги до множини станів $\{_1, _2\}$, зі стану $_3$ — до стану $_0$, тому стан ДСА: $_4 = \{_1, _2, _0\}$.

Стан $s_2 = \{_3\}$: по 0 переходів немає, по 1 — до q_0 , тобто до s_0 , так як такий стан в ДСА уже є.

Стан $s_3 = \{_2\}$: по 0 — до $_2 = \{_3\}$, по 1 — до $_0 = \{_0\}$.

Стан $_4 = \{_1, _2, _0\}$: по 0 — до множини станів $\{_1, _2, _3\}$, такої множини ще немає, тому створюється новий стан ДСА $_5 = \{_1, _2, _3\}$. По 1 — до множини станів $\{_0, _1, _2, _3\}$, новий стан ДСА: $_6 = \{_0, _1, _2, _3\}$. Стан $_5 = \{_1, _2, _3\}$: по 0 — до множини станів $\{_2, _3\}$, новий стан

ДСА: $_7$ = {$_2$, $_3$}, по 1 — до $_4$ = {$_1$, $_2$, $_0$}.

Стан $_6$ = {$_0$, $_1$, $_2$, $_3$}: по 0 — до $_5$ = {$_1$, $_2$, $_3$}, по 1 — до $_6$ = {$_0$, $_1$, $_2$, $_3$}.

Стан $_7$ = {$_2$, $_3$}: по 0 — до $_2$ = {$_3$}, по 1 — до $_0$ = {$_0$}.

Алгоритм закінчує свою роботу, так як більше не існує нових станів ДСА для розгляду. Всі отримані стани ДСА, у яких, у відповідній множині станів, є кінцевий стан $_3$, стають кінцевими станами знайденого ДСА, це стани: $_1$, $_2$, $_5$, $_6$ та $_7$.

Таблиця 8

	0	1
→ 0	1	2
* 1	3	4
* 2	—	0
3	2	0
4	5	6
* 5	7	4
* 6	5	6
* 7	2	0

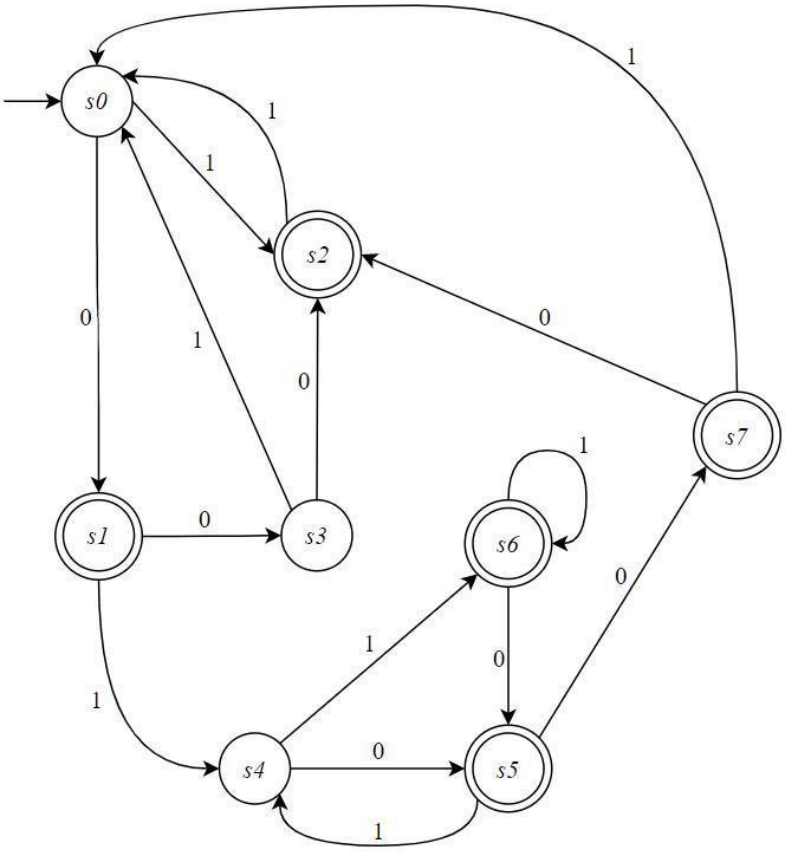

Рис. 8. Синтезований ДСА

Постановка задачі кваліфікаційної роботи

Кваліфікаційна робота присвячена створенню застосунку, що реалізує синтез еквівалентних автоматів, а саме пошук автомата другого роду, тобто автомата Мура, на основі заданого автомата першого роду, автомата Мілі, та пошук ДСА за заданим НСА. На основі результатів синтезу автоматів створеним застосунком було проаналізовано поняття еквівалентності автоматів, як однієї з властивостей автоматів.

Застосунок написаний на мові Python та включає в себе:

— реалізацію алгоритму переходу від автомата Мілі до автомата Мура;

— алгоритм переходу від НСА до ДСА, від ε-НСА до ДСА;

—перевірку еквівалентності на основі порівняння вихідних слів для автоматів Мілі та Мура, отриманих при поданні на вхід однакових вхідних слів;

— реалізацію графічного подання автоматів та таблиць;

— створення посібників “Посібник студента” та “Посібник викладача”.

РОЗДІЛ 2. СТВОРЕННЯ ЗАСТОСУНКУ ДЛЯ СИНТЕЗУ СА МУРА, ЕКВІВАЛЕНТНОГО ДО ЗАДАНОГО СА МІЛІ

Обґрунтування структур даних алгоритму (особливості реалізації для мови Python)

Для повного опису автоматів Мілі та Мура можливо створити один клас опису автомата, проте самі об’єкти класу будуть дещо відрізнятись, в залежності від типу автомата. Для опису автомата клас має містити в собі наступні обов’язкові поля:

— запис початкового стану автомата;

— запис таблиці виходів;

— запис таблиці переходів.

Початкове значення буде зберігатись в змінній числового типу, так як стани зберігаються у вигляді цифр, що відповідають їх нумеруванню.

```
initState = 0
```

Для запису таблиць виходів та переходів краще використати словники, де заданим значенням ключів буде відповідати відповідне значення. Проте при ініціалізації зручніше задати таблицю переходів та виходів за допомогою списків зі списків:

```
transitionFunction:list = []
outputFunction:list = []
```

Таблиця переходів буде списком, де кожен елемент відповідає списку типу [поточний стан, вхідне слово, наступний стан]. Таблиці виходів для автомата Мілі будуть списками з елементами [поточний стан, вхідне слово, вихідне слово], для автомата Мура відповідно — [поточний стан, вихідне слово]. При проходженні по спискам таблиць переходів та виходів можна заповнити потрібні для подальшої роботи словники.

Таблиця переходів для автоматів, як Мілі так і Мура, буде використовувати як ключ набір значень [поточний стан, вхідне слово], відповідним значенням буде виступати наступний стан автомата.

Таблиці виходів автомата Мілі та Мура будуть відрізнятись. Словник, що відповідатиме таблиці виходів Мілі буде використовувати ключ типу [поточний стан, вхідне слово], значенням буде вихідне слово. Для автомата Мура в свою чергу ключем буде тільки поточний стан, значенням — вихідне слово.

Використання таких структур даних, як словники, дозволить зручно ходити по значенням таблиць, виконувати пошук, обхід таблиці.

Наприклад, для обходу таблиць, якщо вхідна послідовність буде налічувати V елементів, середня складність реалізації алгоритму обходу буде становити O(V).

Доступ до даних словників виконується за допомогою ключів. За відсутності колізій хеш функцій ключів, це дозволяє організувати доступ до даних за сталий час, тобто середня складність одного звернення — O(1).

Допоміжними структурами даних виступають вхідний алфавіт, вихідний алфавіт та список станів. Для цього можна використати таку структуру даних, як списки. Задавати ці списки при створенні самого об’єкту класу не обов’язково, так як їх можна буде отримати в подальшому з таблиць переходів та виходів.

```
inAlphabet:list = []
outAlphabet:list = []
states:list = []
```

Імплементація алгоритму переходу від СА Мілі до еквівалентного СА Мура

Для початку роботи алгоритму переходу від автомата Мілі до автомата Мура повинна бути сформована суміщена таблиця переходів-виходів автомату Мілі.

Першим кроком позначаються однакові пари [наступний стан, вихідне слово] в суміщеній таблиці. Це відбувається шляхом проходження по всім коміркам таблиці циклами for по станам автомата (стовпчики) та вхідним сигналам (рядки). Якщо пара вже була позначена, то до суміщеної таблиці просто додається позначення стану автомата Мура.

Якщо пара ще не була позначена, то створюється наступний стан автомата Мура, він додається до суміщеної таблиці в комірку відповідної пари значень [наступний стан, вихідне слово]. За таких умов для позначення суміщеної таблиці переходів-виходів необхідно перебрати всі пари, загальна кількість яких - $N = S * A$, S — кількість станів, A — кількість слів вхідного алфавіту. Та здійснити N запитів до словників переходів-виходів, таким чином середню складність алгоритму можна оцінити як O(N).

Крім цього під час позначення одразу формується таблиця виходів автомата Мура, тобто до новоутвореного стану автомата стає у відповідність вихідна буква алфавіту. Це дозволяє зменшити час виконання алгоритму. Створюються множини еквівалентних автоматів, це словник де ключем є стан автомата Мілі, а значеннями являються списки відповідних станів автомата Мура. Після циклу обходу суміщеної таблиці переходів-виходів визначеними є множини еквівалентних станів, таблиця виходів автомата Мура та помічена суміщена таблиця автомату Мілі.

