НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

Кафедра математичних методів системного аналізу

До захисту допущено:
Завідувач кафедри
_____ Оксана ТИМОЩУК
«___»_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Системний аналіз і управління»
спеціальності 124 «Системний аналіз»
на тему: «Матричні формальні граматики та граматики, які керуються мережами Петрі»

Виконав:
студент IV курсу, групи КА-05
Бичков Денис Віталійович _____

Керівник:
Доцент, к.ф-м.н., Стусь Олександр Вікторович _____

Консультант з економічного розділу:
Доцент, к.е.н., Рощина Надія Василівна _____

Консультант з нормоконтролю:
Доцент, к.ф-м.н., Статкевич Віталій Михайлович _____

Рецензент:
Василик Ольга Іванівна професор кафедри _____
математичного аналізу та теорії ймовірностей,
д.ф.-м.н., доц.

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 124 «Системний аналіз»
Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Оксана ТИМОЩУК
«__» _____2024 р.

ЗАВДАННЯ
на дипломну роботу студенту
Бичкову Денису Віталійовичу

1. Тема роботи «Матричні формальні граматики та граматики, які керуються мережами Петрі», керівник роботи Стусь Олександр Вікторович, доцент, кандидат фізико-математичних наук, затверджені наказом по університету від «___»_____ 20__ р. №_____
2. Термін подання студентом роботи 11.06.2024.
3. Вихідні дані до роботи: матричні формальні граматики, граматики, які керуються мережами Петрі.
4. Зміст роботи: теоретична частина, математична частина, програмна реалізація, функціонально-вартісний аналіз.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) — презентація, “Аналіз результатів”, “Подальші дослідження”.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
економічний	Рощина Н.В., доцент, к.е.н.		

7. Дата видачі завдання _____.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формулювання тематики (напряму) дослідження.	12.03.2024	виконано
2	Ознайомлення з ДСТУ	21.04.2024	виконано
3	Огляд літератури	22.04.2024	виконано
5	Уточнення теми дипломної роботи	22.05.2024	виконано
6	Розробка програмного продукту	22.05.2024 — 29.05.2024	виконано
8	Оформлення пояснювальної записки	29.05.2024 — 02.06.2024	виконано
9	Підготовка презентації для захисту	03.06.2024	виконано
10	Попередній захист дипломної роботи	04.06.2024	виконано

Студент Денис БИЧКОВ

Керівник Олександр СТУСЬ

РЕФЕРАТ

Дипломна робота: 84 с., 28 рис., 7 табл., 2 дод., 11 джерел.

ФОРМАЛЬНА МОВА, ФОРМАЛЬНА ГРАМАТИКА, МЕРЕЖА ПЕТРІ, МАТРИЧНА ГРАМАТИКА, ГРАМАТИКА КЕРОВАНА МЕРЕЖЕЮ ПЕТРІ, КЕРОВАНА ГРАМАТИКА

Об’єктами роботи є граматики, які керуються мережами Петрі та матричні формальні граматиками.

Предметом дослідження є алгоритми, які дозволяють здійснювати перехід між граматиками, які керуються мережами Петрі та матричними формальними граматиками.

Мета роботи полягає в розробці програмного забезпечення для конвертування граматик, що керуються мережами Петрі у матричні формальні граматики та навпаки.

Результатом роботи є розроблене програмне забезпечення на Python, яке є реалізацією розглянутих алгоритмів для переходу між граматиками, що керуються мережами Петрі та матричними формальними граматиками.

ABSTRACT

Bachelor's Thesis: 84 p., 28 fig., 7 tab., 2 app., 11 references.

FORMAL LANGUAGE, FORMAL GRAMMAR, PETRI NET, MATRIX GRAMMAR, PETRI NET CONTROLLED GRAMMAR, CONTROLLED GRAMMAR

The objects of work are grammars controlled by Petri nets and matrix formal grammars.

The subject of research is the set of algorithms that allow transition between grammars controlled by Petri nets and matrix formal grammars.

The goal of the work is to develop software for converting grammars controlled by Petri nets into matrix formal grammars and vice versa.

The result of the work is the developed software (Python), which is the realization of the considered algorithms for the transition between grammars controlled by Petri nets and matrix formal grammars.

ЗМІСТ

ВСТУП……...8
РОЗДІЛ 1 ТЕОРЕТИЧНА ЧАСТИНА…….……...9
1.1 Формальні мови та граматики……...9
1.2 Мережі Петрі……14
1.3 Мови мереж Петрі……21
Висновки до розділу 1……23
РОЗДІЛ 2 МАТЕМАТИЧНА ЧАСТИНА……..24
2.1 Граматики керовані мережею Петрі……...24
2.2 Матричні граматики……26
2.3 Доведення можливості переходу між матричними формальними граматиками та граматиками, які керовані мережами Петрі……..26
2.4 Приклад побудови граматики, яка керована мережею Петрі по матричній граматиці……...34
2.5 Приклад побудови матричної граматики по граматиці, яка керована мережею Петрі……36
Висновки до розділу 2……....37
РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ……....38
3.1 Обґрунтування вибору технологій для програмної реалізації…….38
3.2 Опис реалізованих функцій……40
3.3 Результати програмної реалізації……42
Висновки до розділу 3……45
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ……46
4.1 Постановка задачі проектування……47
4.2 Обґрунтування функцій програмного продукту……47
4.3 Обґрунтування системи параметрів програмного продукту ……...50
4.4 Аналіз експертного оцінювання параметрів……..53

4.5 Аналіз рівня якості варіантів реалізації функцій ……57
4.6 Економічний аналіз варіантів розробки ПП……..59
4.7 Вибір кращого варіанту ПП техніко-економічного рівня…….65
Висновки до розділу 4……66
ВИСНОВКИ……67
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ……..68
ДОДАТОК А ЛІСТИНГ КОДУ……..……69
ДОДАТОК Б ПРЕЗЕНТАЦІЯ...……..……75

ВСТУП

Прогрес у розвитку технологій штучного інтелекту значно виріс за останні кілька років, і у багатьох аспектах це стається саме завдяки дослідженням у галузі математичної логіки. Ми досі потребуємо нових досліджень у цій царині. Наприклад завжди залишається актуальним створення нових більш потужних мов програмування, а якісна формалізація природних мов буде корисною для розвитку мовних моделей штучного інтелекту, на кшталт популярного нині Chat GPT.

У цій роботі було зосереджено увагу на матричних керованих формальних граматиках та граматиках, що керовані мережами Петрі. Це важливі інструменти в області моделювання, які використовуються для опису та аналізу різноманітних систем від біологічних процесів до мереж зв’язку.

Мета даної дипломної роботи — спираючись на теоретичний матеріал з теми, розробити програмний продукт, який реалізує переходи між матричними формальними граматиками та граматиками, які керуються мережами Петрі.

Робота складається з чотирьох розділів: “Теоретична частина”, “Математична частина”, “Програмна реалізація”, “функціонально-вартісний аналіз програмного продукту”. У першому розділі викладено основну теоретичну базу, а саме розглядаються основні концепції формальних мов та граматик й мереж Петрі. Другий розділ присвячений математичному обґрунтуванні переходу між різними типами керованих граматик, також у ньому продемонстровано приклади їх побудови без програмного засобу. У третьому розділі описано етап програмної реалізації, надається опис розробленого продукту і його функціоналу та порівняння між результатами, що були отримані внаслідок роботи програми й від побудови вручну. У четвертому розділі — функціонально-вартісний аналіз.

РОЗДІЛ 1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Формальні мови та граматики

Формальна мова — це штучно створена людиною множина слів, фраз, синтаксичних конструкцій, яка дотримується певних правил та слугує для вирішення завдань або має специфічну сферу застосування. Прикладами таких мов є: математичні формули, мови програмування, хімічні рівняння, тощо. На відміну від природних мов, формальні повинні бути повністю однозначними та для них дуже важлива структура. Штучні мови для комунікації на кшталт міжслов'янської та есперанто не можуть бути формальними, бо в них можлива метафоричність.

Як і природні, формальні мови завжди мають алфавіт — довільну непорожню, скінченну множину, елементи якої називають символами або літерами. Слово — це довільна скінченна, можливо порожня послідовність літер із алфавіта. Довжину слова α, тобто кількість символів у ньому позначають — $|\alpha|$. Якщо $|\alpha| = 0$, це слово є порожнім і позначається ε. Множина всіх слів над алфавітом A позначається A^*, множина без порожнього слова — A^+, множина усіх слів довжиною не більше k — A^k. Правила, яким підпорядковуються формальні мови, називають формальними граматиками.

Формальна граматика - це кортеж вигляду $G =< V, T, P, S >$, де:

- V — алфавіт нетермінальних символів, що не мають конкретного значення для граматики та замінюються;
- T — алфавіт термінальних символів, що мають усталене, незмінне значення, наприклад букви;
- P — непорожня скінченна множина продукцій над алфавітом (правила);

- S — фіксований нетермінальний символ, яким починають опис граматики (джерело), $S \in V$.

Правила із множини P записуються у вигляді $\alpha \to \beta$. Нехай задано алфавіт $A = V \cup T, V \cap T = \varnothing$ та деякі v, u, α, $\beta \in A^*$, тоді слово $\gamma_2 = u\beta v \in A^*$ можна отримати продукцією $\alpha \to \beta$ зі слова $\gamma_1 \in A^*$, якщо $\gamma_1 = u\alpha v$, тобто відбувається заміна деякого входження α на β. Якщо виникає потреба написати продукцію з однаковою лівою частиною, використовують символ "|", наприклад $\alpha \to \beta|\beta\lambda$ означає два правила: $\alpha \to \beta$ та $\alpha \to \beta\lambda$.

Отже, формальною мовою L, що породжена формальною граматикою G називають $L[G]$ — множину слів над термінальним алфавітом T, які можливо отримати із джерела S через скінченну кількість застосувань продукцій із множини P.

Наведемо простий приклад деякої мови та формальної граматики, яка її породжує. Як вже було сказано, математичні формули можна вважати формальними мовами, тоді:

Нехай M — множина арифметичних виразів з цілими числами

$L[G] = \{\alpha \in M\}; G =< V, T, P, S >$

$V = \{S, A, B, O, X\}$

$T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0, +, -, \cdot, \div, (,)\}$

$P = \{S \to A, A \to BOB|\varepsilon$

$B \to X|XB|A,$

$O \to + \mid - \mid \cdot \mid \div,$

$X \to 0|1|2|3|4|5|6|7|8|9|(|)\}$

Нехай $w \in L: w = \{109 \cdot (6 + 36)\}$, тоді щоб отримати це слово матимемо наступний ланцюжок із виконання продукцій:

$S \to A \to BOB \to XB \cdot A \to 1XB \cdot BOB \to 10X \cdot XB + XB \to$

$\to 109 \cdot (X + 3XB \to 109 \cdot (6 + 36X \to 109 \cdot (6 + 36)$

Формальні граматики є потужним інструментом для аналізу та опису структури різноманітних мовних конструкцій, а також для моделювання різних видів обчислень. Вони використовуються в різних областях, насамперед в інформатиці, лінгвістиці та математиці. Наведемо деякі приклади галузей, у яких застосовують формальні граматики.

1. Мови програмування. Формальні граматики використовуються для визначення синтаксису мов програмування. Наприклад, синтаксис таких мов, як C, Java, Python тощо, часто описується за допомогою контекстно-вільних формальних граматик.
2. Розробка компіляторів. Компілятори перекладають мови програмування високого рівня в машинний код. Формальна граматика має важливе значення на етапах лексичного аналізу та розбору компілятора, щоб переконатися, що вихідний код відповідає правилам синтаксису мови.
3. Обробка природної мови. Формальні граматики використовуються для розбору та аналізу структури речень природною мовою.
4. Теорія автоматів. Формальні граматики тісно пов’язані з теорією автоматів, яка вивчає абстрактні машини та обчислювальні проблеми.
5. Моделювання біологічних систем. У біоінформатиці формальні граматики використовуються для опису біологічних послідовностей, таких як ДНК, РНК і білки.

Над формальними мовами можна проводити наступні операції:

- теоретико-множинні операції: об’єднання $L_1 \cup L_2$, перетин $L_1 \cap L_2$, різниця $L_1 \backslash L_2$ та доповнення $\overline{L}$;
- конкатенація мов: $L_1 \cdot L_2 = \{\alpha \cdot \beta : \alpha \in L_1, \beta \in L_2\}$;
- замикання Кліні: $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \cup L^n$, де L^* складається з усіх можливих конкатенацій слів із L, а при $n = 0$: $L^* \in \varepsilon$;

- обертання мови. Дзеркальним відображенням слова називають слово записане у зворотньому порядку, тобто якщо $w = abb$, тоді $w^R = bba$. Обертанням формальної мови L є мова L^R, яка складається із дзеркальних відображень слів мови L, тобто $L^R = \{w^R : w \in L\}$.

Якщо формальні граматики породжують одну й ту саму мову, їх називають еквівалентними: $(G_1 \sim G_2) \Leftrightarrow (L[G_1] = L[G_2])$.

Засновник генеративного напряму у лінгвістиці — Ноам Хомський у 1956 році описав класифікацію формальних граматик, що нині є найпоширенішою та названа його ім'ям [7, 8]. Отже, ієрархія Хомського поділяє формальні граматики на 4 типи:

- загальний тип (граматики типу 0) — будь-яка формальна граматика відноситься до типу 0;
- контекстно-залежні (граматики типу 1) — якщо продукції вигляду: $\gamma_1 A \gamma_2 \to \gamma_1 \alpha \gamma_2$, де $A \in V$, $\alpha \in (V \cup T)^+$, $\gamma_1, \gamma_2 \in (V \cup T)^*$;
- контекстно-вільні (граматики типу 2) — якщо продукції вигляду: $A \to \alpha$, де $A \in V$, $\alpha \in (V \cup T)^*$;
- регулярні (граматики типу 3) — якщо продукції вигляду: $A \to \alpha B | \alpha | \varepsilon$, де $A, B \in V$, $\alpha \in T$.

Також можна зустріти розгалуження регулярних граматик на праволінійні та ліволінійні.

Для ієрархії Хомського справедливі наступні твердження:

- кожна граматика типу 3 є граматикою типу 2;
- кожна граматика типу 2 без ε-продукцій є граматикою типу 1;
- кожна формальна граматика є граматикою типу 0.

Ієрархію Хомського у вигляді діаграми Вена зображено на рисунку 1.1.

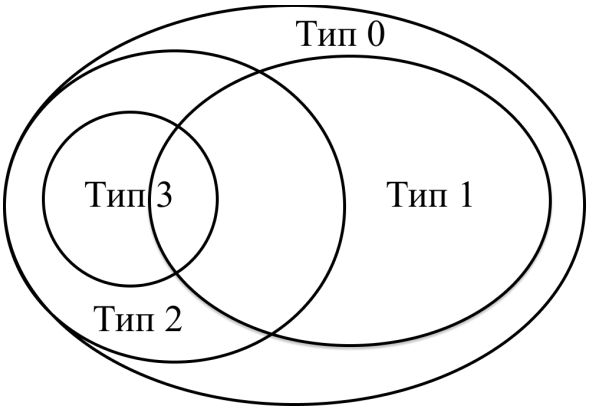


Рисунок 1.1 — Ієрархія Хомського у вигляді діаграми Вена

Формальну мову називають відповідно до найбільш обмеженого типу граматики, яка її породжує. Формальна граматика G, яка була запропонована у якості прикладу, відноситься до граматик типу 2 за ієрархією Хомського, тобто до контекстно-вільних, тому що P має правило: $A \to BOB|\varepsilon$, яке не дозволено у регулярних граматиках, а також у контекстно-залежних, бо є "ε" продукція. Звідси робимо висновок, що формальна мова $L[G]$ (арифметичні вирази з цілими числами) також є контекстно-вільною, за умови, що не існує еквівалентної формальної граматики, що породжує цю мову та відноситься до регулярних.

