

**ФОРМАЛЬНІ ЗАСОБИ МОДЕЛЮВАННЯ
ПАРАЛЕЛЬНИХ ПРОЦЕСІВ ТА СИСТЕМ**

ПРАЦІ
Інституту математики
НАН України
Том 90

Головний редактор А.М. Самойленко

Редакційна рада:

Ю.М. Березанський, М.Л. Горбачук, А.А. Дороговцев,

Ю.А. Дрозд, Ю.Б. Зелінський, В.С. Королюк,

А.М. Кочубей, І.О. Луковський, В.Л. Макаров,

А.Г. Нікітін, В.В. Новицький, М.В. Працьовитий,

О.Л. Рибенко, А.С. Романюк, Ю.С. Самойленко,

С.Г. Солодкий, В.В. Шарко, О.М. Шарковський

Засновано в 1994 році

Б.Б. Нестеренко
М.А. Новотарський

**ФОРМАЛЬНІ ЗАСОБИ МОДЕЛЮВАННЯ
ПАРАЛЕЛЬНИХ ПРОЦЕСІВ ТА СИСТЕМ**

Інститут математики НАН України
Київ 2012

УДК 004.942:519.876.5

Нестеренко Б.Б., Новотарський М.А. Формальні засоби моделювання паралельних процесів та систем // Праці Інституту математики НАН України.— Т.90.— Київ: Ін-т математики НАН України, 2012.— 334 с.

Монографія присвячена проблемам математичного моделювання складних систем та процесів, які включають проблеми створення засобів формалізації, здатних забезпечити достатній рівень подібності математичних моделей за умови їх ефективної реалізації на сучасних обчислювальних засобах.

Наведено короткий огляд моделей, що застосовуються для моделювання складних систем та процесів, розглянуто типовий життєвий цикл та порядок їх створення. Оскільки сучасні математичні моделі потребують застосування паралельних обчислювальних систем, актуальними є паралельні засоби опису моделей. Огляд розвитку паралельних формальних засобів подано в хронологічній послідовності їх виникнення.

Для побудови сучасних математичних моделей складних систем та процесів запропоновано використовувати APRO-мережі. Введено ряд нових структурних елементів, які разом з традиційними забезпечують широкі можливості створення ієрархічних моделей. Розглянуто принципи функціонування APRO-мереж з використанням семантики послідовних кроків, детально описано алгоритмічне забезпечення роботи переходів, розв'язання проблеми транзитивності пересилки міток, способів збору статистичної інформації.

Для аналітичного опису складних систем та процесів запропоновано алгебру процесів, яка надає можливість створення моделей з реальним робочим навантаженням. Наведено опис операцій даної алгебри, семантика яких представлена в стилі нотації Плоткіна. Кожна операція алгебри процесів має свою мережну інтерпретацію, що забезпечує однозначний зв'язок між аналітичним та мережним представленням моделі.

Наведено формальний опис обчислювального середовища для моделювання складних систем та процесів у кластерних системах.

*Затверджено до друку вченою радою
Інституту математики НАН України*

ISBN 978-966-02-2571-7

ISBN 978-966-02-6506-6

© Б.Б. Нестеренко, М.А. Новотарський

ПЕРЕДМОВА

*"Спросим себя:
что доступно непосредственному
наблюдателю? Оказывается, что
не предмет, а границы предмета"*

Л. Гумилев

Збільшення обсягів фундаментальних досліджень та бурхливий розвиток технологій спричинили підвищення інтересу до математичного моделювання як до найбільш ефективного інструменту вивчення навколишнього світу. Математичне моделювання стало також широко використовуватися в різних областях знань, об'єкти досліджень в яких не завжди підлягають точному математичному опису. Це не тільки технічні та економічні науки, де методи математичного моделювання давно вже приносять свої плоди, але й такі науки, як менеджмент, соціально-економічне прогнозування, політологія та ін.