Останнім кроком є створення позначеної таблиці еквівалентного автомату Мура. Для цього відбувається обхід по множинам еквівалентних станів. Створюється словник, де парі ключів [поточний стан, вхідне слово] ставиться у відповідність наступний стан, взятий з поміченої суміщеної таблиці автомата Мілі. Пошук відбувається за ключем [стан з відповідної множини еквівалентних станів, вхідне слово]. Як було зазначено раніше складність такого запису буде дорівнювати O(1).

Перевірка еквівалентності СА Мілі та Мура

Для перевірки еквівалентності автоматів було створено дві функції. Перша призначена для обходу графа, таким чином можливо автоматизувати пошук слів, що будуть подаватись надалі на автомати. Робота функції завершується при попаданні в стан, що вже відвідувався алгоритмом. На вхід функція отримує об’єкт класу, тобто сам автомат, а на виході видає список з вхідних літер:

```
listOfInSignalsList = []
```

Цей список використовується в другій функції, яка, в свою чергу, виконує обхід по автомату з початкового стану, відповідно до вхідних сигналів. Ця функція відповідно до вхідного сигналу виконує перехід с одного стану до наступного, записує цей стан в змінну:

```
curentState = self.getNextState(curentState, inSignal)
```

Також додає вихідний сигнал до списку, що видає функція на виході, `outSignals = []`.

РОЗДІЛ 3. СТВОРЕННЯ ЗАСТОСУНКУ ДЛЯ СИНТЕЗУ ДСА, ЕКВІВАЛЕНТНОГО ДО ЗАДАНОГО НСА

Обгрунтування структур даних алгоритму (особливості реалізації для мови Python)

Для повного опису ДСА та НСА за основу було взято вже створений клас опису автомата для Мілі та Мура. Проте з деякими змінами, а саме: для НСА та ДСА обов’язково мати кінцеві стани, тому було додано поле для їх запису, з іншої сторони ці види автоматів не мають таблиці виходів, так як не мають вихідних сигналів. Для опису НСА та ДСА потрібні наступні обов’язкові поля:

— запис початкового стану автомата;

— запис таблиці переходів;

— запис кінцевих станів.

По аналогії до опису автомата Мілі та Мура, початковий стан записується у вигляді числового значення:

```
initState = 0
```

Таблиця переходів НСА та ДСА записується у вигляді словника або списку зі списків, кожен елемент відповідає множині значень [поточний стан, вхідне слово, наступний стан], де перші два елементи це ключ, а останній — значення, що йому відповідає. Для НСА було також розроблене розширення для запису таблиці переходів, так як на відміну від ДСА, в цьому виді автоматів можливий перехід до більше ніж одного стану по одному вхідному слову. Для НСА можливий запис типу [поточний стан, вхідне слово, (множина наступних станів)].

Запис таблиці переходів ε-НСА є ідентичним до запису відповідної таблиці для НСА, для відображення ε-переходу, як вхідний символ береться значення None:

```
'inAlphabet': (  0,   1,  None, ),
```

Так як для опису НСА та ДСА використовуються такі ж типи змінних, як і для Мілі та Мура, складність реалізації обходу та доступу до значень зі

словнику залишається незмінною, тобто O(V), де V — кількість елементів, та O(1) відповідно.

Додатковими структурами для ініціалізації НСА та ДСА виступають вхідний алфавіт та список станів, для їх запису використовуються списки.

```
inAlphabet:list = []
states:list = []
```

Імплементація алгоритму знаходження ДСА, еквівалентного до недетермінованого

На початку роботи алгоритму створюються основні змінні, якими буде оперувати алгоритм під час роботи, а саме:

— стек для проміжного зберігання станів ДСА, що будуть розглядатися під час роботи алгоритму. Стек - це структура даних, що дозволяє оперувати даними, що там зберігаються по принципу “прийшов останнім — пішов першим”, при повертанні елемента стеку, він видаляється;

— змінна типу *set()* для зберігання станів, що вже були розглянуті, вона використовується для зберігання декількох значень в одній змінній, перевагою використання є те що *set()* не дозволяє однакових елементів в своєму складі;

— змінна типу *list()* для зберігання таблиці переходів синтезованого ДСА;

Алгоритм починає свою роботу з початкового стану НСА, що був заданий, а саме першим кроком додає цей стан до стеку та до множини відвіданих алгоритмом станів.

Синтез відбувається в циклі *while,* що залежить від наявності нових станів у стеку, вкладений цикл *for* відповідає за проходження по всім вхідним символам.

В циклі за вхідною літерою шукаються всі можливі переходи зі стану, що розглядається та створюється множина станів, що в подальшому буде перекодована в новий стан ДСА. До таблиці переходів ДСА додається новий запис типу [поточний стан, вхідне слово, наступний стан або множина станів].

Далі відбуваються перевірки нового стану ДСА, по-перше, чи є вже такий стан в множині відвіданих станів, якщо ні, то він додається. Так як множина

відвіданих станів записана до змінної типу *set()*, що хешує свої значення, перевірка на існування такого стану відбувається зі складністю O(1). При типі даних *list()* це залежало б від кількості елементів N та становило би O(N). По-друге перевіряється чи множина наступних станів та поточний стан однакові, якщо ні, то нова множина станів записується до стеку алгоритму. Цикл закінчить свою роботу, коли в стеку не буде нових елементів, тобто алгоритм більш не буде мати нових станів для розгляду.

Після виходу з циклу останнім кроком алгоритму є пошук кінцевих станів ДСА, для цього відбувається проходження вкладеним циклом по станам записаним у множину відвіданих станів. Якщо у множині станів хоча б один стан є кінцевим станом НСА — він буде кінцевим у ДСА. Алгоритм синтезу повертає початковий стан, таблицю переходів та множину кінцевих станів ДСА.

РОЗДІЛ 4. РОЗРОБКА ТА РОЗГОРТАННЯ ЗАСТОСУНКУ

Інструментальні засоби створення застосунку

Як середовище розробки було обрано Visual Studio Code, як зручне середовище для роботи з мовою програмування Python 3.9.4.

Visual Studio Code - це інтегроване середовище розробки від компанії Microsoft для операційних систем Windows, Linux та macOS. Функції включають підтримку налагодження, підсвічування синтаксису, інтелектуальне завершення коду, фрагменти, рефакторинг коду та вбудований модуль для роботи з системою керування версіями Git.

Розробка коду відбувається у віртуальному оточенні Python. Віртуальне оточення - це інструмент мови програмування Python для зручного управління залежностями та відокремлення проекту. Воно дозволяє зробити проект автономним та легко відтворюваним на будь-якій машині.

Для зручної розробки застосування було вирішено використовувати веб-сервіс GitHub.

GitHub - один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Базується на системі керування версіями Git.

До переваг веб-сервісу GitHub можна віднести його зручність використання, можливість синхронізації коду між персональною машиною та самим сервісом, що дозволяє легко клонувати актуальний код на машину та отримати швидкий доступ до коду. Також GitHub дає змогу просто відслідковувати зміни в коді та планувати подальшу роботу.

GitHub також дає можливість візуалізувати файли в форматі веб-додатку Jupyter Notebook, що є перевагою, так як даний веб-додаток використовується в застосунку.

Jupyter Notebook - це веб-додаток з відкритим кодом, який дозволяє створювати та обмінюватися документами, що містять код, який дозволяє зміни в будь-який момент виконання програми, рівняння, коди для візуалізації та

текст. Використання включає: очищення та перетворення даних, чисельне моделювання, статистичне моделювання, візуалізацію даних, машинне навчання та багато іншого.

Встановивши застосунок Jupyter Notebook для середовища розробки Visual Studio Code, з'являється можливість редагувати та виконувати налагодження записників Jupyter на локальній машині розробника.

Тестування проекту, а саме unit-тести (від англ. unit - модуль, блок), можуть бути виконані як у середовищі програмування Visual Studio Code, так і на вер-сервісі GitHub. Unit-тест - це тест, що дозволяє перевірити на правильність окремі модулі програми. Перевага веб-сервісу GitHub полягає в тому що він дає можливість запуску сценаріїв тестування на віртуальних машинах сервісу замість персональної машини.

Для розгортання проекту було обрано середовище Google Colaboratory, слід зазначити, що компанія Microsoft також пропонує цілу низку сервісів для роботи з записниками Jupyter, але всі вони платні.

Google Colaboratory - це безкоштовне середовище для Jupyter Notebook, яке працює у хмарі та зберігає свої блокноти на сервісі Google Диск або сервісі GitHub. По суті, на час роботи з файлом Jupyter Notebook, Google надає в користування віртуальну машину на якій може виконуватись обробка даних та їх візуалізація.

Для реалізації алгоритму мовою Python було інтегровано декілька додаткових модулів мови програмування. Їх список зберігється в текстовому файлі requirements.txt в корені програми. Створення текстового файлу requirements дозволяє зручно встановити всі потрібні для роботи застосування модулі при початковій активації віртуального оточення Python.

Перший додатковий модуль, який було встановлено, це модуль *ipykernel* - який надає ядро для роботи з додатком Jupyter Notebook. Наступний модуль називається *numpy*, це бібліотека для мови програмування Python, що дає змогу працювати з великими, багатовимірними масивами та таблицями, а також надає багато функцій для роботи з ними. Модуль *graphviz* це модуль з відкритим

кодом для будування графіків, малювання автоматів та іншого. Останній модуль *matplotlib* додає додаткові функції для роботи з масивами модуля *numpy*, а саме побудова графіків на їх основі.