Кожний клас формальних мов, породжених граматиками певного типу, згідно з ієрархією Хомського, яка запропонована в [7, 8], відповідає певному класу абстрактних автоматів. Таку відповідність наочно зображають у вигляді таблиці 1.1 (детальніше, див., наприклад, [1]).

Таблиця 1.1 — Відповідність класу абстрактних автоматів до типів граматик

Тип граматики	Клас формальних мов	Клас абстрактних автоматів
0	Напіврозв'язні	Машини Тьюрінга
1	Контекстно-залежні	Лінійно-обмежені МТ
2	Контекстно-вільні	Магазинні автомати
3	Регулярні	Скінченні автомати

1.2 Мережі Петрі

Мережа Петрі — засіб для моделювання, що був запропонований 1962 року математиком Карлом Адамом Петрі. Цей інструмент використовують для моделювання та аналізу різноманітних паралельних систем і процесів у різних галузях, наведемо їх приклад.

1. Моделювання програмних систем. Мережі Петрі використовуються для визначення структури та поведінки програмних систем, зокрема для аналізу паралельних алгоритмів, конкурентних процесів у програмах, виявлення потенційних проблем з блокуванням, тощо.
2. Розробка систем управління. Мережі Петрі допомагають в проектуванні та аналізі систем управління, таких як автоматизовані системи виробництва, транспортні системи, тощо.
3. Аналіз виробничих процесів. Мережі Петрі допомагають в оцінці ефективності, управлінні запасами, плануванні та оптимізації процесів.
4. Бізнес-процеси. Мережі Петрі можуть бути використані для моделювання та аналізу бізнес-процесів, таких як логістичні потоки, процеси розподілу, а також управління проектами.

Джеймс Пітерсон у своїй праці [2] визначає мережу Петрі як кортеж вигляду: $C =< P, T, I, O >$, де:

- P — множина позицій $P = \{p_1, p_2, \dots p_n\}, n \in \mathbb{N}$;
- T — множина переходів $T = \{t_1, t_2, \dots t_n\}, n \in \mathbb{N}$;
- I — вхідна функція — відображає перехід t_j в множину вхідних позицій переходу $I(t_j)$;
- O — вихідна функція — відображає перехід t_j в множину вихідних позицій переходу $O(t_j)$. Множини P та T ніколи не перетинаються.

Приклад структури мережі Петрі:

$C = < P, T, I, O >$

$P = \{p_1, p_2, p_3, p_4, p_5\}; \ T = \{t_1, t_2, t_3, t_4\}$

$I(t_1) = \{p_1\}; \ O(t_1) = \{p_2\}$

$I(t_2) = \{p_2\}; \ O(t_2) = \{p_3\}$

$I(t_3) = \{p_2\}; \ O(t_3) = \{p_4\}$

$I(t_4) = \{p_3\}; \ O(t_4) = \{p_5\}$

Можна помітити, що подання такого виду не є зручним для моделювання, тому для мереж Петрі частіше використовується спосіб зображення у вигляді графу. Граф мережі Петрі — це дводольний орієнтований мультиграф $G =< V, A >$, де A — комплект направлених дуг, а V — множина вершин, $V = P \cup T$, але $P \cap T = \varnothing$. Приклад графу, який еквівалентний структурі, що описана раніше, зображено на рисунку 1.2.

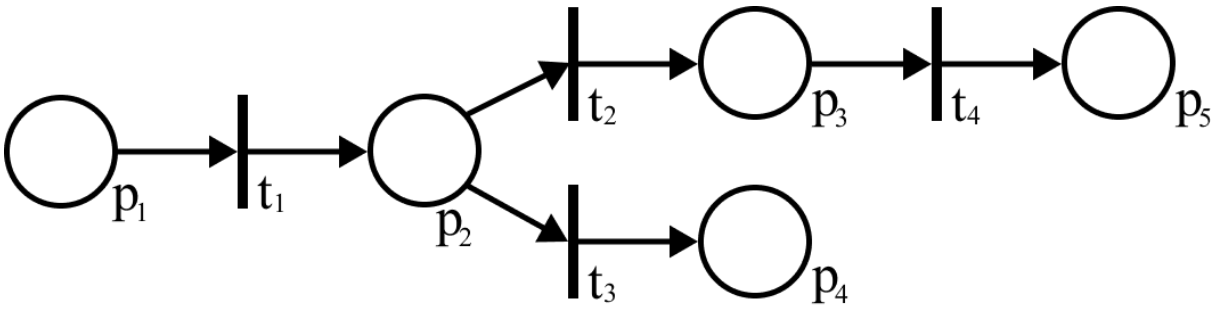


Рисунок 1.2 — Граф мережі Петрі C

Існує поняття двоїстої мережі Петрі, у якій позиції та переходи — це відповідно позиції та переходи іншої мережі Петрі, які поміняли місцями. Тобто двоїста мережа до $C =< P, T, I, O >$ є $\overline{C} =< T, P, I, O >$. Аналогічно структура двоїстої до описаної вище мережі Петрі буде навпаки: $T = \{t_1, t_2, t_3, t_4, t_5\}; \ P = \{p_1, p_2, p_3, p_4\}$.

Приклад графу двоїстої мережі Петрі до вище описаної зображено на рисунку 1.3.

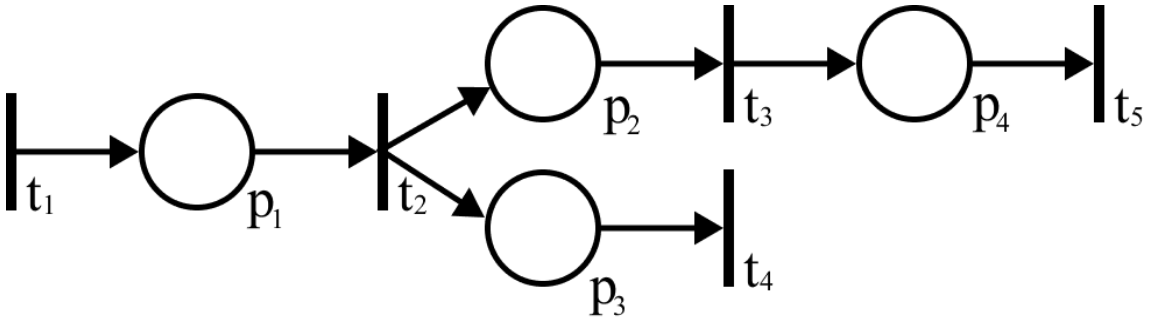


Рисунок 1.3 — Граф двоїстої мережі Петрі $\overline{C}$

Мережі Петрі працюють згідно запусків переходів. Запуски можуть приводити до зміни кількості фішок (токенів) у позиціях. Присвоєння фішок позиції називається маркуванням і позначається μ. μ є функцією, що відображає множину позицій в множину невід'ємних цілих чисел $\mathbb{Z}^{\geq}$, $\mu: P \to \mathbb{Z}^{\geq}$. Також μ часто називають вектором маркування. Задамо мережі Петрі з першого прикладу маркування $\mu = (2, 0, 0, 3, 1)$. На рисунку 1.4 зображено граф маркованої мережі Петрі C.

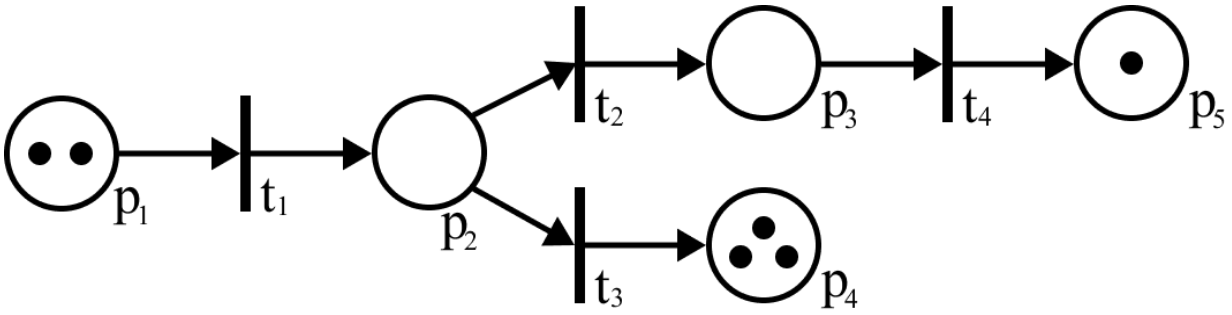


Рисунок 1.4 — Граф маркованої мережі Петрі C

Виконання мережі Петрі відбувається за наступними правилами:

- умова запуску — наявність фішок у вхідних позиціях у кількості не меншій кількості зв'язків (кратності дуги);

- при виконанні умови може відбутись запуск — з вхідних позицій видаляються фішки у кількості рівній кратності дуги;
- перехід не обов’язково повинен бути запущений;
- у вихідні позиції фішки додаються у кількості рівній кратності дуги.

Запуск переходу замінює поточне маркування μ на нове μ'. Для того щоб відслідковувати різні стани мережі Петрі, та досліджувати маркування на досяжність використовують графи досяжності та дерева покриття, приклад якого зображено на рисунку 1.5.

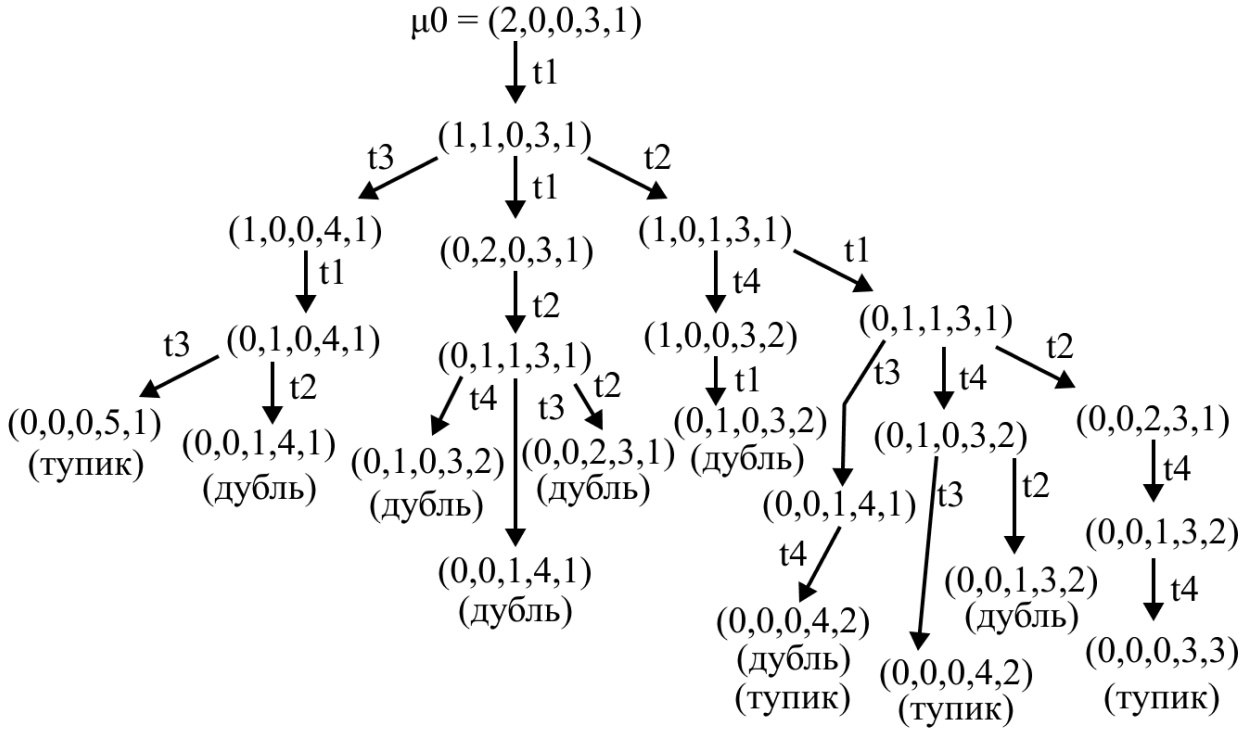


Рисунок 1.5 — Дерево покриття маркованої мережі Петрі C

Варто зауважити, що якщо виконується умова для запуску кількох переходів одночасно — такі переходи називають конфліктними. Конфлікти у мережах Петрі можуть бути розв’язані або за принципом пріоритетності або за випадком.

Також існує матричний спосіб подання мережі Петрі — матриці входів та виходів:

- кожен і-й рядок матриці відповідає переходу T_i;
- кожен j-й стовпчик матриці відповідає позиції P_j;
- матриця входів D^- показує кількість зв'язків від позиції P_j до переходу T_i;
- матриця виходів D^+ показує кількість зв'язків від переходу T_i до позиції P_j.

Для вказаної мережі Петрі C матриці входів та виходів матимуть наступний вигляд:

$$D^- = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad D^+ = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Такий вигляд опису мереж Петрі є зручнішим для аналітичних операцій та програмної реалізації.

Запуск переходу мережі Петрі у матричному вигляді:

Якщо для і-го рядка матриці входів та вектора маркування μ виконується: $\forall j D^-_{ij} \leq \mu_j$, тоді умова для запуску переходу T_i є виконаною і при запуску перехода обчислення відбувається наступним чином:

$$\mu_j' = \mu_j - D_{ij}^- + D_{ij}^+, j \in \overline{1,n}.$$

Приклад запуску переходу t_1 мережі Петрі C з початкового маркування $\mu = (2, 0, 0, 3, 1)$:

$\forall j D^-_{1j} \leq \mu_j$? $\forall j$ $(1\,0\,0\,0\,0) \leq (2\,0\,0\,3\,1) \Rightarrow$ умова запуску виконана;

$\mu' = (2\,0\,0\,3\,1) - (1\,0\,0\,0\,0) + (0\,1\,0\,0\,0) = (1\,1\,0\,3\,1)$

Отриманий математично результат запуску переходу співпадає з тим, що було передбачено за графом та зображено на дереві покриття.

Загальні властивості мереж Петрі:

1. Досяжність. Маркування, до яких не можна дійти з початкового маркування називають недосяжними.
2. k-обмеженість. Якщо в будь-якій позиції кількість фішок не перевищує k — мережа Петрі є k-обмеженою.
3. Збережуваність. Якщо неможливе виникнення і знищення ресурсів — дана мережа Петрі володіє властивістю зберігання. Будь-яка збережувана мережа Петрі є k-обмеженою. Якщо не існує такого k, що мережа Петрі є k-обмеженою — мережа Петрі не є збережуваною.
4. Активність переходів. Переходи мереж Петрі мають різні рівні активності:
   - перехід ніколи не може бути запущений (0 рівень);
   - існує досяжне маркування, яке дозволяє запуск даного переходу (1 рівень);
   - для довільного n існує така послідовність запусків переходів, в якій перехід t присутній принаймні n разів (2 рівень);
   - існує нескінченна послідовність запусків переходів, починаючи з μ_0, в якій перехід t присутній нескінченну кількість разів (3 рівень);
   - для довільного досяжного з μ_0 маркування μ існує послідовність запусків переходів, в якій перехід t присутній нескінченну кількість разів (4 рівень).

Мережа Петрі володіє властивістю активності, якщо з початкового маркування можливий перехід у будь-який інший досяжний стан. Рівень активності мережі Петрі — це найменший з рівнів активності переходів.