Очевидно, що прогрес в традиційних для математичного моделювання областях пов'язаний з істотним зростанням складності моделей та засобів їх реалізації, основними з яких традиційно залишаються обчислювальні системи. В двадцятому столітті розвиток обчислювальних систем значною мірою йшов шляхом зменшення розмірів і відповідного зростання тактової частоти основного елемента обчислювальних систем – процесора. Проте вже майже десятиліття ми є свідками того, що для VLSI-технологій на основі кремнію такий напрямок розвитку обчислювальних систем досяг свого критичного рівня. Відповіддю на цей виклик стали дослідження і роботи, спрямовані, в основному, на розробку багатоядерних процесорів (Intel, AMD) та процесорів з інтеграцією спеціалізованих підпроцесорів (DSP-процесори, нейронні процесори тощо). Використання багатоядерних процесорів в персональних комп'ютерах дало новий поштовх до розвитку паралельних обчислень. До цього часу паралелізм в обчислювальній техніці існував переважно на рівні технічної реалізації обчислювальних засобів, а високорівневий паралелізм, тобто паралелізм на рівні алгоритмів, розвивався в специфічній сфері мультипроцесорних систем, які були важкодоступними для широкого загалу користувачів. І тільки багатоядерні процесори дозволили наблизити

паралельні алгоритми до пересічного користувача. Для професійних обчислень надзвичайно швидко набули широкої популярності кластерні та ґрід-системи, які об'єднують велику кількість комп'ютерів, побудованих переважно на багатоядерних процесорах. З підвищенням доступності цих систем зростає кількість користувачів з різних областей науки та виробництва, що істотно розширило коло математичних моделей складних систем та процесів та загострило проблеми ефективної організації обчислень як на рівні окремої постановки задачі, так і системи в цілому.

Теорія паралельних обчислень, яка покликана розв'язувати згадані проблеми, почала активно розвиватись лише з початку 70-х років, в той час як період розвитку послідовних обчислень нараховує не одне сторіччя. Сьогодні розвиток паралельних обчислень значною мірою потребує переосмислення традиційних підходів не тільки відносно правил формування алгоритмів, але й створення нових паралельних методів прикладної математики.

Теорія послідовних обчислень, базуючись на понятті послідовного алгоритму як скінченній послідовності кроків, виконання яких приводить до одержання бажаного результату та означенні машини Тьюрінга, ідеалізованого автомата, що став прообразом сучасних обчислювальних систем, забезпечила строге доведення універсальності послідовних обчислень та дала визначення оцінки ефективності послідовних алгоритмів.

Тривіальний перехід від послідовних до паралельних обчислень ґрунтується на ідеї розпаралелювання, що полягає у пошуку в послідовному алгоритмі незалежних фрагментів з метою їх паралельного виконання на різних процесорах багатопроцесорної системи. Тому основним критерієм ефективності розпаралелювання є *прискорення*, що чисельно дорівнює відношенню часу виконання послідовного алгоритму на одному процесорі до часу виконання розпаралеленого алгоритму на певній кількості ідентичних процесорів. Оскільки при такій оцінці ефективності в якості параметра виступає кількість паралельних процесорів, то для порівняння ефективності різних алгоритмів часто застосовують *завантаження*, що чисельно дорівнює прискоренню, яке припадає на один процесор багатопроцесорної системи.

Проте перетворення послідовних алгоритмів у паралельні шляхом застосування процедури розпаралелювання далеко не завжди дозволяє одержати бажане прискорення, оскільки переважна більшість відомих алгоритмів є принципово послідовними, тобто такими, що або не допускають взагалі, або лише частково допускають паралельне виконання фрагментів. Це й спричинило активізацію досліджень в області розробки нових обчислювальних методів, орієнтованих на

реалізацію в паралельних обчислювальних системах та розвиток теорії паралельних алгоритмів, що базуються на паралельних методах. Згадані дослідження показали, що, крім прискорення, існує ще один важливий фактор впливу на ефективність таких алгоритмів — *взаємодія*. Цей фактор проявляється через множину параметрів, серед яких найважливішими є інтенсивність взаємодії, схеми передачі даних та способи синхронізації при обміні даними між паралельними фрагментами.