Приклади роботи застосунку

Проект було розгорнуто на платформі Google Colaboratory, слід зауважити що для відкриття та запуску програми потрібно мати обліковий запис Google. На початку роботи з проектом на обраній платформі потрібно підготувати оточення для подальшої роботи зі застосунком. Платформа Google Colaboratory дозволяє легко використовувати всі функції описані в коді застосунку.

Першим кроком встановлюються всі потрібні компоненти, а саме потрібні модулі *graphviz* та *wavedrom*, та саме застосування.

```
!python -m pip install -U graphviz wavedrom > /dev/null
!python -m pip install -U -i https://test.pypi.org/simple/ SciPyFST > /dev/null
```

Наступним кроком є імпорт необхідних бібліотек для роботи застосунку.

```
from SciPyFST import fst, fstUtils
import graphviz, wavedrom
from IPython.display import display, Markdown
from tabulate import tabulate
```

Останнім пунктом є опис додаткової функції, що відповідає за побудову таблиць при роботі застосунку.

```
def printTable(fstIn):
    print(tabulate(fstUtils.toTable(fstIn),
                   headers="firstrow",
                   tablefmt='pretty'))
```

Підготовка оточення відбувається на початку роботи застосунку, кожна частина коду, що виконується виводиться додатково на панель Executions, приклад налаштування показаний на Рис. 9.

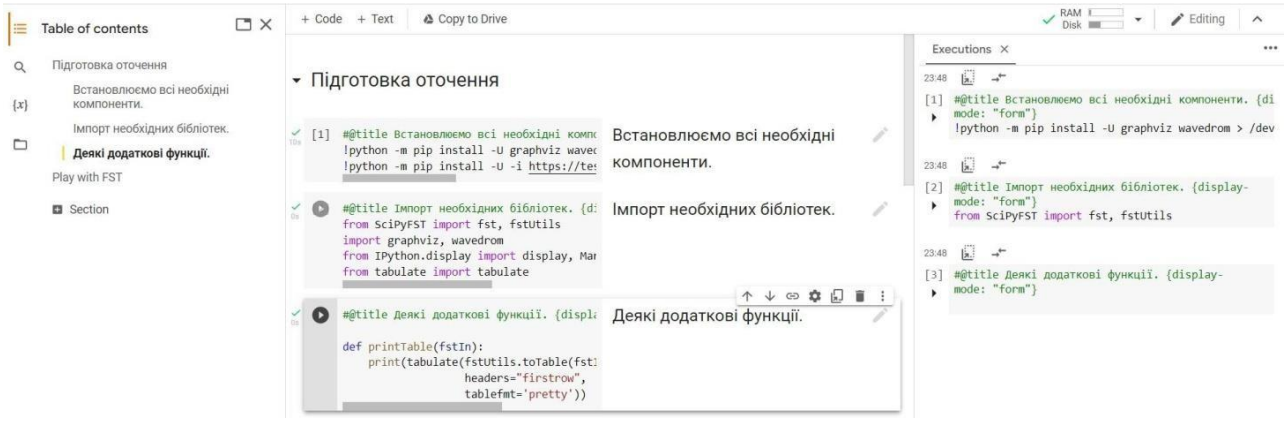

Рис. 9. Приклад підготовки оточення для застосунку

Приклад знаходження еквівалентних СА Мура до Мілі

Перед запуском програми потрібно задати автомат Мілі, а саме задається початковий стан, вхідний алфавіт, таблиця переходів та таблиця виходів. Для зручності читання та задання при записі таблиць використовується табуляція.

Наприклад, як приклад береться автомат Мілі, що був розглянутий в РОЗДІЛІ 1, пункті 1.2.2.:

```
fstInit = {
    'initState': 'q1',
    'inAlphabet':         (  'x1', 'x2',  'x3',  'x4',),
    'transition': {'q1': (   'q1', 'q2',  'q4',  'q6',),
                   'q2': (   'q5', 'q3',  'q2',  'q6',),
                   'q3': (   'q4', 'q3',  'q3',  'q5',),
                   'q4': (   'q5', 'q4',  'q4',  'q4',),
                   'q5': (   'q5', 'q3',  'q2',  'q6',),
                   'q6': (   'q1', 'q5',  'q3',  'q6',),
                  },
    'outputMealy':{'q1': (   'y3', 'y1',  'y3',  'y1',),
                   'q2': (   'y3', 'y2',  'y2',  'y1',),
                   'q3': (   'y3', 'y2',  'y1',  'y1',),
                   'q4': (   'y1', 'y3',  'y1',  'y2',),
                   'q5': (   'y1', 'y1',  'y3',  'y2',),
                   'q6': (   'y2', 'y1',  'y2',  'y2',),
                  },
}
```

Наступна частина коду відповідає за перетворення запису автомата у потрібний для застосунку вигляд, а саме створення словників, побудову суміщеної таблиці, Рис. 10, та малювання графу автомата Мілі, Рис. 11.

```
mealy = fstUtils.fstFromDict(fstInit)
printTable(mealy) ### Рис. 10
graphviz.Source(fstUtils.toDot(mealy)) ### Рис. 11
```

State\Input	x1	x2	x3	x4
->q1	q1/y3	q2/y1	q4/y3	q6/y1
q2	q5/y3	q3/y2	q2/y2	q6/y1
q3	q4/y3	q3/y2	q3/y1	q5/y1
q4	q5/y1	q4/y3	q4/y1	q4/y2
q5	q5/y1	q3/y1	q2/y3	q6/y2
q6	q1/y2	q5/y1	q3/y2	q6/y2

Рис. 10. Суміщена таблиця автомату Мілі

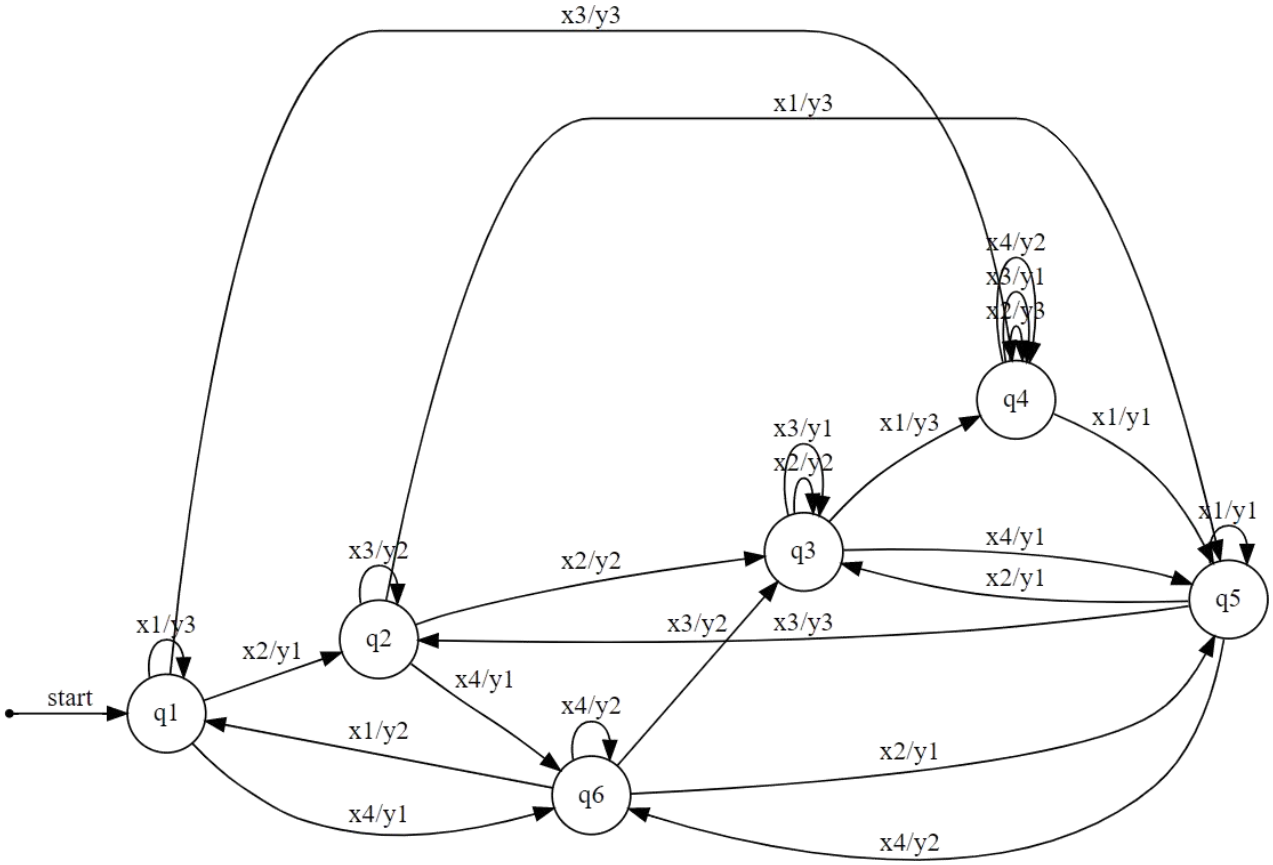

Рис. 11. Графоїд автомату Мілі

Останнім кроком є синтез еквівалентного автомата Мура з можливістю задання літери для кодування нових станів, також побудова позначеної таблиці, Рис.12, та малювання графоїду Мура, Рис. 13.

```
moore = fstUtils.copyWithRenameStates(mealy.asMoore(), prefix='s')
printTable(moore) ### Рис. 12
graphviz.Source(fstUtils.toDot(moore)) ### Рис. 13
```