При моделюванні систем за допомогою мереж Петрі, керуються принципом “передумова → подія → післяумова” у якому в якості передумови виступають входи, подіями — позиції, а післяумовами — виходи [2].

Для деяких завдань потужність мережі Петрі може виявитись невиправданою, тому виділяють кілька підкласів з меншою складністю, які відрізняються накладеними на них обмеженнями.

Підкласи мереж Петрі (МП) [3]:

1. Автоматна мережа (АМ) — це така мережа Петрі, у якій кожен перехід має рівно одну вхідну позицію і рівно одну вихідну позицію.
2. Узагальнена автоматна мережа (УАМ) — це мережа Петрі, у якій кожен перехід має не більше однієї вхідної позиції і не більше однієї вихідної позиції.
3. Маркований граф (МГ) — це мережа Петрі, у якій кожна позиція має рівно один вхідний перехід і рівно один вихідний перехід.
4. Узагальнений маркований граф (УМГ) — це мережа Петрі, у якій кожна позиція має не більше одного вхідного переходу і не більше одного вихідного переходу.
5. Проста мережа (ПМ) — узагальнений маркований граф, кожен підграф якого не є циклом.
6. Мережа вільного вибору (МВВ)— мережа Петрі, у якій переходи можуть спрацьовувати незалежно один від одного, доки потрібні маркери доступні у місцях введення.
7. Розширена мережа вільного вибору (РМВВ) — це мережа вільного вибору, яка має додаткові модифікації: розширені переходи, зважені дуги, початкове розподілення маркерів.
8. Асиметричний вибір (АВ) — це мережа Петрі у якій переходи мають кілька місць введення, але лише одне місце виведення, переходи не мають дуг, що сполучаються між собою, кожна дуга з’єднує лише два місця або місце та перехід.

Аналогічно як й ієрархію Хомського, класифікацію мереж Петрі можна зобразити за допомогою діаграми Вена, що зображені на рисунку 1.6.

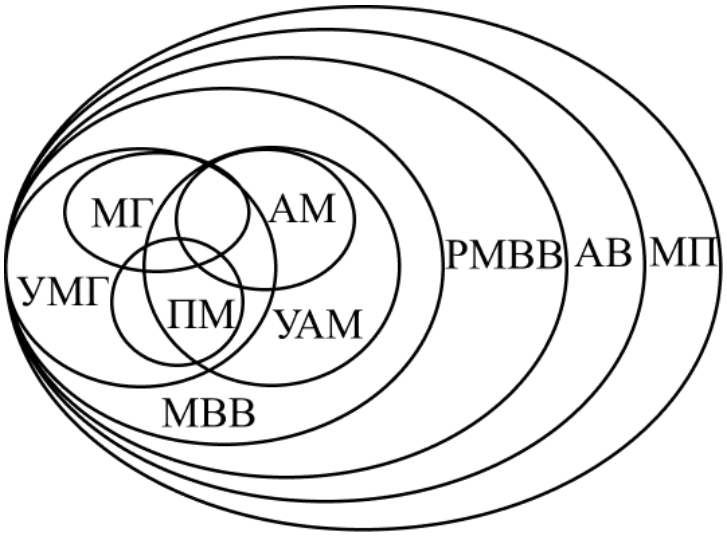


Рисунок 1.6 — Класифікація мереж Петрі у вигляді діаграми Вена [3]

1.3 Мови мереж Петрі

Одним із фундаментальних підходів у галузі вивчення формальних мов є розгляд мережі Петрі як генератора формальних мов [3].

Мова мережі Петрі визначається наступним чином: нехай $N =< P, T, F, \mu_0 >$ є мережею Петрі, де:

- P — множина місць;
- T — множина переходів;
- $F \subseteq (P \times T) \cup (T \times P)$ — множина дуг;
- μ_0 — початкове маркування.

Тоді мова мережі Петрі $L(N)$ є множиною всіх слів над алфавітом T, які можуть бути отримані шляхом виконання послідовностей переходів, починаючи з початкового маркування μ_0.

Для мережі Петрі, яка є генератором формальної мови, виникає потреба надавати позиціям та переходам деякий зміст. Для вирішення цього питання вводять поняття міченої мережі Петрі. Мічена мережа Петрі — це

розширення звичайних МП, до яких додається функція мічення $l: T \to \Sigma$, яка присвоює кожному переходу із множини T символ з алфавіту Σ або порожній рядок ε. Тобто мічена мережа Петрі — це кортеж $N =< P, T, F, \mu, l >$.

Класифікація мов, які згенеровані міченими мережами Петрі:

- вільна — якщо всі переходи мають різні мітки, і жоден перехід не позначений порожнім рядком;
- λ-вільна — якщо жоден перехід не позначено порожнім рядком;
- довільна — якщо на функцію мічення l не накладено обмежень.

Типи мов мереж Петрі за заключним маркуванням:

1. Тип P — заключними маркуваннями МП є усі маркування, які досяжні з початкового.
2. Тип L — мова включає лише скінченну кількість маркувань, які досяжні з початкового маркування.
3. Тип G — мова включає маркування, кількість фішок в яких є більшим або рівним кількості фішок у заданому маркуванні.
4. Тип T — мова складається з усіх кінцевих маркувань мережі, тобто тих, з яких більше не можна здійснити жодного переходу.

Ш. Тураєв у своїй роботі [3] наводить ілюстрацію, яка показує взаємозв'язок між сім'ями мереж Петрі (рисунок 1.7)

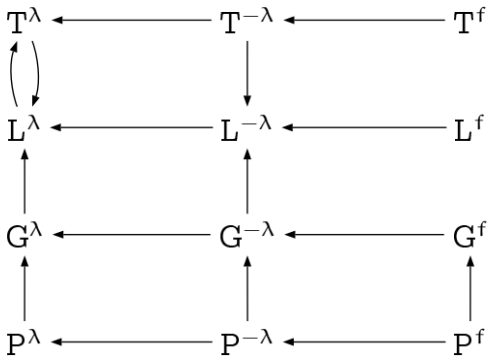


Рисунок 1.7 — Взаємозв'язок між сім'ями мереж Петрі [3]

Висновки до розділу 1

У даному розділі було викладено основну теоретичну базу з тем “Формальні мови та граматики” та “Мережі Петрі”. Розглянуто основні поняття формальних мов і граматик, включаючи їх класифікацію та застосування в різних областях. Також було детально розглянуто мережі Петрі, їх структуру та функціональні можливості.

РОЗДІЛ 2 МАТЕМАТИЧНА ЧАСТИНА

2.1 Граматики керовані мережею Петрі

Керовані формальні граматики — це формальні граматики (зазвичай контекстно-вільні) для яких додатково використовують певний інструмент для обмеження або контролю породження слів. Такими інструментами можуть виступати дуже різні засоби від предикатів до скінченних автоматів. Конкретно у цій роботі розглядаються тільки граматики керовані мережами Петрі та матричні.

Граматика, що керована мережею Петрі — це контекстно-вільна граматика, яка обладнана мережею Петрі. Переходи цієї мережі позначаються продукціями граматики або порожнім словом. Мова, що породжується цією граматикою, складається з усіх скінченних слів, які можуть бути отримані в граматиці [4].

Наведемо приклад формальної граматики, яка керована мережею Петрі за допомогою описаної раніше контекстно-вільної граматики, що породжує мову запису арифметичних виразів з цілими числами:

Отже ми маємо кортеж: $G =< V, T, P, S >$, де $V = \{S,\ A,\ B,\ O,\ X\}$;

$T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0, +, -, \cdot, \div, (,)\}$;

$P = \{S \to A, A \to BOB|\varepsilon,$

$B \to X|XB|A\ , O \to + \mid - \mid \cdot \mid \div,$

$X \to 0|1|2|3|4|5|6|7|8|9|(|)\}$.

Мережа Петрі, що відповідає продукціям граматики зображено на рисунку 2.1.

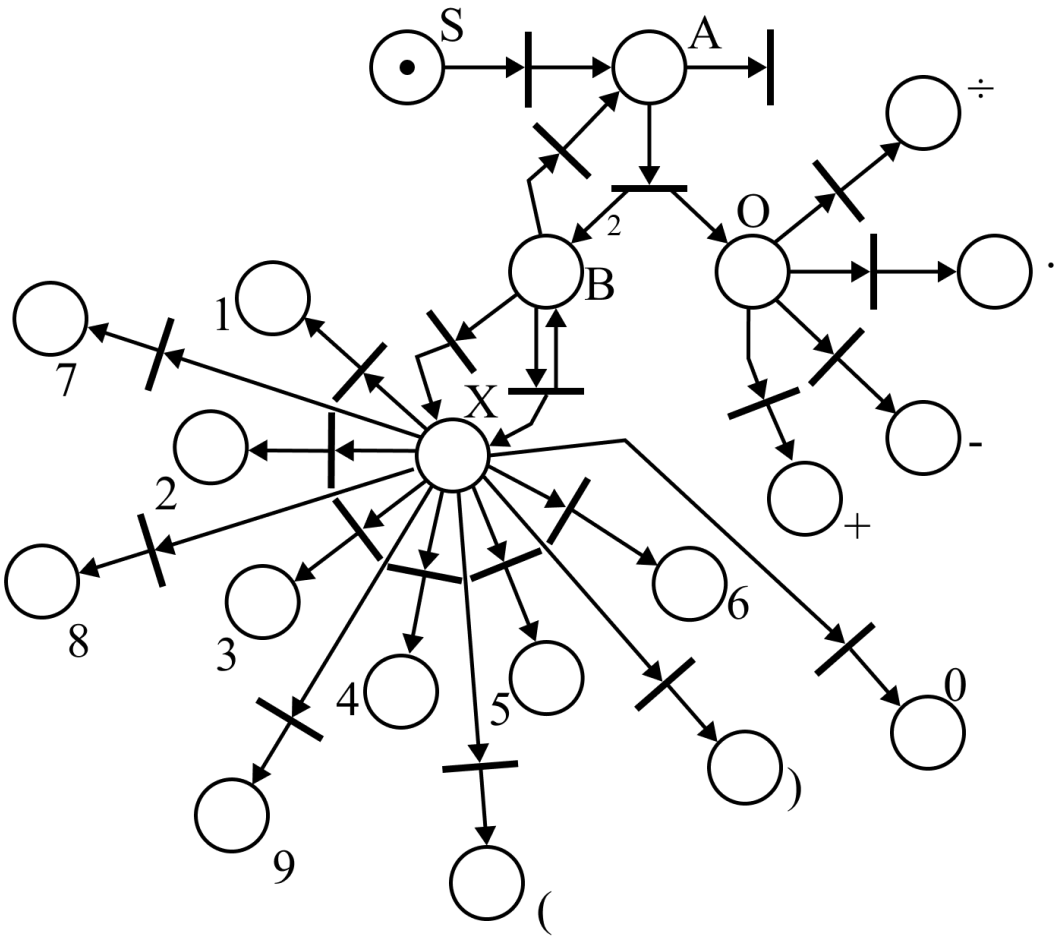


Рисунок 2.1 — Мережа Петрі для КВ граматики G

Математично таку мережу Петрі можна описати так:

$N =< P, T, I, O,$ μ, β, γ $>$, де β та γ — функції мічення, які співставляють до позицій та переходів термінальні/нетермінальні символи та продукції граматики відповідно.

Описана формальна граматика G функціонує й без мережі Петрі, але їх також застосовують для того щоб досліджувати контекстно-вільні граматики. Наприклад можна проводити аналіз на нескінченність породжуваної мови за допомогою дерева покриття мережі Петрі [5].

2.2 Матричні граматики

Матрична формальна граматика, або формальна граматика, що керована матрицями — це це представлений Абрахамом С. [6] тип керованих формальних граматик, у яких продукції записуються не по одній, а записуються послідовно у групи — матриці. Виконання продукцій обов'язково відбувається в тій послідовності, в якій вони записані у матрицях. Матриці належать множині M, яка виступає у кортежі заміною множини P.

Приклад:

$G =< V, T, S, M >$, де $V = \{S, X, Y\}$;

$T = \{a, b, c\}$;

$M = \{m_0 = [S \to XY],\ m_1 = [X \to aXb,\ Y \to cY],\ m_2 = [X \to ab,\ Y \to c]\}$

Граматика породжує мову: $L[G] = \{a^n b^n c^m : n, m \geq 1\}$.

Слово $w = \{a^4 b^4 c^4\}$ породжується внаслідок такої послідовності виконання матриць: $m_0 m_1^2 m_2$, а для створення найкоротшого слова $w = \{abc\}$ достатньо лише $m_0 m_2$.

2.3 Доведення можливості переходу між матричними формальними граматиками та граматиками, які керовані мережами Петрі

Для того щоб написати програмний продукт для переходу між двома типами керованих формальних граматик, варто впевнитися, що такий перехід є допустимим. Професор Шерзод Тураєв у своїй дисертації [3]

приводить наступні леми 5.11 [3] та 5.12 [3] (див. далі формули (2.1) та формула (2.2)).

Варто зауважити, що Ш. Тураєв використовує позначення, які відрізняються від тих, що були описані у попередніх розділах, тому наводимо їх перелік у пристосуванні до наведених вище позначень.

1. $PN^{\lambda}(x, y)$ — граматика, яка керована мережею Петрі з певним обмеженням маркування x та кінцевими маркуваннями y.
2. $x \in \{f, -\lambda, \lambda\}$ визначає тип функції маркування, де:
   - f — вільне маркування, де кожен перехід має унікальну мітку та немає переходів, що позначені порожньою міткою;
   - $-\lambda$ — маркування, яке не може мати порожню мітку;
   - λ — маркування, яке може мати порожню мітку.
3. MAT^{λ} — родина мов, що генерується матричними формальними граматиками з λ-операціями, де λ може бути порожнім рядком (ε).
4. $MAT^{[\lambda]}$ — матричні граматики, де λ-операції дозволені, але обмежені певним чином.
5. $PN^{[\lambda]}$— граматики, де λ-операції дозволені, але обмежені певним чином.
6. позначає матричні граматики з обмеженими λ-операціями.
7. $y \in \{r, t, g\}$ — визначає тип кінцевих маркувань, де:
   - r — досяжні маркування;
   - t — скінченна множина досяжних маркувань;
   - g — скінченна множина маркувань така, що для кожного маркування існує таке маркування, яке є або рівним або більшим за кількістю фішок у кожній позиції мережі.
8. V, Σ — термінальний та нетермінальний алфавіти.
9. S, R — стартовий символ та множина продукцій граматики.
10. N, γ, M' — мережа Петрі, функція переходів, маркування.

11. P, T, F, φ, ι — множини позицій, переходів, дуг, функція мічення та початкове маркування мережі Петрі.

Лема 1:

Для $x \in \{f, -\lambda, \lambda\}$ та $y \in \{r, t, g\}$,

$PN^{\lambda}(x, y) \subseteq MAT^{\lambda}$. [3]

Формула 2.1 — Лема 1

Доведення:

Нехай :$G = < V, \Sigma, S, R, N, \gamma, M' >$ буде, (x, y)-МП керована граматика з $N = < P, T, F, \varphi, \iota >$, де

$x \in \{f, -\lambda, \lambda\}$ та $y \in \{r, t, g\}$

Нехай: $P = \{p_1, p_2, ..., p_n\}$.

Покладемо $V' = V \cup \overline{P} \cup \{S', B\}$, де $\overline{P} = \{\overline{p} \mid p \in P\}$ — це набір нових нетермінальних символів, а S' та B — нові нетермінальні символи.

