МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ВОЙТЕНКО МАКСИМ ОЛЕКСАНДРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
канд. техн. наук, доцент

_____О. В. Зелінська
«_ __ »_____2024 р.

РЕАЛІЗАЦІЯ МЕТОДУ АВТОМАТИЗОВАНОГО ПРОЄКТУВАННЯ МІКРОПРОГРАМНОГО АВТОМАТА МОВОЮ PYTHON

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (магістерська) робота

Науковий керівник:
Р. М. Бабаков, професор кафедри
інформаційних технологій,
докт. техн. наук, доцент

_____
(підпис)

Оцінка: _____ /_____ /_____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК:_____
(підпис)

Вінниця 2024

АНОТАЦІЯ

Войтенко М. О. Реалізація методу автоматизованого проєктування мікропрограмного автомата мовою Python. Спеціальність 122 «Комп'ютерні науки», Освітня програма «Комп'ютерні технології обробки даних (Data Science)». Донецький національний університет імені Василя Стуса, 2024.

У кваліфікаційній роботі вирішено актуальну науково-практичну проблему програмної реалізації методу алгебраїчного синтезу мікропрограмного автомата з операційним автоматом переходів. Даний метод є основною складовою методу автоматизованого синтезу даного типу мікропрограмного автомата.

Досліджено структуру і принцип функціонування мікропрограмного автомата з операційним автоматом переходів; основні етапи синтезу автомата, зокрема етап алгебраїчного синтезу за заданою множиною операцій переходів; сучасні інструменти та технології швидкої обробки матриць за допомогою мови Python.

Розроблено алгоритмічне і програмне забезпечення для проведення алгебраїчного синтезу мікропрограмного автомата з операційним автоматом переходів. Отримана програмна реалізація дозволяє знаходити повні і часткові розв'язки задачі алгебраїчного синтезу автомата з операційним автоматом переходів, заданого у вигляді файла формату KISS.

Ключові слова: мікропрограмний автомат, операції переходів, алгебраїчний синтез, алгоритм, Python.

Кваліфікаційна робота складається з 3 розділів, основна частина складає 80 сторінок, 1 таблицю, 38 рисунків, 54 використаних джерела інформації.

ABSTRACT

Voitenko M. O. Implementation of Method of Automated Design of a Microprogram Automaton Using Python. Specialty 122 «Computer science», Educational program «Data Science». Vasyl' Stus Donetsk National University, 2024.

In the qualification work, the actual scientific and practical problem of software implementation of the method of algebraic synthesis of a microprogram automaton (finite state machine) with datapath of transitions is solved. This method is the main component of the method of automated synthesis of this type of finite state machine.

Studied: the structure and principle of operation of a microprogram finite state machine with datapath of transitions; the main stages of state machine synthesis, in particular, the stage of algebraic synthesis based on a given set of transition operations; modern tools and technologies for fast matrix processing using the Python language.

Developed: algorithms and software for carrying out the algebraic synthesis of a microprogram finite state machine with datapath of transitions. The resulting software implementation allows you to find complete and partial solutions to the problem of algebraic synthesis of a finite state machine with datapath of transitions, given as a file in KISS format.

Keywords: finite state machine, transition operations, algebraic synthesis, algorithm, Python.

The qualification work consists of 3 sections, the main part consists of 80 pages, 1 table, 38 figures. 54 sources of information used.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГСА – граф-схема алгоритму;

МПА – мікропрограмний автомат;

ОАП – операційний автомат переходів;

ОП – операція переходів;

ПК – пристрій керування;

a_i – поточний стан автомата;

a_j – стан переходу;

B – кількість автоматних переходів;

B_{min} – мінімальна кількість автоматних переходів, не покритих ОП;

B' – кількість знайдених переходів, не покритих ОП;

K – множина допустимих кодів станів;

$K(a_i)$ – код поточного стану;

$K(a_j)$ – код стану переходу;

L – кількість вхідних сигналів;

M – кількість станів автомата;

N – кількість вихідних сигналів;

$N_{ОП}$ – кількість елементів множини O;

N_V – кількість варіантів кодування станів у випадку повного перебору;

O – множина ОП;

O_i – елемент множини ОП;

R – розрядність кодів станів;

T – код поточного стану автомата;

X – множина вхідних сигналів;

x_i – елемент множини X;

Y – множина вхідних сигналів;

y_i – елемент множини Y.

ВСТУП

Цифрові обчислювальні системи використовуються в усіх сферах людського життя. Окрім систем універсального призначення (комп'ютерів), важливе місце займають вузькоспеціалізовані системи, які мають обмежений функціонал, але й низьку вартість. В умовах збройного конфлікту спеціалізовані цифрові системи використовуються у складі роботизованої зброї, в дронах, у системах зв'язку тощо. Актуальною науково-технічною проблемою для промисловості цифрових спеціалізованих систем є здешевшення кінцевої продукції та спрощення процесу виробництва. Розв'язання цієї проблеми можливе, в тому числі, за рахунок автоматизації прєктування таких систем.

Однією з невід'ємних складових цифрової системи є пристрій керування, який координує роботу усіх блоків системи. Одним із різновидів пристроїв керування є мікропрограмний автомат з операційним автоматом переходів (МПА з ОАП), схема якого характеризується високою швидкодією та зменшеною вартістю порівняно з іншими класами пристроїв керування. Одним із етапів проєктування МПА з ОАП є так званий алгебраїчний синтез автомата, що може здійснюватись за різними методами та алгоритмами. Наукове дослідження, проведене в даній роботі, спрямоване на розробці і дослідженні ефективності програмної реалізації одного із відомих методів алгебраїчного синтезу мікропрограмного автомата з операційним автоматом переходів засобами мови програмування Python.

Об'єкт дослідження: алгебраїчний синтез мікропрограмного автомата з операційним автоматом переходів.

Предмет дослідження: програмна реалізація алгоритму синтезу автомата мовою Python.

У кваліфікаційній роботі вирішуються наступні **завдання**:

1. Огляд структури мікпропрограмного автомата з операційним автоматом переходів та процесу його алгебраїчного синтезу.

2. Аналіз методу алгебраїчного синтезу МПА з ОАП на основі матричного підходу.

3. Розробка алгоритму алгебраїчного синтезу МПА з ОАП на основі повного перебору способів кодування станів.

4. Програмна реалізація зазначеного алгоритму.

5. Тестування розробленої програми і дослідження її можливостей в контексті алгебраїчного синтезу МПА з ОАП.

Для досягнення цілей використані наступні наукові підходи та методи:

- огляд літератури;
- алгоритмічна реалізація;
- обчислювальне моделювання;
- емпіричне тестування.

Наукова новизна роботи полягає у дослідженні ефективності алгоритму алгебраїчного синтезу МПА з ОАП за допомогою розробленої програмної реалізації алгоритму.

Практична цінність полягає у тому, що розроблена програмна реалізація алгебраїчного синтезу МПА з ОАП дозволяє автоматизувати процес синтезу автомата і може бути застосована у складі спеціалізованої САПР цифрових пристроїв керування.

.

РОЗДІЛ 1

ТЕОРЕТИЧНІ АСПЕКТИ АЛГЕБРАЇЧНОГО СИНТЕЗУ МІКРОПРОГРАМНОГО АВТОМАТА

1.1 Принцип роботи мікропрограмного автомата з операційним автоматом переходів

Цифрові системи сьогодні застосовуються в будь-якій сфері людської діяльності [1]. Для організації процесу функціонування цифрової системи використовується пристрій керування (ПК). Його функцією є координація роботи інших блоків системи шляхом формування керуючих сигналів відповідно до заданих алгоритмів [2]. Сучасні підходи до проектування ПК цифрових систем є розвитком теорії автоматів, основи якої були закладені в роботах відомих учених С. Кліні, Г. Мілі, Е. Мура, Дж. фон Неймана, В. М. Уілкса, В. М. Глушкова, Д. Хафмена та інших [3-5]. Розвиток досліджень у цій області привів до появи спеціальних методів структурного синтезу для різних класів ПК.

Одним зі структурних різновидів ПК є мікропрограмний автомат (МПА, англ. FSM – Finite State Machine), який також називають автоматом з «жорсткою» логікою (англ. Hardware Logic). Під «жорсткою» логікою розуміється те, що алгоритм керування, який реалізується автоматом, відтворений не у вигляді окремої програми, що пишеться програмістом і зберігається у запам’ятовуючому пристрої, а у вигляді електронної схеми. Така схема проєктується одноразово під кожну конкретну задачу, а характеристики схем багато в чому визначають характеристики цифрової системи в цілому. Схема МПА здатна здійснювати так звані багатоспрямовані мікропрограмні переходи за один крок (такт) роботи автомата, тоді як в інших типах ПК такі переходи здійснюються за декілька тактів роботи. Ця особливість робить МПА найбільш швидким типом ПК, що дозволяє застосовувати МПА в тих системах, де швидкодія є критичним

фактором (інтернет речей, системи швидкого реагування, робототехніка тощо).

Недоліком МПА є те, що його схема, завдяки апаратній реалізації алгоритму керування, має найбільшу складність порівняно з іншими типами ПК [6-10]. Складність схеми вимагає великих апаратурних витрат за її побудову, що збільшує кінцеву вартість пристрою. Крім того, висока складність схеми приводить до збільшення енергоспоживання схеми, збільшення фізичних розмірів (габаритів) пристрою та зменшення надійності його роботи. Всі ці фактори не дозволяють розглядати МПА як ідеальний тип ПК і певним чином обмежують область його використання.

В теперішній час складність алгоритмів керування, що використовуються в ПК, постійно зростає. Відповідно, зростає складність ПК і їх вартість. Це приводить до нової науково-практичної проблеми, яка полягає в розробці і дослідженні нових способів побудови пристроїв керування [11-19]. В рамках цієї проблеми був розроблений так званий мікропрограмний автомат з операційним автоматом переходів (МПА з ОАП) [20]. Схема МПА з ОАП за певних умов має менші витрати апаратури (і, відповідно, меншу вартість) порівняно з іншими типами ПК [21, 22]. Це надає переваги даному пристрою при використанні в якості пристрою керування сучасних цифрових систем.

Розглянемо на загальному рівні структуру і принцип роботи мікропрограмного автомата з операційним автоматом переходів [20, 23].

Структурна схема МПА з ОАП зображена на рис. 1.1.

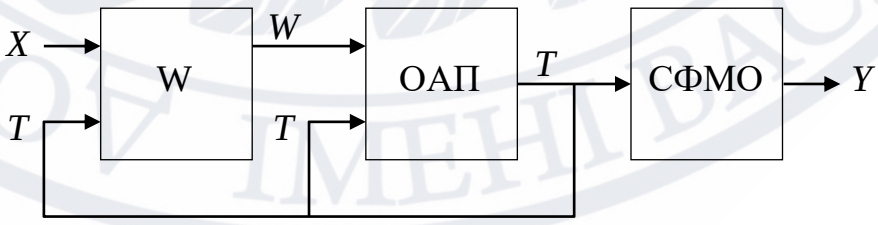

Рисунок 1.1 – Структурна схема МПА з ОАП

Як і будь-який пристрій керування, МПА з ОАП отримує ззовні певні вхідні сигнали, які на рис. 1.1 позначені символом *X*. На виході автомата формується множина вихідних сигналів *Y*, які надходять в об'єкт управління автомата (в інші блоки цифрової системи). Вхідні сигнали *X* прийнято також називати логічними умовами, вихідні сигнали *Y* – мікрооперaціями [2, 6-10, 12, 17].

Символом *T* позначений поточний стан МПА, а точніше – код поточного стану. Автомат на кожному кроці своєї роботи переходить з одного стану в інший, змінюючи значення коду *T*. В даній структурі зміна коду стану здійснюється в блоці ОАП (операційний автомат переходів) шляхом виконання над кодом *T* певної арифметичної або логічної операції. Якщо над кодом виконується арифметична операція, код інтерпретується як скалярне значення, представлене у тому чи іншому форматі (наприклад, як ціле число без знаку). Якщо над кодом виконується логічна операція, код T розглядається як вектор із *R* двійкових розрядів. Наприклад, двійковий вектор 0101_2 може ототожнюватись з беззнаковим цілим числом 5_{10}.

Операції, виконувані над кодом стану *T*, називають операціями переходів (ОП) [20-23]. Кожна ОП реалізується у вигляді комбінаційної схеми, сукупність яких утворює операційний автомат переходів. Окрім множини комбінаційних схем, до складу ОАП входить регістр пам'яті автомата, який зберігає код поточного стану *T*. В кожному такті роботи над кодом *T* виконується тільки одна із наявної множини ОП. Вибір виконуваної ОП здійснюється під керуванням спеціальних сигналів *W*. Дані сигнали називають кодом операції переходів.

Формуванням сигналів *W* займається однойменний блок W. На входи блока подаються вхідні сигнали *X* і код поточного стану *T*. Отже, блок W реалізує функцію

$$W = W(X, T) \qquad (1.1)$$

і синтезується за відповідною системою булевих рівнянь [23, 24].

Множина мікрооперацій Y формується схемою формування мікрооперацій (блок СФМО). У загальному випадку формування мікрооперацій можливе за принципом автомата Мілі або за принципом автомата Мура [2, 7, 9, 14, 18] і для вирішуваної задачі не є принциповим. В даній роботі розглядається виключно автомат Мура, для якого функція виходів реалізується відповідно до рівняння (1.2).

$$Y = Y(T). \qquad (1.2)$$

Більш докладно структура і принцип роботи МПА з ОАП розглянуті в працях [20-24]. В даній кваліфікаційній роботі робота схеми автомата розглядається лише в розрізі функції переходів, а саме в контексті перетворення кодів станів за допомогою певної множини операцій переходів. Функція виходів автомата, а також питання проектування схеми автомата, в даній роботі не розглядаються.

1.2 Способи опису мікропрограмного автомата

Вхідними даними для синтезу МПА є алгоритм керування, який має бути імплементований схемою автомата. Існують різни способи завдання алгоритму керування, одним з яких вважається так звана граф-схема алгоритму (ГСА) [2, 7, 19, 22]. ГСА представляє собою орієнтований зв’язний граф, що містить початкову, операторні, умовні та кінцеву вершини. В операторних вершинах записуються набори одночасно виконуваних мікрооперацій $Y_i \subseteq Y$, в умовних вершинах – логічні умови (вхідні сигнали) $x_j \in X$.

Розглянемо приклад ГСА, наведений на рис. 1.2. Дана ГСА (будемо позначати її символом G_1) містить множину мікрооперацій $Y = \{y_1, y_2, y_3, y_4, y_5\}$ і множину логічних умов $X = \{x_1, x_2, x_3\}$. ГСА позначена станами автомата Мура [23, 24] і містить 8 станів $a_0, ..., a_7$. Також ГСА G_1 містить $B = 12$ автоматних переходів:

$a_0 \to a_1$, $a_1 \to a_2$, $a_1 \to a_3$, $a_2 \to a_3$, $a_3 \to a_4$, $a_3 \to a_6$,
$a_4 \to a_7$, $a_5 \to a_6$, $a_6 \to a_5$, $a_6 \to a_7$, $a_7 \to a_1$, $a_7 \to a_0$.

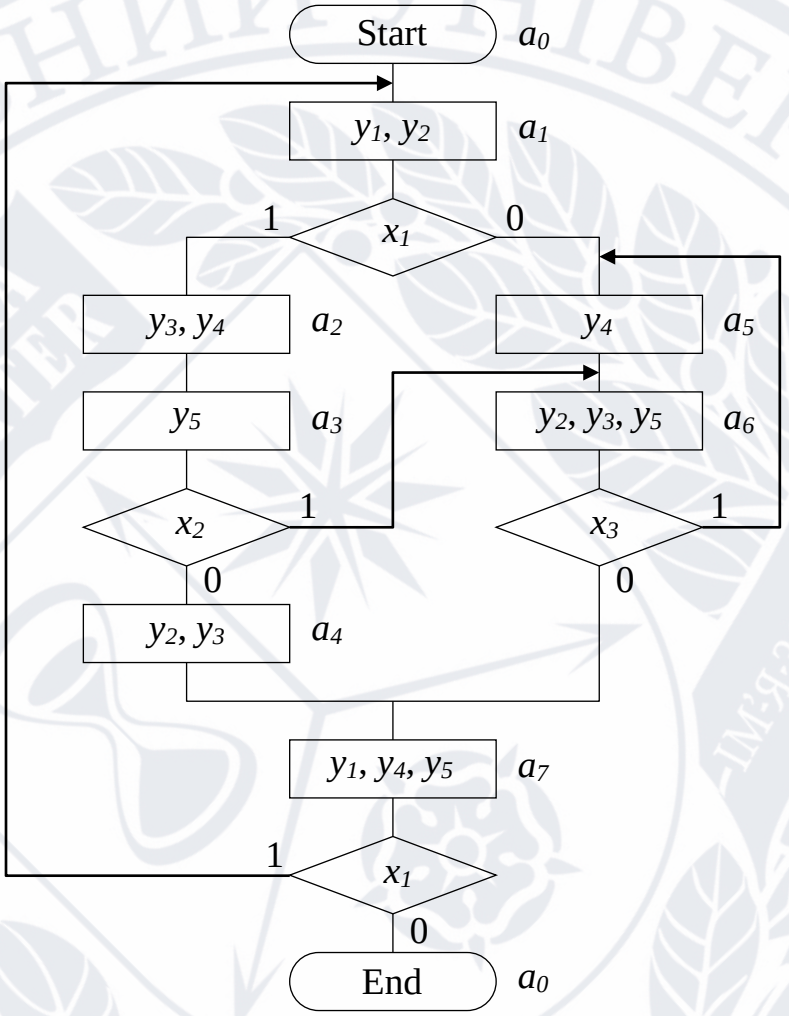

Рисунок 1.2 – Граф-схема алгоритму G_1

Частина переходів не залежать від логічних умов (є безумовними): $a_0 \to a_1$, $a_2 \to a_3$ тощо. Частина – залежать від логічних умов (умовні переходи), наприклад, $a_1 \to a_5$, $a_6 \to a_7$. Початкова і кінцева вершини позначені єдиним станом a_0, що забезпечує циклічність роботи алгоритму керування.

Отже, в будемо характеризувати довільну такими параметрами:

M – кількість станів автомата;

L – кількість аналізованих логічних умов;

N – кількість мікрооперацій;

B – кількість автоматних переходів.

Для ГСА G_1 (рис. 1.2) маємо $M = 8$, $L = 3$, $N = 5$, $B = 12$.

Графічне завдання МПА є зручним для людини, але не для комп'ютера. Автоматизоване проєктування схеми МПА потребує представлення ГСА у вигляді текстового опису, який може бути заданий у вигляді файлу. Одним із відомих форматів текстового опису ГСА є так званий формат «KISS2», який використовується у наборі тестових схем LGSynth93 [25]. Перші декілька рядків файлу визначають характеристики ГСА, після чого розташовуються рядки, що описують окремі автоматні переходи. Розглянемо структуру файлу формату KISS2 на прикладі опису ГСА G_1 (рис. 1.3).

```
1   .start_kiss
2   .i 3
3   .o 5
4   .p 12
5   .s 8
6   .r a0
7   --- a0 a1 00000
8   1-- a1 a2 11000
9   0-- a1 a5 11000
10  --- a2 a3 00110
11  -1- a3 a6 00001
12  -0- a3 a4 00001
13  --- a4 a7 01100
14  --- a5 a6 00010
15  --1 a6 a5 01101
16  --0 a6 a7 01101
17  1-- a7 a1 10011
18  0-- a7 a0 10011
19  .end_kiss
```

Рисунок 1.3 – Вміст файлу у форматі KISS2 для ГСА G_1

В рядках 1 і 19 вказані заголовок і футер файлу відповідно. Ці рядки не містять безпосереднього опису ГСА і можуть бути корисними при програмному аналізі вмісту файлу. Допускається відсутність цих рядків.

В рядку 2 вказаний параметр «.i» і його значення 3, які відповідають кількості вхідних сигналів автомата *L* (сигнали x_1, x_2, x_3).

В рядку 3 вказаний параметр «.o» і його значення 5, які відповідають кількості вихідних сигналів (мікрооперацій) *N* (мікрооперації y_1, ..., y_5).

В рядку 4 вказаний параметр «.p» і його значення 12, які відповідають кількості автоматних *B*.