State\Input	x1	x2	x3	x4
->q0/-	q1	q2	q3	q4
q1/y3	q1	q2	q3	q4
q2/y1	q6	q7	q8	q4
q3/y3	q10	q3	q11	q12
q4/y1	q5	q10	q7	q13
q5/y2	q1	q2	q3	q4
q6/y3	q10	q14	q9	q13
q7/y2	q3	q7	q14	q10
q8/y2	q6	q7	q8	q4
q9/y3	q6	q7	q8	q4
q10/y1	q10	q14	q9	q13
q11/y1	q10	q3	q11	q12
q12/y2	q10	q3	q11	q12
q13/y2	q5	q10	q7	q13
q14/y1	q3	q7	q14	q10

Рис. 12. Позначена таблиця синтезованого автомата Мура

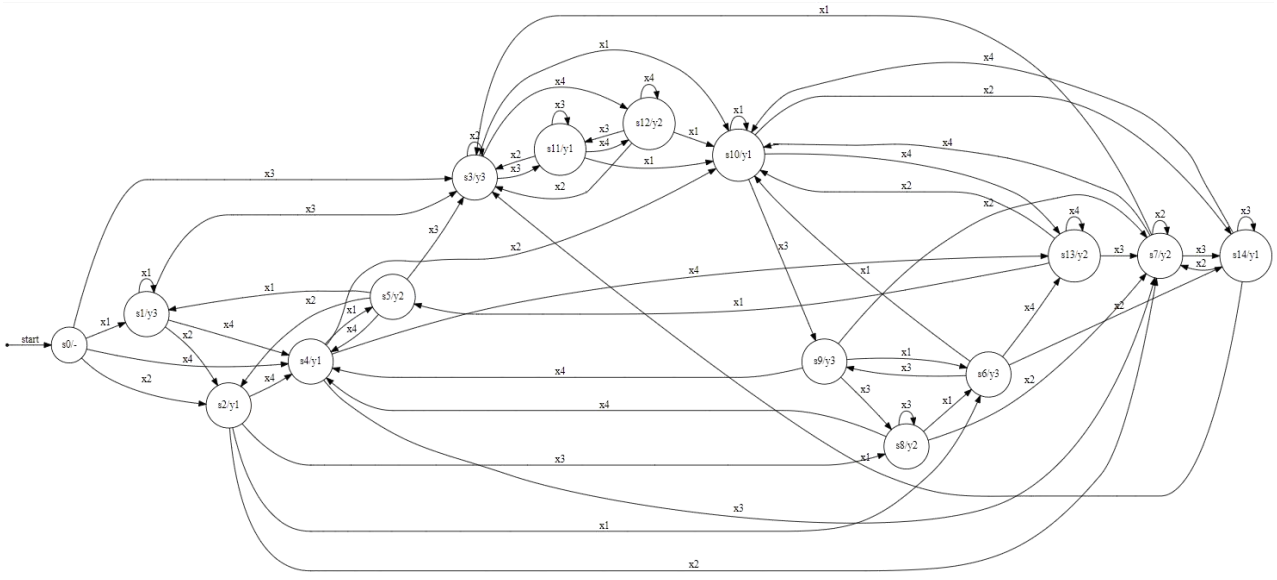

Рис. 13. Графоїд синтезованого автомата Мура

Приклад знаходження еквівалентних ДСА до НСА

На початку задається НСА для синтезу, задаються початковий та кінцеві стани, а також вхідний алфавіт та таблиця переходів, приклад з РОЗДІЛУ 1, 1.3.2.:

```
nfaInit = {
    'initState': 'q0',
    'finalStates': ('q3', ),
    'inAlphabet':           (  0,           1, ),
    'transition': { 'q0':   ( ('q1','q3'),  'q3', ),
                    'q1':   (  'q2',        ('q1','q2'), ),
                    'q2':   (  'q3',        'q0', ),
                    'q3':   (  None,        'q0', ),},
}
```

Аналогічно до синтезу автомата Мура запис перетворюється в словники для роботи застосунку, будується таблиця переходів, Рис. 14, та графоїд, Рис. 15.

```
nfa = fstUtils.fstFromDict(nfaInit)
printTable(nfa) ### Рис. 14
graphviz.Source(fstUtils.toDot(nfa)) ### Рис. 15
```

State\Input	0	1
->q0	q1,q3	q3
q1	q2	q1,q2
q2	q3	q0
*q3	-	q0

Рис. 14. Таблиця переходів НСА

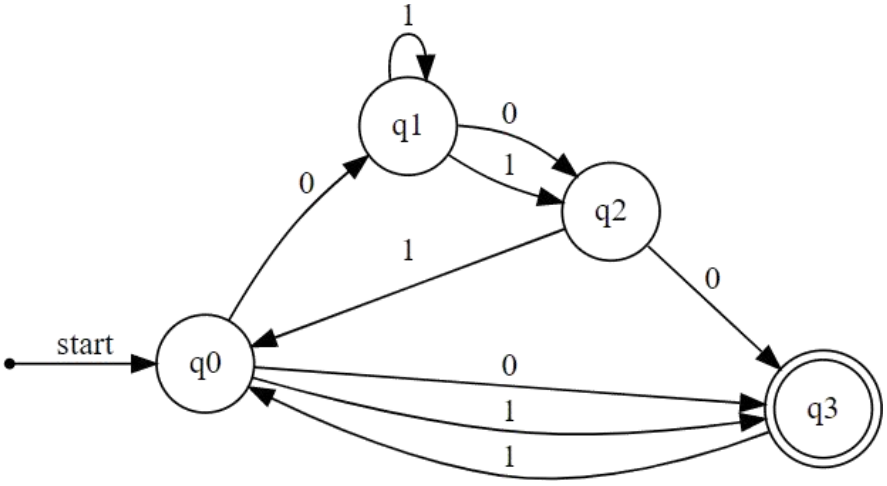

Рис. 15. Графоїд НСА

Пошук еквівалентного ДСА до заданого НСА з заданням літери для перекодування станів, побудова таблиці, Рис. , та графоїда, Рис. :

```
dfa = fstUtils.copyWithRenameStates(nfa.asDFA(), prefix='s')
printTable(dfa) ### Рис.
graphviz.Source(fstUtils.toDot(dfa)) ### Рис.
```

State\Input	0	1
->q0	q1	q2
*q1	q3	q4
*q2	-	q0
q3	q2	q0
q4	q5	q6
*q5	q7	q4
*q6	q5	q6
*q7	q2	q0

Рис. 16. Таблиця переходів синтезованого ДСА

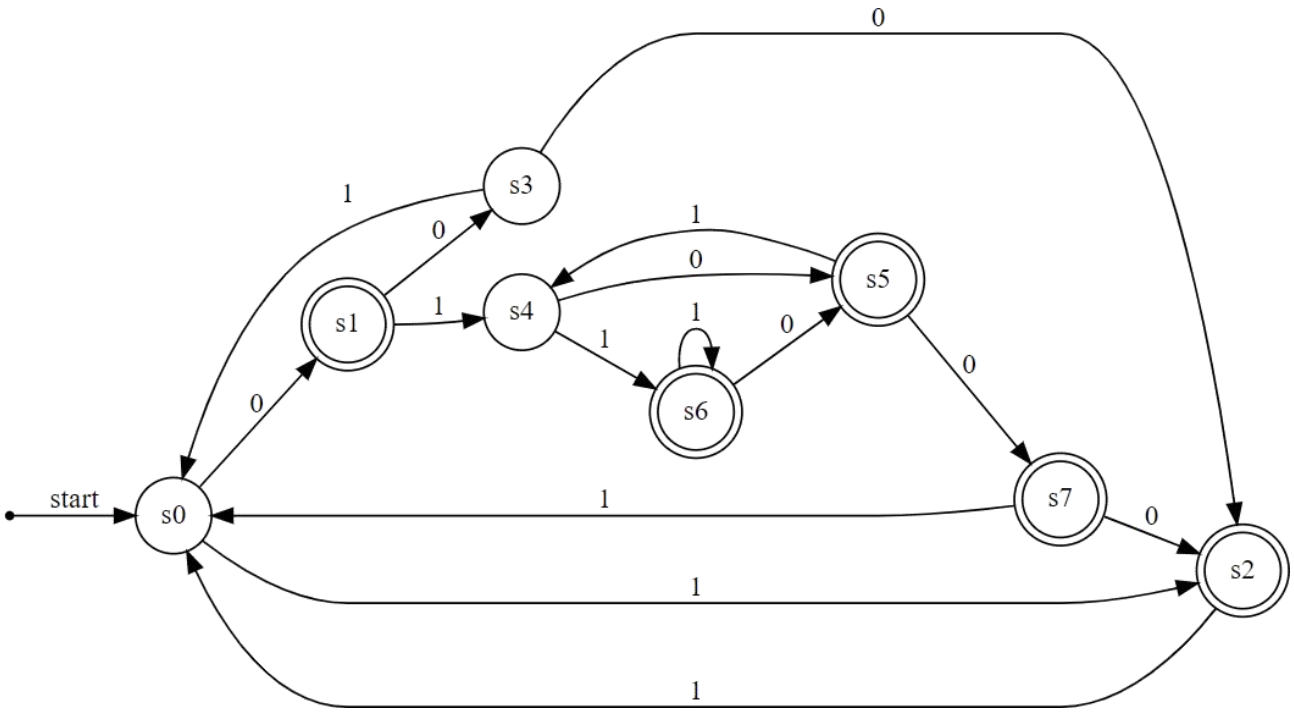

Рис. 17. Графоїд синтезованого ДСА

Приклад знаходження еквівалентних ДСА до ε-НСА

ε-НСА задається так само як і НСА, єдине що до множини вхідних слів додається значення None, що і відповідає ε-переходу.

```
nfaInit = {
    'initState': 'S0',
    'finalStates': ('S1', 'S3'),
    'inAlphabet':         ( 0,     1,     None, ),
    'transition': { 'S0': ( None,  None,  ('S1', 'S3', ), ),
                    'S1': ( 'S2', 'S1',   None, ),
                    'S2': ( 'S1', 'S2',   None, ),
                    'S3': ( 'S3', 'S4',   None, ),
                    'S4': ( 'S4', 'S3',   None, ), },
}
```