Нехай для $t \in T$: $\dot{t} = \{p_{i_1}, p_{i_2}, ..., p_{i_k}\}$ та $t\dot{} = \{p_{j_1}, p_{j_2}, ..., p_{j_m}\}$

Ми пов'язуємо наступні послідовності правил з кожним переходом $t \in T$ (рисунок 2.2).

$$\sigma_{i_1} : \underbrace{\overline{p}_{i_1} \to \lambda, \overline{p}_{i_1} \to \lambda, \ldots, \overline{p}_{i_1} \to \lambda}_{\varphi(p_{i_1}, t)} \qquad a)$$

$$\sigma_{i_2} : \underbrace{\overline{p}_{i_2} \to \lambda, \overline{p}_{i_2} \to \lambda, \ldots, \overline{p}_{i_2} \to \lambda}_{\varphi(p_{i_2}, t)} \qquad b)$$

$$\ldots$$

$$\sigma_{i_k} : \underbrace{\overline{p}_{i_k} \to \lambda, \overline{p}_{i_k} \to \lambda, \ldots, \overline{p}_{i_k} \to \lambda}_{\varphi(p_{i_k}, t)} \qquad c)$$

$$\sigma_B : B \to B\overline{p}_{j_1}^{\varphi(t,p_{j_1})}\overline{p}_{j_2}^{\varphi(t,p_{j_2})} \cdots \overline{p}_{j_m}^{\varphi(t,p_{j_m})} \qquad d)$$

Рисунок 2.2 — Вирази a), b), c), d). [3]

і визначаємо матрицю e) (рисунок 2.3)

$$m_r = (\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_k}, \sigma_B, r) \qquad e)$$

Рисунок 2.3 — Вираз e) [3]

де $r = A \to \alpha = \gamma(t) \in R$. Крім того, додаємо вихідну матрицю f)(рисунок 2.4)

$$m_0 = (S' \to SB \cdot \prod_{p \in P} \overline{p}^{|\iota(p)|}) \qquad f)$$

Рисунок 2.4 – Вираз f) [3]

За типами наборів кінцевих маркувань розглянемо три випадки правила стирання:

Випадок: $y = t$ для кожного $\tau \in M'$ (рисунок 2.5),

$$m_{\tau,\lambda} = (B \to \lambda, \underbrace{\overline{p}_1 \to \lambda, \ldots, \overline{p}_1 \to \lambda}_{\tau(p_1)}, \ldots, \underbrace{\overline{p}_n \to \lambda, \ldots, \overline{p}_n \to \lambda}_{\tau(p_n)}). \quad g)$$

Рисунок 2.5 — Вираз g) [3]

Випадок: $y = r$ (рисунок 2.6).

$$m_{p,\lambda} = (\overline{p} \to \lambda) \text{ для кожного } p \in P \text{ та } m_{B,\lambda} = (B \to \lambda) \quad j)$$

Рисунок 2.6 — Вираз j) [3]

Випадок: $y = g$. Тут ми розглядаємо матриці g) разом з матрицями j). Розглянемо матричну граматику: $G' =< V', \Sigma, S', M >$, де M складається усіх матриць e)-f) і матриць g) (випадок y = t), матриць j) (випадок y = r) або матриці g)-j) (випадок y = g).

Нехай: $D: S \Rightarrow^{r_1 \ldots r_n} w \in \Sigma^*$ буде виведенням в G.

Тоді $v = t_1 t_2 \ldots t_s$, де $\gamma(v) = r_1 r_2 \ldots r_n$ є послідовністю входжень переходів.

Будуємо виведення D' в G', яке імітує виведення D.

Виведення D' починається в $S' \Rightarrow SB \cdot \prod_{p \in P} \overline{p}^{-|\iota(p)|}$,застосувавши матрицю f), тоді для кожної пари переходів t в v та відповідного правила $r = \gamma(t)$, ми обираємо матрицю e). Коли термінальний рядок $w \in \Sigma^*$ — породжено, щоб стерти залишені символи $\overline{P}$ та B використовуємо матриці g), j) або g) разом з j) в залежності від $y \in \{r, t, g\}$.

Нехай: $D'\colon S' \Rightarrow^{m_0} SB \cdot \prod_{p\in P} \overline{p}^{|\iota(p)|} \Rightarrow^{m_{i1}\dots m_{in}} w_n = w \in \Sigma^*$ буде виведенням в G'.

Оскільки $V \cap \overline{P} = \varnothing$, ми можемо описати виведення:

$D''\colon S \Rightarrow^{r_1\dots r_n} w_{j_k} = w \in \Sigma^*$,

де r_{j_i} є правилом нестирання матриці $m_{r_{j_i}}$, $1 \le i \le k$ в D' і ми пропускаємо ті кроки в D' в яких використовуються стираючі матриці. Застосування матриці m_r виду е) в D' показує, що там принаймні $\varphi(p_{i_2}, t)$ частин $\overline{p}_{i_1}$, і т. д. та принаймні $\varphi(p_{i_k}, t)$ частин $\overline{p}_{i_k}$ у формі речення, тобто вхідні позиції $p_{i_1}, p_{i_2}, p_{i_k}$ переходу t мають принаймні $\varphi(p_{i_1}, t), \varphi(p_{i_2}, t),\dots, \varphi(p_{i_k}, t)$ фішок відповідно.

Ми можемо побудувати успішне входження послідовності $\iota \rightarrow^{t_{j_1}\dots t_{j_k}} \mu_k$, де $\gamma(t_{j_i}) = r_{j_i}$, $1 \le i \le k$. Отже, D'' — це виведення в G, тоді $L(G') \subseteq L(G)$.

Тепер нехай: $E : S \Rightarrow^{r_1\dots r_n} w_{j_k} = w \in \Sigma^*$ буде виведенням в G. Далі ми також маємо виведення $E'\colon S' \Rightarrow^{m_0} SB \Rightarrow^{m_{j_1}\dots m_{k_k}} w'_{j_k} B$ в G', де w'_{j_k} відрізняється від w_{j_k} тільки літерами $\overline{p}$ з $p \in P$. Ці літери та B можуть бути стерті матрицями g), j) або g) разом з j) в залежності від $y \in \{r, t, g\}$. Отже, $L(G) \subseteq L(G')$.

Лема 2:

$$MAT^{[\lambda]} \subseteq PN^{[\lambda]}(-\lambda, r) \quad [3]$$

Формула 2.2 — Лема 2

Доведення:

Нехай: $G =< V, \Sigma, S, M >$ буде матричною граматикою і:

$M = \{m_1, m_2,..., m_n\}$, де $m_i = (r_{i,1}, r_{i,2},..., r_{i,k(i)})$, $1 \le i \le n$.

Ми можемо вважати, що G не має повторів.

Нехай: $R = \{r_{i,j} | 1 \le i \le n, 1 \le n, 1 \le j \le k(i)\}$.

Покладемо $\overline{\Sigma} = \{\overline{\alpha} \mid \alpha \in \Sigma\}$, де для $\alpha \in \Sigma$, $\overline{\alpha}$ — новий нетермінальний символ.

Визначаємо бієкцію: $\psi : V \cup \Sigma \to V \cup \overline{\Sigma}$ за $\psi(x) = x$, якщо $x \in V$ та $\psi(x) = \overline{x}$, якщо $x \in \Sigma$ і для кожного правила $r = A \to x_1 x_2 ... x_1 \in R$ вводимо нове правило: $\overline{r} = A \to \psi(x_1)\psi(x_2)... \psi(x_1)$.

Нехай: $\overline{M} = \{\overline{m_1}, \overline{m_2},..., \overline{m_n}\}$, де $\overline{m_i}$=$(\overline{r_{i,1}}, \overline{r_{i,2}},..., \overline{r_{i,k(i)}})$, $1 \le i \le n$,

$M_\Sigma = \{(\overline{\alpha} \to \alpha) \mid \alpha \in \Sigma\}$.

Побудуємо матричну граматику $G' =< V \cup \overline{\Sigma}, \Sigma, S, \overline{M} \cup M_\Sigma >$, очевидно, що $L(G) = L(G')$.

Визначаємо $(-\lambda, r)$-МП керовану граматику:

$G'' =< V \cup \overline{\Sigma}, \Sigma, S, R', N, \gamma, M' >$, де

$R' = R \cup \{\overline{\alpha} \to \alpha \mid \alpha \in \Sigma\}$ та $N =< P, T, F, \varphi, \iota >$ — керуюча МП, де множини позицій, переходів і дуг відповідно визначаються:

$P = \{p_0\} \cup \{p_{i,j} \mid 1 \le i \le j \le k(i) - 1\}$,

$T = \{t_\alpha \mid \alpha \in \Sigma\} \cup \{t_{i,j} \mid 1 \le i \le n, 1 \le j \le k(i)\}$,

$F = \{(p_0, t_0), (t_\alpha, p_0) | \alpha \in \Sigma\} \cup \{(p_0, t_{i,1}), (t_{i,k(i)}, p_0) | 1 \le i \le n\}$

$\cup \{(t_{i,j}, p_{i,j}) | 1 \le i \le n, 1 \le j \le k(i) - 1\}$

$\cup \{(p_{i,k(i)-1}, t_{i,k(i)}) | 1 \le i \le n\}$.

Вагова функція визначається за $\varphi(x, y) = 1$, для всіх $(x, y) \in F$, а початкове маркування визначається: $\iota(p_0) = 1$, та $\iota(p) = 0$ для всіх

$P' - \{p_0\}$. Функція мічення γ: $T \to R'$ визначається за $\gamma(t_\alpha) = \overline{\alpha} \to \alpha$ для всіх $\alpha \in \Sigma$ та $\gamma(t_{i,j}) = r_{i,j}$, $1 \le i \le n, 1 \le j \le k(i)$

Нехай: $S = w_0 \Rightarrow^{m_{i_1}} w_1 \Rightarrow^{m_{i_2}} \ldots \Rightarrow^{m_{i_\iota}} w_\iota = w \in \Sigma^*$ буде виведенням в G', де m_{i_j}, $1 \le j \le \iota$ — елементи $\overline{M}$ або M_Σ та $w_{j-1} \Rightarrow^{m_{i_j}} w_j$: $w_{j-1} \Rightarrow^{\overline{r_{i,1}}, \overline{r_{i,2}}, \ldots, \overline{r_{i,k(i)}}} w_j$ або $w_{j-1} \Rightarrow^{\overline{\alpha} \to \alpha} w_j$ для деяких $\alpha \in \Sigma$

Тоді за визначенням функції γ, $\mu_{j-1} \to^{v_j} \mu_j$, де $v_j = t_{i,1} t_{i,2} \ldots t_{i,k(i_j)}$ або $v_j = t_\alpha$ та $\mu_j = \iota$ для всіх $1 \le j \le l$. Отже, можна побудувати послідовність: $\iota \to^{v_1 \ldots v_\iota} \iota$ успішних запусків переходів мережі N, тому $S \Rightarrow^{\pi_1 \ldots \pi_l} w_\iota \in \Sigma^*$ — це виведення в G'' для всіх $1 \le j \le l$, $\pi_j = \overline{r_{i,1}}, \overline{r_{i,2}}, \ldots, \overline{r_{i,k(i)}}$ або $\pi_j = \overline{\alpha} \to \alpha$ для деяких $\alpha \in \Sigma$.

Нехай: $D : S \Rightarrow^{r_1 \ldots r_l} w \in \Sigma^*$ буде виведенням в G'', де $v = t_1 t_2 \ldots t_l$, $\gamma(t_i) = r_i$, $1 \le j \le l$ —- послідовність успішних запусків переходів мережі N.

Якщо $t_{i,1}$ $1 \le i \le l$ запускається в v тоді наступні кроки: $t_{j,1}$, $t_{j,2}$, … $t_{i,k(i)}$ можуть бути запущені лише в такому порядку.

Інші $t_{j,1}$, $1 \le i \le l$ або t_α для деяких $\alpha \in \Sigma$ можна запустити після того як відпрацює $t_{i,k(i)}$.

За визначенням γ відповідні правила мічення $\overline{r_{i,1}}, \overline{r_{i,2}}, \ldots, \overline{r_{i,k(i)}}$ — елементи однієї матриці $\overline{m_i} \in \overline{M}$, тобто $\overline{m_i} = (\overline{r_{i,1}}, \overline{r_{i,2}}, \ldots, \overline{r_{i,k(i)}})$.

Таким чином застосування матриць G' може модулюватися відповідними переходами мережі Петрі N, звідси робимо висновок, що D є виведенням в G'. Можна підсумувати зміст цих лем та їх доведення сформулювавши наступну теорему (формула 2.3).

Для $x \in \{f, -\lambda, \lambda\}\, y \in \{r, t, g\}$,

$$MAT \subseteq PN(x, r) = PN(x, g) \subseteq PN(x, t) \subseteq PN^{\lambda}(x, y) = MAT^{\lambda} \quad [3]$$

Формула 2.3 — Теорема 1

Доведення конструктивні, отже є алгоритми як побудувати матричну граматику по граматиці, яка керована мережею Петрі та граматику, яка керована мережею Петрі, по матричній граматиці.

2.4 Приклад побудови граматики, яка керована мережею Петрі по матричній граматиці

Щоб продемонструвати алгоритм переходу від граматики, що керована мережею Петрі до матричної, розглянемо граматику, що була наведена у якості прикладу в підрозділі 1.4:

$G =< V, T, S, M >$, де $V = \{S, X, Y\}$;

$T = \{a, b, c\}$;

$M = \{m_0 = [S \to XY],\ m_1 = [X \to aXb,\ Y \to cY],\ m_2 = [X \to ab,\ Y \to c]\}$

Вже на даному етапі можливо побудувати граф мережі Петрі з фішкою у позиції S так як було показано у підрозділі 1.3, але так як у подальшому буде створюватися програмна реалізація цього алгоритму, краще покроково розписати порядок дій переведення продукцій із матриць керованої граматики у рядки матриць входів та виходів для мережі Петрі.

Крок 1. Оцінити кількість позицій та переходів. Матрична граматика G має 3 нетермінальні символи, 3 термінальні символи та 5 продукцій у матрицях. Отже мережа Петрі буде мати 6 позицій — сума кількості термінальних та нетермінальних символів, та 5 переходів — кількість

продукцій. Тоді матриці входів та виходів будуть мати 6 стовпчиків та 5 рядків.

Крок 2. Заповнення матриць. Рядки та стовпчики відповідають термінальним/нетермінальним символам та продукціям у тому порядку, у якому вони записані у граматиці. Матриця входів заповнюється за таким принципом: якщо ліва частина продукції, які відповідає рядок має в собі символ, якому відповідає стовпчик — ця клітинка матриці заповнюється, числом, яке відповідає кількості цього символу (тобто кількості дуг у мережі Петрі). Матриця виходів заповнюється аналогічно, тільки по правій частині продукції.

Отже матриці входів та виходів D^- та D^+ матимуть такий вигляд:

$$D^- = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \qquad D^+ = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Для наочності можна побудувати граф мережі Петрі (рисунок 2.7)

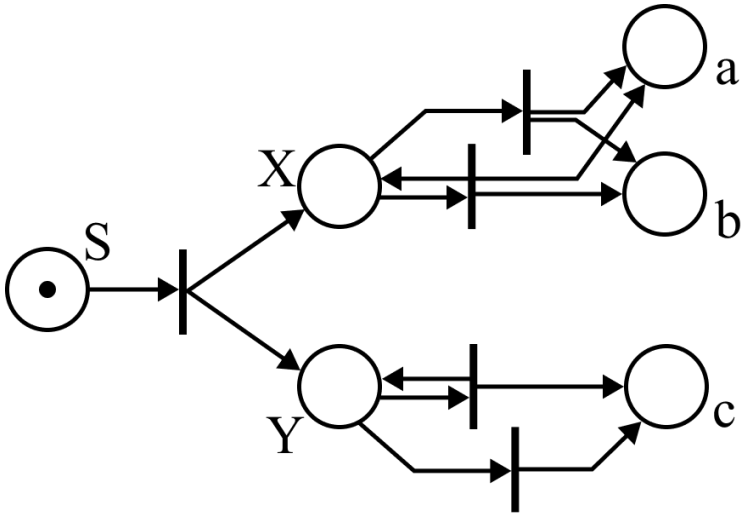


Рисунок 2.7 — Граф мережі Петрі, що керує граматикою

Також можна побудувати загальну контекстно-вільну граматику:

$G =< V, T, S, P >$, де $V = \{S, X, Y\}; T = \{a, b, c\};$

$P = \{ S \to XY$

$X \to aXb \mid ab$

$Y \to cY \mid c\}$

Отже, вдалося здійснити перехід від матричної формальної граматики до граматики, яка керована мережею Петрі.