Інтенсивність взаємодії може визначатись, наприклад, частотою комунікацій та обсягом даних, переданих в ході комунікацій. Паралельні методи та алгоритми їх реалізації з високою інтенсивністю взаємодії інколи називають *тісно зв'язаними*. Такі алгоритми для ефективною реалізації потребують паралельних обчислювальних систем з потужними швидкісними каналами зв'язку, до яких можна віднести сучасні кластерні системи. Низька інтенсивність взаємодії характеризує паралельні алгоритми як *слабко зв'язані*. Такі алгоритми менше залежать від засобів комунікації, тому можуть бути реалізовані в розподілених обчислювальних системах, зокрема в ґрід-системах.

Схеми зв'язків між паралельними фрагментами значною мірою пов'язані з показником інтенсивності взаємодії. Мінімальної інтенсивності можна досягти при *локальній* схемі передачі даних, оскільки в цьому випадку кожен фрагмент алгоритму робить свій внесок у розв'язання загальної задачі шляхом обміну даними тільки з фрагментами, розташованими на сусідніх обчислювальних ресурсах. Якщо область обміну даними збільшити, то, очевидно, зростатиме навантаження на комунікаційні канали, а відповідно, і інтенсивність взаємодії. Граничного значення інтенсивність взаємодії досягає при застосуванні *глобальної* схеми передачі даних, для якої кожний фрагмент алгоритму повинен приймати та передавати дані всім фрагментам алгоритму розв'язування глобальної задачі.

За відносно невеликий історичний період розвитку паралельних обчислень накопичено значний досвід розробки способів синхронізації передачі даних, які доцільно розділити на дві групи: *синхронні та асинхронні* способи взаємодії. При синхронній взаємодії операції передачі даних виконуються тільки за умови готовності всіх учасників і завершуються лише після повного закінчення всіх комунікаційних дій. Асинхронна взаємодія використовує принцип поштової скриньки, який дозволяє учасникам взаємодії не чекати повного завершення дій з передачі даних.

Цей короткий огляд властивостей взаємодії показує, що проблема ефективною організації паралельних обчислень є набагато складнішою у порівнянні з організацією обчислень на послідовних обчислювальних системах. Ефективність послідовних обчислень зале-

жить від параметрів послідовних алгоритмів, тоді як характеристики засобу їх реалізації залишаються незмінними і можуть варіюватись тільки параметрами архітектури фон Неймана. Істотна відмінність організації паралельних обчислень полягає у тому, що оцінка їх ефективності залежить не тільки від вибору паралельного алгоритму та архітектури паралельного обчислювального засобу але, головним чином, від того, наскільки вимоги алгоритму відповідають можливостям паралельної обчислювальної системи.

Аналіз відомих паралельних алгоритмів показує, що спосіб синхронізації радикально впливає на характер обчислень і повинен братись до уваги ще на ранніх етапах створення обчислювального методу. Крім того, кожний з паралельних алгоритмів характеризується певною інтенсивністю взаємодії, отже, потребує відповідної схеми передачі даних. Ці та ряд інших факторів сприяють тому, що певний паралельний алгоритм може бути ефективно реалізований тільки на паралельній обчислювальній системі з заданою архітектурою. Проте згадану ситуацію важко реалізувати на практиці, особливо в кластерних системах, коли завдання виконуються в пакетному режимі під управлінням локального менеджера ресурсів. За таких умов складно не тільки розмістити фрагменти паралельного алгоритму на заданих фізичних процесорах, але й одержати необхідну кількість процесорів для реалізації паралельного алгоритму на момент запиту.