Наступним кроком робиться перетворення заданого автомата у вигляд словників та візуалізація таблиці, Рис. 18, та графоїда, Рис. 19, ε-НСА.

```
nfa = fstUtils.fstFromDict(nfaInit)
printTable(nfa) ### Рис. 18
graphviz.Source(fstUtils.toDot(nfa)) ### Рис. 19
```

State\Input	0	1	ε
->S0	-	-	S1,S3
*S1	S2	S1	-
S2	S1	S2	-
*S3	S3	S4	-
S4	S4	S3	-

Рис. 18. Таблиця переходів ε-НСА

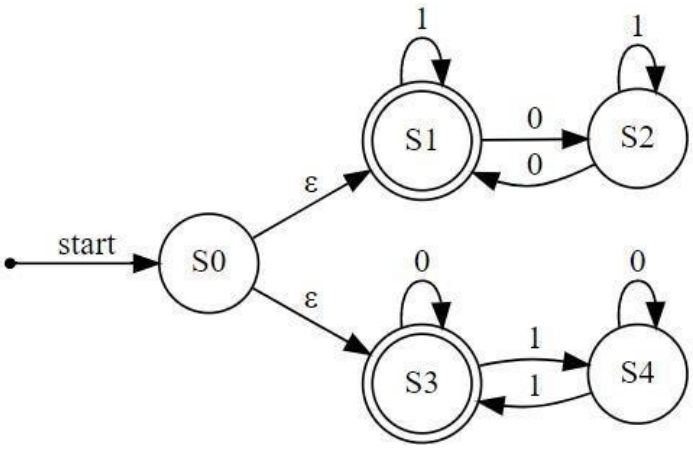

Рис. 19. Графоїд ε-НСА

Синтезування заданого ε-НСА:

```
dfa = fstUtils.copyWithRenameStates(nfa.asDFA(), prefix='q')
printTable(dfa) ### Рис. 20
graphviz.Source(fstUtils.toDot(dfa)) ### Рис. 21
```

State\Input	0	1
->*q0	q1	q2
*q1	q4	q3
*q2	q3	q4
q3	q2	q1
*q4	q1	q2

Рис. 20. Таблиця переходів синтезованого ДСА

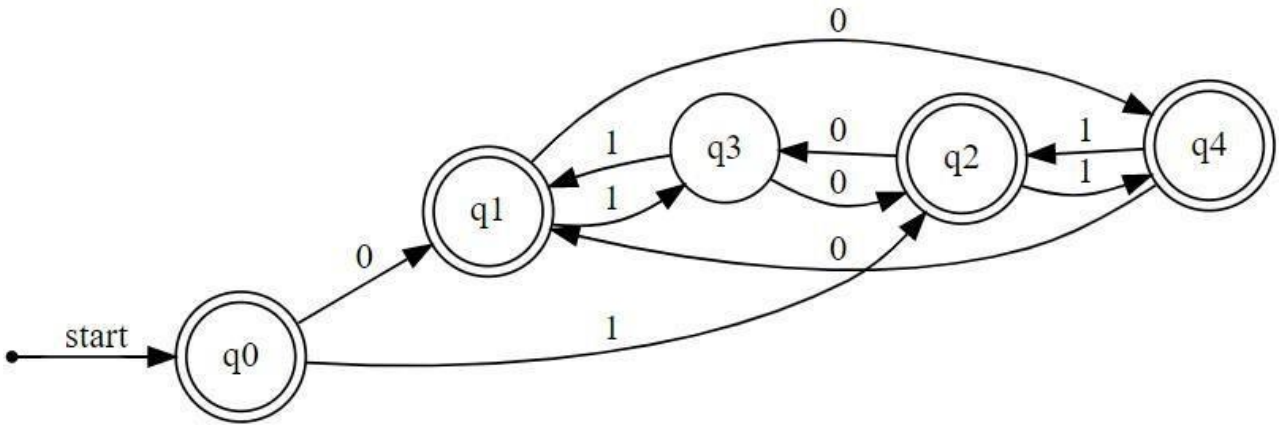

Рис. 21. Графоїд синтезованого ДСА

ВИСНОВКИ

1. В ході виконання роботи було досліджено та проаналізовано поняття еквівалентності автоматів. В результаті проведеного аналізу спроектовано структури даних та алгоритми трансформації з відображенням в мову програмування Python:

— автомата першого роду Мілі до еквівалентного автомата другого роду Мура;

— заданих недетермінованих та ε-недетермінованих автоматів до еквівалентних детермінованих автоматів.

2. Після етапу алгоритмічного проектування створено застосунок мовою програмування Python, що реалізує спроектовані алгоритми переходу та дозволяє автоматизувати процес побудови еквівалентних автоматів замість ручних розрахунків. Обсяг створеного застосунку — 534 рядки вихідного коду на мові Python (лістинг коду застосунку наведено у додатку 1).

3. Підготовлено матеріали для створення методичного посібника «Розрахункові роботи з курсу Дискретної математики» (загальна кількість – 60 задач). Структура матеріалу відповідає сучасній концепції підготовки фахівців у провідних ІТ-корпораціях світу і включає дві частини: «Посібник студента» та «Посібник викладача».

СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ

1. Андерсон Д. Дискретна математика та комбінаторика / Дж. Андерсон. — Москва: Видавничий будинок ”Вільямс”, 2004. — 941 с.

2. Чорней Р. Дискретна математика [Електронний ресурс] / Р. Чорней — Режим доступу до ресурсу: https://www.ukma.edu.ua/~bogd/Discrete%20Mathematics/Applied%20Math/TeorAvtomat.pdf (дата звернення: 14.11.2021)

3. Биков М. М. Дискретний аналіз і теорія автоматів / М. М. Биков, В. Д. Черв'яков. — Суми: Сумський державний університет, 2016. — 354 с.

4. Погорілий С.Д. Програмне конструювання / С. Д. Погорілий. — Київ: ВПЦ "Київський університет", 2007. — 440 с. — (Друге видання)

5. Хопкрофт Д., Мотвані Р., Ульман Дж. Введение в теорию автоматов, языков и вычислений / Дж. Е. Хопкрофт, Р. Мотвані, Дж. Д. Ульман. — Москва: Видавничий будинок “Вільямс”, 2008. — 528 с. — (Друге видання)

6. Виноградова М., Ткачев С., Кандаурова И. Особенности процедуры детерминизации конечных автоматов / Виноградова М., Ткачев С., Кандаурова И. — 2017, № 4 — с. 1-17 — Режим доступу до ресурсу: https://doi.org/10.24108/mathm.0417.0000067 (дата звернення: 17.02.2022)

7. Зайцев Д.А. Математичні моделі дискретних систем: Навч. посібник. / Д.А. Зайцев – Одеса: ОНАЗ ім. О.С. Попова, 2004. – 40 с.

8. Анісімов А.В., Дорошенко А.Ю., Погорілий С.Д., Дорогий Я.Ю. Програмування числових методів мовою Python: Підручник / А.В. Анісімов, А.Ю. Дорошенко, С.Д. Погорілий, Я.Ю Дорогий – Київ.: ВПЦ “Київський університет”, 2014. — 640 с.