2.5 Приклад побудови матричної граматики по граматиці, яка керована мережею Петрі

Нехай є така мережі Петрі $N =< P, T, I, O, \mu, \beta, \gamma >$,
яка керує формальною граматикою. Її граф (рисунок 2.8):

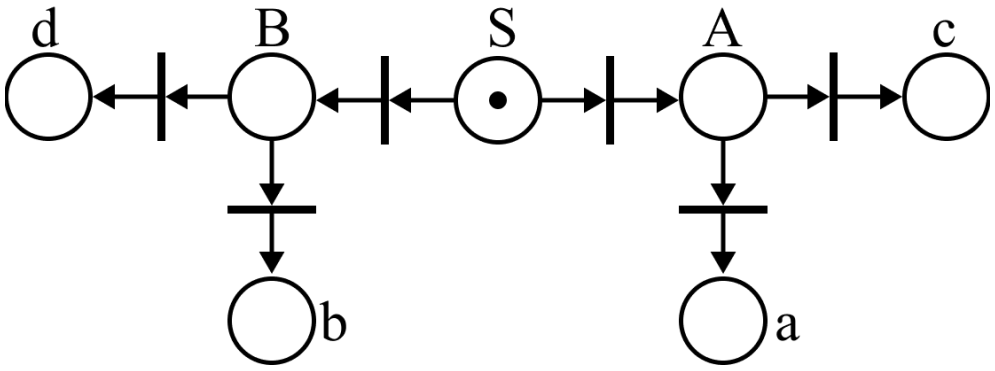


Рисунок 2.8 — Граф мережі Петрі, що керує граматикою

Матриці входів та виходів будуть:

$$D^- = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \quad D^+ = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Позиції (стовпчики) розташовані у такому порядку: стартовий нетермінал S, інші нетермінали алфавітному порядку, термінальні символи в алфавітному порядку.

Щоб здійснити перехід від даної формальної граматики до матричної треба:

Крок 1. За матрицями входів та виходів зіставити переходи мережі Петрі з продукціями граматики. Кожен i-й рядок — це відповідний перехід t_i. Заповнена клітина у матриці входів — ліва частина продукції (число відповідає кількості символів), заповнена клітина у матриці виходів — права частина продукції.

Отже маємо:

$t_1\colon S \to A,\ t_2\colon S \to B,\ t_3\colon A \to a,\ t_4\colon A \to c,\ t_5\colon B \to b,\ t_6\colon B \to d$

Крок 2. Розподілити продукції по матрицях. У матриці не може бути продукцій з однаковою лівою частиною. Нехай будуть наступні матриці:

$M = \{m_1 = [S \to A,\ A \to a]$

$m_2 = [S \to A, A \to c], m_3 = [S \to B, B \to b], m_4 = [S \to B, B \to d\ ]\}$

Отже, вдалося здійснити перехід від граматики, яка керована мережею Петрі до матричної формальної граматики.

Висновки до розділу 2

У розділі 2 було представлено доведення можливості переходу між різними типами керованих формальних граматик, а саме матричних та керованих мережами Петрі. Також було запропоновано алгоритми для реалізації такого переходу, які були проілюстровані на прикладі.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Обґрунтування вибору технологій для програмної реалізації

Для програмної реалізації алгоритмів переходу між матричними формальними граматиками та граматиками, що керовані мережами Петрі було обрано мову програмування Python.

Python – це потужна та високорівнева мова програмування, яка набула широкого використання завдяки численним перевагам, які вона пропонує. Її синтаксис дуже простий і зрозумілий, внаслідок чого код написаний цією мовою зазвичай коротший, що полегшує спільну роботу над проектами та полегшує розуміння коду іншим програмістам, а також значно підвищує швидкість розробки. Python підходить для різноманітних завдань, включаючи веб-розробку, наукові обчислення, штучний інтелект, обробку даних, робототехніку та багато іншого. Це робить його пріоритетним вибором для багатьох програмістів та науковців у різних галузях. У цієї мови є велика та активна спільнота, що розробляє різноманітні бібліотеки, фреймворки та інструменти. Це дозволяє використовувати готові рішення для багатьох завдань та швидко розвивати свої проекти. Код, написаний на Python є кросплатформним, тобто він може легко переноситися між різними операційними системами без змін. Це робить його ідеальним вибором для розробки програм, які мають мати можливість бути запущеними на різних платформах. У Python є величезна кількість бібліотек та фреймворків для різних завдань. Наприклад, бібліотека NumPy дозволяє проводити ефективні обчислення над масивами даних, а фреймворк Django пропонує потужні засоби для швидкої розробки веб-додатків. Для наукових обчислень Python має багато високопродуктивних бібліотек, таких як NumPy, SciPy, Pandas та Matplotlib. Python дозволяє використовувати код, написаний на мовах програмування C та C++, що значно розширює можливості мови,

наприклад, бібліотека NumPy, яка використовується для ефективних обчислень над масивами даних у Python, фактично має внутрішню реалізацію на мові C.

Але також Python має низку недоліків:

- швидкодія;
- гранулярність управління пам'яттю;
- обмеження у мобільній розробці;
- гранична підтримка для паралельного програмування;
- відсутність низькорівневої оптимізації.

У якості середовища програмування було обрано Visual Studio Code — редактор коду, створений компанією Microsoft. Це безкоштовний редактор з відкритим вихідним кодом, який нині є одним із найпопулярніших інструментів для розробки на Python. Однією з його переваг є легкість налаштування. Завдяки підтримці розширень, користувачі можуть легко додавати нові функціональні можливості, налаштовувати середовище під свої потреби та встановлювати розширення для підтримки різних мов програмування, включаючи Python. Розширення для Python, що доступне у цьому редакторі, забезпечує підсвічування синтаксису, автодоповнення, перевірку коду на помилки та можливості відлагодження. Також VS Code підтримує інтерактивні нотатники Jupyter, що особливо корисно для аналітики даних та машинного навчання. Вбудований термінал у VS Code дозволяє швидко запускати скрипти, встановлювати пакети та виконувати інші команди, не виходячи з редактора. Дуже важлива перевага VS Code — інтеграція з Git. Вона дозволяє легко керувати версіями коду, робити коміти, відслідковувати зміни та працювати з віддаленими репозиторіями. Завдяки цьому розробники можуть ефективніше співпрацювати у команді та слідкувати за історією змін у проекті. VS Code також підтримує роботу з віртуальними середовищами Python, такими як venv чи conda. Це допомагає утримувати проекти ізольованими, що важливо для управління залежностями та запобігання конфліктам між різними версіями бібліотек.

3.2 Опис реалізованих функцій

Засобами мови програмування Python та середовища розробки Google Colab було запрограмовано алгоритм, що представлений у підрозділі 2.2. Код програми представлений у додатку А “Лістинг програми”.

Програма складається із трьох класів MatrixToPetri, PetriToMatrix, Application та їх методів.

Клас MatrixToPetri — відповідає за перехід від матричної формальної граматики до граматики, яка керується мережею Петрі.

Методи:

1. __init__(self, non_terminals, terminals, productions) — конструктор класу. Ініціалізує список нетермінальних символів, термінальних символів та продукцій та створює порожні матриці входів та виходів.
2. fill_matrices(self) — заповнює матриці входів та виходів на основі заданих продукцій.
3. get_matrices(self) — повертає заповнені матриці входів та виходів.

Клас PetriToMatrix — відповідає за конвертацію граматики, керованої мережею Петрі у матричну формальну граматику.

Методи:

1. __init__(self, non_terminals, terminals, input_matrix, output_matrix) — конструктор класу.
2. get_productions(self) — перетворює матриці входів та виходів у список продукцій граматики.

Клас Application — клас, що відповідає за графічний інтерфейс користувача програми для виконання переходу між матричними формальними керованими граматиками та граматиками, які керуються мережами Петрі.

Методи:

1. __init__(self) — конструктор класу. Ініціалізує головне вікно програми, задає його назву та розмір, викликає створення початкових віджетів.
2. create_widgets(self) — створює початкові віджети для вибору операції.
3. next_step(self) — визначає, яку операцію вибрав користувач, і викликає відповідний метод для введення даних.
4. clear_window(self) — очищає поточні віджети з вікна.
5. input_matrix_to_petri(self) — створює віджети для введення даних для переходу від матричної граматики до граматики, керованої мережею Петрі.
6. convert_matrix_to_petri(self) — збирає введені користувачем дані, конвертує їх у матриці входів та виходів (використовуючи клас MatrixToPetri), і відображає результати.
7. plot_petri_net(self) — викликає метод для побудови графу мережі Петрі на основі матриць входів та виходів.
8. plot_petri_graph(self, input_matrix, output_matrix) — створює граф мережі Петрі на основі матриць входів та виходів та відображає його.
9. input_petri_to_matrix(self) — створює віджети для введення даних для конвертації граматики, керованої мережею Петрі, у матричну граматику.
10. convert_petri_to_matrix(self) — збирає введені користувачем дані, конвертує їх у список продукцій (використовуючи клас PetriToMatrix) та відображає результати.
11. back_to_start(self) — повертає користувача на початковий екран вибору операції.

3.3 Результати програмної реалізації

Перше, що бачить користувач після запуску програми — це вікно вибору операції, яку він бажає провести (рисунок 3.1).

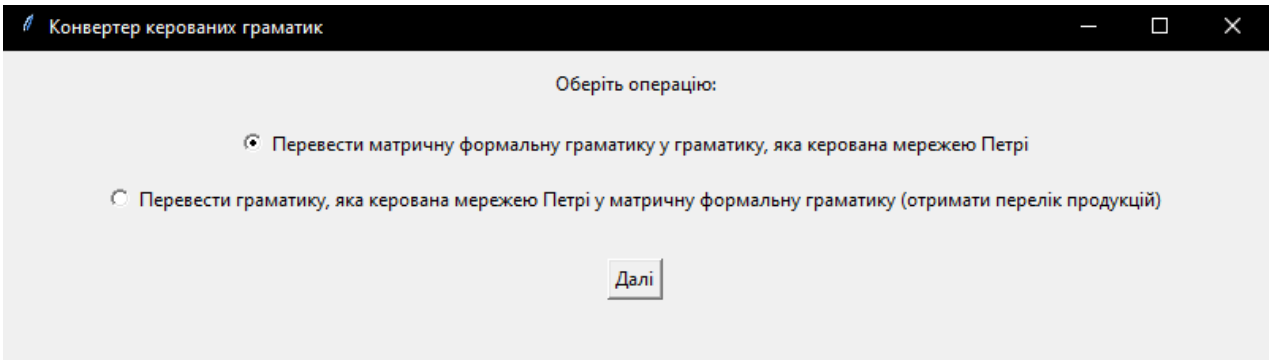


Рисунок 3.1 — Вікно вибору операції

Після вибору операції користувач отримує можливість ввести вхідні дані, які потрібні для відпрацювання алгоритму (рисунок 3.2 та рисунок 3.3). Якщо помилково була обрана операція протилежна до потрібної, є можливість повернутися на початковий екран, натиснувши “Назад”.

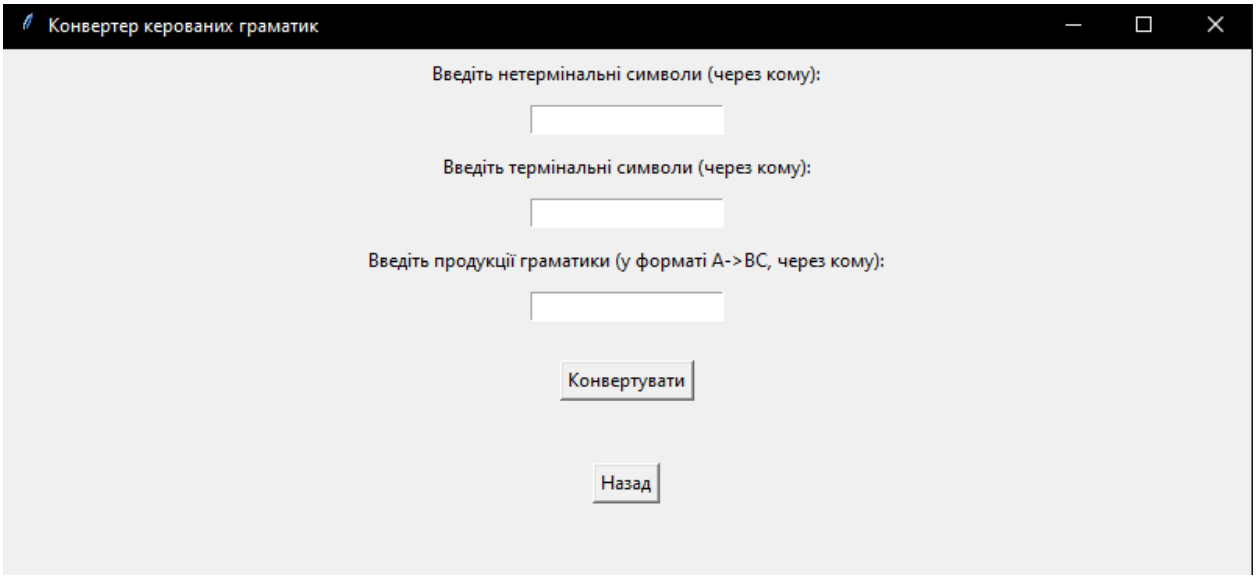


Рисунок 3.2 — Екран введення даних для конвертування матричної граматики у керовану мережею Петрі

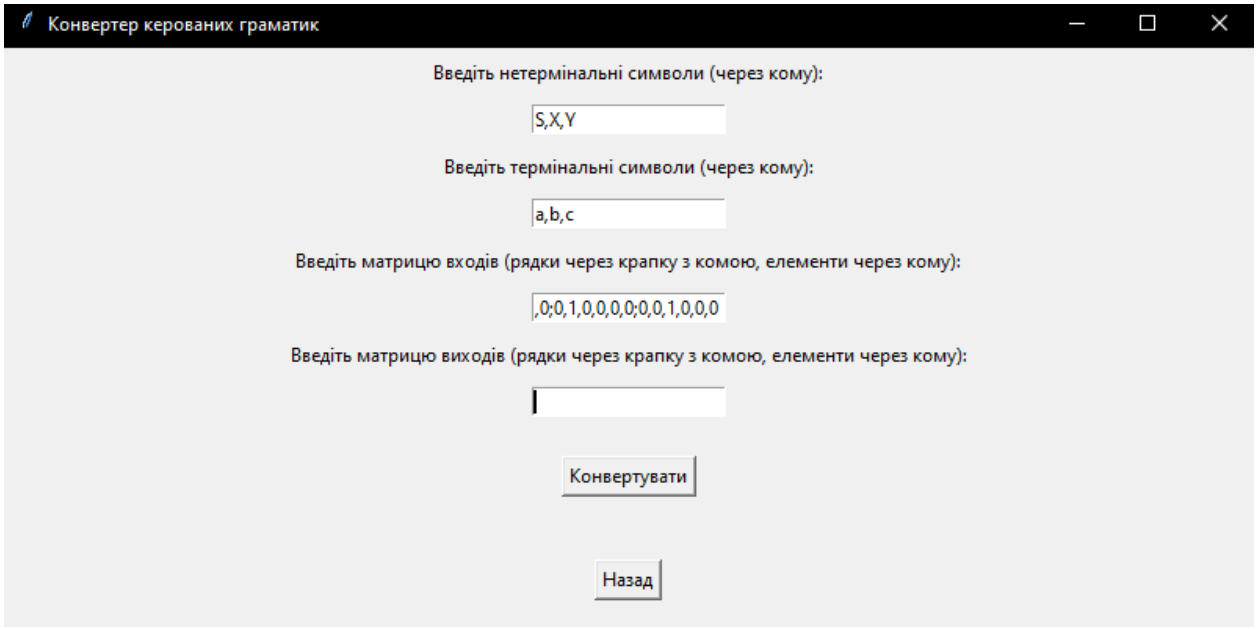


Рисунок 3.3 — Екран введення даних для конвертування граматики, яка керується мережею Петрі у матричну формальну граматику

Далі, якщо дані введено коректно, користувач отримує перелік продукцій граматики (рисунок 3.4) або матриці входів та виходів мережі Петрі (рисунок 3.5). А також у другому випадку є можливість накреслити граф мережі Петрі (рисунок 3.6).