В рядку 5 вказаний параметр «.s» і його значення 8, які відповідають кількості станів автомата *M*.

В рядку 6 вказаний параметр «.r» і його значення a0, які відповідають початковому стану автомата (в нашому прикладі – стан a_0). Використання цього параметру дозволяє змінювати «точку входу» в ГСА без зміни опису її структури.

Рядки 7-18 містять опис автоматних переходів. Кожен рядок складається із наступних чотирьох компонент:

1. Шаблон опису значень логічних умов (вхідних сигналів), що відповідають цьому переходу. Наприклад, в рядку 12 вказаний шаблон «–0–», який означає, що даний перехід здійснюється за умови $x_2 = 0$ незалежно від значень логічних умов x_1 та x_3. Для безумовних автоматних переходів, які не залежать від логічних умов, шаблон складається лише з прочерків (як, наприклад, в рядку 7).

2. Символьне ім’я поточного стану (з якого здійснюється перехід). В загальному випадку для позначення станів допустимо використовувати довільні унікальні набори символів, що не містять пробілів, оскільки пробіли вважаються роздільниками окремих компонент рядка. Один і той самий стан в межах усього файлу повинен позначатись одним ідентифікатором.

3. Символьне ім’я стану переходу (стану, в який здійснюється перехід). Правила його завдання такі самі, як і для імені поточного стану автомата.

4. Шаблон опису мікрооперацій, що формуються в даному переході. В кожному такті роботи МПА окрема мікрооперація може дорівнювати або нулю, або одиниці. Відповідно, цей шаблон складається лише з нулів або одиниць. Використання інших символів в шаблоні неприпустимо. Для будь-якого переходу кількість символів в шаблоні є однаковою і дорівнює кількості мікрооперацій *N* (або значенню параметра «.o», що міститься у перших рядках файлу). Також відзначимо, що у випадку автомата Мура набір формованих мікрооперацій залежить тільки від поточного стану автомата і не залежить від значень вхідних сигналів. Тому в переходах з однаковим поточним станом (наприклад, в рядках 11 і 12) використовується однаковий шаблон мікрооперацій.

1.3 Алгебраїчний синтез мікропрограмного автомата з операційним автоматом переходів

Перш, ніж проєктувати схему МПА з ОАП, необхідно провести так званий алгебраїчний синтез автомата [23, 24]. Він може бути виконаний в різний спосіб із різним результатом. Від результату алгебраїчного синтезу залежить таке:

– яка кількість операцій переходів буде використана (чим ця кількість менша, ти менша складність схеми автомата);

– яка кількість автоматних переходів буде реалізована за допомогою операцій переходів (чим ця кількість більша, тим менша складність схеми автомата);

– якою буде розрядність коду стану (чим розрядність менша, тим менша складність схеми);

– які саме коди будуть використані для кодування станів (впливає на синтез схеми формування мікрооперацій);

– які саме операції переходів будуть використані (чим операції простіші, тим менша складність схеми автомата).

В процесі алгебраїчного синтезу відбуваються такі дії.

1. Станам автомата ставляться у відповідність унікальні коди із певної множини кодів станів.

2. Автоматним переходам ставляться у відповідність певні операції переходів. Використання однієї ОП для реалізації декількох автоматних переходів є припустимим і сприяє зменшенню загальної кількості використовуваних ОП та, відповідно, зменшенню апаратурних витрат в схемі ОАП. Ті переходи, які не можуть бути реалізовані жодною із заданих ОП, в подальшому реалізуються в канонічний спосіб за системою булевих рівнянь.

Розглянемо даний процес на прикладі. Нехай МПА з ОАП заданий ГСА G_1 (рис. 1.2). Також задані три операції переходів O_1, O_2 і O_3, які визначаються виразами (1.3) – (1.5) відповідно.

$$O_1: \quad K(a_j) = (K(a_i) + 1_{10}); \tag{1.3}$$

$$O_2: \quad K(a_j) = (K(a_i) + 2_{10}); \tag{1.4}$$

$$O_3: \quad K(a_j) = (K(a_i) \oplus 101_2). \tag{1.5}$$

А даних виразах $K(a_i)$ відповідає коду поточного стану, $K(a_j)$ – коду стану переходу. В кожній операції над кодом поточного стану виконується певна арифметична або логічна операція. Так, в ОП O_1 до коду поточного стану додається десяткова константа 1, в ОП O_2 додається десяткова константа 2, в ОП O_3 над кодом поточного стану відбувається операція XOR з двійковою константою 101.

ГСА G_1 містить $M = 8$ станів, для кодування яких достатньо три двійкових розряди. Отже, у виразах (1.3) і (1.4) коди $K(a_i)$ і $K(a_j)$ повинні «укладатись» у три розряди, тобто в діапазон чисел [0; 8]. Математично це виражається в тому, що над результатом будь-якої ОП повинна виконуватись операція «mod 8», тобто взяття остачі від ділення результату націло на 8. Для ОП O_1 і O_2 це можна формально записати у вигляді виразів (1.6) і (1.7) відповідно. У виразі (1.3) над кодом поточного стану виконується порозрядна логічна операція XOR, тож потреби у виконанні «mod 8» немає.

$$O_1: \quad K(a_j) = (K(a_i) + 1_{10}) \bmod 8; \qquad (6)$$

$$O_2: \quad K(a_j) = (K(a_i) + 2_{10}) \bmod 8; \qquad (7)$$

Відзначимо, що операція «mod 8» не потребує додаткових витрат на реалізацію схеми ОП і досягається тим, що усі старші розряди результату відкидаються, залишаючи молодші R розрядів в якості коду стану переходу.

Задані ОП можуть бути використані для реалізації переходів автомата. Операція переходів O_k реалізує перехід зі стану a_i в стан a_j в тому випадку, коли код $K(a_i)$ перетворюється в код $K(a_j)$ за допомогою цієї ОП. На рис. 1.4 показаний фрагмент ГСА G_1, який відображає переходи зі стану a_1. На фрагменті всередині операторних вершин замість мікрооперацій показані десяткові коди станів і їх двійкові еквіваленти. Гілці кожного переходу зіставлена певна ОП. Переходу $a_1 \to a_2$ зіставлена ОП O_1 (умовно показана як «+ 1»), переходу $a_1 \to a_5$ зіставлена ОП O_3 (умовно показана як «⊕ 101»).

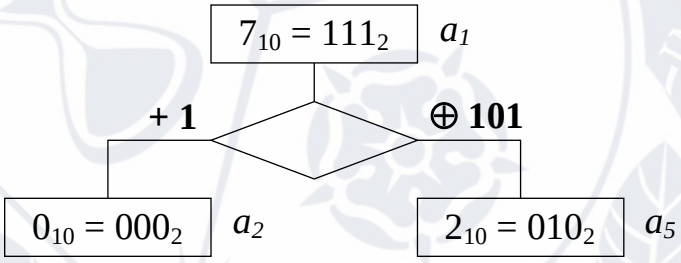

Рисунок 1.4 – Приклад операційної реалізації переходів

На рис. 1.4 також не показаний вміст умовної вершини та умови переходів по її гілкам. Це пов’язане з тим, що для алгебраїчного синтезу важлива лише можливість реалізації того чи іншого переходу за допомогою заданої множини ОП. Саме набір використовуваних ОП визначає структуру операційного автомата переходів. Аналіз логічних умов буде здійснюватись окремо блоком W (рис. 1.1) відповідно до результатів алгебраїчного синтезу.

Рис. 1.4 показує таке. Перехід $a_1 \to a_2$ за своєю суттю є перетворенням десяткового коду $K(a_1) = 7_{10}$ у десятковий код $K(a_2) = 0_{10}$. Це можливо за допомогою ОП O_1, оскільки

$$K(a_2) = (K(a_1) + 1_{10}) \bmod 8 = (7_{10} + 1_{10}) \bmod 8 = 8 \bmod 8 = 0.$$

Відповідно, перехід $a_1 \to a_5$ є перетворенням двійкового коду $K(a_1) = 111_2$ у двійковий код $K(a_5) = 010_2$. Це можливо за допомогою ОП O_3, оскільки

$$K(a_5) = K(a_1) \oplus 101_2 = 111_2 \oplus 101_2 = 010_2.$$

Таким чином, у фрагменті ГСА на рис. 1.4 обидва переходи можуть бути реалізовані за допомогою заданих ОП. Така реалізація переходів називається операційною реалізацією переходів або операційним перетворенням кодів станів [ ].

Якщо у фрагменті на рис. 1.4 для стану a_5 обрати код $K(a_5) = 4_{10} = 100_2$, то перехід $a_1 \to a_5$ не можна буде реалізувати жодною з ОП (1.3) – (1.5). Аби цей перехід був операційно реалізований, доведеться змінювати або код стану a_1, або набір заданих операцій переходів. Обидва ці способи можуть привести до того, що в межах усієї ГСА деякі інші переходи не зможуть отримати операційну реалізацію.

Під час алгебраїчного синтезу коди станів і використовувані ОП обираються проєктувальником пристрою. Перед проєктувальником постає наступна задача: які коди призначити станам автомата і які ОП призначити автоматним переходам, щоб усі переходи автомата мали операційну реалізацію? Ця задача називається задачею алгебраїчного синтезу МПА з ОАП і на сьогоднішній день не має оптимального способу розв’язання [ ]. Розглянемо приклад розв’язання задачі алгебраїчного синтезу МПА з ОП, заданого ГСА G_1, і множини ОП, що утворена ОП (1.3) – (1.5): $O = \{O_1, O_2, O_3\}$ (рис. 1.5).

Розв’язок, наведений на рис. 1.5, називають формальним розв’язком задачі алгебраїчного синтезу МПА з ОАП [23, 24, 26]. Термін «формальний» тут означає, що даний розв’язок не гарантує зменшення вартості схеми автомата у порівнянні з іншими відомими методами синтезу МПА. Формальність розв’язку лише підкреслює, що за заданих умов усі переходи автомата можуть бути реалізовані в операційний спосіб.

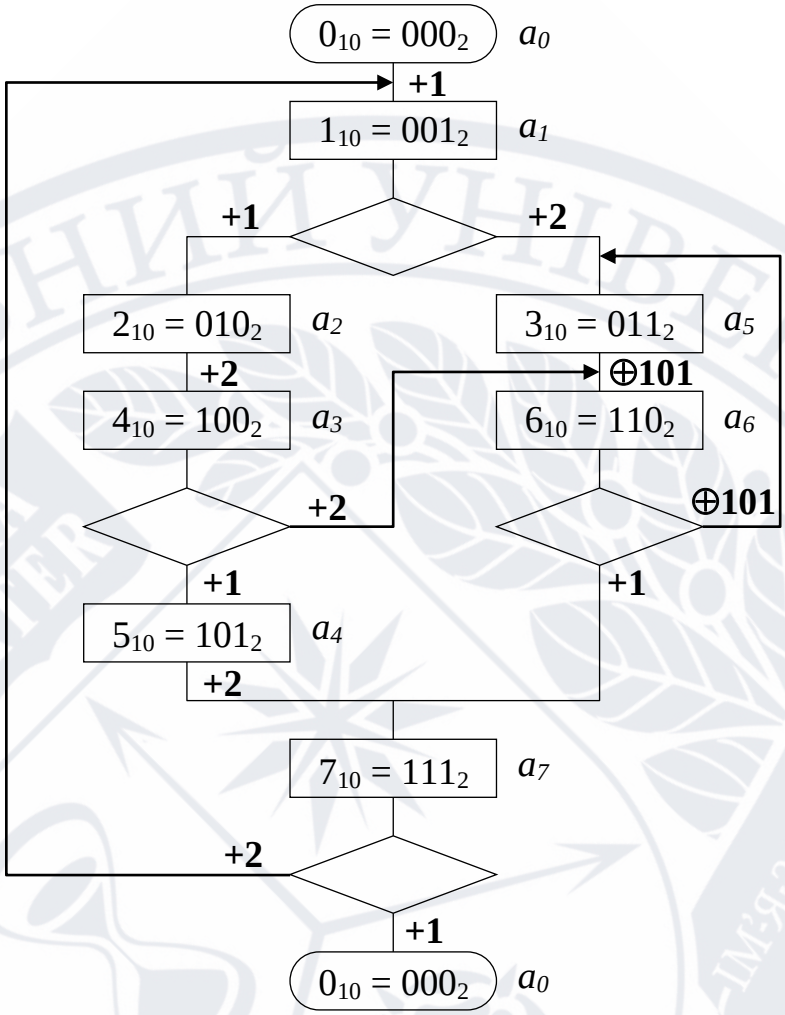

Рисунок 1.5 – Графічне представлення результату алгебраїчного синтезу МПА з ОАП за ГСА G_1 для ОП (1.3) – (1.5)

В загальному випадку для заданих ГСА і множини ОП можливе існування декількох формальних розв'язків задачі алгебраїчного синтезу МПА з ОАП. Рисунок 1.6 демонструє ще два розв'язки, подібні до розв'язку на рис. 1.5. Усі ці розв'язки можна вважати тотожними, оскільки для будь-якого з них схема операційного автомата переходів буде містити блоки трьох операцій – «+1», «+2» і «⊕ 101» – і мати однакову складність.

Усі наведені формальні розв'язки вимагають спеціального кодування станів. Якщо стани закодувати інакше, формальний розв'язок можна не отримати. Наприклад, при $K(a_0) = 3_{10} = 011_2$ і $K(a_1) = 7_{10} = 111_2$ перехід $a_0 \to a_1$ не може бути реалізований за допомогою жодної з ОП $O_1 – O_3$.

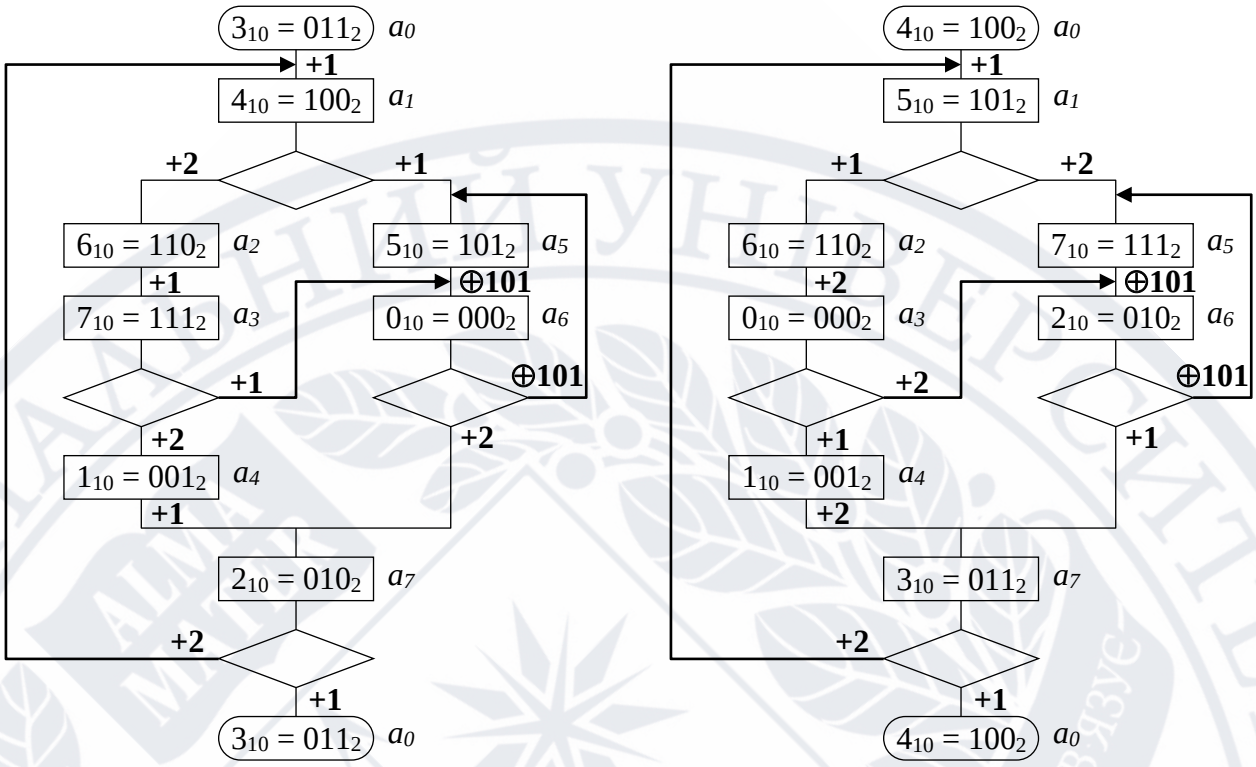

Рисунок 1.6 – Додаткові приклади формальних розв’язків

Проблема пошуку формальних розв’язків задачі алгебраїчного синтезу є нетривіальною. Невідомо, які операції переходів треба обрати та як закодувати стани автомата, щоб:

– кількість ОП була мінімально можливою;
– обрані ОП були максимально простими за схемною реалізацією;
– усі переходи автомата мали операційну реалізацію.

В роботі [26] показано, що ситуація, коли декілька автоматних переходів не мають операційної реалізації, також може вважатись формальним розв’язком задачі алгебраїчного синтезу. На рис. 1.7 наведені два приклади кодування станів з переходами, які не можуть бути реалізовані за допомогою ОП (1.3) – (1.5). На рис. 1.7, *а)* немає операційної реалізації для переходу $a_7 \to a_0$, на рис. 1.7, *б)* немає операційної реалізації для переходів $a_2 \to a_3$ і $a_6 \to a_5$. Наявність операційно нереалізованих переходів не є бажаною, оскільки для їх реалізації потрібен додатковий блок в схемі ОАП, який збільшує складність і вартість схеми. Будемо називати розв’язок задачі алгебраїчного синтезу, в якому є операційно нереалізовані переходи,

частковим розв'язком задачі алгебраїчного синтезу МПА з ОАП. Натомість, розв'язок, в якому усі переходи реалізуються операційно, називатимемо повним розв'язком задачі алгебраїчного синтезу.

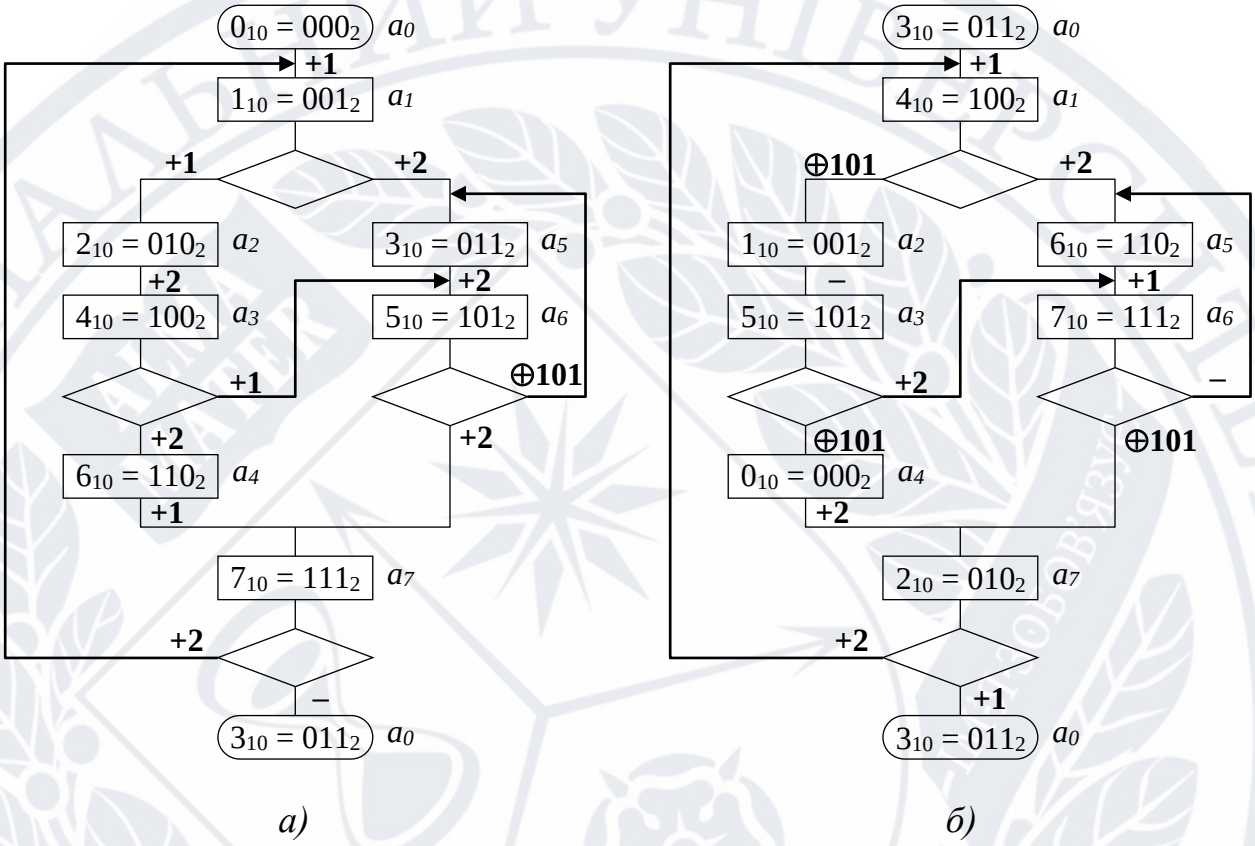

Рисунок 1.7 – Формальні розв'язки задачі алгебраїчного синтезу з одним (а) і двома (б) нереалізованими переходами

Будь-який операційно нереалізований автоматний перехід завжди можна реалізувати операційним шляхом, додавши до множини ОП додаткову операцію. Наприклад, для реалізації переходу $a_7 \to a_0$ в ГСА на рис. 1.7, *а)* можна використати операцію «⊕ 100» або «+4» або якусь іншу, яка здатна перетворити код $K(a_7) = 7_{10} = 111_2$ в код $K(a_0) = 3_{10} = 011_2$. Ця операція повинна бути додана до множини ОП, а її схема має бути додана до складу операційного автомата переходів. Як наслідок, схема ОАП стане більш складною, і це лише заради одного автоматного переходу. Додавання нової ОП у множину ОП зазвичай доцільно в тих випадках, коли:

– ОП, що додається, реалізує велику кількість автоматних переходів;

– ОП, що додається, реалізує малу кількість автоматних переходів на фоні дуже великої загальної кількості переходів автомата, які вже реалізовані операційним шляхом.