Тому постає проблема адаптації паралельного алгоритму до архітектури паралельного обчислювального засобу. Важливими складовими процесу адаптації є *масштабування* та *модифікація* схеми розпаралелювання. Необхідність у масштабуванні виникає, коли кількість наявних паралельних фрагментів відрізняється від числа запланованих до використання процесорів. У випадку перевищення відбувається скорочення кількості фрагментів шляхом укрупнення обчислень. Правила укрупнення полягають у об'єднанні спочатку тих фрагментів паралельного алгоритму, які характеризуються максимальною інтенсивністю взаємодії за умови збереження однакової обчислювальної складності укрупнених фрагментів. При необхідності завантаження додаткових процесорів масштабування полягає в подрібненні існуючих фрагментів за умови мінімального зростання інтенсивності взаємодії між новоутвореними фрагментами.

Модифікація схеми розпаралелювання тісно пов'язана з масштабуванням, оскільки після зміни кількості паралельних фрагментів виникає потреба в їх розміщенні на доступних процесорах таким чином, щоб забезпечити максимальне завантаження при збереженні мінімальної інтенсивності взаємодії.

Описані дії з адаптації паралельного алгоритму до архітектури паралельного обчислювального засобу шляхом масштабування та модифікації схеми розпаралелювання потребують значних затрат часу та високої кваліфікації користувача. З метою автоматизації процедур адаптації використовують спеціалізоване програмне забезпечення, яке разом з виділеними ресурсами паралельного обчислювального засобу утворює *обчислювальне середовище*. Згадане програмне забезпечення виконує роль прошарку між прикладним програмним забезпеченням та ресурсами кластера, реалізуючи модель обчислень, яка імітує ідеальні для паралельного алгоритму умови реалізації. В той же час, програмне забезпечення, виконуючи процедури масштабування та модифікації, здійснює ефективне використання виділених ресурсів кластерної системи.

Отже, математичне моделювання на сучасних кластерних системах, як правило, представлено двома взаємопов'язаними моделями. Одна математична модель описує складну систему або досліджуваний фізичний процес і, як правило, представлена базовою системою рівнянь та паралельним методом її розв'язування. В якості другої моделі виступає обчислювальне середовище, що використовує попередню математичну модель як своє робоче навантаження. Така організація обчислень в кластерних системах істотно пришвидшує постановку задачі та збільшує ефективність процесу моделювання. Проте при цьому зростають вимоги до формального опису згаданого тандему моделей. Тому одним з головних завдань теорії моделювання за таких умов можна вважати пошук нових інструментів формального опису, які відрізнялися б компактністю і гнучкістю опису взаємодії компонент складних систем, а також простими засобами задавання алгоритмів функціонування цих компонент. Такі характеристики формального опису дають можливість використовувати його як основу для створення програмних модулів, які моделюють роботу складних дискретних систем. При цьому дуже важливо забезпечити еквівалентність моделі у формальному описі та програмному вигляді для того, щоб мати можливість модифікувати формальний опис за результатами тестування програмної моделі. Для успішної реалізації процесу адаптації важливою характеристикою формального представлення моделі є масштабування. Інструмент формального опису з можливістю рекурсивного повторення процедур масштабування та модифікації поточної схеми розпаралелювання дає можливість ефективно адаптувати паралельний алгоритм до структури виділених для його реалізації ресурсів кластерної системи.

Для формального опису моделей створено велику кількість формальних засобів. Наприклад, для визначення, візуалізації, проекту-

вання та документування програмних систем використовують уніфіковану мову моделювання UML разом з мовою об'єктних обмежень OCL [1]. Ці мови також застосовують при моделюванні бізнес-процесів, системному проектуванні та відображенні організаційних структур. Завдяки використанню таких понять, як клас, компонент, узагальнення (generalization), об'єднання (aggregation) і поведінка, UML дозволяє сконцентруватися на проектуванні або вивченні архітектури об'єкта, не заглиблюючись в особливості функціонування компонент і в механізми взаємодії між ними.