ДОДАТКИ

Додаток 1. Лістинг коду застосунку мовою Python

```
from copy import deepcopy

### Клас для задання автомата
class fst:
    def__init__(self,
                states: list = [],
                initState=None,
                inAlphabet: list = [],
                outAlphabet: list = [],
                transitionFunction: list = [],
                outputFunction: list = [],
                finalStates: list = [], ):

    ### Запис станів автомата
        self.states = list(dict.fromkeys(states))
        """ states = [0,1,2] """
    ### Запис початкового стану
        self.initState = deepcopy(initState)
        """ initState = 0 """
    ### Запис вхідного алфавіту
        self.inAlphabet = list(dict.fromkeys(inAlphabet))
        """ inAlphabet = [0,1] """
    ### Запис вихідного алфавіту
        self.outAlphabet = list(dict.fromkeys(outAlphabet))
        """ outAlphabet = [0,1,2] """
    ### Запис таблиці переходів
        self.transitionFunction = deepcopy(transitionFunction)
        """ transitionFunction [ [State, inAlphabet, nextState], ...]\n
        transitionFunction = [ [0,0,1], [1,0,1], [1,1,2] ] """
    ### Запис таблиці виходів
        self.outputFunction = deepcopy(outputFunction)
        """
        outputFunction Moore [ [State, outAlphabet], ...]\n
        outputFunction = [ [0,0], [1,0], [2,2]]\n
        outputFunction Mealy [ [State, inAlphabet, outAlphabet], ...]\n
        outputFunction = [ [0,1,0], [1,1,0], [2,2,2]]
        """
    ### Запис кінцевих станів
        self.finalStates = list(dict.fromkeys(finalStates))
        """ finalStates = [0,1,2] """

        self.__type = self.__detTypeByOutputFunction()

        self.states = list(dict.fromkeys(self.states +
        self._getStatesFromTransitionAndOutputFunction()))

        self.inAlphabet = list(dict.fromkeys(self.inAlphabet +
        self._getInAlphabetFromTransitionAndOutputFunction()))

        self.outAlphabet = list(dict.fromkeys(self.outAlphabet +
        self._getOutAlphabetFromTransitionAndOutputFunction()))

    ### Створення словника та запис до словника переходів та виходів
      self.trFuncDict = dict()
        for (curentState, inSignal, nextState) in self.transitionFunction:
```

```
            if (curentState, inSignal) in self.trFuncDict:
                temp = self.trFuncDict[curentState, inSignal] \
                    if isinstance(self.trFuncDict[curentState, inSignal], list) \
                    else [self.trFuncDict[curentState, inSignal]]
                self.trFuncDict[curentState, inSignal] = temp + [nextState]
            else:
                self.trFuncDict[curentState, inSignal] = nextState


        self.outFuncDict = dict()
        if self.isMoore():
            for (curentState, outSignal) in self.outputFunction:
                self.outFuncDict[curentState] = outSignal
        else:
            for (curentState, inSignal, outSignal) in self.outputFunction:
                self.outFuncDict[curentState, inSignal] = outSignal

        self.comboStateAndOutDict = dict()
        for (curentState, inSignal, nextState) in self.transitionFunction:
            if self.isMoore():
                self.comboStateAndOutDict[curentState, inSignal] = [nextState,
        self.outFuncDict.get(curentState)]
            else:
                self.comboStateAndOutDict[curentState, inSignal] = [nextState,
        self.outFuncDict.get((curentState, inSignal))]

### Допоміжні функції для отримання типу автомата, алфавітів вхідних та вихідних сигналів
    def_detTypeByOutputFunction(self):
        if self.outputFunction:
            if len(self.outputFunction[0]) == 2:
                return 'Moore'
            return 'Mealy'
        return 'FSM'

    def_getStatesFromTransitionAndOutputFunction(self):
        toOut = []
        if self.initState is not None:
            toOut.append(self.initState)
        for (curentState, inSignal, nextState) in self.transitionFunction:
            if curentState is not None:
                toOut.append(curentState)
            if nextState is not None:
                toOut.append(nextState)
        if self.isMoore():
            for (curentState, outSignal) in self.outputFunction:
                if curentState is not None:
                    toOut.append(curentState)
        else:
            for (curentState, inSignal, outSignal) in self.outputFunction:
                if curentState is not None:
                    toOut.append(curentState)
        return list(dict.fromkeys(toOut))

    def_getInAlphabetFromTransitionAndOutputFunction(self):
        toOut = []
        for (curentState, inSignal, nextState) in self.transitionFunction:
            if inSignal is not None:
                toOut.append(inSignal)
        if self.isMealy():
            for (curentState, inSignal, outSignal) in self.outputFunction:
                if inSignal is not None:
                    toOut.append(inSignal)
```

```
        return list(dict.fromkeys(toOut))

    def _getOutAlphabetFromTransitionAndOutputFunction(self):
        if self.isFSM():
            return []
        toOut = []
        if self.isMoore():
            for (curentState, outSignal) in self.outputFunction:
                if outSignal is not None:
                    toOut.append(outSignal)
        else:
            for (curentState, inSignal, outSignal) in self.outputFunction:
                if outSignal is not None:
                    toOut.append(outSignal)
        return list(dict.fromkeys(toOut))

### Функція, що повертає тип автомату, Мілі або Мура, тільки для ДСА
    def getType(self):
        """ Return FST type as string - "Moore" or "Mealy" """
        return self.__type

### Допоміжна функція для копіювання параметрів об’єкту класу
    def deepcopy(self):
        fstOut = fst(self.states, self.initState, self.inAlphabet,
                     self.outAlphabet, self.transitionFunction,
         self.outputFunction, self.finalStates)
        return fstOut

### Функція, що дозволяє додавати стани до існуючого автомату
    def addState(self, state):
        if state not in self.states:
            self.states.append(state)
            self.states = list(dict.fromkeys(self.states))
        return True

### Функція, що дозволяє додавати переходи до існуючого автомату
    def addTransition(self, curentState, inSignal, nextState):
        self.addState(curentState)
        self.addState(nextState)
        if [curentState, inSignal, nextState] not in self.transitionFunction:
            self.transitionFunction.append([curentState, inSignal,
    nextState])
        self.trFuncDict[curentState, inSignal] = nextState
        return True

### Функції для перевірки типу автомата, повертає булеве значення
    def isMoore(self):
        """ Return True if self.getType() == 'Moore' else False """
        return True if self.getType() == 'Moore' else False
    def isMealy(self):
        """ Return True if self.getType() == 'Mealy' else False """
        return True if self.getType() == 'Mealy' else False
    def isFSM(self):
        """ Return True if self.getType() == 'FSM' else False """
        return True if self.getType() == 'FSM' else False

### Функції для перевірки наявності ε-переходів
    def withEpsilon(self):
        """ Return True if epsilon(None) in transitionFunction(inAlphabet) """
        for (state, inSignal, nextState) in self.transitionFunction:
            if inSignal is None:
                return True
```

```
        return False

### Функція для отримання наступного стану автомату, проходження по автомату для ДСА
    def getNextState(self, curentState, inSignal, ifNotInDict=None):
        nextSate = self.trFuncDict.get((curentState, inSignal), ifNotInDict)
        if nextSate is None:
            return ifNotInDict
        return nextSate

### Функція для отримання наступного стану автомату, проходження по автомату для НСА
    def getNextStates(self, curentStates, inSignal, ifNotInDict=None):
        """
        return set of next states for set of current states & input signal,
        ε-closure is NOT INCLUDED!
        if input signal == None return set of ε-accessible in one step states
        """
        states = set(curentStates) if isinstance(curentStates, (list, set)) else
        set([curentStates, ])
        nextStates = set()
        for state in states:
            if (state, inSignal) in self.trFuncDict:
                temp = self.trFuncDict.get((state, inSignal))
                nextStates.update(temp if isinstance(temp, (list, set)) elseset([temp, ]))
            elif inSignal is not None:
                nextStates.update([None, ])
        return nextStates

### Функція для отримання вихідного сигналу автомату
    def getOutSignal(self, curentState, inSignal, ifNotInDict=None):
        if self.isMoore():
            outSignal = self.outFuncDict.get(curentState, ifNotInDict)
        else:
            outSignal = self.outFuncDict.get((curentState, inSignal),
    ifNotInDict)
        if outSignal is None:
            return ifNotInDict
        return outSignal

### Функція для отримання вихідного слова по вхідному слову
    def playFST(self, inSignals: list, startState=None):
        outSignals = []
        curentState = self.initState if startState is None else startState
        outStates = []
        for inSignal in inSignals:
            outStates.append(curentState)
            outSignals.append(self.getOutSignal(curentState, inSignal, -1))
            curentState = self.getNextState(curentState, inSignal)
        outStates.append(curentState)
        if self.isMoore():
            outSignals.append(self.getOutSignal(curentState, inSignal, -1))
            return outSignals[1:], outStates
        return outSignals, outStates

### Функція дозваляє отримати всі стани куди можна потрапити по ε-переходу
    def getEpsilonClosure(self, states, returnType=None):
        """ return ε-closure of a state or set of states """
        inStates = set(states) if isinstance(states, (list, set)) else
    set([states, ])
        while True:
            nextStates = self.getNextStates(inStates, None)
            if set(nextStates).issubset(inStates):
                break
```

```
            inStates.update(nextStates)
        if returnType == 'sortedTupleWithoutNone':
            inStates.discard(None)
            return tuple(sorted(inStates, key=str))
        if returnType == 'sortedTuple':
            return tuple(sorted(inStates, key=str))
        return inStates

### Функція проходження в ширину НСА
    def playFSM(self, inSignals: list, debug=None):
        def_debug(a):
            if debug:
                print("--> " + a)
        __debug("Start print debug info for playFSM():")
        curentStates = set(self.initState) \
            if isinstance(self.initState, (list, set)) \
            else set([self.initState, ])
        curentStates = self.getEpsilonClosure(curentStates)
        for inSignal in inSignals:
            __debug("  curent state(s): " + str(curentStates))
            __debug("  input signal: " + str(inSignal))