Якщо б не існувало обмежень на кількість і тип використовуваних ОП, алгебраїчний синтез будь-якої ГСА був би тривіальною задачею. Можна було б обрати довільні коди станів (унікальні в межах заданої ГСА), після чого для кожного переходу підібрати підходящу ОП, яка забезпечує перетворення коду поточного стану в код стану переходу. На рис. 1.8 наведений приклад тривіального розв'язання задачі алгебраїчного синтезу для ГСА G_1.

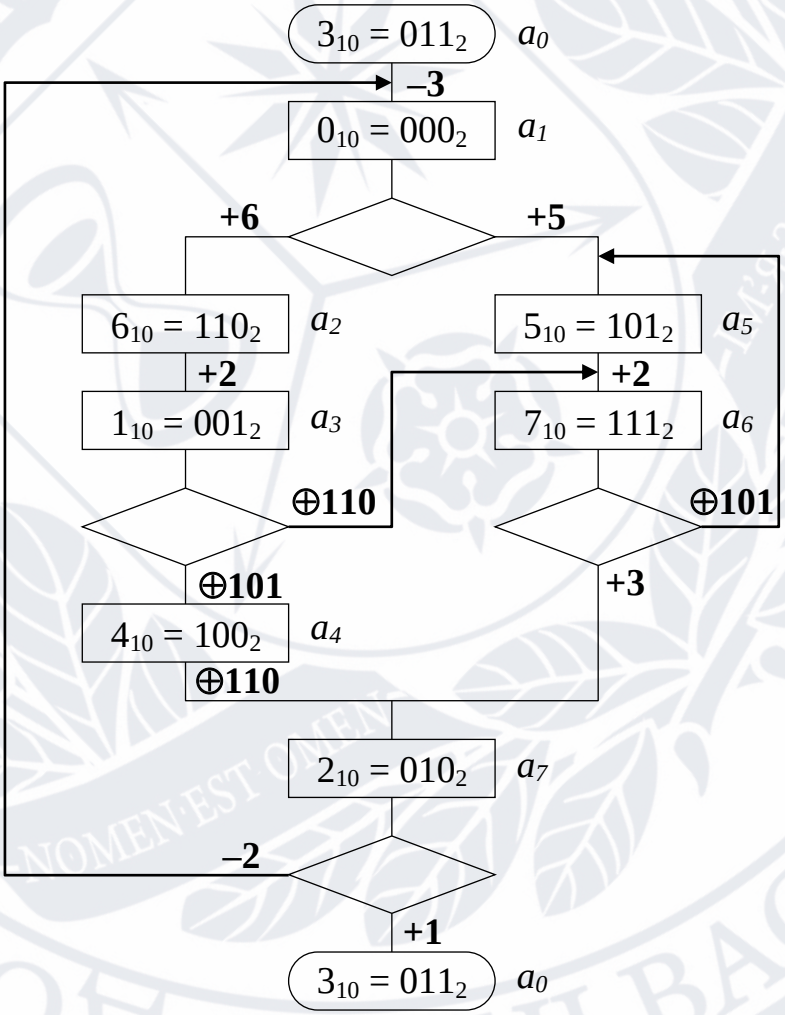

Рисунок 1.8 – Приклад тривіального розв'язку задачі алгебраїчного синтезу для ГСА G_1

Як можна бачити, для реалізації 12 автоматних переходів використані дев’ять операцій: «–3», «+6», «+5», «⊕110», «⊕101», «+2», «+3», «–2», «+1». Можна умовно вважати, що складність схеми автомата в даному випадку буде у 3 рази складніше, ніж у випадку використання трьох ОП (рис. 1.5).

Отже, актуальною науково-практичною проблемою вважається пошук такої множини ОП, яка дозволяє реалізувати якнайбільшу кількість автоматних переходів при якнайменшій кількості ОП.

1.4 Вибір засобів розробки

Алгебраїчний синтез мікропрограмного автомата з операційним автоматом переходів є математичною задачею. Розв’язання цієї задачі «вручну» можливе лише для ГСА невеликого розміру, тоді як великі ГСА слід аналізувати на комп’ютері. Потрібно враховувати те, що на сьогодні відсутні відомі методи і алгоритми алгебраїчного синтезу МПА з ОАП. Для розробки таких алгоритмів, а також для їх тестування і практичного застосування доцільно використовувати високорівневі мови програмування, що мають розвинутий математичний апарат.

В наші дні однією з таких мов є Python. Це вільно розповсюджувана універсальна мова програмування з великою кількістю вільно розповсюджуваних модулів і бібліотек для створення програм майже будь-якого типу (наукові обчислення, інженерні розрахунки, робота з базами даних, візуалізація даних, проєктування користувацьких інтерфейсів, веб-додатки тощо). Синтаксично Python – інтерпретована, об’єктно-орієнтована високорівнева мова програмування сценаріїв із динамічною семантикою. Розвинені внутрішні структури даних у поєднанні з динамічною типізацією та динамічним зв’язуванням роблять її дуже привабливою для швидкої розробки застосунків, а також для використання в якості скриптової мови, що «склеює» разом наявні компоненти.

Вважається, що мова Python має низький поріг входження і її неважко вивчити, через що можна використовувати Python для розв'язання поставлених задач через нетривалий час підготовки. Синтаксис Python є простим і зручним для розуміння. Ця мова близька до мови, вбудованої в Matlab, і є зручною для програмування математичних обчислень. До того ж, Python вміє працювати з такими мовами, як Fortran, C, C++, які вже тривалий час використовуються при проведенні розрахунків.

Можна виділити такі математичні можливості мови Python:

– існує чотири вбудованих типи числових даних: булеві значення, цілі числа, дійсні числа (числа з плаваючою крапкою) та комплексні числа;

– підтримуються усі математичні операції і дії;

– потужна стандартна бібліотека Python дозволяє задовольнити найбільш розповсюджені потреби користувачів, наприклад, завантажити файл з Інтернету, розпакувати архів

– доступні спеціалізовані математичні модулі: math, random, decimal, fractions, numbers та інші;

– доступні бібліотеки для складних обчислень і візуалізації даних: Numpy, Numeric, Pandas, Matplotlib, Dislim та інші.

Між тим, до недоліків Python порівняно з іншими мовами програмування можна віднести наступні:

– низька продуктивність за рахунок інтерпретації команд замість компіляції;

– обмежена підтримка багатопотоковості;

– слабка підтримка розробки мобільних додатків.

Мова Python створена нідерландським програмістом Гвідо ван Россумом (Guido van Rossum) 1991 році. У 2008 році вийшов реліз Python 3.0, який вніс значні зміни у розвиток мови. Наразі актуальною версією вважається Python 3.13, який вийшов 7 жовтня 2024 р. Мова Python випускається за безкоштовною ліцензією і може вільно використовуватись у навчальному процесі при підготовці фахівців з комп'ютерних наук.

Підтримка швидких математичних обчислень в Python здійснюється, зокрема, за допомогою бібліотеки Numpy. Ця бібліотека не входить до стандартної поставки Python і повинна бути інстальована окремо. Бібліотека має ряд особливостей, які зробили її популярним математичним інструментом.

По-перше, її вихідний код зберігається на GitHub у вільному доступі, тому Numpy називають open-source модулем Python.

По-друге, функції бібліотеки написані на мовах C++ і Fortran, за рахунок чого обчислення відбуваються значно швидше, аніж стандартними засобами Python.

Numpy використовується в наступних сферах:

1. Наукові обчислення. Бібліотекою користуються вчені для розв'язання багатовимірних задач з математики, фізики, біоінформатики, обчислювальної хімії і навіть когнітивної психології.

2. Data Science. Бібліотека може використовуватись на всіх етапах роботи з даними: отримання та перетворення, очищення, аналіз, моделювання і оцінка, репрезентація.

3. Machine Learning. Бібліотеки для машинного навчання Scikit-learn та SciPy також працюють завдяки обчислювальним потужностям Numpy.

4. Візуалізація даних. Якщо порівнювати безпосередньо з Python, можливості Numpy дозволяють дослідника візуалізовувати набори даних, які є значно більшими за обсягом. Наприклад, Numpy лежить в основі системи PyViz, яка включає в себе десятки програм для візуалізації.

Цифрові графічні зображення можна представити у вигляді простих двовимірних масивів чисел, що відображають яскравість пікселів відповідної області; аудіо записи представляються як одновимірні масиви інтенсивності звуку у відповідні моменти часу; текст можна перетворити у числове представлення різними шляхами. Незалежно від характеру даних першим кроком до їх аналізу є перетворення в числові масиви. Ефективне зберігання і робота з числовими масивами дуже важливі для процесу дослідження даних.

Бібліотека Numpy містить багатовимірні масиви і матричні структури даних. Numpy можна використовувати для виконання множини математичних операцій з масивами. На відміну від стандартних списків Python, усі дані в масивах Numpy мають бути однорідними, тобто належати до одного типу. За рахунок однорідності даних масиви Numpy є більш ефективними і компактними порівняно зі списками Python. Масиви споживають менше пам’яті, що дуже важливо для оптимізації швидкості обчислень та використовуваної потужності процесора.

В даній кваліфікаційній роботі для реалізації алгоритмів пошуку формальних розв’язків задачі алгебраїчного синтезу будемо використовувати мову Python разом із бібліотекою Numpy.

1.5 Постановка основних задач дослідження

На сьогоднішній день методи і алгоритми алгебраїчного синтезу мікропрограмного автомата з операційним автоматом переходів залишаються нерозробленими. За відсутності алгоритмів відсутня програмна реалізація алгебраїчного синтезу. Це не дозволяє автоматизувати процес проєктування цифрових систем на базі МПА з ОАП. Як наслідок, зменшується область використання даного класу мікропрограмних автоматів.

Складність алгебраїчного синтезу полягає у його широкій варіативності. Окрім алгоритму керування, що задається у вигляді ГСА або в інший спосіб, немає жодних початкових умов для проведення синтезу.

По-перше, дозволяється обирати будь-які операції переходів. Кількість операцій також може бути довільною. В деяких випадках для реалізації усіх переходів автомата може бути достатньо використати три-чотири ОП. В інших випадках може бути недостатньо і десяти ОП. Враховуючи те, що великі ГСА можуть містити десятки і сотні вершин, маючи при цьому різноманітну структуру, не можливо визначити жодних правил для формування ОП. Якщо вважати, що коди станів мають інтерпретацію цілих

чисел без знаку, над ними можна виконувати такі операції, як додавання константи, віднімання константи, множення або ділення на константу, а також декілька подібних дій одразу. Яку саме константу використовувати в кожному випадку – також є невідомим. Якщо код стану складається із 10 двійкових розрядів, константа може варіюватись від 1 до 2^{10}–1. При цьому використання різних констант з однією і тією ж операцією (наприклад, з операцією додавання) породжує різні операції переходів.

Окрім арифметичних операцій, можливе використання різноманітних логічних операцій. До них належать кон'юнкція з двійковою константою, диз'юнкція з константою, XOR з константою, інверсія, зсув вліво або вправо, циклічний зсув, заміна або переставлення окремих розрядів тощо. Так само використання кожної конкретної константи породжує окрему ОП. Наприклад, операції «$\oplus\ 1011_2$» та «$\oplus\ 0101_2$» вважаються різними. Якщо розрядність коду стану $R = 10$, можливе використання будь-якої із (2^{10}–1) двійкових констант. Більш того, допустимо створювати складені операції, в яких над кодом стану спочатку виконується, наприклад, диз'юнкція з константою, після чого результат зсувається циклічно на два розряди вліво. Теоретично можна створювати подібні послідовності із більш ніж двох операцій.

Оскільки кожна ГСА має власну структуру, неможливо по вигляду ГСА встановити, які операції переходів будуть більш затребуваними в даному конкретному випадку. Однак чим більше ОП буде обрано і чим більш складними вони будуть, тим складнішою буде результуюча схема МПА з ОАП. Отже, стратегія вибору ОП полягає в тому, щоб знайти мінімально достатню або близьку до неї множину ОП для кожної заданої ГСА.

По-друге, невідомо, яким чином обирати коди станів автомата. Якщо множина ОП вже задана, кодування станів слід проводити так, щоб забезпечити операційну реалізацію усіх або більшості автоматних переходів (як, наприклад, на рис. 1.4). Якщо множина ОП не задана, можна спочатку задати певні коди станів, згідно з якими підібрати множину ОП (наприклад,

як на рис. 1.8). Також теоретично можливо об’єднати процеси завдання кодів станів і вибору ОП в один процес, в ході якого кодування станів і підбір ОП здійснюються поступово в межах усієї ГСА.

Коди станів обираються із певної множини, яка містить усі допустимі коди станів. Якщо ГСА містить 10 станів, для їх кодування достатньо 4 двійкових розряди. Отже, множина допустимих кодів станів містить 2^4=16 кодів, від 0 до 15. Кожен стан автомата може бути закодований одним із цих кодів, причому в межах ГСА коди станів не повинні повторюватись. Між тим, для кодування 10 станів можна використати не 4, а 5 двійкових розрядів (або більше). За умови 5 двійкових розрядів множина допустимих кодів станів міститиме не 16, а 32 різних коди. З одного боку, більша кількість допустимих кодів станів збільшує кількість допустимих варіантів кодування. Зі іншого боку, такий підхід може спростити операційну реалізацію автоматних переходів при фіксованому наборі ОП. Таким чином, питання кодування станів також не є однозначним. Не існує очевидного способу оптимального кодування станів в кожному конкретному випадку.

Усі розглянуті підходи потребують розробки окремих алгоритмів алгебраїчного синтезу та їх подальшої програмної реалізації. Цій проблематиці присвячена дана кваліфікаційна робота. В її межах необхідно розв’язати такі задачі:

1. Розробка алгоритму пошуку повних формальних розв’язків задачі алгебраїчного синтезу МПА з ОАП за допомогою повного перебору способів кодування станів при заданій множині ОП.

2. Модифікація алгоритму, розробленого в п. 1, для пошуку часткових формальних розв’язків задачі алгебраїчного синтезу. Під частковими розуміються розв’язки, в яких один або більше автоматних переходів не отримують операційної реалізації в межах заданої множини ОП.

3. Розробка програмної реалізації цих алгоритмів.

4. Дослідження розроблених алгоритмів і їх програмної реалізації на тестових граф-схемах алгоритмів.

Для розв’язання зазначених задач заплановано використовувати мову програмування Python разом із математичною бібліотекою Numpy.

РОЗДІЛ 2
АЛГОРИТМИ АЛГЕБРАЇЧНОГО СИНТЕЗУ МІКРОПРОГРАМНОГО АВТОМАТА З ОПЕРАЦІЙНИМ АВТОМАТОМ ПЕРЕХОДІВ

2.1 Оцінка складності алгоритмів на основі повного перебору

Враховуючи значну варіативність вхідних даних для проведення алгебраїчного синтезу МПА з ОАП, введемо наступні обмеження:

1. Множину операцій переходів будемо вважати заданою. Нехай множина ОП містить I операцій переходів: $O = \{O_1, ..., O_I\}$. В якості ОП можуть використовуватись арифметичні або логічні операції на кодом поточного стану автомата.
2. Розрядність кодів станів автомата будемо вважати мінімально достатньою для кодування M станів і рівною, відповідно, $R = \lceil log_2 M \rceil$, тобто округлену до найближчого більшого цілого.
3. Множина допустимих кодів станів автомата визначається значеннями, що лежать в діапазоні $[0; 2^R–1]$. Кожне число з цього діапазону має десяткову інтерпретацію та двійкову інтерпретацію у вигляді цілого R-розрядного числа без знаку.

Нехай задана ГСА містить M станів, для кодування яких достатньо R двійкових розрядів. Кількість припустимих кодів станів, кожен з яких може бути зіставлений одному із M станів автомата, дорівнює 2^R. Кількість варіантів зіставлення 2^R припустимих кодів M станам можна визначити виразом (2.1) згідно з комбінаторною формулою розміщення без повторів:

$$N_1 = \mathrm{A}_{2^R}^{M} = \frac{(2^R)!}{(2^R - M)!}. \qquad (2.1)$$

У випадку ГСА G_1 з параметрами $M = 8$, $R = 3$ маємо $N_1 = 40320$ варіантів кодування. Для ГСА з $M = 100$ станами $N_1 = 1{,}26*10^{186}$.

Нехай задана ГСА містить B переходів, кожному з яких може бути зіставлена одна з N_I операцій переходів. Оскільки в МПА з ОАП окрема ОП

може бути зіставлена будь-якій кількості переходів, для визначення кількості таких зіставлень слід скористатись виразом (2.2) згідно з комбінаторною формулою розміщень з повторами.

$$N_2 = (N_I)^B. \qquad (2.2)$$

Для ГСА G_1 при $B = 12$, $N_I = 3$ маємо $N_2 = 531\,441$ варіант зіставлення. Якщо ГСА містить $B = 100$ переходів, а множина ОП містить $N_I = 10$ операцій, кількість способів зіставлення ОП автоматним переходам дорівнюватиме $N_2 = 10^{100}$, причому додавання одного автоматного переходу збільшуватиме це число у десять разів.

У найпростішому випадку процес кодування станів може бути не пов'язаний із зіставленням ОП переходам автомата. Для кожного з N_1 можливих варіантів кодування станів можливе одне з N_2 зіставлень операцій автоматним переходам. І навпаки: для кожного з N_2 варіантів зіставлення ОП можливі N_1 способів кодування станів. При цьому загальна кількість можливих ситуацій N_V дорівнює добутку значень N_1 і N_2:

$$N_V = \frac{(2^R)!}{(2^R - M)!} \times (N_I)^B. \qquad (2.3)$$

Вираз (2.3) визначає кількість варіантів повного перебору, кожен з яких може виявитись формальним розв'язком задачі алгебраїчного синтезу. Для ГСА G_1 із 12 переходами і множини із трьох ОП кількість таких варіантів дорівнюватиме близька 21.5 мільярди, що само по собі є дуже великим числом. Для ГСА більшого розміру або більшої кількості ОП значення N_V має тенденцію до нелінійного зростання.