Мови UML, ARIS [2] та інші мови об'єктного моделювання, через універсальність, мають ряд недоліків, серед яких можна відзначити їхню надмірність та неточну семантику. Саме ці недоліки часто стають перешкодою під час визначення еквівалентності моделі та об'єкта моделювання при масштабуванні та модифікації схем розпаралелювання.

Виконання строгих семантичних правил набагато легше забезпечити у низькорівневих інструментах формального опису. Тому в сучасних підходах до автоматизації формування еквівалентних моделей використовують засоби низькорівневого опису, розвиток яких щодо паралельних моделей почався у 60-х роках з мереж Петрі. Активне застосування та бурхливий розвиток теорії мереж Петрі зумовлений вдалим поєднанням графічного та аналітичного опису, що дає можливість розглядати складну модель як сукупність процесів, які розвиваються паралельно і певним чином взаємодіють між собою. Сучасні мережі Петрі дозволяють досліджувати моделі з синхронним і асинхронним характером взаємодії складових процесів, що розвиваються у часі та просторі.

Досвід моделювання складних об'єктів за допомогою мереж Петрі з часом дозволив визначити основні вузькі місця, одним з яких можна вважати фіксований характер правил спрацювання переходів, який призводить до значного росту розміру моделі. Для подолання цього недоліку створюють нові правила спрацювання переходів, які задовольняють потреби конкретної моделі. В результаті реалізації такого підходу відома дуже велика кількість версій мереж Петрі, які мало сумісні одна з одною. В рамках теорії мереж Петрі сформовано ряд підходів, більшість з яких базуються на застосуванні методики визначення досяжності. Такі види еквівалентності дістали назву еквівалентності маркування та трасової еквівалентності. Дослідження показали, що обмеження застосування еквівалентності впливають з фіксованого набору правил роботи мереж. Важливим результатом став також висновок про неможливість гарантовано застосовувати критерії еквівалентного маркування до класичних мереж Петрі, які мають багаторівневу структуру.

Механізм встановлення еквівалентності набагато краще, ніж у мережах Петрі, реалізовано в алгебрі процесів. Процесне числення (алгебра процесів) є великою групою підходів до формального опису складних систем, що включають інструментарій ієрархічного опису компонентів систем, а також алгоритмів взаємодії між ними. У термінах алгебри процесів компоненти систем представлені за допомогою агентів або процесів, що складаються з активностей. Управління процесами виконується за алгебраїчними правилами, визначеними для кожної версії алгебри процесів, в залежності від конкретних цілей моделювання. Спільні риси алгебр процесів: визначення взаємодій між компонентами системи через процедури передачі повідомлень, опис складних систем за допомогою комбінацій з обмеженого набору операцій і операторів, визначення правил перетворення операторів, що дозволяють описувати виконання процесів за допомогою еквівалентних виразів.

Отже, мережні засоби формального опису відрізняються високими показниками відображення структурних властивостей об'єктів моделювання завдяки можливості графічної інтерпретації зв'язків між компонентами складної моделі. В той же час, більш строгим представленням складних систем є їх аналітичний опис за допомогою алгебри процесів. Протиріччя між потребою підвищення наочності задавання характеристик моделі та необхідністю слідувати строгим правилам збереження еквівалентності поведінки є основним двигуном розвитку формальних засобів опису моделей.

Дана монографія присвячена розгляду нових підходів до створення мережного та аналітичного опису моделей. Запропоновано спосіб їх об'єднання з метою еквівалентного представлення за умови збереження опису структурної організації моделі.

В розділі 1 розглянуто загальні риси моделей складних дискретних систем, що мають певний перелік властивостей та характеризуються заданою структурою. Інструментом їх дослідження є моделі. Моделі також можуть бути об'єктами дослідження, якщо множина їх параметрів має спільні елементи з множиною параметрів складної дискретної системи.