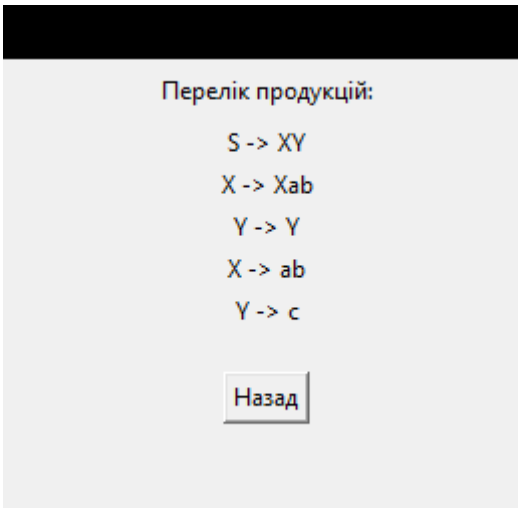


Рисунок 3.4 — Результат роботи алгоритму переходу від граматики, яка керується мережею Петрі до матричної граматики (перелік продукцій)

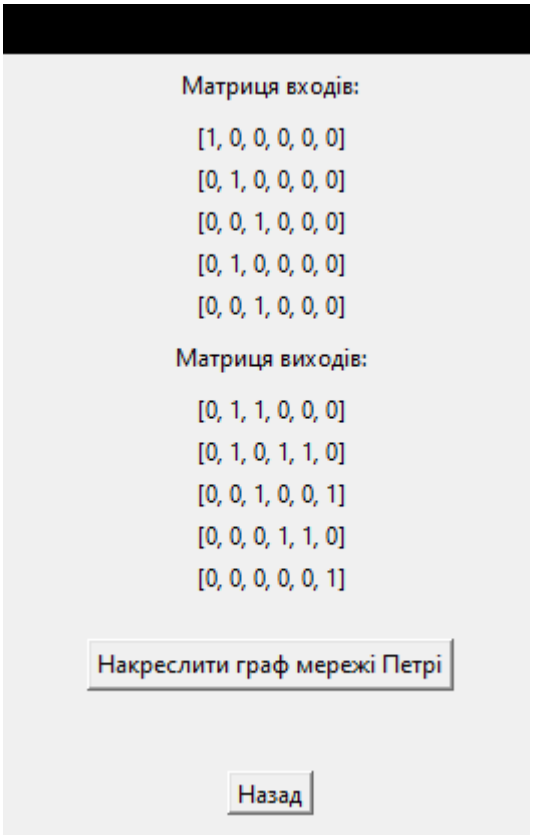


Рисунок 3.5 — Результат роботи алгоритму переходу від матричної граматики до керованої мережею Петрі (матриці входів та виходів)

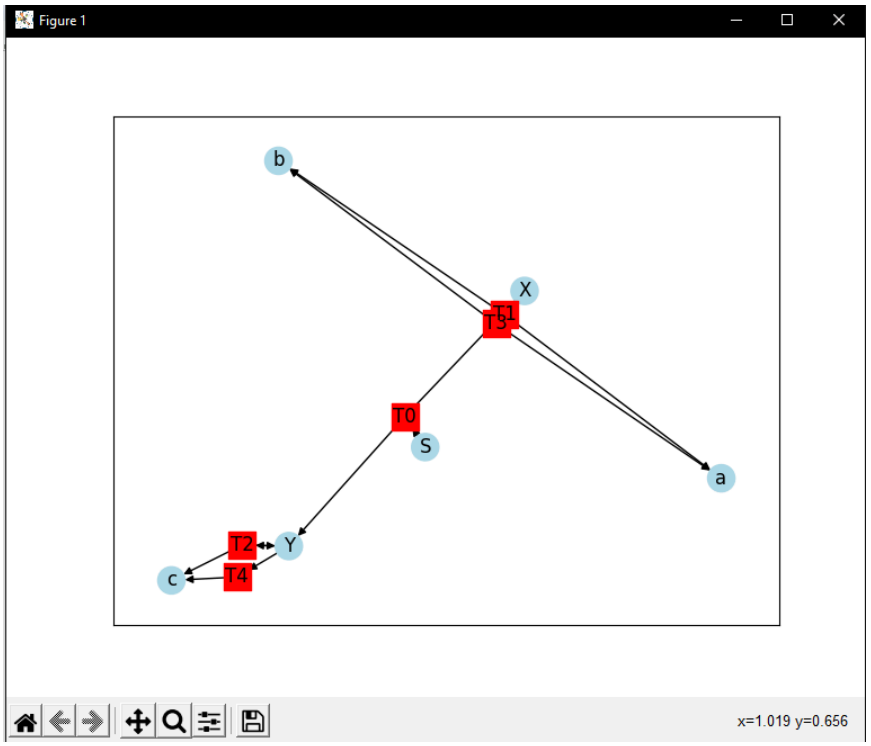


Рисунок 3.6 — граф мережі Петрі, що накреслений програмою

Візьмемо матричну граматику з прикладу виконання алгоритму у підрозділі 2.2. На рисунку 3.7 можна помітити, що результат входів та виходів співпадають з тими, які були отриманими вручну (рисунок 3.8).

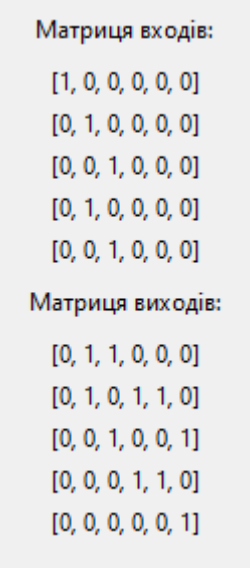


Рисунок 3.7 — Матриці входів та виходів, що отримані програмою

$$D^- = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \qquad D^+ = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Рисунок 3.8 — Матриці входів та виходів, що отримані вручну

А також, граф мережі Петрі на рисунку 3.6 є еквівалентним графу на рисунку 2.6.

Висновки до розділу 3

У розділі 3 засобами мови програмування Python було написано програму з графічним інтерфейсом, яка виконує перехід від матричної формальної граматики до граматики, яка керована мережею Петрі.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В даному розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні демографічного стану.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється, впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп’ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір самої програми.

F_2 – якісний аналіз даних.

F_3 – графічні показники.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Eviews.

б) R.

Функція F_2.

а) Застосування вбудованих функцій.

б) Створення своїх обчислень значень.

Функція F_3:

а) Використання шаблонних графіків.

б) Створення своїх.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

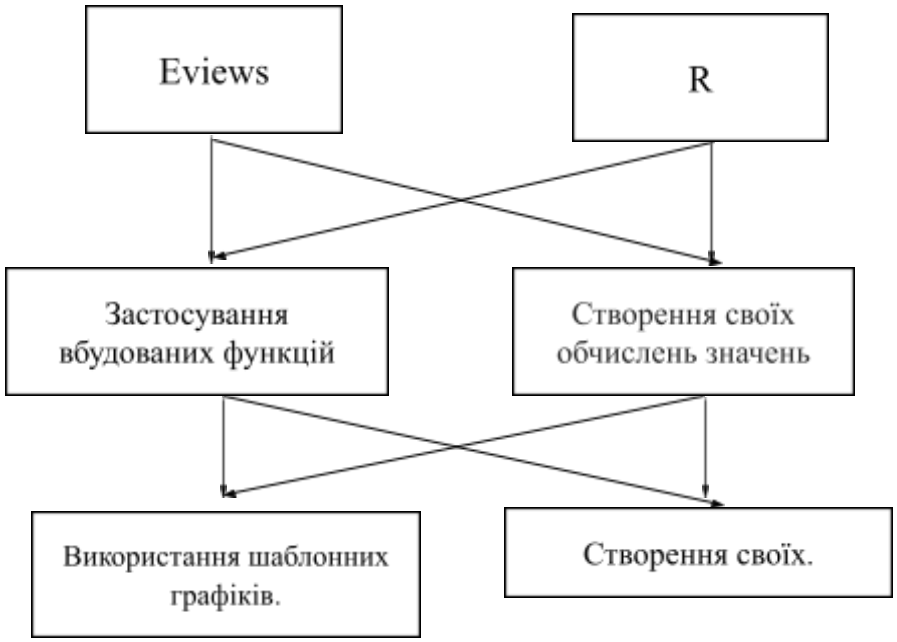


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Загальнодоступна програма, доступність багатьох бібліотек	Необхідність повної реалізації алгоритму
	Б	Доступна в реалізації програма для різних обчислень	На написання коду необхідно мати базові навички та вміння
F_2	*А*	Доступність та легкість при написанні	Іноді не відповідає задачі яку треба розв’язати
	Б	Ідеально описують усі необхідні характеристики	Достатньо затратно реалізовувати свої алгоритми для подальшої реалізації
F_3	*А*	Загальноприйнята реалізація	Іноді не відповідає очікуваним значенням
	Б	При виконанні власних досліджень краще може передавати висновки	Необхідно достатньо багато часу для написання програми для побудови та знаходження всього необхідного в задачі.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу даємо загальнодоступності. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2:

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Функція F_3:

Реалізація першого варіанту є сприйнятливою для програми. Це варіант А.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2б – F_3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для обчислень та збереження даних;
- *X3* – час навчання даних;
- *X4* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	оп/мс	60	80	110
Об’єм пам’яті	X2	Мб	60	50	30
Час попередньої обробки даних	X3	мс	80	70	60
Потенційний об’єм програмного коду	X4	кількість рядків коду	35	25	20