2.2 Метод виявлення формального розв'язку задачі алгебраїчного синтезу

Вираз (2.3) визначає, скільки існує варіантів, щоб розмістити в заданій ГСА коди станів та операції переходів. Після того, як черговий спосіб

отриманий (станам надані певні коди, а переходам зіставлені певні ОП), необхідно перевірити, чи є даний спосіб формальним розв'язком задачі алгебраїчного синтезу МПА з ОАП. Для цього треба виконати процедуру перевірки усіх автоматних переходів на «коректність», тобто для кожного переходу переконатись, що код поточного стану перетворюється у код стану переходу за допомогою ОП, що зіставлена цьому переходу. З математичної точки зору при перевірці кожного переходу треба взяти код поточного стану, підставити його у вираз відповідної ОП, обчислити вираз і порівняти результат із кодом стану переходу. Якщо усі переходи виявляться «коректними», формальний розв'язок знайдений.

Назвемо цю процедуру виявленням формального розв'язку задачі алгебраїчного синтезу МПА з ОАП. Хоча процедура доволі проста, вона вимагає багаторазового виконання арифметико-логічних операцій, що утворюють множину ОП. У випадку повного перебору така процедура повинна проводитись щоразу для кожного із варіантів перебору, що може вимагати значних обчислювальних ресурсів. Для спрощення процесу виявлення формального розв'язку задачі алгебраїчного синтезу може бути використаний наступний підхід.

Представимо систему переходів заданого МПА у вигляді квадратної матриці, організованої в такий спосіб:

1. Кількість рядків і кількість стовпців дорівнюють кількості припустимих кодів станів, тобто значенню 2^R. Рядки і стовпці позначаються послідовними кодами від 0 до 2^R–1.

2. Рядку і стовпцю з однаковим кодом зіставляється стан автомату, що закодований цим кодом. Стан вказується ліворуч від номеру рядка та над номером стовпця. Якщо якийсь код не використовується для кодування станів, біля відповідного рядка і стовпця стан автомату не вказується.

3. В комірці (i, j) ставиться одиниця, якщо існує перехід зі стану з кодом, вказаним в рядку i, у стан з кодом, вказаним в стовпці j. Якщо такого

переходу не існує, в комірці записується нуль (або комірка залишається порожньою).

Приклад такої матриці для ГСА G_1, де коди станів задані згідно рис. 1.5, наведений на рис. 2.1. Така матриця називається *матрицею переходів*, оскільки вона відображає множину переходів МПА.

		a_0	a_1	a_2	a_5	a_3	a_4	a_6	a_7
		0	1	2	3	4	5	6	7
a_0	0		1						
a_1	1			1	1				
a_2	2					1			
a_5	3							1	
a_3	4						1	1	
a_4	5								1
a_6	6				1				1
a_7	7	1	1						

Рисунок 2.1 – Матриця переходів для ГСА G_1

В рядку 4 вказані переходи зі стану з кодом 4, тобто зі стану a_3. Оскільки зі стану a_3 є два умовних переходи, рядок містить дві одиниці, що відповідають переходам в стан a_4 з кодом 5 та в стан a_6 з кодом 6. Стовбець 3 містить дві одиниці, оскільки в стан a_5 з кодом 3 є два переходи: зі стану a_1 з кодом 1 та зі стану a_6 з кодом 6. Якщо б якийсь код не використовувався для кодування станів (наприклад, станів було б 7, а допустимих кодів – 8), то рядок і стовпець, що відповідають цьому коду, не містили б одиниць.

Нехай задана деяка ОП. Представимо її у вигляді квадратної матриці, організованої наступним чином:

1. Матриця є квадратною, і її розміри матриці співпадають з розмірами матриці переходів. Рядки і стовпці позначені послідовними кодами від 0 до 2^R.

2. В комірці (*i*, *j*) записується одиниця, якщо значення *i* перетворюється за допомогою заданої ОП у значення *j*. В іншому випадку в комірці записується нуль (або комірка залишається порожньою). Оскільки ОП повинна являти собою відношення функціонального типу, в кожному рядку може бути не більше одного одиничного значення.

Назвемо матрицю подібного типу *матрицею операції*. Приклади таких матриць для ОП O_1, O_2, O_3, що задаються виразами (1.3) – (1.5), наведені на рис. 2.2 – 2.4 відповідно.

	0	1	2	3	4	5	6	7
0		1						
1			1					
2				1				
3					1			
4						1		
5							1	
6								1
7	1							

Рисунок 2.2 – Матриця операції «+1»

	0	1	2	3	4	5	6	7
0			1					
1				1				
2					1			
3						1		
4							1	
5								1
6	1							
7		1						

Рисунок 2.3 – Матриця операції «+2»

	0	1	2	3	4	5	6	7
0						1		
1					1			
2								1
3							1	
4		1						
5	1							
6				1				
7			1					

Рисунок 2.4 – Матриця операції «⊕ 101»

Пояснимо вміст матриці на рис. 2.4. Рядок 3 відповідає двійковому коду 011_2. При виконанні операції «⊕ 101» над кодом 011_2 ми отримуємо двійковий код 110_2, який відповідає десятковому числу 6. Отже, в рядку 3

одиниця ставиться в стовпці 6, оскільки операція «⊕ 101» перетворює число 3 в число 6.

На основі матриць операцій можна побудувати так звану об’єднану матрицю операцій, розмір якої співпадає з розміром матриць операцій. Вміст кожної комірки формується за наступним принципом: якщо хоча б в одній матриці операції (рис. 2.2 – 2.4) відповідна комірка містить одиницю, в комірку об’єднаної матриці операцій записується одиниця; якщо в усіх матрицях операцій відповідні комірки містять нулі (є порожніми), комірка об’єднаної матриці операцій дорівнюватиме нулю (залишається порожньою). Даний принцип схожий на виконання логічної операції диз’юнкції над значеннями відповідних комірок усіх матриць операцій.

Об’єднана матриця операцій, побудована на основі матриць операцій, зображених на рис. 2.2 – 2.4, представлена на рис. 2.5.

	0	1	2	3	4	5	6	7
0		1	1			1		
1			1	1	1			
2				1	1			1
3					1	1	1	
4		1				1	1	
5	1						1	1
6	1			1				1
7	1	1	1					

Рисунок 2.5 – Об’єднана матриця операції для ОП $O_1 – O_3$.

Можливість побудови об’єднаної матриці операцій виходить з того, що під час алгебраїчного синтезу дозволяється зіставити будь-якому автоматному переходу довільну ОП. Немає значення, за допомогою якої ОП

буде реалізований той чи інший перехід. Головне, щоб він був реалізований операційним шляхом. Таким чином, об'єднана матриця операцій являє собою «мапу» усіх можливих перетворень кодів станів за допомогою заданої множини операцій переходів.

Скористаємось об'єднаною матрицею операцій для виконання процедури виявлення формального рішення задачі алгебраїчного синтезу МПА з ОАП. Для цього виконаємо зіставлення матриці переходів (рис. 2.1) і об'єднаної матриці операцій (рис. 2.5). Алгоритм зіставлення буде таким:

1. Якщо в комірках (*i*, *j*) обох матриць записані одиниці, перехід зі стану з кодом *i* в стан з кодом *j* може бути реалізований за допомогою мінімум однієї із заданих ОП. Такий перехід будемо вважати покритим заданою множиною ОП, тобто таким, що може бути реалізований операційно.

2. Якщо в комірці (*i*, *j*) матриці переходів записана одиниця, а в комірці (*i*, *j*) об'єднаної матриці операцій записаний нуль, перехід зі стану з кодом *i* в стан з кодом *j* не може бути реалізований за допомогою будь-якої із заданих ОП. Такий перехід будемо вважати непокритим заданою множиною ОП.

Результат зіставлення матриць представимо у вигляді матриці покриття (рис. 2.6). В цій матриці одиничні значення відповідають одиницям в матриці переходів (рис. 2.1), а зафарбовані комірки відповідають одиницям в об'єднаній матриці операцій (рис. 2.5). Як можна бачити, у даному прикладі усі одиниці розташовані у зафарбованих комірках, тобто усі автоматні переходи покриваються заданою множиною ОП. Отже, можна зробити висновок, що обраний спосіб кодування станів і задана множина ОП дають повний формальний розв'язок задачі алгебраїчного синтезу.

Розглянемо приклад, коли стани автомата закодовані кодами, що співпадають з індексами станів. При тих самих ОП $O_1 - O_3$ об'єднана матриця операцій відповідає рис. 2.6, а матриця переходів і матриця покриття наведені на рис. 2.7 і 2.8 відповідно.

		a_0	a_1	a_2	a_5	a_3	a_4	a_6	a_7
		0	1	2	3	4	5	6	7
a_0	0		1						
a_1	1			1	1				
a_2	2					1			
a_5	3							1	
a_3	4						1	1	
a_4	5								1
a_6	6				1				1
a_7	7	1	1						

Рисунок 2.6 – Матриця покриття з усіма покритими переходами

		a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
		0	1	2	3	4	5	6	7
a_0	0		1						
a_1	1			1			1		
a_2	2				1				
a_3	3					1		1	
a_4	4								1
a_5	5							1	
a_6	6						1		1
a_7	7	1	1						

Рисунок 2.7 – Матриця переходів при кодуванні станів згідно їх індексів

		a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
		0	1	2	3	4	5	6	7
a_0	0		1						
a_1	1			1			1		
a_2	2				1				
a_3	3					1		1	
a_4	4								1
a_5	5							1	
a_6	6						1		1
a_7	7	1	1						

Рисунок 2.8 – Матриця покриття при кодуванні станів згідно їх індексів

Рис. 2.8 демонструє, що при обраних кодах станів і операціях переходів три переходи, а саме $a_1 \to a_5$, $a_4 \to a_7$ і $a_6 \to a_5$ залишаються непокритими, тобто не можуть бути реалізовані жодною із заданих ОП. Таким чином, обраний спосіб кодування станів дає частковий розв'язок задачі алгебраїчного синтезу.

Перевага розглянутого способу виявлення формальних розв'язків задачі алгебраїчного синтезу полягає в такому. Коли матриця переходів зіставляється з об'єднаною матрицею операцій, кожному переходу зіставляються одночасно усі ОП із множини O. Зіставлення матриць еквівалентне виконанню кількості переборів варіантів, що визначається виразом (2.2). В результаті у виразі (2.3) частка, що відповідає виразу (2.2), замінюється на одиницю, під якою розуміється лише одне зіставлення матриць незалежно від кількості ОП і автоматних переходів. Вираз (2.3) приймає вигляд (2.4). Відсутність у виразі (2.4) параметрів B та N_I говорить про те, що при використанні матричного способу виявлення формальних

рішень кількість автоматних переходів та кількість операцій переходів не впливають на складність пошуку формальних рішень.

$$N_V = \frac{(2^R)!}{(2^R - M)!} \times 1 = \frac{(2^R)!}{(2^R - M)!}. \qquad (2.4)$$

Слід відзначити, що вираз (7) відповідає кількості варіантів пошуку формальних рішень саме у випадку повного перебору варіантів кодування станів МПА. Використання методів алгебраїчного синтезу, що використовують який-небудь частковий перебір варіантів, дозволить зменшити значення N_1, додатково скоротивши загальну кількість варіантів N_V. Це дозволяє вважати розглянутий матричний спосіб виявлення формальних розв'язків задачі алгебраїчного синтезу універсальним для використання у різних методах синтезу МПА з ОАП.

Також зіставлення матриці переходів і об'єднаної матриці операцій не передбачає виконання жодної ОП. Операції переходів використовуються на етапі формування матриць операцій, а це відбувається одноразово на початку алгебраїчного синтезу. Об'єднана матриця операцій не дозволяє визначити, яка конкретна операція зіставляється певному автоматному переходу, але дозволяє швидко визначити знаходження формального розв'язку задачі алгебраїчного синтезу. Якщо в результаті зіставлення матриць якийсь автоматний перехід виявився покритим, достатньо переглянути усі матриці операцій та обрати будь-яку з тих ОП, які мають одиничне значення у комірці, співпадаючій з даним переходом.

2.3 Алгоритм пошуку розв'язків задачі алгебраїчного синтезу методом повного перебору

Будемо розглядати випадок, коли множина операцій переходів вважається заданою і незмінюваною в процесі алгебраїчного синтезу МПА з ОАП. За цієї умови єдиним чинником, що впливає на отримання

формального розв’язку, залишається спосіб кодування станів. В п. 1.3 показано, що в загальному випадку може існувати декілька формальних розв’язків, які відрізняються способом кодування станів та способом зіставлення ОП автоматним переходам.

При фіксованій множині ОП фіксованими залишаються витрати апаратури в схемі операційного автомата переходів. Кодування станів та кодування операцій переходів впливають лише на логіку схеми блока, що формує коди операцій переходів (блок W на рис. 1.1). Якими б не були коди станів та коди ОП, кількість вхідних і вихідних сигналів блоку ОП залишається фіксованою. Це дозволяє вважати, що витрати апаратури в схемі блоку W не залежать або залежать несуттєво від того чи іншого знайденого формального розв’язку задачі алгебраїчного синтезу.

Отже, для синтезу схеми МПА з ОАП за умови заданої і фіксованої множини ОП достатньо знайти будь-який формальний розв’язок задачі алгебраїчного синтезу. Після того, як один формальний розв’язок знайдений, пошук інших розв’язків слід припинити, оскільки додатково знайдені розв’язки не дадуть суттєвої практичної користі, втім вимагатимуть зайвих витрат часу та обчислювальних потужностей для їх пошуку.

За незмінної множини ОП незмінною є і об’єднана матриця операцій. Отже, пошук формальних розв’язків зводиться до пошуку такого способу кодування станів, при якому усі автоматні переходи виявляються покритими заданими ОП. Це дозволяє застосувати алгоритм пошуку формальних розв’язків, який містить наступні кроки:

1. Вибір розрядності двійкових кодів станів R.

2. Формування множини K усіх можливих кодів станів.

3. Завдання множини операцій переходів O.

4. Формування об’єднаної матриці операцій.

5. Організація перебору способів кодування станів автомата кодами із множини K.

6. Вибір чергового способу кодування станів.

7. Формування матриці переходів.

8. Зіставлення матриці переходів та об’єднаної матриці операцій.

9. Якщо повний формальний розв’язок знайдений (усі переходи покриті ОП) і пошук інших розв’язків не потрібен, здійснити перехід до кроку 10. Інакше здійснити перехід до кроку 6.

10. Кінець.

Дамо коротке пояснення цих кроків.

Крок 1. Зазвичай розрядність кодів станів R слід брати мінімально достатньою для кодування M станів, тобто рівною $\lceil log_2 M \rceil$. Втім, за потреби можна взяти більшу кількість розрядів, що відповідним чином збільшить кількість допустимих кодів станів. Такий підхід може збільшити кількість повних розв’язків і знайти їх там, де їх немає при $R = \lceil log_2 M \rceil$. Однак при цьому суттєво збільшиться кількість допустимих варіантів кодування, що збільшить час на пошук повних розв’язків. До того ж, збільшення розрядності кодів станів жодним чином не гарантує знаходження повних формальних розв’язків.

Крок 2. В загальному випадку множина K містить усі двійкові вектори розрядності R. Якщо певні коди з якихось причин не можуть бути використані для кодування станів, їх слід прибрати із множини K.

Крок 3. В якості ОП зручно використовувати операції, що виконуються між кодом поточного стану і певною константою. Це можуть бути або арифметичні операції (додавання, віднімання), або логічні операції (XOR, AND, OR, зсуви вліво чи вправо). Такі операції мають відносно просту схемну імплементацію, що спрощує проєктування схеми операційного автомата переходів. Після формування множини O елементи множини K (допустимі коди станів) отримують інтерпретацію відповідно до обраних ОП. Зазвичай це цілі числа без знака в діапазоні $[0; 2^R–1]$ та R-розрядні двійкові вектори.

Крок 4. Даний процес розглянутий в п. 2.1.

Крок 5. Для автомата з M станами для кодування станів можуть бути використані 2^R різних кодів. Загальна кількість способів зіставлення 2^R кодів M станам визначається виразом (2.4), що відповідає комбінаторній формулі визначення числа розміщень.

Перебір способів кодування станів може бути організований по-різному. Найпростішим варіантом є повний перебір усіх можливих способів кодування, кількість яких визначається виразом (3). Однак вже при $R = 5$ чисельник виразу (2.4) має порядок 10^{35}, що говорить про практичну неможливість пошуку формальних розв'язків шляхом повного перебору. Тож залишається пошук різних способів часткового перебору, які б дозволяли знайти формальне рішення за приємний час. Пошук таких способів виходить за межі цієї кваліфікаційної і планується як напрямок майбутніх досліджень.

Крок 6. Кожен стан автомата має бути закодований унікальним кодом із множини K. Якщо $|K| > M$, частина допустимих кодів не буде використана для кодування станів.

Крок 7. Формування матриці переходів розглянуте в [16] та проілюстровано прикладом, наведеним вище.

Крок 8. Результатом зіставлення матриці переходів і об'єднаної матриці операцій є матриця покриття, розглянута в п. 2.1.

Крок 9. В даному алгоритмі будемо шукати повний формальний розв'язок задачі алгебраїчного синтезу, тобто коли усі автоматні переходи покриті операціями із множини O. Як було відзначено, зазвичай знаходження одного формального розв'язку є достатнім для синтезу схеми автомата. Втім, з метою дослідження таких характеристик алгоритму, як універсальність, продуктивність, результативність тощо, може виявитись корисним пошук усіх можливих формальних розв'язків.

Крок 10. Результатом роботи алгоритму можуть бути:

– знайдений повний розв'язок задачі алгебраїчного синтезу МПА з ОАП;

– множина знайдених повних розв’язків (якщо дозволений пошук декількох повних розв’язків);

– висновок про відсутність повних розв’язків для заданих ГСА, розрядності кодів станів і множини ОП.

Блок-схема запропонованого алгоритму наведена на рис. 2.9.

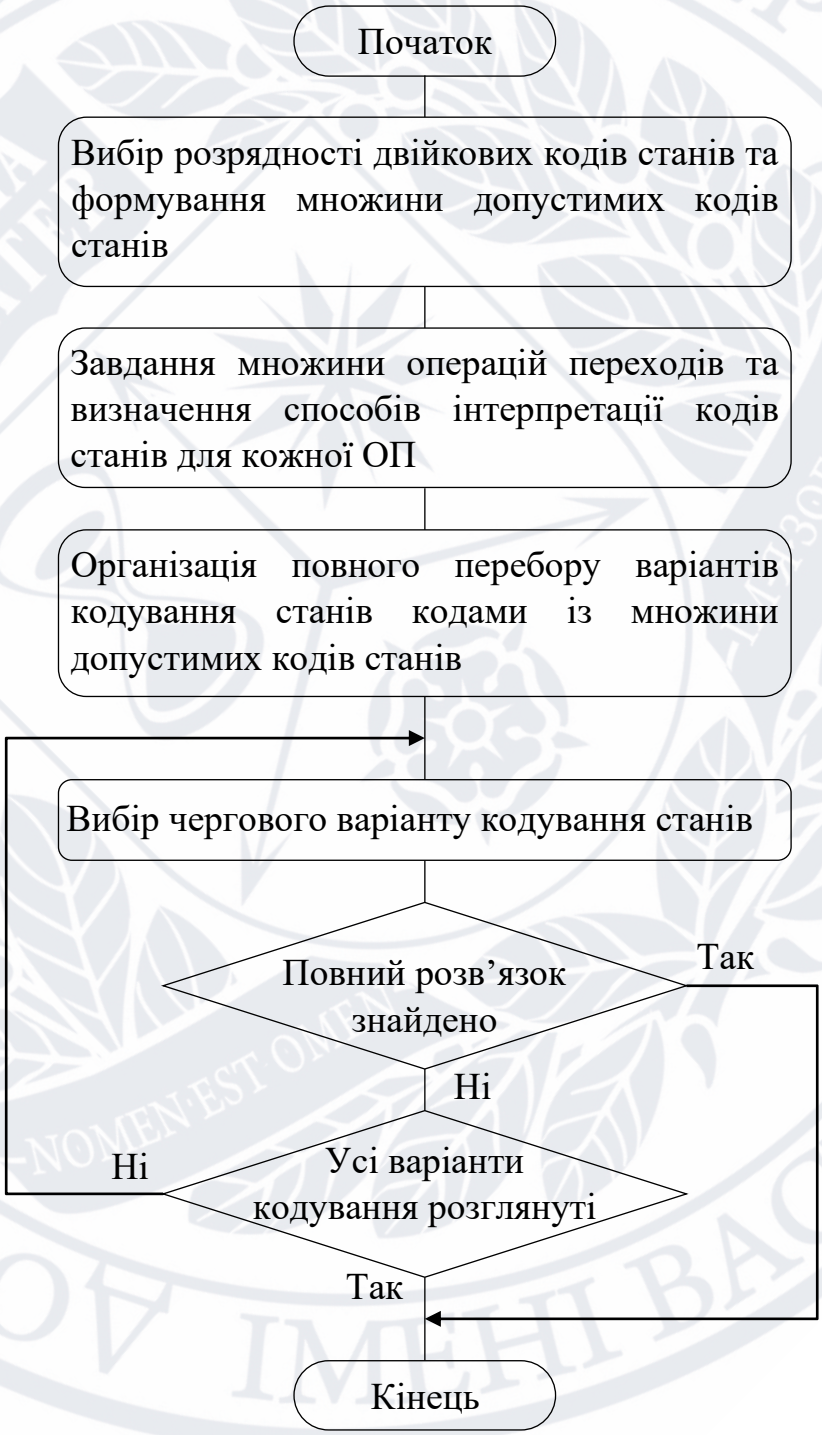

Рисунок 2.9 – Блок-схема алгоритму пошуку повних розв’язків задачі алгебраїчного синтезу МПА з ОАП

Назвемо цей алгоритм алгоритмом пошуку формального розв'язку задачі алгебраїчного синтезу МПА з ОАП. Його особливістю є те, що пошук відбувається до тих пір, поки перший розв'язок не буде знайдений. При цьому алгоритм не гарантує знаходження повного розв'язку, оскільки для заданої ГСА і заданої множини ОП повний розв'язок може не існувати.