            nextStates = self.getNextStates(curentStates, inSignal)
            __debug("    next state(s): " + str(nextStates))
            curentStates = self.getEpsilonClosure(nextStates)
            __debug("      + ε-closure: " + str(curentStates))
        if curentStates & set(self.finalStates):
            __debug("accepting state(s): " + str(curentStates & set(self.finalStates)))
            return True
        __debug("last states: " + str(curentStates) + ", all accepting states:       " +
        str(self.finalStates))
        return False

### Функція, що реалізує алгоритм переходу від автомату Мілі до автомату Мура
    def asMoore(self):
        if self.isMoore():
            return self.deepcopy()

        initStateMoore = 0
        newMooreState_qi = initStateMoore
        markedTranOut = dict()
        dictForSearchNewState = dict()
        eqStatesFor_Si = dict()
        outputFunctionMoore = []
        eqStatesFor_Si[self.initState] = [newMooreState_qi]

        ### Позначення новими станами таблиці Мілі, створення множин еквівалентності
        for topHeaderState_Si in self.states:
            for leftHeaderSignal_Xj in self.inAlphabet:
                tableState_Sm = self.getNextState(topHeaderState_Si, leftHeaderSignal_Xj)
                if tableState_Sm is None:
                    continue
                tableSignal_Yn = self.getOutSignal(topHeaderState_Si, leftHeaderSignal_Xj)
                existMooreState_qi = dictForSearchNewState.get((tableState_Sm,
                tableSignal_Yn))
                if existMooreState_qi is None:
                    newMooreState_qi += 1
                    existMooreState_qi = newMooreState_qi
                    dictForSearchNewState[(tableState_Sm, tableSignal_Yn)] =
                    existMooreState_qi
                    outputFunctionMoore.append([existMooreState_qi,   tableSignal_Yn])
                    tempList = eqStatesFor_Si.get(tableState_Sm, [])
                    tempList.append(newMooreState_qi)
```

```
                    eqStatesFor_Si[tableState_Sm] = tempList
                markedTranOut[topHeaderState_Si, leftHeaderSignal_Xj] = existMooreState_qi

        ### Створення таблиці переходів та виходів автомата Мура
        transitionFunctionMoore = []
        for key_Si, list_qj in eqStatesFor_Si.items():
            for qj in list_qj:
                for signal in self.inAlphabet:
                    transitionFunctionMoore.append([qj, signal, markedTranOut.get((key_Si,
                    signal))])
        return fst([], initStateMoore, [], [], transitionFunctionMoore,
        outputFunctionMoore)

### Функція, що реалізує алгоритм переходу від НСА до ДСА
    def asDFA(self):
        Qstack_dfa = list()
        Qvisited_dfa = set()
        T_dfa = list()
        q0_dfa = self.getEpsilonClosure(
            set(self.initState) if isinstance(self.initState, (list, set, frozenset))
            else set([self.initState, ]))
        q0_dfa.discard(None)
        q0_dfa = tuple([c for b, c in sorted([(str(a), a) for a in q0_dfa])])
        Qstack_dfa.append(q0_dfa)
        Qvisited_dfa.add(q0_dfa)
        while Qstack_dfa:
            curentQ_dfa = Qstack_dfa.pop()
            for inSimbol in self.inAlphabet:
                nextQ_dfa =
            set(self.getEpsilonClosure(self.getNextStates(set(curentQ_dfa), inSimbol)))
                nextQ_dfa.discard(None)
                nextQ_dfa = tuple([c for b, c in sorted([(str(a), a) for a in nextQ_dfa])])
                if tuple() == nextQ_dfa:
                    continue
                T_dfa.append(tuple([curentQ_dfa, inSimbol, nextQ_dfa]))
                if nextQ_dfa not in Qvisited_dfa:
                    Qvisited_dfa.add(nextQ_dfa)
                    if nextQ_dfa != curentQ_dfa:
                        Qstack_dfa.append(nextQ_dfa)
        T_dfa = list(dict.fromkeys(T_dfa))
        finalQ_dfa = list()
        for state_d in Qvisited_dfa:
            for state_n in state_d:
                if state_n in self.finalStates:
                    finalQ_dfa.append(state_d)
                    continue
        return fst(initState=q0_dfa, transitionFunction=T_dfa, finalStates=finalQ_dfa)

### Функція для дебагу результатів програми
    def isNotEqual(self, fst: 'fst', check0equal=True, debug=False):
        def_debug(a):
            if debug:
                print("--> " + a)

        appendPair = "append pair of states {} {} to stack, path={}"
        popPair = "pop pair of states {} {} from stack, path={}"
        checkEqual = "in signal={}; out pair {} {}; next states {} {}"
        inVisited = "{} {} in visited states"

        selfType = 'FSM' if self.isFSM() else 'FST Moore' if self.isMoore() else 'FST Mealy'
        fstType = 'FSM' if fst.isFSM() else 'FST Moore' if fst.isMoore() else 'FST Mealy'
        __debug("compare {} and {}".format(selfType, fstType))
```

```
if self.isFSM() != fst.isFSM():
    return 'FSM != FST'
states_stack = list()
states_visited = set()
final_state_self = set(self.finalStates)
final_state_fst = set(fst.finalStates)

init_state_self = self.getEpsilonClosure(self.initState,returnType='sortedTuple')
init_state_fst = fst.getEpsilonClosure(fst.initState, returnType='sortedTuple')
# place for check 0-equivalent init_state_self and init_state_fst
# have sens for FSM compare to FSM or FST Moore compare to FST Moore
if check0equal:
    __debug("check 0-equivalent {} and {}".format(selfType, fstType))
    if self.isFSM() and fst.isFSM():
        self_out = final_state_self.isdisjoint(set(init_state_self))
        fst_out  =  final_state_fst.isdisjoint(set(init_state_fst))
        if self_out != fst_out:
            __debug("    " + "{} != {}".format(str(self_out), str(fst_out)))
            return 'check 0-equivalent two FSM not pass'
    elif self.isMoore() and fst.isMoore():
        self_out = self.getOutSignal(init_state_self[0], None)
        fst_out = fst.getOutSignal(init_state_self[0], None)
        if self_out != fst_out:
            __debug("    " + "{} != {}".format(str(self_out), str(fst_out)))
            return 'check 0-equivalent two Moore FST not pass'
    else:
        __debug("  can`t check 0-equivalent {} and {}".format(selfType, fstType))

states_stack.append((init_state_self, init_state_fst, tuple()))
__debug(appendPair.format(str(init_state_self), str(init_state_fst),
str(tuple())))
states_visited.add((init_state_self, init_state_fst))

while states_stack:
    (state_self, state_fst, in_path) = states_stack.pop()
    __debug(" " + popPair.format(str(state_self), str(state_fst), str(in_path)))
    for inSimbol in self.inAlphabet:
        curent_path = in_path + (inSimbol, )
        next_state_self =
        self.getEpsilonClosure(self.getNextStates(set(state_self), inSimbol),
        returnType='sortedTuple')
        next_state_fst = fst.getEpsilonClosure(fst.getNextStates(set(state_fst),
        inSimbol), returnType='sortedTuple')
        # place for check equivalent next_state_self and next_state_fst
        if self.isFSM() and fst.isFSM():
            self_out = final_state_self.isdisjoint(set(state_self))
            fst_out = final_state_fst.isdisjoint(set(state_fst))
        else:
            self_out = self.playFST([inSimbol, ], state_self[0])[0]
            fst_out = fst.playFST([inSimbol, ], state_fst[0])[0]
        __debug("    " + checkEqual.format(str(inSimbol),
        str(self_out), str(fst_out), str(next_state_self), str(next_state_fst)))
        if self_out != fst_out:
            __debug("    " + "{} != {}".format(str(self_out), str(fst_out)))
            return curent_path

        if (next_state_self, next_state_fst) not in states_visited:
            __debug("      " + appendPair.format(str(next_state_self),
            str(next_state_fst), str(curent_path)))
            states_stack.append((next_state_self, next_state_fst, curent_path))
            states_visited.add((next_state_self, next_state_fst))
        else:
```

```
                    __debug("      " + inVisited.format(str(next_state_self),
            str(next_state_fst)))

        return False

### Функція для знаходження слів для перевірки еквівалентності
    def getTestSignal(self):
        listOfInSignalsList = []

        def recGetNext(curentState, visitedStates: dict, inSignals: list):
            if visitedStates.get(curentState) is None:
                visitedStates[curentState] = 1
                for inSignal in self.inAlphabet:
                    copyOfInSignals = deepcopy(inSignals)
                    copyOfInSignals.append(inSignal)
                    nextCurentState = self.getNextState(curentState, inSignal)
                    if nextCurentState is None:
                        listOfInSignalsList.append(copyOfInSignals)
                        return
                    else:
                        recGetNext(nextCurentState, deepcopy(visitedStates),
            copyOfInSignals)
            else:
                listOfInSignalsList.append(inSignals)
                return
        recGetNext(self.initState, dict(), [])
        return listOfInSignalsList

    def _isContains(self, fst: 'fst'):
        listOfInSignalsList = fst.getTestSignal()
        for inSignalList in listOfInSignalsList:
            if self.playFST(inSignalList)[0] != fst.playFST(inSignalList)[0]:
                return False
        return True

    def isContains(self, fst: 'fst'):
        def recGetNext(curentState, visitedStates: dict, inSignals: list):
            if visitedStates.get(curentState) is None:
                visitedStates[curentState] = 1
                for inSignal in fst.inAlphabet:
                    copyOfInSignals = deepcopy(inSignals)
                    copyOfInSignals.append(inSignal)
                    nextCurentState = fst.getNextState(curentState, inSignal)
                    if nextCurentState is None:
                        return self.playFST(copyOfInSignals)[0] ==
            fst.playFST(copyOfInSignals)[0]
                    else:
                        if not recGetNext(nextCurentState,
            deepcopy(visitedStates), copyOfInSignals):
                            return False
            else:
                return self.playFST(inSignals)[0] ==
    fst.playFST(inSignals)[0]
            return True
        return recGetNext(fst.initState, dict(), [])

    def isSimilar(self, fst: 'fst'):
        return self.isContains(fst) and fst.isContains(self)

### Функція для пошуку недоступних станів автомата
    def getUnreachableStates(self):
        unreachableStates = dict.fromkeys(self.states)
```

```
def recGetNext(curentState, visitedStates: dict):
    unreachableStates.pop(curentState, None)
    if visitedStates.get(curentState) is None:
        visitedStates[curentState] = 1
        for inSignal in self.inAlphabet:
            nextCurentState = self.getNextState(curentState, inSignal)
            if nextCurentState is None:
                return
            else:
                recGetNext(nextCurentState, deepcopy(visitedStates))
    else:
        return
recGetNext(self.initState, dict())
return list(unreachableStates)
```