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

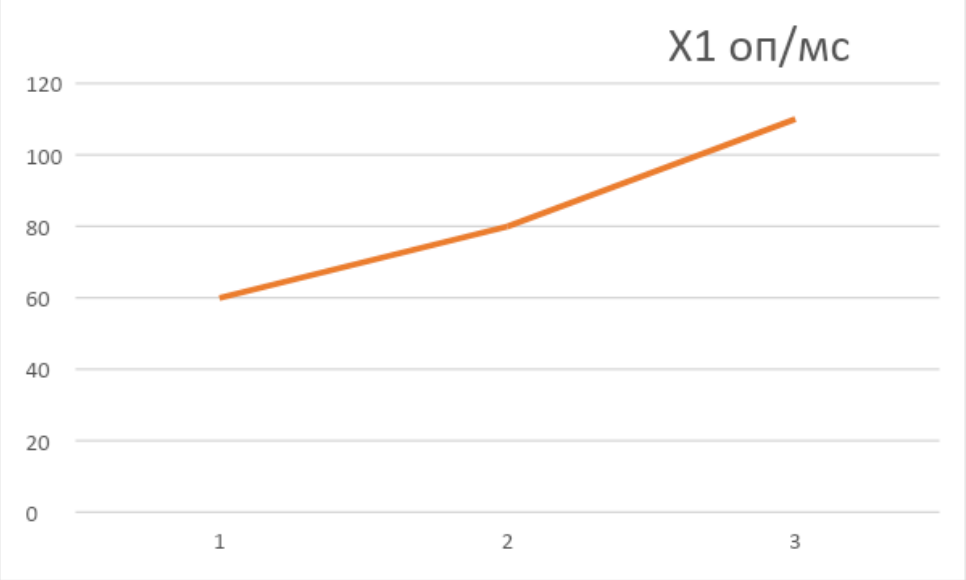


Рисунок 4.2 – X1, швидкодія мови програмування

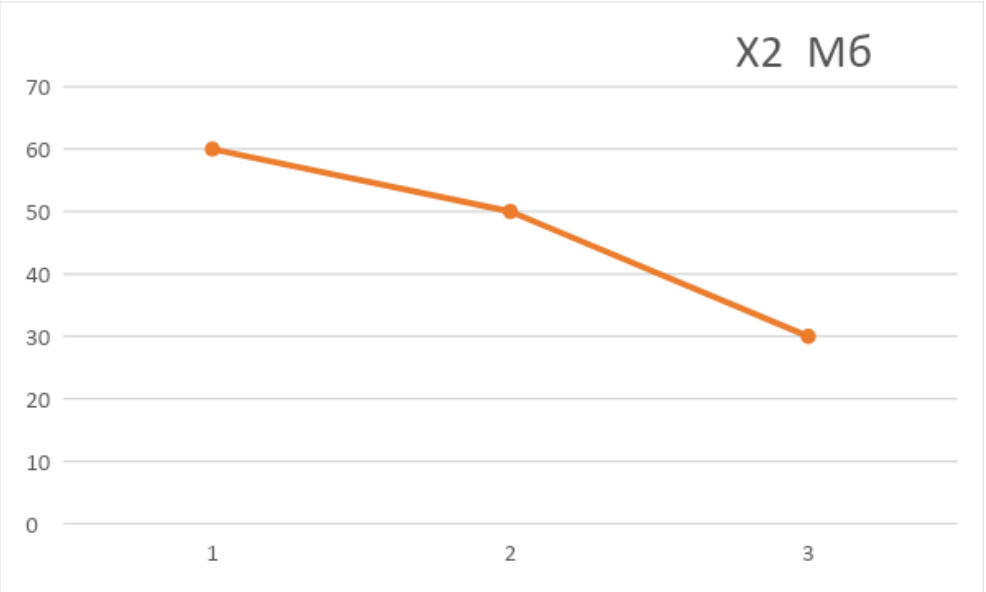


Рисунок 4.3 – X2, об’єм пам’яті

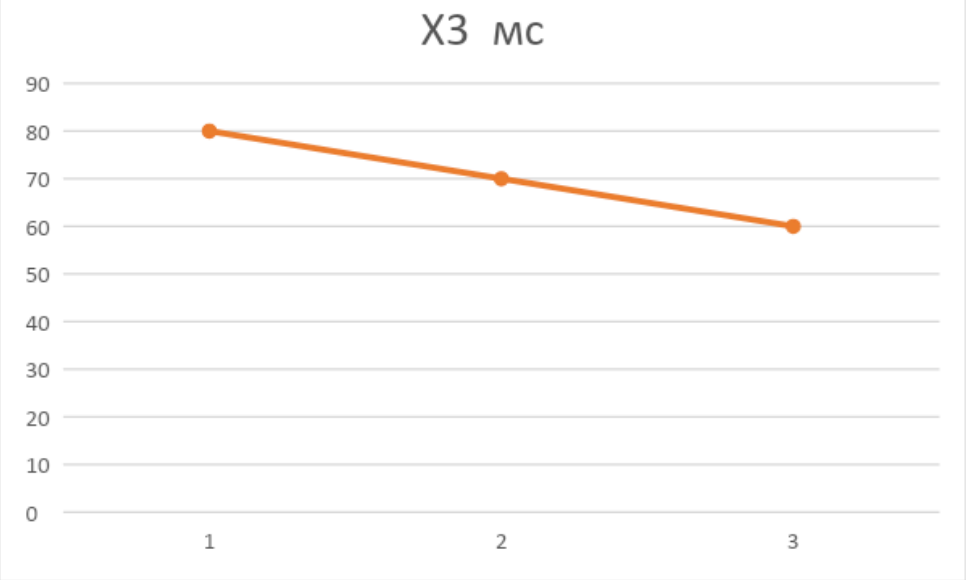


Рисунок 4.4 – X3, час попередньої обробки даних

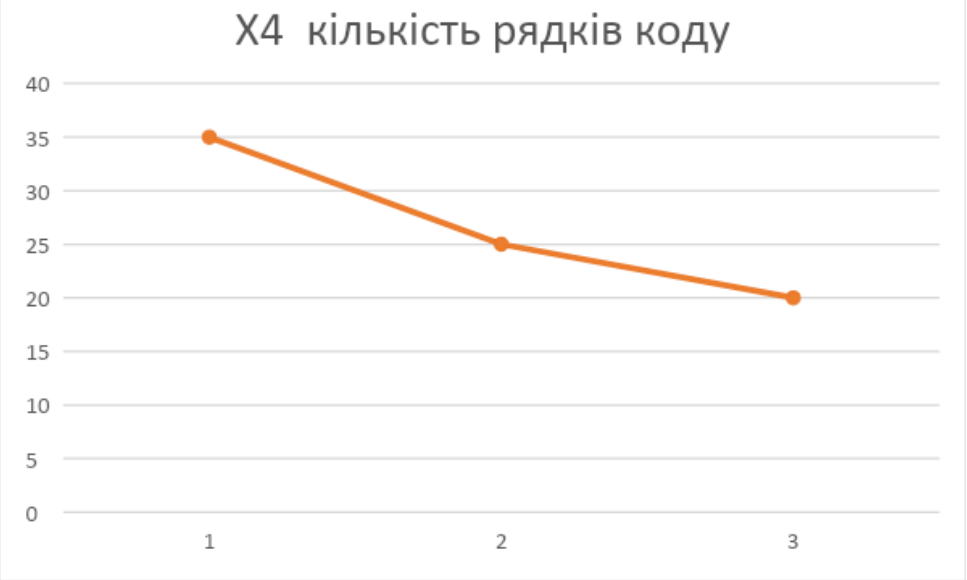


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;

- визначення оцінки попарного пріоритету параметрів;

- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Параметр	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програм.	Оп/мс	1	2	2	1	1	1	2	10	-7,5	56,25
X2	Об'єм пам'яті	Mb	3	4	3	3	4	3	4	24	6,5	42,25
X3	Час попередньої обробки даних	мс	2	1	1	2	2	2	1	11	-6,5	42,25
X4	Потенційний об'єм коду	Кільк. рядків коду	4	3	4	4	3	4	3	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	197

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \qquad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5 \tag{4.2}$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \tag{4.3}$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 197. \tag{4.4}$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2\left(n^3 - n\right)} = \frac{12 \cdot 197}{7^2\left(4^3 - 4\right)} = 0,754 > W_k = 0,67. \tag{4.5}$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0,5
X1 і X3	<	>	>	<	<	<	>	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	<	>	<	<	>	<	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \{1.5 \text{ при } X_i > X_j \; 1,0 \text{ при } X_i = X_j \; 0,5 \text{ при } X_i < X_j . \qquad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A=\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{ві}} = \frac{b_i}{\sum_{i=1}^{n} b_i} \qquad (4.7)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \qquad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{ві}} = \frac{b_i^{'}}{\sum_{i=1}^{n} b_i^{'}}, \quad (4.9)$$

$$b_i^{'} = \sum_{i=1}^{N} a_{ij} b_j \quad (4.10)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{ві}}$	b_i^1	$K_{\text{ві}}^1$	b_i^2	$K_{\text{ві}}^2$
X1	1	0,5	0,5	0,5	2,5	0,16	9,25	0,16	34,125	0,16
X2	1,5	1	1,5	0,5	4,5	0,28	16,25	0,28	59,125	0,28
X3	1,5	0,5	1	0,5	3,5	0,22	12,25	0,21	41,875	0,2
X4	1,5	1,5	1,5	1	5,5	0,34	21,25	0,35	77,875	0,36
Всього:					16	1	59	1	213	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2* (Об’єм пам’яті), *X3* (час попередньої обробки даних) та *X4* (потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X1* (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j}, \quad (4.11)$$

де n – кількість параметрів;

$K_{\text{в}i}$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	100	25	0,16	6
F2	А	X2	87	29	0,28	11,12
	Б	X3	27	19	0,2	4,6
F	А	X4	25	23	0,36	9,28

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}\left[F_{1k}\right] + K_{\text{ТУ}}\left[F_{2k}\right] + \ldots + K_{\text{ТУ}}\left[F_{zk}\right], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 6 + 11{,}12 + 9{,}28 = 26{,}4 ;$$

$$K_{K2} = 6 + 4{,}6 + 9{,}28 = 19{,}88 .$$

Як видно з розрахунків, кращим є 2 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$Т_О = Т_Р \cdot К_П \cdot К_{СК} \cdot К_М \cdot К_{СТ} \cdot К_{СТ.М} \qquad (4.13)$$

де $Т_Р$ – трудомісткість розробки ПП;

$К_П$ – поправочний коефіцієнт;

$К_{СК}$ – коефіцієнт на складність вхідної інформації;

$К_М$ – коефіцієнт рівня мови програмування;

$К_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ = 37 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1,8. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0,9. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 37 \cdot 1,8 \cdot 0,9 = 59,94 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =29 людино-днів, $К_П$ =0,9, $К_{СК}$ = 1, $К_{СТ}$ =0.8:

$$Т_2 = 29 \cdot 0,9 \cdot 0,8 = 20,88 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (59,94 + 20,88 + 4,8 + 20,88) \cdot 8 = 852 \text{ людино-годин.}$$

$$Т_{II} = (59,94 + 20,88 + 6,91 + 20,88) \cdot 8 = 868,88 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 30000 грн., один аналітик в області даних з окладом 40000. Визначимо середню зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} грн., \quad (4.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$СЧ = \frac{30000+30000+40000}{3 \cdot 21 \cdot 8} = 198,41 \text{ грн.} \quad (4.15)$$

Тоді, обчислимо заробітну плату за формулою:

$$СЗП = С_ч \cdot Т_i \cdot КД \quad (4.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 198,41 \cdot 852 \cdot 1,2 = 202854,38$ грн.

II. $С_{ЗП} = 198,41 \cdot 868.88 \cdot 1,2 = 206873,38$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 202854,38 \cdot 0,22 = 44627,96$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0,22 = 206873,38 \cdot 0,22 = 45512,14$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 30000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 30000 \cdot 0,2 = 72000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 72000 \cdot (1 + 0,2) = 86400 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 72000 \cdot 0,22 = 15840 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 50000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1,4 \cdot 0,25 \cdot 50000 = 17500 \text{ грн.},$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1,4 \cdot 10000 \cdot 0,08 = 1120 \text{ грн.},$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0,35 =$$
$$= \ 627,2 \ \text{години},$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 627,2 \ \cdot 0,2 \cdot 0,3 \cdot 5,23 = 196,82 \ \text{грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 = 6700 \quad \text{грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н, \qquad (4.17)$$

$С_{ЕКС}$ = 86400 + 15840 +17500 + 1120 + 196.82 + 6700 = 127756,82 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 127756.82 \ / \ 627,2 = 203,69 \ \text{грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_М = C_{М-Г} \cdot T, \quad (4.18)$$

I. $C_М = 203{,}69 \cdot 852 = 173543{,}88$ грн.

II. $C_М = 203{,}69 \cdot 868{,}88 = 176982{,}17$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0{,}67, \quad (4.19)$$

I. $C_Н = 202854{,}38 \cdot 0{,}67 = 135912{,}44$ грн.

II. $C_Н = 206873{,}38 \cdot 0{,}67 = 138605{,}17$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н, \quad (4.20)$$

I. $C_{ПП} = 202854{,}38 + 15840 + 173543{,}88 + 135912{,}44 = 528150{,}7$ грн.

II. $C_{ПП} = 206873{,}38 + 15840 + 176982{,}17 + 138605{,}17 = 538300{,}72$ грн.

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР j} = К_{Кj}/С_{Фj}, \quad (4.21)$$

$$К_{ТЕР1} = 20,4 / 297435,63 = 6,851 \cdot 10^{-5},$$

$$К_{ТЕР2} = 16,08 / 303328,49 = 5,3011 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1} = 6,851 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розробляється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 6,851 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- вибір програмного продукту – Eviews;
- реалізація важливої постановки з допомогою вбудованих функцій;
- використання стандартного інтерфейсу для побудови значень.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидку реалізацію програми та доступний функціонал для роботи.

Висновки до розділу 4

В даній частині було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розробляється, було визначено та проведено оцінку основних функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

Отже, під час даної роботи було опрацьовано матеріали з теми “Матричні формальні граматики та граматики, які керуються мережами Петрі” та викладено теоретичну базу. Також було представлено доведення можливості переходу між керованими формальними граматиками матричного типу та тими, що керуються мережами Петрі та запропоновано алгоритми для його реалізації. Засобами мови програмування Python було написано програмний продукт, який може конвертувати матричну формальну граматику у граматику, яка керується мережею Петрі та навпаки керовану мережею Петрі у матричну. Програма має графічний інтерфейс для зручнішого отримання результатів, а також через який користувач має можливість швидко вносити вхідні дані. В розділі 4 дипломної роботи було проведено повний функціонально-вартісний аналіз програмного продукту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Спекторський І.Я., Статкевич В.М. Формальні мови та автомати: підруч. Київ, КПІ ім. Ігоря Сікорського, 2019. 167 с.
2. Peterson J. L. Petri Net Theory and the Modeling of Systems. Englewood Cliffs : Prentice Hall, 1981. 264 p.
3. Turaev S. Petri net controlled grammars, PhD Dissertation / Taragona, Spain, 2010. 179 p.
4. Dassow J., Turaev S. Petri net controlled grammars: the case of special Petri nets. Journal of Universal Computer Science. 2009. Vol. 15, No. 14. P. 2808–2835.
5. Спекторський І.Я. Застосування мереж Петрі для аналізу КВ-граматик. Системні дослідження та інформаційні технології. 2011. № 4, С. 129 – 133.
6. Ábrahám, S. Some questions of language theory. Proceedings of the 1965 conference on Computational linguistics (COLING). May 1965. P. 1–11.
7. Chomsky N. Three models for the description of language. IEEE Transactions on Information Theory. 1956. Vol. 2:3, P. 113–124.
8. Chomsky, N. On Certain Formal Properties of Grammars. Information and Control, 1959, 2(2), P. 137–167.
9. Hopcroft J. E., Motwani R., Ullman J. D. Introduction to Automata Theory, Languages, and Computation (3rd ed.). Addison-Wesley. 2006.
10. Hopcroft J. E., Ullman J. D. Formal languages and their relation to automata. Reading, Massachusetts : Addison-Wesley Publishing Company, 1969. 242 p.
11. Теорія цифрових автоматів та формальних мов. Вступний курс : навч. посіб. / С.Ю. Гавриленко, А.М Клименко, Н.Ю. Любченко та ін. Харків : НТУ “ХПІ” , 2011. 176 с.