Для даного алгоритму слід вважати справедливими наступні твердження:

1. Імовірність знаходження повного розв'язку збільшується зі збільшенням кількості використовуваних операцій переходів. Це пов'язано з тим, що більша кількість ОП приводить, як правило, до збільшення кількості одиничних комірок в об'єднаній матриці операцій. Це, в свою чергу, може сприяти більшій кількості покритих автоматних переходів.

2. Імовірність знаходження повного розв'язку зменшується зі збільшенням кількості станів автомата при фіксованій множині операцій переходів. Більша кількість автоматних переходів збільшує кількість одиничних комірок в матриці переходів. Це, при фіксованій об'єднаній матриці операцій, сприятиме збільшенню кількості непокритих переходів.

Із першого твердження виходить, що чим меншою є кількість використовуваних ОП, тим меншою є ймовірність знаходження повного розв'язку. Однак економія апаратурних витрат в схемі МПА з ОАП досягається як раз за рахунок зменшення кількості ОП і спрощення схеми операційного автомата переходів. Отже, прагнення до зменшення кількості використовуваних ОП сприяє незнаходженню повних розв'язків.

Із другого твердження виходить, що збільшення складності ГСА також не сприяє знаходженню повних розв'язків. Складність алгоритму в даному випадку вимірюється кількістю його станів, яка впливає на розрядність коду стану R. Отже, чим складнішим є автомат, тим важче покрити усі його переходи заданою множиною ОП, оскільки кількість варіантів кодування станів зростає, згідно (2.4), у нелінійній залежності.

Таким чином, прагнення зменшити кількість ОП та зростання складності алгоритмів керування зменшують імовірність і збільшують час знаходження повних розв'язків задачі алгебраїчного синтезу. Алгоритм, наведений на рис. 2.9, слід використовувати лише в тому випадку, якщо з якихось причин неприпустимо мати непокриті переході. В якості таких причин можна передбачити наступні:

– поставлено задачу знизити складність схеми автомата до мінімально можливого рівня;

– в якості ОАП використовується заздалегідь спроєктований пристрій, в який не можна вносити зміни.

Отже, можлива ситуація, коли алгоритм (точніше, його програмна реалізація) працює надто довго, не знаходячи при цьому жодного повного розв'язку. Чим довше працює алгоритм, тим менш імовірно, що повний розв'язок буде знайдений. Для підвищення ефективності роботи алгоритму пропонується наступне:

– обмежити алгоритм в часі виконання;

– протягом виконання алгоритму запам'ятовувати частковий розв'язок, що має найменшу кількість непокритих переходів;

– по завершенні роботи алгоритму результатом його роботи вважати частковий розв'язок з найменшою кількістю непокритих переходів, знайдений на поточний момент.

Даний алгоритм містить наступні кроки:

1. Вибір розрядності двійкових кодів станів R та формування множини K усіх можливих кодів станів.

2. Завдання множини операцій переходів O та формування об'єднаної матриці операцій.

3. Мінімальна кількість непокритих переходів B_{min} береться рівною загальній кількості переходів B.

4. Введення максимально допустимого часу роботи алгоритму (йдеться про час роботи його програмної реалізації).

5. Початок відрахунку часу роботи алгоритму.

6. Організація перебору способів кодування станів автомата кодами із множини *K*.

7. Вибір чергового способу кодування станів.

8. Формування матриці переходів відповідно до обраного кодування станів.

9. Зіставлення матриці переходів та об’єднаної матриці операцій.

10. Якщо знайдений повний формальний розв’язок (усі переходи покриті ОП) і пошук інших розв’язків не потрібен, здійснити перехід до кроку 14.

11. Якщо в отриманому розв’язку кількість непокритих переходів B' менша за B_{min}, прийняти B_{min} рівною B' та вважати частковий розв’язок, знайдений на даній ітерації, найкращим.

12. Якщо допустимий час роботи алгоритму вичерпаний, здійснити перехід до кроку 14.

13. Якщо усі варіанти кодування станів розглянуті, перейти до кроку 14. Інакше перейти до кроку 7.

14. Кінець.

Даному алгоритму відповідає блок-схема, наведена на рис. 2.10.

Відмінності цього алгоритму від алгоритму пошуку повного розв’язку полягають в наступному.

1. Алгоритм завжди дає результат, достатній для проєктування схеми автомата. Цей результат може бути як повним, так і частковим розв’язком, однак він завжди буде. Алгоритм пошуку повного розв’язку (рис. 2.9) може взагалі не дати результату, що зробить неможливим алгебраїчний і подальший синтез автомата.

Рисунок 2.10 – Блок-схема алгоритму пошуку часткових розв'язків

2. Якщо задані обмеження в часі, алгоритм за відведений час знаходить серед розглянутих часткових розв’язків такий, що має мінімальну кількість непокритих переходів. Це дозволяє максимально спростити схему автомата.

3. Якщо обмеження в часі не задані, алгоритм після перебору усіх можливих варіантів кодування станів дає найкращий результат, який тільки є можливим для заданої ГСА при заданій множині ОП. Це може бути або повний, або частковий розв’язок, який можна вважати оптимальним за заданих умов алгебраїчного синтезу.

Як і в попередньому алгоритмі, при розробці алгоритму пошуку часткових розв’язків було використане припущення про те, що усі повні розв’язки, які існують для заданої ГСА і множини ОП, приводять до однакових або близьких значень складності схеми автомата. Отже, у випадку знаходження повного розв’язку алгоритм завершує свою роботу, а знайдений результат вважається найкращим серед усіх можливих.

РОЗДІЛ 3
РОЗРОБКА І ДОСЛІДЖЕННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АЛГОРИТМІВ

3.1 Вимоги до розроблюваної програми

В розділі 2 розроблені алгоритми пошуку повних і часткових формальних розв'язків задачі алгебраїчного синтезу мікропрограмного автомата з операційним автоматом переходів. В їх основі лежить велика кількість переборів варіантів кодування станів автомата. Такий перебір на практиці може бути здійснений лише за допомогою комп'ютера. Отже, для практичного впровадження алгоритмів і для проведення подальших досліджень в цьому напрямку необхідною складовою є програмна реалізація запропонованих алгоритмів. Розроблювану програмну реалізацію алгоритмів алгебраїчного синтезу будемо в подальшому називати терміном «програма».

Будемо дотримуватись наступних тверджень:

1. Початковими даними для роботи програми є граф-схема заданого алгоритму керування і множина операцій переходів.

2. Результатом роботи програми є формальний розв'язок (розв'язки) задачі алгебраїчного синтезу у вигляді закодованої множини станів автомата. Наприклад, для формального розв'язку, наведеного на рис. 1.5, результат роботи програми може бути таким, як на рис. 3.1.

$a_0 - 0,\ a_1 - 1,\ a_2 - 2,\ a_3 - 4,\ a_4 - 5,\ a_5 - 3,\ a_6 - 6,\ a_7 - 7$

Рисунок 3.1 – Приклад представлення формального розв'язку задачі алгебраїчного синтезу за умови відомої структури ГСА та множини ОП

Знаючи структуру ГСА (рис. 1.3) та множину ОП (вирази (1.3) – (1.5)), можна відтворити зіставлення ОП автоматним переходам. Наприклад,

перехід зі стану a_2 з кодом $2_{10}=010_2$ в стан a_3 з кодом $4_{10}=100_2$ може бути виконаний за допомогою ОП O_2, що задається виразом (1.4). Якщо якийсь перехід може бути реалізований за допомогою декількох ОП, достатньо вибрати будь-яку із них. Таким чином, наведений приклад результату роботи програми є достатнім для відтворення формального розв'язку задачі алгебраїчного синтезу, наведеного на рис. 1.5.

На етапі дослідження алгоритмів буде цілком достатньо, якщо програма матиме консольний інтерфейс. Це обґрунтовується прикладом результату роботи програми, що наведений на рис. 3.1. На етапі практичного впровадження програми буде зручно, якщо програма в якості вхідних даних прийматиме ГСА, задану у форматі KISS2 (подібно до рис. 1.3). Це дозволяє сформулювати наступні вимоги до розроблюваної програми:

1. Програма розробляється із використанням мови Python.
2. Вхідні дані задаються файлом у форматі KISS2.
3. Для виведення результатів використовується консольний інтерфейс.
4. Результати програми мають містити інформацію про спосіб кодування станів, що дає формальний розв'язок задачі алгебраїчного синтезу.

Інші особливості програми не є надто принциповими. В рамках майбутніх досліджень можлива розробка графічного інтерфейсу із використанням Python-бібліотеки *tkiner* або подібних. Однак в даній роботі графічний інтерфейс не застосовується.

Розглянемо її окремі структурні елементи розробленої програми.

3.2 Структури даних програми та їх створення

Структура даних *GSA*, яка описує довільну ГСА в контексті алгебраїчного синтезу, наведена на рис. 3.2. Пояснимо основні її складові.

В розділах 1 і 2 цієї роботи кількість переходів автомата позначена символом *B*. У файлі формату KISS2 цій змінній відповідає змінна *p*. Для

ототожнення структури *GSA* із вмістом файлу KISS2 кількість переходів автомата позначається в розроблюваній програмі змінною *p*.

```
class GSA:
    i = 0               # Кількість вхідних сигналів
    o = 0               # Кількість вихідних сигналів
    p = 0               # Кількість переходів
    s = 0               # Кількість станів
    r = ""              # Початковий стан
    R = 0               # Розрядність коду стану
    slist = list()      # Список текстових імен станів
    clist = list()      # Список кодів станів
    m = 0               # Матриця переходів
    mm = 0              # Модифікована матриця переходів
    tr_list = list()    # Список переходів
```

Рисунок 3.2 – Опис класу (структури даних) для опису граф-схеми алгоритму

Як в теорії цифрових автоматів, так і в файлі формату KISS2 стани автомату можуть позначатись довільними наборами символів. Для зберігання імен станів, прочитаних з файлу KISS2, використовується список *slist*, кожен елемент якого є текстовим значенням імені стану. Наприклад, список *slist* може містити такі елементи, як «a0», «a1», «a2» тощо.

Список *clist* призначений для зберігання десяткових значень кодів станів. Вміст списку формується не відразу, а в процесі алгебраїчного синтезу. Вважається, що елементи з однаковими індексами у списках *clist* і *slist* відповідають одному стану автомата. На початку роботи програми елементи списку *clist* отримують послідовні цілі значення, починаючи з нуля. Це можна вважати початковим варіантом кодування станів.

Список переходів *tr_list* також заповнюється в процесі читання файлу KISS2, а також обробляється в процесі алгебраїчного синтезу. Елементами списку *tr_list* є двоелементні списки, кожен з яких складається із коду початкового стану і коду стану переходу. Наприклад, перехід зі стану з кодом 5 в стан з кодом 7 описується як список [5, 7]. Кількість елементів в списку *tr_list* дорівнює кількості переходів автомата.

Елементи *m* і *mm* за своєю структурою відповідають матриці переходів автомата. В них міститься та сама інформація, що й у списку *tr_list*, але у вигляді квадратних матриць (рис. 2.1). Матриця *m* містить початкову інформацію про систему переходів, матриця *mm* є її дублем і використовується в процесі перебору варіантів кодування станів.

Для заповнення екземпляра класу GSA інформацією із файлу формату KISS2 розроблено функцію *read_kiss2*. Вона починається зі створення екземпляра класу та ініціалізації його атрибутів-списків (рис. 3.3).

```
def read_kiss2(filename):
    """Reads information from file in kiss2 format"""

    G = GSA()             # Екземпляр класу GSA
    f = open(filename, "r")

    G.slist = list()      # Ініціалізація списку імен станів
    G.clist = list()      # Ініціалізація списку кодів станів
    G.tr_list = list()    # Ініціалізація списку переходів
```

Рисунок 3.3 – Початок функції *read_kiss2*

Останні три команди на рис. 3.3 необхідні для того, щоб кожен екземпляр класу GSA мав власні списки, окремі від списків самого класу. В мові Python при створенні екземпляра класу ініціалізація атрибутів-списків не відбувається.

На рис. 3.4 наведений код основного циклу функції *read_kiss2*, задачею якого є читання вмісту файлу по окремим рядкам. Цикл є нескінченим, оскільки кількість рядків файлу заздалегідь невідома. На кожній ітерації циклу відбуваються такі дії.

1. Читається черговий рядок файлу. Якщо прочитаний останній рядок файлу, цикл примусово завершується оператором *break*.

2. Прочитаний рядок розділяється на окремі фрагменти, що розділені пробілами.

```
    while True:
        s = f.readline()
        if s=="": break                 # Якщо кінець файлу
        s = s.replace("\n", "")
        if s=="": continue              # Якщо порожній рядок

        l = s.split(" ")

        if ((l[0])[0]=="."):                    # Якщо крапка

            if l[0]==".i":                      # Якщо ".i"
                G.i = int(l[1])
                print(G.i, "inputs")
            if l[0]==".o":                      # Якщо ".o"
                G.o = int(l[1])
                print(G.o, "outputs")
            if l[0]==".p":                      # Якщо ".p"
                G.p = int(l[1])
                print(G.p, "transitions")
            if l[0]==".s":                      # Якщо ".s"

                G.s = int(l[1])
                print(G.s, "states")

                if G.s==0:                      # Якщо ".s" немає
                    print(".s = 0, thats wrong!")
                    return -1

                # Розрахунок розрядності кодів станів
                G.R = math.ceil(math.log2(G.s))
                # Створення матриці переходів
                G.m = np.zeros((2**G.R, 2**G.R), dtype = bool)

            if l[0]==".r":                      # Якщо ".r"
                G.r = l[1]
                print("Initial state: ", G.r)

        else: # Якщо прочитаний рядок з автоматним переходом

            s1 = l[1]
            s2 = l[2]
            if G.r=="": G.r = s1        # Якщо ".r" не було в файлі

            if s1 not in G.slist:       # Якщо стан s1 не у списку
                code1 = len(G.slist)    # Створюємо новий код стану s1
                G.slist.append(s1)      # Додаємо стан s1 у список
                G.clist.append(code1) # Додаємо код стану s1 у список
            else:  # Якщо стан s1 вже нам відомий
                code1 = G.slist.index(s1)    # Шукаємо код стану s1 у списку

            if s2 not in G.slist:           # Якщо стан s2 не у списку
                code2 = len(G.slist)        # Створюємо новий код стану s2
                G.slist.append(s2)          # Додаємо стан s2 у список
                G.clist.append(code2)       # Додаємо код стану s2 у список
            else:  # Якщо стан s1 вже нам відомий
                code2 = G.slist.index(s2)    # Шукаємо код стану s2 у списку

            G.m[code1][code2] = 1 # Заповнюємо комірку матриці переходів
            G.tr_list.append([code1, code2]) # Додаємо перехід у список
```

Рисунок 3.4 – Головний цикл функції *read_kiss2*

3. Аналізується фрагмент з індексом [0]. Якщо він починається з символу крапки, то це один із «ключів»: «.i», «.o», «.p» тощо. В цьому випадку в залежності від другого символу відповідна характеристика ГСА заповнюється значенням фрагменту з індексом [1]. Якщо прочитано ключ «.s», додатково визначається розрядність коду стану *G.R* та формується матриця переходів *G.m*, заповнена нулями.

4. Якщо прочитаний рядок починається не з крапки, то це опис чергового переходу автомата. В цьому випадку виконуються такі дії.

4.1. В змінні *s1* і *s2* прочитуються імена поточного стану і стану переходів, взяті із елементів рядка з індексами [1] і [2]. Елемент з індексом [0], згідно формату KISS2, відповідає вхідним сигналам автомата, що аналізуються в даному переходів. Він має використовуватись в процесі синтезу блоку W (рис. 1.1), але в процесі алгебраїчного синтезу не використовується.

4.2. Якщо стан *s1* відсутній у списку *slist*, тобто зустрівся вперше, йому надається черговий послідовний код, після чого стан і код додаються до списків *slist* та *clist* відповідно. Якщо стан *s1* знайдений у списку *slist*, визначаємо його код (звертаємось за однойменним індексом до списку *clist*).

4.3. Виконуємо пункт 4.2 для стану *s2*.

4.4. Після того, як у пунктах 4.2 і 4.3 були визначені код *code1* стану *s1* і код *code2* стану *s2*, у матрицю переходів *G.m* за координатами [*code1*][*code2*] заноситься 1.

4.5. У список переходів *tr_list* додається черговий автоматний перехід у вигляді списку [*code1*, *code2*].

5. Після завершення головного циклу виконуються команди, наведені на рис. 3.5. Це закриття файлу KISS2, копіювання матриці *G.m* у матрицю *G.mm*, а також певні перевірки взаємної коректності прочитаних даних.

```
    f.close()

    G.mm = G.m.copy()

    if G.s != len(G.slist) or G.s != len(G.clist) or
       len(G.slist) != len(G.clist):
        print("Some errors during reading GSA.")
        print("Please check content of input file.")
        exit(0)
```

Рисунок 3.5 – Кінцеві дії функції *read_kiss2*

3.3 Підготовка матриць

Розглянуті в розділі 2 алгоритми передбачають виконання підготовчих етапів, пов'язаних із формування матриць переходів і матриць операцій. Матриця переходів формується в процесі роботи функції *read_kiss2*. Розглянемо формування матриць операцій та об'єднаної матриці операцій.

Операції переходів будемо задавати у вигляді окремих функцій Python. Приклад завдання ОП $O_1 - O_3$, що визначаються виразами (1.3) – (1.5), наданий на рис. 3.6.

```
def O1(x):          # ОП O1
    return (x+1)

def O2(x):          # ОП O2
    return (x+2)

def O3(x):          # ОП O3
    return (x^5)
```

Рисунок 3.6 – Приклад програмного завдання операцій переходів

Дані функції використовуються при формуванні матриць операцій. Матриці операцій формуються функцією *op_matrix*, яка наведена на рис. 3.7. Її аргументами є одна із функцій ОП та розрядність коду стану *R*.

```
def op_matrix(func, R):
    ### Building matrix with size 2**R for function func ###

    m = np.zeros((2**R, 2**R), dtype = bool) # Матриця нулів

    for i in range(0, 2**R):          # Заповнення матриці
        y = int(func(i)) % (2**R)     # Виклик функції ОП
        m[i][y] = 1

    return m
```

Рисунок 3.7 – Лістинг функції *op_matrix*

У функції *op_matrix* спочатку створюється матриця *m*, яка представляє собою двовимірний масив типу *ndarray* (масив бібліотеки *Numpy*) зі стороною, рівню 2^R. У випадку ГСА G_1 (рис. 1.2) матриця *m* матиме розміри 8 х 8 елементів.