Додаток 2. Приклади розділів задачників «Посібник студента» та «Посібник викладача» для побудови еквівалентних автоматів Мура та еквівалентного ДСА

9 Варіант

Знайти еквівалентний автомат Мура до заданого автомата Мілі.
Початковий стан автомата Мілі - Q1;
Вхідний алфавіт - ['x1', 'x2', 'x3', 'x4'];
Вихідний алфавіт - ['y1', 'y2', 'y3', 'y4', 'y5']
Слово на вході: ['x4', 'x3', 'x2', 'x2', 'x1', 'x4', 'x3', 'x4', 'x2'] - Слово на виході: ???

Input	State					
	Q1	Q2	Q3	Q4	Q5	Q6
x1	Q1/y3	Q5/y3	Q4/y3	Q5/y1	Q5/y1	Q1/y2
x2	Q2/y1	Q3/y2	Q3/y2	Q4/y3	Q3/y1	Q5/y1
x3	Q4/y3	Q2/y2	Q3/y1	Q4/y1	Q2/y3	Q3/y2
x4	Q6/y1	Q6/y1	Q5/y1	Q4/y2	Q6/y2	Q6/y2

Табл. 9.1: Таблиця переходів-виходів автомата Мілі.

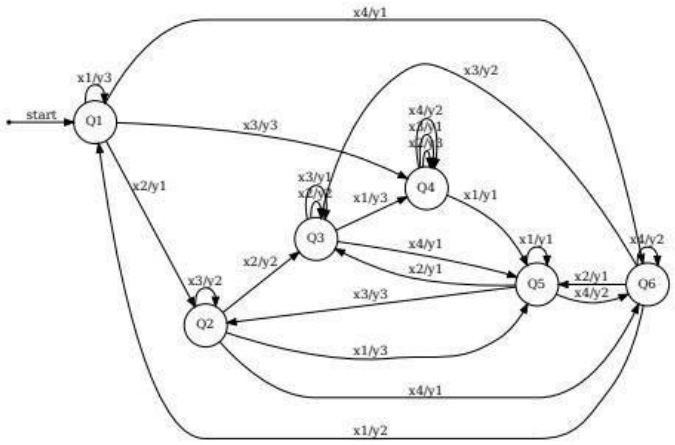

Рис. 9.1: Автомат Мілі.

Рис. 22. Приклад посібника студента для Мілі/Мура

9 Варіант

Слово на вході: ['x4', 'x3', 'x2', 'x2', 'x1', 'x4', 'x3', 'x4', 'x2']
Слово на виході: ['y1', 'y2', 'y2', 'y2', 'y3', 'y2', 'y1', 'y2', 'y3']

Input	State					
	Q1	Q2	Q3	Q4	Q5	Q6
x1	Q1/y3	Q5/y3	Q4/y3	Q5/y1	Q5/y1	Q1/y2
x2	Q2/y1	Q3/y2	Q3/y2	Q4/y3	Q3/y1	Q5/y1
x3	Q4/y3	Q2/y2	Q3/y1	Q4/y1	Q2/y3	Q3/y2
x4	Q6/y1	Q6/y1	Q5/y1	Q4/y2	Q6/y2	Q6/y2

Табл. 9.1: Таблиця переходів-виходів автомата Мілі.

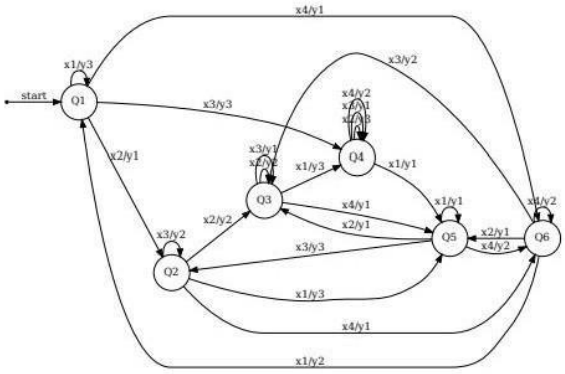

Рис. 9.1: Автомат Мілі.

Input	State														
	0/-	1/y3	10/y1	11/y2	12/y3	13/y2	14/y2	2/y1	3/y3	4/y1	5/y3	6/y2	7/y2	8/y1	9/y1
x1	1	1	9	9	5	14	1	5	9	14	9	3	5	3	9
x2	2	2	3	3	6	9	2	6	3	9	8	6	6	6	8
x3	3	3	10	10	7	6	3	7	10	6	12	8	7	8	12
x4	4	4	11	11	4	13	4	4	11	13	13	9	4	9	13

Табл. 9.2: Таблиця переходів-виходів автомата Мура.

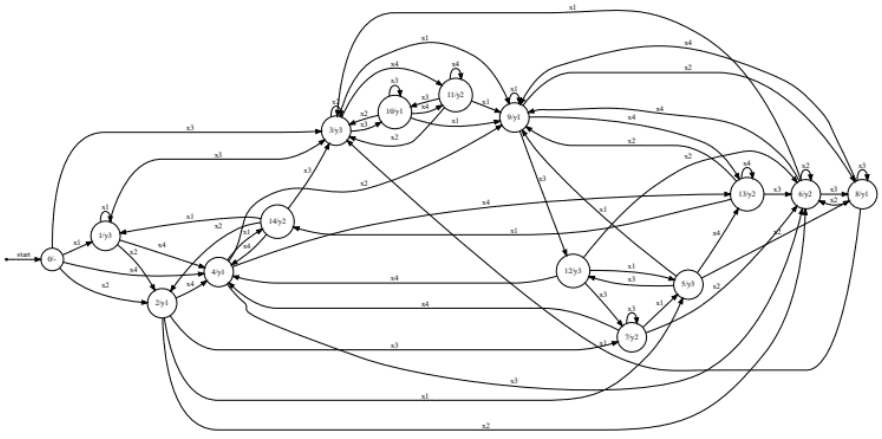

Рис. 9.2: Автомат Мура.

Рис. 23. Приклад посібника викладача для Мілі/Мура

1 Варіант.

Для недермінованого скіченного автомата, заданого матрицею переходів, побудувати еквівалентний детермінований скіченний автомат.
Початковий стан НСА - q0;
Кінцеві/допустимі стани НСА - q3;
Вхідний алфавіт - [0, 1];
Нехай стани НСА перекодовуються в стани ДСА вигляду {a1, a2,...}.

State	Input	
	0	1
→q0	q1,q3	q3
q1	q2	q1,q2
q2	q3	q0
*q3	-	q0

Табл. 1.1: Таблиця переходів НСА.

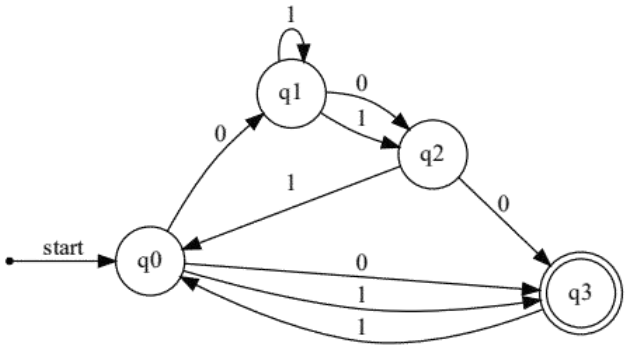

Рис. 1.1: НСА автомат.

Рис. 24. Приклад посібника студента для НСА/ДСА

1 Варіант.

State	Input	
	0	1
→q0	q1,q3	q3
q1	q2	q1,q2
q2	q3	q0
*q3	-	q0

Табл. 1.1: Таблиця переходів НСА.

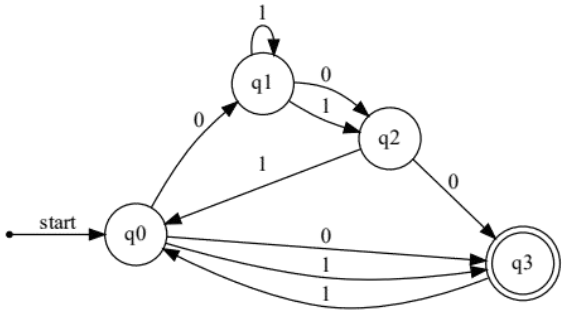

Рис. 1.1: НСА автомат.

State	Input	
	0	1
→a0	a1	a2
*a1	a3	a4
*a2	-	a0
a3	a2	a0
a4	a5	a6
*a5	a7	a4
*a6	a5	a6
*a7	a2	a0

Табл. 1.2: Таблиця переходів ДСА.

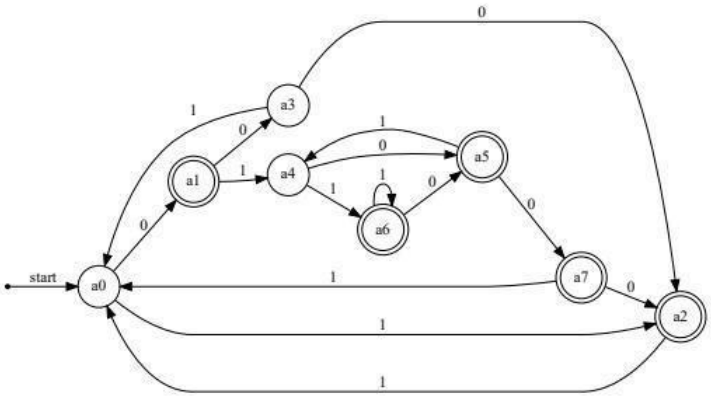

Рис. 1.2: ДСА автомат.

Рис. 25. Приклад посібника викладача для НСА/ДСА