ДОДАТОК А ЛІСТИНГ КОДУ

```
from tkinter import messagebox
import networkx as nx
import matplotlib.pyplot as plt

class MatrixToPetri:
    def __init__(self, non_terminals, terminals, productions):
        self.non_terminals = non_terminals
        self.terminals = terminals
        self.productions = productions
        self.num_non_terminals = len(non_terminals)
        self.num_terminals = len(terminals)
        self.num_productions = len(productions)

        self.input_matrix = [[0] * (self.num_non_terminals + self.num_terminals) for _
in range(self.num_productions)]
         self.output_matrix = [[0] * (self.num_non_terminals + self.num_terminals) for
_ in range(self.num_productions)]

    def fill_matrices(self):
        for i, production in enumerate(self.productions):
            left, right = production
            for j, symbol in enumerate(self.non_terminals + self.terminals):
                self.input_matrix[i][j] = left.count(symbol)
                self.output_matrix[i][j] = right.count(symbol)

    def get_matrices(self):
        return self.input_matrix, self.output_matrix

class PetriToMatrix:
    def __init__(self, non_terminals, terminals, input_matrix, output_matrix):
        self.non_terminals = non_terminals
        self.terminals = terminals
        self.input_matrix = input_matrix
        self.output_matrix = output_matrix
        self.num_non_terminals = len(non_terminals)
        self.num_terminals = len(terminals)
        self.num_productions = len(input_matrix)

    def get_productions(self):
        productions = []
        symbols = self.non_terminals + self.terminals
```

```
        for i in range(self.num_productions):
            left = []
            right = []
            for j in range(len(symbols)):
                count_input = self.input_matrix[i][j]
                count_output = self.output_matrix[i][j]
                left.append(symbols[j] * count_input)
                right.append(symbols[j] * count_output)

            left_side = ''.join(left)
            right_side = ''.join(right)
            productions.append((left_side, right_side))

        return productions

class Application(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Конвертер керованих граматик")
        self.geometry("800x400")
        self.create_widgets()

    def create_widgets(self):
        self.label = tk.Label(self, text="Оберіть операцію:")
        self.label.pack(pady=10)

        self.choice_var = tk.StringVar()
        self.choice_var.set("matrix_to_petri")

           self.matrix_to_petri_radio = tk.Radiobutton(self, text="Перевести матричну формальну граматику у граматику, яка керована мережею Петрі", variable=self.choice_var, value="matrix_to_petri")
        self.matrix_to_petri_radio.pack(pady=5)

         self.petri_to_matrix_radio = tk.Radiobutton(self, text="Перевести граматику, яка керована мережею Петрі у матричну формальну граматику (отримати перелік продукцій)", variable=self.choice_var, value="petri_to_matrix")
        self.petri_to_matrix_radio.pack(pady=5)

        self.next_button = tk.Button(self, text="Далі", command=self.next_step)
        self.next_button.pack(pady=20)

    def next_step(self):
        choice = self.choice_var.get()
        self.clear_window()

        if choice == "matrix_to_petri":
```

```
            self.input_matrix_to_petri()
        else:
            self.input_petri_to_matrix()

    def clear_window(self):
        for widget in self.winfo_children():
            widget.destroy()

    def input_matrix_to_petri(self):
             self.label = tk.Label(self, text="Введіть нетермінальні символи (через
кому):")
        self.label.pack(pady=5)
        self.non_terminals_entry = tk.Entry(self)
        self.non_terminals_entry.pack(pady=5)

        self.label = tk.Label(self, text="Введіть термінальні символи (через кому):")
        self.label.pack(pady=5)
        self.terminals_entry = tk.Entry(self)
        self.terminals_entry.pack(pady=5)

           self.label = tk.Label(self, text="Введіть продукції граматики (у форматі
A->BC, через кому):")
        self.label.pack(pady=5)
        self.productions_entry = tk.Entry(self)
        self.productions_entry.pack(pady=5)

                        self.submit_button = tk.Button(self, text="Конвертувати",
command=self.convert_matrix_to_petri)
        self.submit_button.pack(pady=20)

        self.back_button = tk.Button(self, text="Назад", command=self.back_to_start)
        self.back_button.pack(pady=20)

    def convert_matrix_to_petri(self):
        non_terminals = self.non_terminals_entry.get().split(',')
        terminals = self.terminals_entry.get().split(',')
        productions_input = self.productions_entry.get().split(',')
        productions = [tuple(prod.split('->')) for prod in productions_input]

        self.converter = MatrixToPetri(non_terminals, terminals, productions)
        self.converter.fill_matrices()

        self.clear_window()

        self.label = tk.Label(self, text="Матриця входів:")
        self.label.pack(pady=5)
        for row in self.converter.input_matrix:
```

```
            self.label = tk.Label(self, text=str(row))
            self.label.pack()

        self.label = tk.Label(self, text="Матриця виходів:")
        self.label.pack(pady=5)
        for row in self.converter.output_matrix:
            self.label = tk.Label(self, text=str(row))
            self.label.pack()

            self.draw_button = tk.Button(self, text="Накреслити граф мережі Петрі",
command=self.plot_petri_net)
        self.draw_button.pack(pady=20)

        self.back_button = tk.Button(self, text="Назад", command=self.back_to_start)
        self.back_button.pack(pady=20)

    def plot_petri_net(self):
        input_matrix, output_matrix = self.converter.get_matrices()
        self.plot_petri_graph(input_matrix, output_matrix)

    def plot_petri_graph(self, input_matrix, output_matrix):
        G = nx.DiGraph()

        non_terminals = self.converter.non_terminals
        terminals = self.converter.terminals
        symbols = non_terminals + terminals
        num_productions = len(input_matrix)

        for symbol in symbols:
            G.add_node(symbol, shape='o', color='lightblue')

        for t in range(num_productions):
            transition_name = f"T{t}"
            G.add_node(transition_name, shape='s', color='red')

            for i in range(len(symbols)):
                if input_matrix[t][i] > 0:
                    G.add_edge(symbols[i], transition_name)
                if output_matrix[t][i] > 0:
                    G.add_edge(transition_name, symbols[i])

        pos = nx.spring_layout(G)
        nx.draw_networkx_edges(G, pos)
        nx.draw_networkx_labels(G, pos)

        node_shapes = set((aShape[1]["shape"] for aShape in G.nodes(data=True)))
```

```
        for shape in node_shapes:
            node_list = [sNode[0] for sNode in G.nodes(data=True) if sNode[1]["shape"]
== shape]
                nx.draw_networkx_nodes(G, pos, node_shape=shape, nodelist=node_list,
node_color=[G.nodes[node]['color'] for node in node_list])

        plt.show()

    def input_petri_to_matrix(self):
             self.label = tk.Label(self, text="Введіть нетермінальні символи (через
кому):")
        self.label.pack(pady=5)
        self.non_terminals_entry = tk.Entry(self)
        self.non_terminals_entry.pack(pady=5)

        self.label = tk.Label(self, text="Введіть термінальні символи (через кому):")
        self.label.pack(pady=5)
        self.terminals_entry = tk.Entry(self)
        self.terminals_entry.pack(pady=5)

        self.label = tk.Label(self, text="Введіть матрицю входів (рядки через крапку з
комою, елементи через кому):")
        self.label.pack(pady=5)
        self.input_matrix_entry = tk.Entry(self)
        self.input_matrix_entry.pack(pady=5)

         self.label = tk.Label(self, text="Введіть матрицю виходів (рядки через крапку
з комою, елементи через кому):")
        self.label.pack(pady=5)
        self.output_matrix_entry = tk.Entry(self)
        self.output_matrix_entry.pack(pady=5)

                        self.submit_button = tk.Button(self, text="Конвертувати",
command=self.convert_petri_to_matrix)
        self.submit_button.pack(pady=20)

        self.back_button = tk.Button(self, text="Назад", command=self.back_to_start)
        self.back_button.pack(pady=20)

    def convert_petri_to_matrix(self):
        non_terminals = self.non_terminals_entry.get().split(',')
        terminals = self.terminals_entry.get().split(',')
                    input_matrix = [list(map(int, row.split(','))) for row in
self.input_matrix_entry.get().split(';')]
                   output_matrix = [list(map(int, row.split(','))) for row in
self.output_matrix_entry.get().split(';')]
```

```
                converter = PetriToMatrix(non_terminals, terminals, input_matrix,
output_matrix)
        productions = converter.get_productions()

        self.clear_window()

        self.label = tk.Label(self, text="Перелік продукцій:")
        self.label.pack(pady=5)
        for left, right in productions:
            self.label = tk.Label(self, text=f"{left} -> {right}")
            self.label.pack()

        self.back_button = tk.Button(self, text="Назад", command=self.back_to_start)
        self.back_button.pack(pady=20)

    def back_to_start(self):
        self.clear_window()
        self.create_widgets()

if __name__ == "__main__":
    app = Application()
    app.mainloop()
```

ДОДАТОК Б ПРЕЗЕНТАЦІЯ

Матричні формальні граматики та граматики, які керуються мережами Петрі

Виконав: студент IV курсу, групи КА-05
Бичков Денис Віталійович

Керівник: доцент, к. ф.-м. н., Стусь Олександр Вікторович

Мета роботи

- Об’єктами роботи є граматики, які керуються мережами Петрі та матричні формальні граматиками.
- Предметом дослідження є алгоритми, які дозволяють здійснювати перехід між граматиками, які керуються мережами Петрі та матричними формальними граматиками.
- Мета роботи полягає в розробці програмного забезпечення для конвертування граматик, які керуються мережами Петрі у матричні формальні граматики та навпаки.

Актуальність

- Потреба у нових дослідженнях з математичної логіки через великий прогрес у галузі ШІ.
- Матричні формальні граматики та граматики, що керуються мережами Петрі, є важливими інструментами для моделювання та аналізу.
- Розробка програмного забезпечення для конвертування дозволяє автоматизувати процеси переходу між різними типами граматик.

Постановка задачі

- Запропонувати алгоритми для переходу між матричними формальними граматиками та граматиками, які керуються мережами Петрі.
- Засобами мови програмування Python реалізувати ці алгоритми.

Теоретичні відомості

- Граматика, що керована мережею Петрі — це контекстно-вільна граматика, яка обладнана мережею Петрі.
- Переходи цієї мережі Петрі позначаються продукціями граматики або порожнім словом.
- Мова, що породжується цією граматикою, складається з усіх скінченних слів, які можуть бути отримані в граматиці.

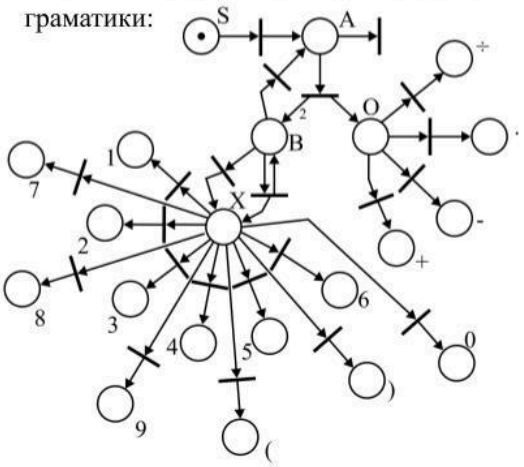
5

Теоретичні відомості

КВ граматика:

G=<V,T,P,S> , де V = {S, A, B, O, X};
T = {1,2,3,4,5,6,7,8,9,0,+,-,·,:,(,)};
P={S → A,
A → BOB|ε,
B → X|XB|A ,
O → +|-|·|:,
X → 0|1|2|3|4|5|6|7|8|9|(|)}.

Мережа Петрі, що відповідає продукціям граматики:



6

Теоретичні відомості

- Матрична формальна граматика — тип керованих формальних граматик, у яких продукції записуються не по одній, а записуються послідовно у групи — матриці.
- Виконання продукцій обов'язково відбувається в тій послідовності, в якій вони записані у матрицях.

Приклад:

G=<V,T,S, M> , де

V = {S, X, Y};

T = {a, b, c};

M = {m0=[S → XY],

m1=[X → aXb, Y → cY],

m2=[X → ab, Y → c]}

Опис алгоритмів та порівняння результатів

Продемонструємо алгоритм переходу від матричної граматики G (з попереднього слайду) до граматики, яка керується МП. Крок 1:

- Оцінити кількість позицій та переходів.
- Матрична граматика G має 3 нетермінальні символи, 3 термінальні символи та 5 продукцій у матрицях. Отже мережа Петрі буде мати 6 позицій — сума кількості термінальних та нетермінальних символів, та 5 переходів — кількість продукцій.
- Тоді матриці входів та виходів будуть мати 6 стовпчиків та 5 рядків.

Опис алгоритмів та порівняння результатів

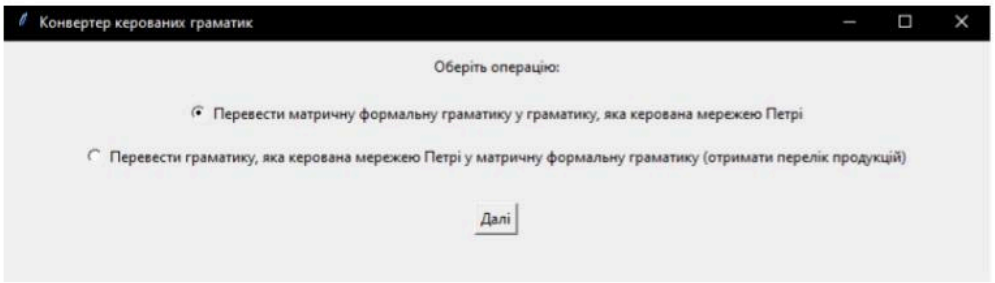
Крок 2 (заповнення матриць):

- Рядки та стовпчики відповідають термінальним/нетермінальним символам та продукціям у тому порядку, у якому вони записані у граматиці.
- Матриця входів заповнюється за таким принципом: якщо ліва частина продукції, які відповідає рядок має в собі символ, якому відповідає стовпчик — ця клітинка матриці заповнюється, числом, яке відповідає кількості цього символу (тобто кратності дуги у МП).
- Матриця виходів заповнюється аналогічно, тільки по правій частині продукції.

Опис алгоритмів та порівняння результатів

Конвертація через програму виглядає наступним чином:

1. Вибір операції



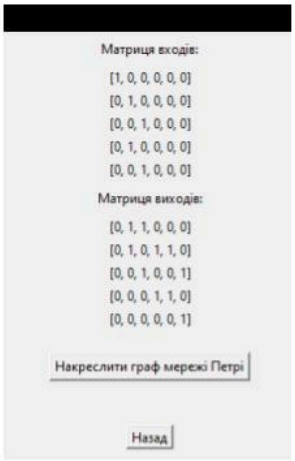
Опис алгоритмів та порівняння результатів

2. Введення вхідних даних

11

Опис алгоритмів та порівняння результатів

3. Результат роботи програми та порівняння його з результатом отриманим вручну

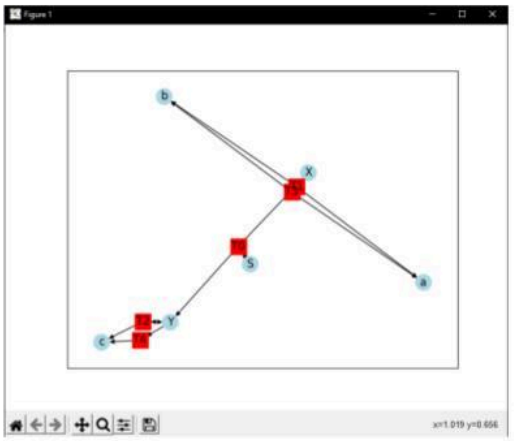


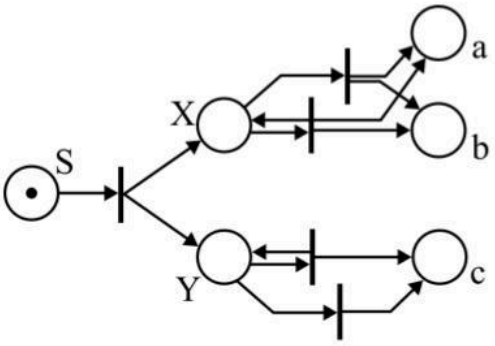
$$D^- = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad D^+ = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

12

Опис алгоритмів та порівняння результатів

3. Результат роботи програми та порівняння його з результатом отриманим вручну



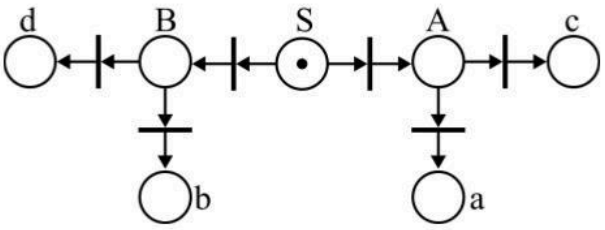


13

Опис алгоритмів та порівняння результатів

Продемонструємо алгоритм переходу від граматики, яка керується МП до матричної.

Граф МП:



Матриці входів та виходів:

$$D^- = \begin{pmatrix} 1&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&0&1&0&0&0&0 \end{pmatrix} \quad D^+ = \begin{pmatrix} 0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1 \end{pmatrix}$$

14

Опис алгоритмів та порівняння результатів

Крок 1:

- За матрицями входів та виходів зіставити переходи мережі Петрі з продукціями граматики.
- Кожен і-й рядок — це відповідний перехід t_i.
- Заповнена клітина у матриці входів — ліва частина продукції (число відповідає кількості символів), заповнена клітина у матриці виходів — права частина продукції.

Крок 2 — розподілити продукції по матрицях.

- У матриці не може бути продукцій з однаковою лівою частиною.

15

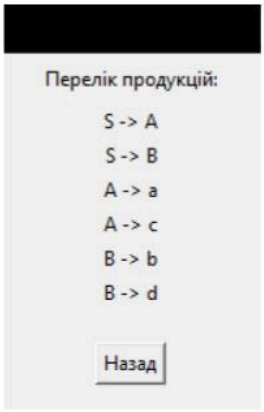
Опис алгоритмів та порівняння результатів

1. Введення вхідних даних

16

Опис алгоритмів та порівняння результатів

2. Результат роботи програми та порівняння його з результатом отриманим вручну



t1: S → A,
t2: S → B,
t3: A → a,
t4: A → c,
t5: B → b,
t6: B → d

M = {m1=[S → A, A → a]
m2 = [S → A, A→c],
m3 = [S → B, B → b],
m4 = [S → B, B → d]}

17

Аналіз результатів

- Запропоновані алгоритми можливо реалізувати програмно.
- Програмний продукт виконує свої функції справно.
- Отримані матриці входів та виходів співпадають з отриманими вручну.

18

Висновки

Під час даної роботи було запропоновано алгоритми для переходу між керованими формальними граматиками матричного типу та тими, що керуються мережами Петрі .

Засобами мови програмування Python було розроблено програмний продукт, який може конвертувати матричну формальну граматику у граматику, яка керується мережею Петрі та навпаки керовану мережею Петрі у матричну. Програма має графічний інтерфейс для зручнішого отримання результатів, а також через який користувач має можливість швидко вносити вхідні дані.

Подальші дослідження

За результатами даної бакалаврської роботи можливі такі подальші дослідження:

- Дослідження зв’язку між іншими типами керованих граматик.
- Розширення функціоналу програмного продукту.
- Застосування в інших областях.