Далі перебираються усі допустимі коди станів, що лежать у діапазоні $[0; 2^R]$. Для кожного коду стану викликається функція *func*, що передана в функцію *op_matrix* в якості аргументу. Результат функції приводиться до цілого числа і обмежується останніми *R* двійковими розрядами. В результаті код стану переходу *y* завжди лежатиме у діапазоні допустимих кодів станів $[0; 2^R]$. Обчислений в такий спосіб код стану переходу відповідає стовпцю матриці. Далі в матрицю за координатами [i, y] записується одиничне значення.

Функція *op_matrix* повертає в якості результату повністю сформовану матрицю операції для ОП, що задана аргументом *func*. У випадку декількох використовуваних ОП слід виконати декілька викликів цієї функції з різними аргументами *func*, отримавши декілька матриць операцій. Далі ці матриці слід зібрати в об'єднану матрицю переходів (приклад – на рис. 2.5). Це робиться за допомогою двох наступних функцій.

Функція *op_matrix_3d* (рис. 3.8) представляє набір двовимірних матриць операцій у вигляді тривимірного масиву, кожен шар якого є однією з

матриць операцій. Окремі матриці операцій передаються у функцію єдиним списком, тому мають бути сформовані до її виклику.

```
def op_matrix_3d(m_list):
    ### Creating multylayer ndarray from list of op_matrixes ###

    if len(m_list) == 0:   # If list is empty
        print("op_matrix_3d: input list is EMPTY.")
        return -1

    x = m_list[0].shape[0]
    y = m_list[0].shape[1]
    z = len(m_list)

    m = np.zeros( (z, x, y), dtype = bool )

    for i in range(0, len(m_list)):
        m[i] = m_list[i]

    return m
```

Рисунок 3.8 – Лістинг функції *op_matrix_3d*

Результатом функції *op_matrix_3d* є тривимірний масив. Об’єднана матриця операцій робиться як «проєкція» усіх шарів тривимірного масиву в один шар. Проєкція відбувається за принципом диз’юнкції: якщо певна комірка дорівнює одиниці хоча б в одному шарі тривимірного масиву, відповідна комірка в об’єднаній матриці операцій також дорівнюватиме одиниці. Подібні дії виконуються в функції *op_matrix_flat* (рис. 3.9).

Аргументом цієї функції є тривимірний масив типу *ndarray*, попередньо сформований функцією *op_matrix_3d*. На початку роботи функції формується матриця *m*, заповнена нулями. Далі в циклі ця матриця поелементно об’єднується з кожним шаром тривимірного масиву *m3d*, який передається у функцію в якості аргументу. В якості результату функція повертає двовимірний масив, що містить об’єднану матрицю переходів.

```
def op_matrix_flat(m3d):
    ### Creating OR matrix from array of matrixes ###

    # Підготовка об'єднаної матриці операцій
    m = np.zeros(m3d[0].shape, dtype = int)

    for i in m3d:
        m = np.logical_or(m, i)

    m = m.astype(int)

    return m
```

Рисунок 3.9 – Функція для побудови об’єднаної матриці операцій

Якщо перед останньою командою функції додати команду виведення на екран матриці *m*:

```
    print(m)
```

то для трьох ОП, наведених на рис. 3.6, результат виводу на екран буде таким:

```
[[0 1 1 0 0 1 0 0]
 [0 0 1 1 1 0 0 0]
 [0 0 0 1 1 0 0 1]
 [0 0 0 0 1 1 1 0]
 [0 1 0 0 0 1 1 0]
 [1 0 0 0 0 0 1 1]
 [1 0 0 1 0 0 0 1]
 [1 1 1 0 0 0 0 0]]
```

Неважко переконатись, що цей результат співпадає з об’єднаною матрицею операцій, наведеною на рис. 2.6. Підкреслимо, що вміст цієї матриці не залежить від граф-схеми алгоритму автомата і визначається лише заданою множиною операцій переходів.

3.4 Перебір варіантів кодування станів

Обидва алгоритми, розглянуті в розділі 2, використовують повний перебір варіантів кодування станів автомата. Алгоритм пошуку повних розв'язків зупиняється тільки тоді, коли усі можливі варіанти перебрані. Алгоритм пошуку часткових розв'язків може зупинятись за досягнення певних критеріїв, будь то допустимий час роботи програми або знайдений частковий розв'язок.

Для пошуку повних розв'язків задачі алгебраїчного синтезу розроблено функцію permutation, початок якої наведений на рис. 3.10.

```
def permutation(G, m_flat):
    ### Генерація усіх можливих перестановок кодів станів ###

    L = list(range(0, 2**G.R))
    # Створення генератору перестановок
    gen = itertools.permutations(L, G.s)

    permut = 0  # Кількість перебраних перестановок кодів станів
    n_sol = 0   # Кількість знайдених повних розв'язків
```

Рисунок 3.10 – Початок функції *permutation*

На початку створюється список *L*, який містить множину усіх допустимих кодів станів (кодів в діапазоні [0; 2^R]). Далі викликається функція *permutations* модуля *itertools*, яка із елементів списку *L* генерує усі можливі послідовності по *G.s* елементів (тут *G.s* – кількість станів автомата). Слід відзначити, що функція *itertools.permutations* не генерує усі послідовності відразу, оскільки їх може бути дуже багато. Замість цього функція створює спеціальний об'єкт (змінна *gen*), до якого можна звертатись багаторазово, щоразу отримуючи чергову послідовність кодів станів.

Опис головного циклу перебору варіантів представлений на рис. 3.11.

```
    for i in gen:       # Цикл перебору перестановок
        permut += 1     # Кількість знайдених повних розв'язків
        G.mm = G.m.copy()   # Створення копії матриці переходів
        clist = G.clist.copy()  # Створення копії кодів станів

        for j in range(0, G.s):   # Цикл по кількості станів

            # Перекодування станів
            code_new = i[j]     # Новий код для стану G.slist[j]
            code_old = clist[j]# Старий код для стану G.slist[j]

            if (code_new != code_old):  # Якщо вони різні

                # Зміна кодів в локальному списку кодів
                if code_new in clist:
                    index = clist.index(code_new)
                    clist[index] = code_old
                clist[j] = code_new

                # Перероблення матриці переходів G.mm
                # з урахуванням змінених кодів станів
                G.mm[:, code_new], G.mm[:, code_old] =
                        G.mm[:, code_old].copy(),
                        G.mm[:, code_new].copy()
                G.mm[code_new, :], G.mm[code_old, :] =
                        G.mm[code_old, :].copy(),
                        G.mm[code_new, :].copy()

        # Перевірка: чи знайдений формальний розв'язок
        n = compare(G.mm, m_flat) # Підрахунок кількості
                                  # нереалізованих переходів
        if (n==0): # Якщо усі переходи реалізуються операційно
            n_sol += 1    # Знайдено черговий повний розв'язок

            print("Solution", n_sol, ":")
            print(G.slist)   # Виводимо список станів
            print(clist)     # Виводимо список кодів станів
            print()

    print("Повних розв'язків:", n_sol)
```

Рисунок 3.11 – Головний цикл пошуку повних розв’язків

Головний цикл представляє собою послідовний перебір усіх внутрішніх елементів ітерованого об’єкту *gen*. Кожним таким елементом, що кладеться у змінну *i*, є список з *M* кодів станів, де *M* – кількість станів

автомата, $M \leq 2^R$. Індекси списку *i* ототожнюються з індексами списку станів *G.clist*.

Як було показано в розділі 2, після вибору кодів станів необхідно побудувати матрицю переходів. Ця побудова відбувається всередині внутрішнього циклу

```
for j in range(0, G.s):   # Цикл по кількості станів
```

кількість повторів якого дорівнює кількості станів автомата.

Суть виконуваних дій полягає в наступному. Об'єкт *G* (задана ГСА) містить всередині вже побудовану матрицю переходів *mm*. Однак ця матриця побудована за таким принципом: черговий стан, прочитаний з файлу KISS2, отримує черговий код, починаючи з 0. Першим прочитаним станом може бути a_0, наступним – a_3, потім – a_1 і так далі. Нам же потрібно перекодувати усі стани, замінивши їх поточні коди кодами зі списку *G.clist*. На основі нових кодів станів слід побудувати оновлену матрицю переходів.

Цей процес не складним з точки зору програмної реалізації. Проблема полягає в тому, щоб зробити цей процес якомога більш швидким. Перебудова матриці переходів, а також зіставлення перебудованої матриці переходів і об'єднаної матриці операцій, є тими діями, які виконуються в кожній ітерації головного циклу функції *permutation*. Чим швидше буде відбуватись створення матриці переходів, тим більшою буде загальна швидкодія програми.

У наведеному варіанті реалізований наступний спосіб побудови матриці переходів. Якщо стан a_i з кодом $K_1(a_i)$ отримує зі списку *G.cmas* код $K_2(a_i)$, який не дорівнює $K_1(a_i)$, в поточній матриці переходів слід зробити дві дії:

1) обміняти місцями вміст рядків, що відповідають кодам $K_1(a_i)$ і $K_2(a_i)$;

2) обміняти місцями вміст стовпців, що відповідають кодам $K_1(a_i)$ і $K_2(a_i)$.

На рис. 3.11 це робиться за допомогою команд, що використовують можливості зрізів масивів Numpy:

```
G.mm[:, code_new], G.mm[:, code_old] =
        G.mm[:, code_old].copy(),
        G.mm[:, code_new].copy()
G.mm[code_new, :], G.mm[code_old, :] =
        G.mm[code_old, :].copy(),
        G.mm[code_new, :].copy()
```

Такий обмін рядків та стовпців відбувається для кожного стану, код якого був змінений у новій сгенерованій послідовності кодів станів. Якщо у списку *G.clist* код якогось стану збігається з його «старим» кодом, обмін рядків і стовпців не відбувається.

Можливо, можна знайти більш швидкий спосіб побудови матриці переходів. Це питання не є об'єктом дослідження цієї кваліфікаційної роботи і може бути розглянуте в майбутньому.

Зіставлення матриці переходів і об'єднаної матриці операцій здійснюється командою

```
n = compare(G.mm, m_flat) # Підрахунок кількості
```

Тут викликається функція *compare*, лістинг якої наведений на рис. 3.12.

Функція *compare* приймає в якості аргументів матрицю переходів *m_tr* та об'єднану матрицю операцій *m_flat*. Матриця переходів формується в першій частині функції *permutation*, об'єднана матриця операцій готується заздалегідь за допомогою розглянутих вище функцій *op_matrix*, *op_matrix_3d*, *op_matrix_flat*.

```
def compare(m_tr, m_flat):
    ### Зіставлення матриці переходів m_tr   ###
    ### і об'єднаної матриці операцій m_flat ###

    m0 = m_flat.copy()

    m0 = np.logical_and(m0, m_tr)
    m0 = np.logical_xor(m0, m_tr)

    n = np.count_nonzero(m0)   # Кількість непокритих станів

    return n
```

Рисунок 3.12 – Лістинг функції *compare*

Спочатку обидві матриці зіставляються поелементно за принципом кон'юнкції. В результаті утворюється матриця, в якій комірки з одиничними значеннями відповідають покритим переходам. Однак нас більше цікавить інформація не про покриті, а про непокриті переходи, тобто про ті переходи, які при обраному варіанті кодування станів не можуть бути реалізовані жодною із заданих ОП. Для виявлення нереалізованих переходів поточна матриця *m0* (що містить покриті переходи) зіставляється з матрицею переходів *m_tr* за принципом XOR. В результаті ті комірки, які в обох матрицях дорівнювали 1 (покриті переходи), перетворюються на нулі, а інші одиниці матриці *m_tr* (непокриті переходи) залишаються одиницями. Як наслідок, результуюча матриця *m0* буде містити одиниці лише в тих комірках, які відповідають непокритим переходам. Нам залишається порахувати кількість таких клітин за допомогою Numpy-функции *count_nonzero*. Порахована в такий спосіб кількість непокритих переходів є тим значенням, що повертається функцією *compare*.

Після виклику функції *compare* продовжує роботу функція *permutation* (рис. 3.11). Здійснюється перевірка: якщо кількість непокритих переходів *n* дорівнює нулю, поточний варіант кодування станів дає нам черговий повний формальний розв'язок задачі алгебраїчного синтезу. Згідно алгоритму пошуку повних розв'язків (рис. 2.9) програма після знаходження першого

повного розв’язку має завершитись, однак, відповідно до лістингу на рис. 3.11, функція *permutation* продовжує працювати аж до повного перебору усіх варіантів кодування станів. Це зроблено з дослідницькою метою і дозволяє дізнатись, скільки загалом існує повних розв’язків для заданої ГСА і заданої множини ОП. Таке дослідження може бути проведене лише для невеликих ГСА (не більше 16 станів), тоді як час аналізу більших ГСА є надто великим для повного перебору.

У разі необхідності переривати програму після знаходження першого повного розв’язку слід після команд *print* додати команду примусового переривання циклу *break*. Самі команди *print* виводять на екран вміст списку станів і списку кодів станів, що при відомій множині ОП є достатнім для представлення формального розв’язку задачі алгебраїчного синтезу.

На рис. 3.13 наведений блок команд головної частини програми, які здійснюють виклик розглянутих вище функцій.

```
G = read_kiss2("test.kiss2")

m1 = op_matrix(O1, G.R)
m2 = op_matrix(O2, G.R)
m3 = op_matrix(O3, G.R)

m3d = op_matrix_3d([m1, m2, m3])    # Поєднання матриць операцій

m_flat = op_matrix_flat(m3d)        # Об’єднана матриця операцій

permutation(G, m_flat)              # Основний алгоритм

print("Done.")
```

Рисунок 3.13 – Головний блок команд програми

В даному блоці на початку створюється екземпляр класу GSA (змінна *G*), вміст якого формується на основі файлу «test.kiss2». Далі на основі заданих функцій *O1*, *O2*, *O3* створюються три матриці операцій *m1*, *m2*, *m3*. На їх основі спочатку створюється «тривимірна» матриця операцій *m3d*, яка

далі перетворюється на об’єднану матрицю операцій *m_flat*. Граф-схема *G* і матриця *m_flat* передаються у функцію *permutation*. Ця функція здійснює повний перебір усіх варіантів кодування станів і виводить на екран усі знайдені повні розв’язки задачі алгебраїчного синтезу.

3.5 Пошук часткових розв’язків

Часткові розв’язки задачі алгебраїчного синтезу можуть бути знайдені за допомогою модифікованої функції *permutation* (рис. 3.11). До функції мають бути внесені наступні зміни.

1. Програма на рис. 3.11 веде підрахунок кількості знайдених повних розв’язків, яку зберігає в змінній *n_sol*. При пошуку часткових розв’язків необхідно також зберігати інформацію їх кількість. Для цього можна використовувати або єдину змінну, що підраховує загальну кількість часткових розв’язків, або окремі змінні для підрахунку кількості знайдених розв’язків з одним непокритим переходом, з двома непокритими переходами і так далі. Оберемо другий варіант і додамо після команди

```
n_sol = 0   # Кількість знайдених формальних розв’язків
```

додамо команди

```
n_sol_1 = 0 # Кількість розв’язків з 1 непокритим переходом
n_sol_2 = 0 # Кількість розв’язків з 2 непокритими переходами
n_sol_3 = 0 # Кількість розв’язків з 3 непокритими переходами
```

Подібним чином можна рахувати кількість знайдених розв’язків з будь-яким числом непокритих переходів.

Безпосередній підрахунок цих значень відбуватиметься тоді, коли буде відома кількість непокритих переходів. Для цього після команди

```
        n = compare(G.mm, m_flat) # Підрахунок кількості
                                  # нереалізованих переходів
```

слід додати такі команди:

```
        if (n==1):
            n_sol_1 += 1

        if (n==2):
            n_sol_2 += 1

        if (n==3):
            n_sol_3 += 1
```

У команді

```
        if (n==0): # Якщо усі переходи реалізуються операційно
```

константа 0 означає ту кількість непокритих переходів, при якій розв'язки будуть виводитись на екран. Значення 0 відповідає повним розв'язкам, в яких немає непокритих переходів. Якщо є потреба у виведенні на екран часткових розв'язків, слід замінити в цій команді вираз «**==0**» на вираз «**<=1**» або подібний. Наприклад, при використанні команди

```
        if (n<=3):
```

на екран будуть виводитись усі повні розв'язки та усі неповні розв'язки, в яких кількість непокритих переходів не перебільшує 3.

В самому кінці функції слід вивести ці значення на екран:

```
    print("Повних розв'язків:", n_sol)
    print("З 1 непокритим переходом:", n_sol_1)
    print("З 2 непокритими переходами:", n_sol_2)
    print("З 3 непокритими переходами:", n_sol_3)
```

Алгоритм, наведений на рис. 2.10, передбачає перевірку часу виконання програми і переривання програми у випадку виходу за встановлений ліміт часу. Цей механізм може бути легко реалізований за допомогою функціоналу модуля *time*. Однак введення обмеження по часу доцільне у тому випадку, коли програма використовується з практичною метою у складі САПР пристроїв керування. На етапі дослідження розроблених алгоритмів очікується, що людина-дослідник може вручну перервати роботу програми у будь-який потрібний момент. Тому у програмі, розробленій в даній кваліфікаційній роботі, обмеження в часі роботи не реалізоване.

3.6 Тестування роботи програми і дослідження розроблених алгоритмів

Розглянемо роботу програми на прикладі ГСА G_1 (рис. 1.2), що задається у вигляді KISS2 файлу, вміст якого наведений на рис. 1.3.

Нехай множина ОП утворена операціями O_1 – O_3, що визначаються виразами (1.3) – (1.5) і програмно задаються відповідно до рис. 3.6. Програма виконується близька двох секунд і дає результат, наведений на рис. 3.14.

Перші 5 рядків результату формуються функцією *read_kiss2* і відображають основні параметри ГСА, прочитаної з файлу. Далі виводяться результати роботи функції *permutation*. По цим результатам можна відзначити таке.

1. В результаті повного перебору варіантів кодування станів було знайдено 8 повних формальних розв’язків задачі алгебраїчного синтезу. На рис. 3.14 розв’язки позначені як «Solution 1» – «Solution 8».

2. Формат виведення розв’язків подібний до рис. 3.1 і містить лише інформацію про кодування станів. Слід зауважити, що стани виводяться не у порядку збільшення індексів, а у тому порядку, в якому вони зустрічались під час читання файлу KISS2.

3. Перший знайдений формальний розв’язок (Solution 1) відповідає розв’язку, наведеному на рис. 1.5.

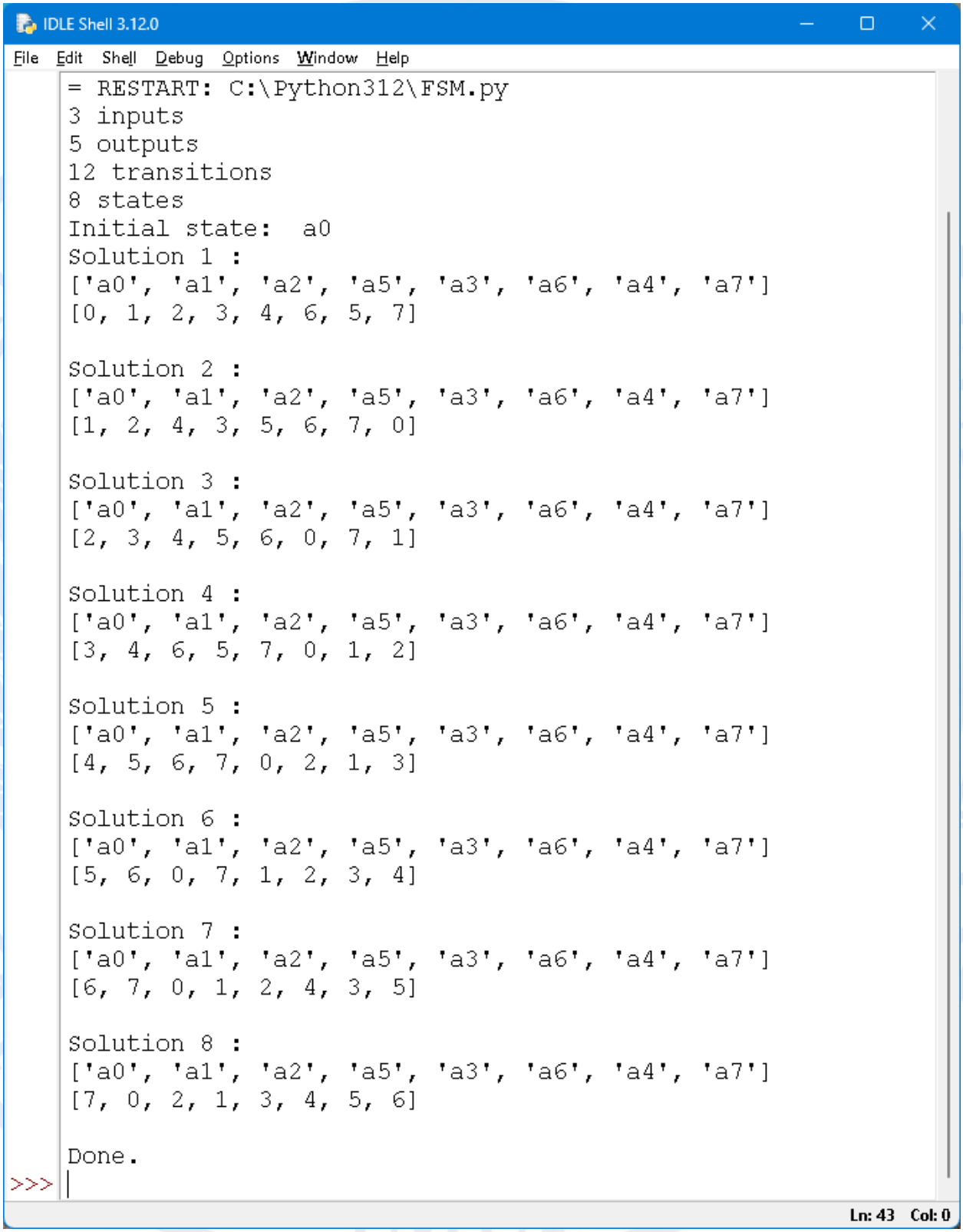

Рисунок 3.14 – Результат роботи програми для ГСА G_1 і ОП $O_1 – O_3$

Якщо операцію «⊕5» замінити на операцію «⊕3», буде знайдено чотири повних розв’язки (рис. 3.15).

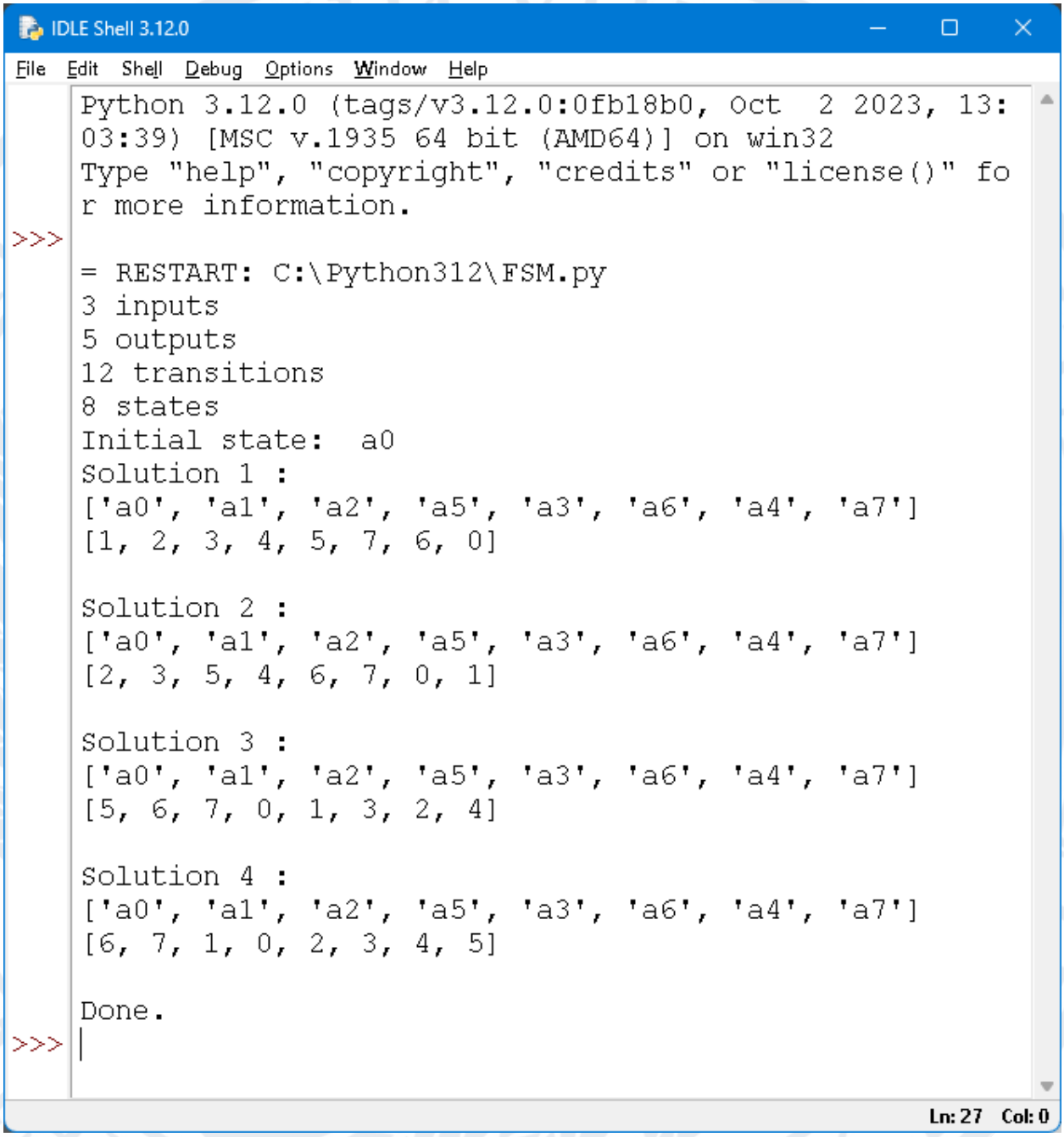

Рисунок 3.15 – Приклад знаходження чотирьох повних розв’язків

Якщо замість операцій «+1», «+2», «⊕5» обрати операції «+1», «+3», «⊕4», жодного повного розв’язку знайдено не буде і результат роботи програми буде відповідати рис. 3.16. Як можна бачити, в програмі немає виведення окремого повідомлення про відсутність повних розв’язків.

Подібне повідомлення планується реалізувати в майбутньому при розробці графічного інтерфейсу програми.

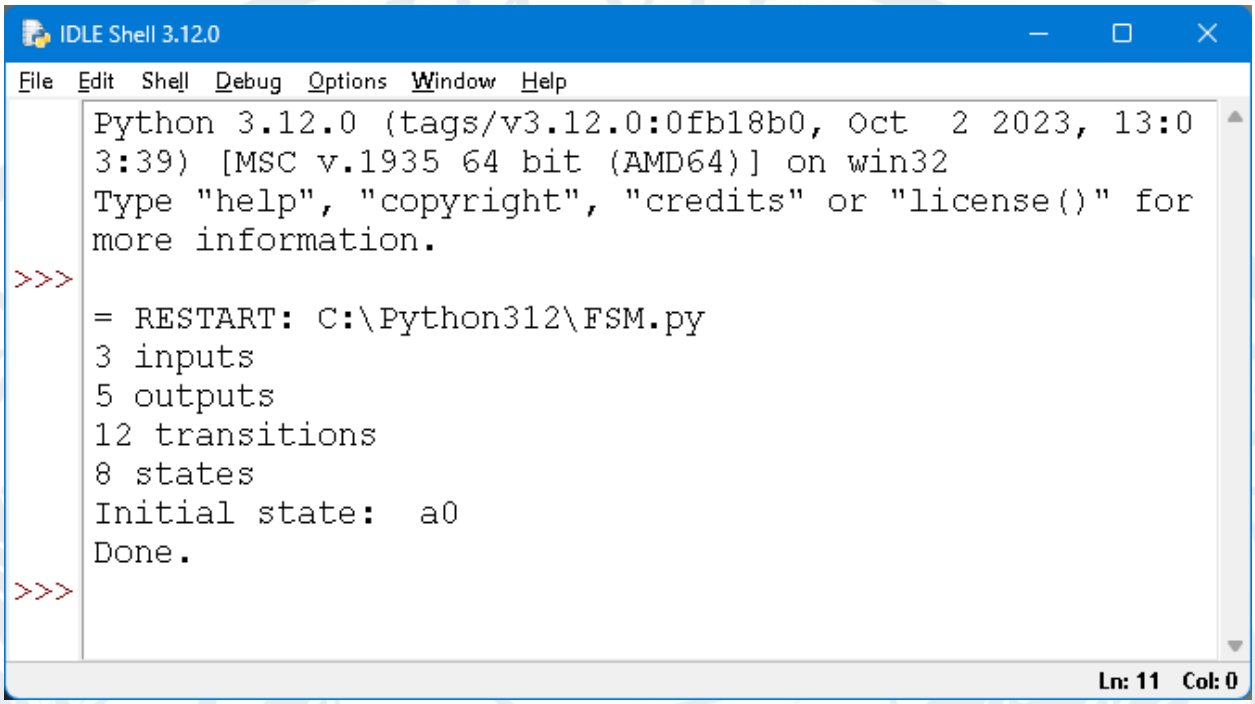

Рис. 3.16 – Результат роботи програми за відсутності повних розв’язків

Утворимо множину ОП із десяти наступних операцій:

«+1», «+2», «+3», «+4», «+5», «⊕1», «⊕2», «⊕3», «⊕4», «⊕5»

В результаті програма знаходить 8436 повних розв’язків (рис. 3.17). При цьому час роботи програми складає близька півтори хвилини. Збільшення часу пов’язане виключно з виведенням інформації на екран. Якщо у функції *permutaton* відключити (закоментувати) команди виведення розв’язків на екран, програма виконується близька 2 секунд незалежно від кількості знайдених розв’язків.

Отже, розроблена програма коректно реалізує алгоритм пошуку повних розв’язків задачі алгебраїчного синтезу МПА з ОАП і дозволяє в майбутньому провести дослідження наступних факторів:

– вплив на кількість повних розв’язків типу і кількості операцій переходів;

– час пошуку розв’язків на різних комп’ютерних системах;

– вплив розрядності коду стану на кількість повних розв’язків.

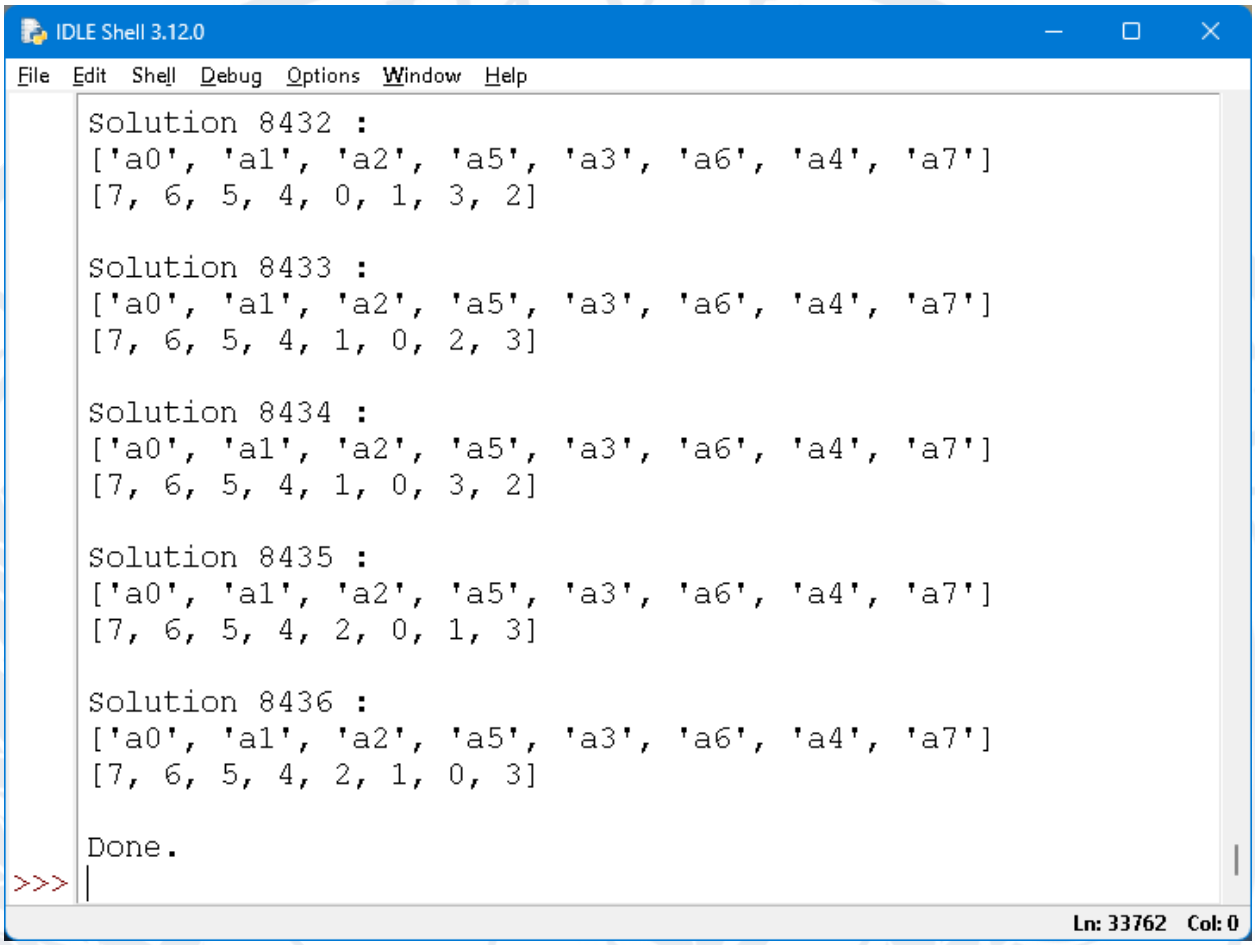

Рисунок 3.17 – Приклад результату роботи програми для десяти ОП

Розглянемо роботу програми для пошуку часткових розв’язків задачі алгебраїчного синтезу (п. 3.5). Для ГСА G_1 і ОП (1.3) – (1.5) результат роботи програми наведений на рис. 3.18.

В процесі роботи програми виводиться інформація про знайдені повні розв’язки, а також збирається інформація про кількість знайдених часткових розв’язків з одним, двома і трьома непокритими переходами. Наприкінці програма зібрана статистика виводиться на екран. Рис. 3.18 показує, що для проаналізованої ГСА існує 8 повних розв’язків, 56 часткових розв’язків з одним непокритим переходом, 200 часткових розв’язків з двома непокритими переходами та 760 часткових розв’язків з трьома непокритими переходами.

Інформація про кількість розв’язків з більшою кількістю непокритих переходів у проведеному експерименті не збиралась, враховуючи малий розмір ГСА.

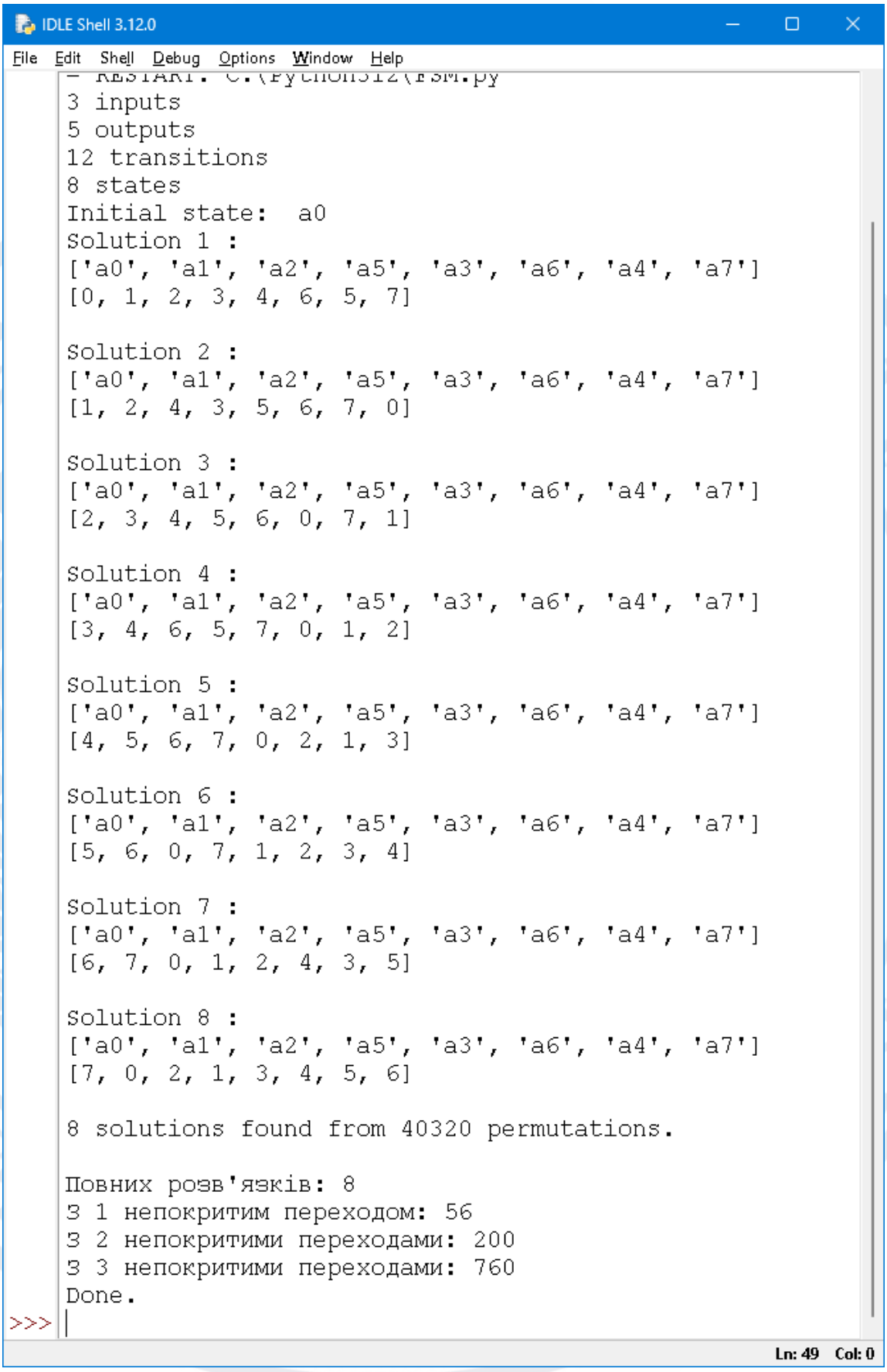

Рисунок 3.18 – Результат роботи програми для пошуку часткових розв’язків задачі алгебраїчного синтезу (ГСА G_1, ОП (1.3) – (1.5))

Утворимо множину ОП із десяти наступних операцій:

«+1», «+2», «+3», «+4», «+5», «⊕1», «⊕2», «⊕3», «⊕4», «⊕5».

Також закоментуємо у функції *permutation* команди, що відповідають за виведення знайдених розв'язків на екран. Тепер результат роботи програми відповідатиме рис. 3.19.

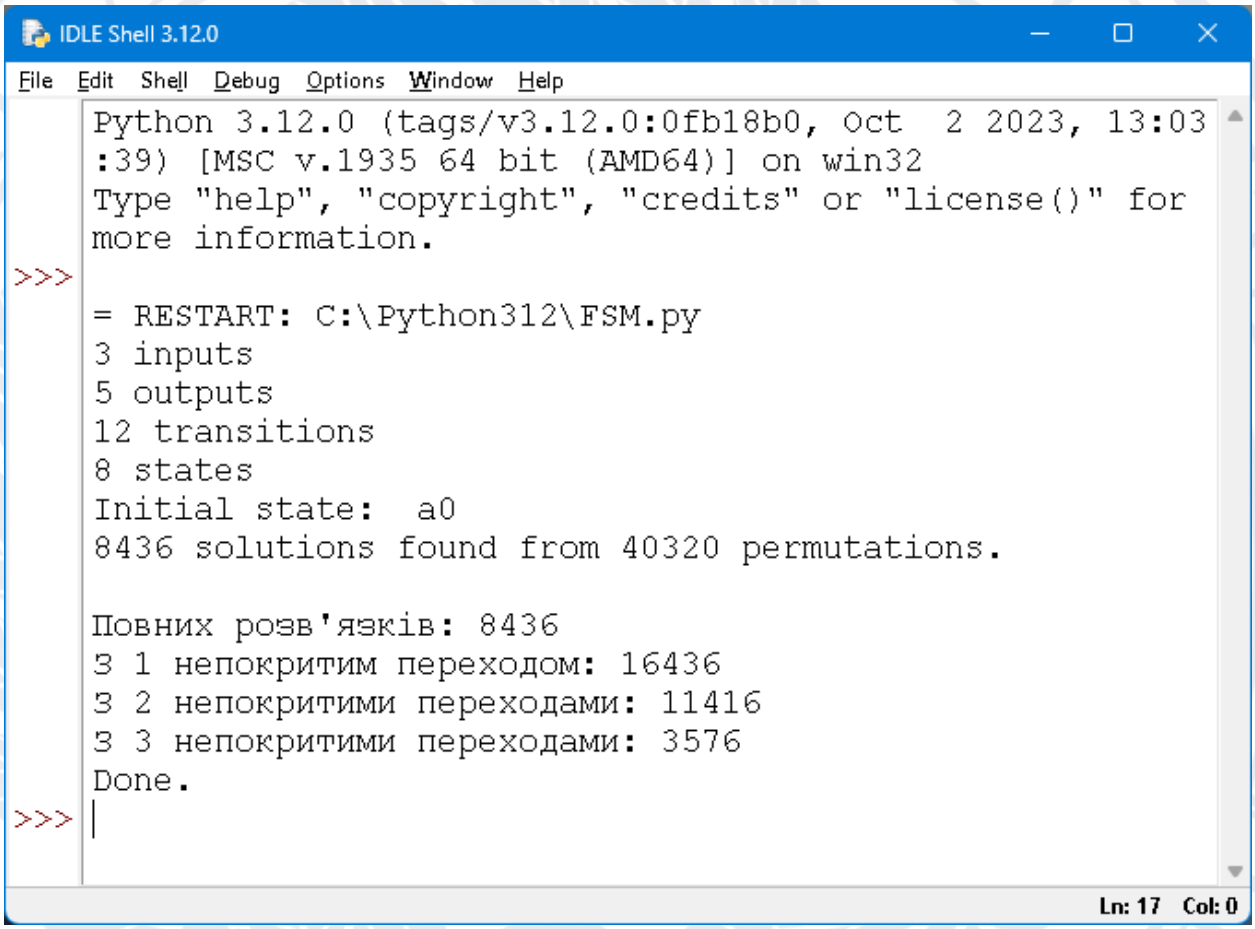

Рисунок 3.19 – Результат роботи програми для десяти ОП

Аналіз рис. 3.19 показує наступне. Загальна кількість перестановок дорівнює 40320. Якщо скласти кількості знайдених повних і часткових розв'язків, отримаємо 8436 + 16436 + 11416 + 3576 = 39864 розв'язки. При цьому залишається лише (40320 – 39864) = 456 часткових розв'язків, у яких кількість непокритих переходів більша за 3. Отже, навіть для відносно великої кількості наявних ОП кількість часткових розв'язків значно перебільшує кількість повних розв'язків.

Розглянемо випадок, коли повні розв’язки відсутні. Для цього оберемо множину ОП: «+1», «+3», «⊕4». Результат роботи програми наведений на рис. 3.20.

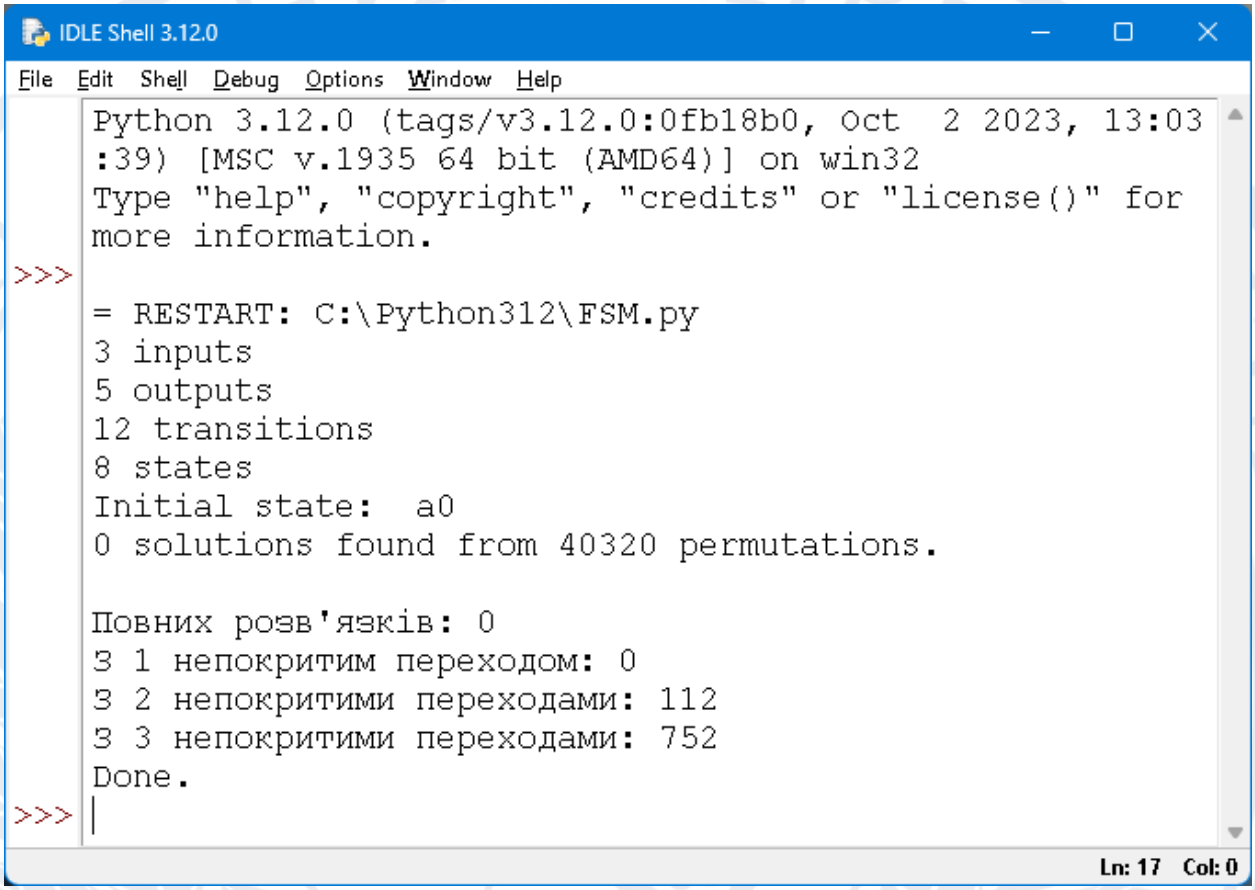

Рисунок 3.20 – Результат роботи програми за відсутності повних розв’язків

Як можна бачити, в цьому випадку відсутні не тільки повні розв’язки, але й часткові розв’язки з одним непокритим переходом. Такий результат є неочевидним і не може бути отриманий без використання розробленої програми.

В процесі дослідження розробленої програми були виконані спроби запуску програми для різних наборів операцій переходів. Підсумкові результати цього експерименту наведені в табл. 3.1. У першому стовпці вказується номер експерименту, у трьох наступних стовпцях зазначаються використовувані операції переходів (символ « – » означає, що ця ОП не використовувалась в поточному експерименті), в останніх чотирьох стовпцях

міститься статистична інформація про кількість знайдених розв'язків («0» – кількість повних розв'язків; «–1», «–2», «–3» – кількість часткових розв'язків з одним, двома і трьома непокритими переходами відповідно).

Таблиця 3.1 – Результати пошуку часткових розв'язків для ГСА G_1 і різних наборів операцій переходів

№	ОП 1	ОП 2	ОП 3	0	–1	–2	–3
1	+1	+2	–	0	8	24	64
2	+2	+3	–	0	0	0	24
3	+3	+5	–	0	0	0	0
4	+1	⊕5	–	0	0	0	36
5	+1	&2	–	0	0	2	7
6	+1	∨4	–	0	0	0	4
7	&3	∨4	–	0	0	0	0
8	+3	+7	–	0	0	0	64
9	+5	∨2	–	0	0	0	4
10	+1	∨6	–	0	0	0	1
11	+1	+2	+3	0	40	312	1024
12	+1	+2	+4	8	16	184	840
13	+2	+3	+5	0	8	232	728
14	+2	+5	+6	8	8	160	720
15	+1	+4	⊕3	0	4	56	322
16	+1	+3	⊕6	0	36	228	956
17	⊕1	⊕3	⊕5	0	0	96	720
18	∨1	&2	⊕6	0	0	10	72
19	+6	&3	⊕7	0	4	26	176
20	+1	⊕2	⊕5	0	14	114	734

Таблиця 3.1 містить результати двадцяти експериментів. В перших десяти з них (рядки таблиці з номерами 1 – 10) використано дві ОП, в

останніх десяти (рядки таблиці з номерами 11 – 20) використано три ОП. Вміст таблиці наочно демонструє, що при трьох ОП кількість розв’язків (повних та неповних) є значно більшою, ніж при двох ОП. Також можна відзначити, що кількість знайдених часткових розв’язків суттєво залежить від обраних ОП.

В цілому результати роботи розроблених програм демонструють коректність алгоритмів пошуку формальних розв’язків задачі алгебраїчного синтезу МПА з ОАП. Враховуючи відносно низьку швидкодію перебору варіантів кодування станів, на практиці доцільно застосовувати алгоритм пошуку часткових розв’язків. Він може бути легко налаштований на пошук розв’язків з будь-якою кількістю непокритих автоматних переходів. Загалом можна відзначити, що чим меншою буде задана кількість непокритих переходів, тим довший час потрібен програмі на пошук підходящого розв’язку.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну проблему програмної реалізації методу алгебраїчного синтезу мікропрограмного автомата з операційним автоматом переходів. Даний метод оснований на повному переборі варіантів кодування станів і є однією із складових методу автоматизованого синтезу даного типу мікропрограмного автомата. В рамках зазначеної проблеми виконано наступне.

1. Досліджено структурну організацію і принцип функціонування мікропрограмного автомата з операційним автоматом переходів. Проаналізовано процес алгебраїчного синтезу автомата як один із етапів проєктування схеми автомата. Обрано мову Python і бібліотеку Numpy як інструмент для розробки високоефективних програм, що використовують обробку квадратних матриць.

2. Розроблено алгоритм пошуку повних розв’язків задачі алгебраїчного синтезу мікропрограмного автомата з операційним автоматом переходів. Алгоритм дозволяє знайти усі повні розв’язки шляхом повного перебору варіантів кодування станів автомата. Недоліком алгоритму є відносно низька швидкодія, яка ускладнює пошук повних розв’язків для автоматів великої складності.

3. Розроблено алгоритм пошуку часткових розв’язків задачі алгебраїчного синтезу. Алгоритм дозволяє знаходити розв’язки з будь-якою кількістю переходів, не покритих заданою множиною операцій. При завданні кількості непокритих переходів рівною нулю алгоритм вироджується у алгоритм пошуку повних розв’язків.

4. Здійснено програмну реалізацію алгоритмів пошуку повних і часткових розв’язків задачі алгебраїчного синтезу за допомогою мови Python і бібліотеки Numpy. В якості вхідних даних виступає граф-схема алгоритму, задана у файлі формату KISS2.

5. Проведене тестування розробленої програмної реалізації. Воно підтвердило здатність алгоритмів знаходити як повні розв’язки, так і часткові розв’язки із заданою кількістю нереалізованих автоматних переходів.

Кваліфікаційну роботу виконано в рамках ініціативної науково-дослідної теми «Методи, алгоритми та засоби автоматизованого проектування пристроїв управління обчислювальних систем» (реєстраційний номер 0224U002156), яка ведеться на кафедрі інформаційних технологій ДонНУ імені Василя Стуса під керівництвом докт. техн. наук, професора Бабакова Р.М.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bailliul J., Samad T. Encyclopedia of Systems and Control. *Springer: London, UK*, 2015. 1554 p.

2. Baranov S. Finite State Machines and Algorithmic State Machines. *Seattle, WA, USA: Amazon*, 2018. 185 p.

3. Wilkes M. The best way to design an automatic calculation machine. *Manchester University Computer Inaugural Conf. Proc.*, 1951. P. 16–28.

4. Gajski D. D., Abdi S., Gerstlauer A., Schirner G. Embedded System Design: Modeling, Synthesis and Verification, 1st ed. *New York, NY, USA: Springer*, 2009.

5. Hartmanis J., Stearns R. E. Algebraic Structure Theory of Sequential Machines. *New Jersey: Prentice-Hall*, 1966. 211 p.

6. DeMicheli G. Synthesis and Optimization of Digital Circuits. *NY: McGraw-Hill*, 1994. 576 p.

7. Adamski M., Barkalov A. Architectural and Sequential Synthesis of Digital Devices, *Zielona Gora: University of Zielona Gora Press*, 2006. 199 p.

8. Barkalov A. A., Titarenko L. A., Babakov R. M., Baiev A. V. Logic synthesis for FPGA-based finite state machines: monograph. *Vinnitsya: DonNU Vasyl' Stus*. 2016. 195 p.

9. Barkalov A., Titerenko L. Logic synthesis for FSM-based control units. *Berlin: Springer*, 2009. 233 p.

10. Barkalov A., Wegrzyn M. Design of control units with programmable logic. *Zielona Gora: University of Zielona Gyra Press*. 2006. 150 p.

11. Barkalov A., Titarenko L., Barkalov A., Jr. Structural decomposition as a tool for the optimization of an FPGA-based implementation of a Mealy FSM. *Cybernetics and Systems Analysis*. 2012. №48 (2). P. 313–322.

12. Czerwinski R., Kania D. Finite State Machine Logic Synthesis for Complex Programmable Logic Devices. *Berlin: Springer*. 2013. 172 p.

13. Garcia-Vargas I., Senhadji-Navarro R. Finite state machines with input multiplexing: A performance study. *IEEE Transactions a Computer-Aided Design of Integrated Circuits and Systems*. 2015. № 34 (5). P. 867–871.

14. Kolopienczyk M., Titarenko L., Barkalov A. Design of EMB-based Moore FSMs. *Journal of Circuits, Systems, and Computers*. 2017. № 26 (7). P. 1–23.

15. Luba T. Synthesis of Logic Devices. *Warsaw: Warsaw University of Technology Press*. 2005.

16. Mano M. Digital design (4th Edition). *New Jersey: Prentice Hall*. 2006. 624 p.

17. Minns P., Elliot I. FSM-Based Digital Design Using Verilog HDL. *New Jersey: J. Wiley & Sons*. 2008. 408 p.

18. Popovskij V., Barkalov A., Titarenko L. Control and adaptation in telecommunication systems: Mathematical Foundations. Lectures Notes in Electrical Engineering. *Berlin: Springer*. 2011. № 94. 173 p.

19. Sklyarov V., Skliarova I., Barkalov A., Titarenko L. Synthesis and Optimization of FPGA-Based Systems. *Berlin: Springer*. 2014. 432 p.

20. Barkalov A.A., Babakov R.M. Operational formation of state codes in microprogram automata. *Cybernetics and Systems Analysis*. Volume 47 (2), 2011. P. 193-197.

21. Barkalov A.A., Babakov R.M. Determining the Area of Efficient Application of a Microprogrammed Finite-State Machine with Datapath of Transitions. *Cybernetics and Systems Analysis*. Volume 54 (3), 2018. P. 366-375.

22. Barkalov A.A., Titarenko L.A., Babakov R.M. Test Graph-Schemes of the Algorithms of Finite State Machines Work for Assessing the Efficiency of Automated Synthesis In Xilinx Vivado CAD. *Radio Electronics, Computer Science, Control*. 2023. No. 3 (109). P. 120-129.

23. Barkalov A.A., Titarenko L.A., Babakov R.M. Synthesis of the Finite State Machine with Datapath of Transitions According to the Operational Table of

Transitions. *Radio Electronics, Computer Science, Control*. 2022. No. 3 (109). P. 109-119.

24. Barkalov A.A., Titarenko L.A., Babakov R.M. Synthesis of VHDL Model of a Finite State Machine with Datapath of Transitions. *Radio Electronics, Computer Science, Control*. 2023. No. 4. P. 135-147.

25. LGSynth'93 Benchmark Information. *URL: http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.cbl.ncsu.edu/CBL_Docs/lgs93.html*

26. Barkalov A.A., Babakov R.M. A Matrix Method for Detecting Formal Solutions to the Problem of Algebraic Synthesis of a Finite-State Machine with a Datapath of Transitions. *Cybernetics and Systems Analysis*. Volume 59 (2), 2023. P. 190-198.

27. Васильєв О. Програмування мовою Python. *Тернопіль: Навчальна книга – Богдан*, 2019. 504 с.

28. Копей В.Б. Мова програмування Python для інженерів і науковців: навчальний посібник. *Івано-Франківськ : ІФНТУНГ*, 2019. 272 с.

29. Langtangen H. A Primer on Scientific Programming with Python. 5th Edition. *Springer*, 2016. 942 p.

30. Kiusalaas J. Numerical Methods in Engineering with Python. 2ed. *Cambridge University Press*, 2010. 434 p.

31. Hellmann D. Python Module of the Week. *URL: https://pymotw.com/2/*

32. Unpingco J. Python for Probability, Statistics, and Machine Learning. *Springer*, 2016. 288 p.

33. Desai P. Python Programming for Arduino. *Birmingham: Packt Publishing*, 2015. 57 p.

34. Torbert S. Applied Computer Science. Second Edition. *Fairfax: Springer*, 2016. 291 p.

35. Haslwanter T. An Introduction to Statistics with Python: With Applications in the Life Sciences. *Springer*, 2016. 285 p.

36. Al-Taie M, Kadry S. Python for Graph and Network Analysis. – Springer, 2017. 211 p.

37. PyFlo – The beginners guide to becoming a Python programmer. *URL: https://pyflo.net/*

38. Singh A. Master Python Using Version 3.11: Learn Python Like Never Before. *Independently published*, 2023. 463 p.

39. Whitington J. Python from the Very Beginning. Cambridge, Coherent Press, 2020. 238 p.

40. Schroeder A. The Book of Dash: Build Dashboards with Python and Plotly. San Francisco, No Starch Press, 2022. 224 p.

41. Dunn N. Python 3.8. *Webucator*, 2020. 556 p.

42. Idris I. Numpy 1.5 Beginner's Guide. *Packt Pub Ltd*, 2011. 234 p.

43. Sheppard C. Genetic Algorithms with Python. *CreateSpace Independent Publishing Platform*, 2016. 423 p.

44. Lutz M. Python Pocket Reference, 5th Edition. O'Reilly Media, 2014. 262 p.

45. Hellmann D. The Python 3 Standard Library by Example. *Addison-Wesley Professional*, 2017. 1456 p.

46. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming, 2nd Edition. *O'Reilly Media*, 2022. 1012 p.

47. Slatkin B. Effective Python: 90 Specific Ways to Write Better Python (Effective Software Development Series), 2nd Edition. *Addison-Wesley Professional*, 2019. 480 p.

48. Sweigart A. The Big Book of Small Python Projects: 81 Easy Practice Programs. *No Starch Press*, 2021. 432 p.

49. Reitz, K., Schlusser T. The Hitchhiker's Guide to Python: Best Practices for Development. *O'Reilly Media*, 2016, 336 p.

50. Hillard D. Practices of the Python Pro. *Manning*, 2020. 248 p.

51. Official Numpy documentation. *URL: https://numpy.org/doc/stable/*

52. W3 schools: Numpy Tutorial. *URL: https://www.w3schools.com/python/numpy/default.asp*

53. NumPy Tutorial – Python Library. *URL: https://www.geeksforgeeks.org/numpy-tutorial/*

54. Мізюк О. Путівник мовою програмування Python. *URL: https://pythonguide.rozh2sch.org.ua/*

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)