Наведено узагальнену схему класифікації сучасних моделей, яка дозволяє зорієнтуватися у виборі типу моделі та множині її характеристик, ефективних для конкретного об'єкта дослідження. Описано життєвий цикл моделей, який включає: постановку задачі, створення моделі, застосування моделі та інтерпретацію результатів. Створення моделі представлено як багаторівневий процес, у якому головним є вибір інструменту формального опису, що забезпечує еквівалентне представлення даного об'єкта.

В розділі 2 показано, що узагальнений опис розглянутих об'єктів моделювання базується на основних положеннях загальної теорії систем, яка розглядає досліджувані об'єкти як сукупність станів та переходів.

Використання узагальненого опису дає можливість формалізувати поняття *подібності* моделі та об'єкта моделювання. Розрізняють пряму подібність, обернену подібність та взаємну подібність. Виходячи з факту, що модель є подібною, але спрощеною копією складної системи, пропонується спрощена схема відкритого життєвого циклу моделі, який включає можливість задавання робочого навантаження.

Дано короткий огляд типових формальних засобів опису моделей з дискретними подіями, що включає скінченні автомати, кусочно-лінійні агрегати, специфікації та графічні формальні засоби. Загальним недоліком даних формальних засобів є орієнтація на послідовний характер процесу моделювання. Показано, що принципово новим кроком у створенні формальних засобів опису моделей стали мережні засоби. Найважливіша властивість мережних засобів полягає у можливості представлення моделі складної дискретної системи як сукупності паралельних процесів, які паралельно розвиваються і взаємодіють між собою. Формальні мережні засоби знайшли свою реалізацію у вигляді різноманітних версій мереж Петрі, E-мереж та PRO-мереж.

Сучасний підхід до створення моделей включає застосування різних версій алгебр процесів. Наведено огляд основних принципів функціонування відомих алгебр процесів з дискретними подіями.

В розділі 3 запропоновано нову версію мереж для формального опису дискретних систем з асинхронною взаємодією компонент – APRO-мережі (асинхронні PRO-мережі). Дано опис структури APRO-мереж у вигляді кортежу, що включає множини позицій, переходів, ребер, маркувань та глобальних змінних. Детально описано склад і функціональне призначення елементів цих множин. На конкретних прикладах обґрунтовано головні засади функціонування структурних елементів APRO-мереж. За аналогією з мережами Петрі запропоновано означення T-невиродженої, простої та чистої мереж, які задають співвідношення згаданих структурних елементів.

Принципи функціонування APRO-мереж описано за допомогою семантики послідовних кроків з використанням мультимножин переходів, що утворюють кроки спрацювань, які виникають під дією поточних маркувань. Розглянуто загальні питання досяжності маркування та складного кроку, що виникають в ході еволюції APRO-мереж.

Головна увага приділена алгоритмічним аспектам функціонування APRO-мереж, оскільки саме вони дозволяють забезпечити якісно новий рівень формального опису складних дискретних систем у порів-

нянні з відомими мережними засобами. Особливість алгоритмічних аспектів функціонування полягає у тому, що кожен з переходів APRO-мережі представлений у вигляді дерева процесів, листками якого є активності. Процеси переходів, які описують функціонування компонентів складної системи, можуть взаємодіяти між собою, передаючи або приймаючи інформацію у вигляді міток. Мітки відображають поточний стан моделі, зберігаються на позиціях і можуть бути передані на перехід або одержані з переходу вздовж ребер, які моделюють канали зв'язку між компонентами моделі. Асинхронний характер взаємодії між компонентами складної системи моделюється алгоритмом проходження інформації в структурі перехід—позиція—перехід. З метою створення алгоритмів функціонування кожен зі структурних елементів APRO-мережі представлений відповідними структурами даних та внутрішніми процедурами. Детально описано алгоритми, які забезпечують функціонування переходу на етапах активації, роботи, деактивації та в умовах транзитної пересилки міток. Показано, що згадані алгоритми та структури даних істотно залежать від вибору способу просування модельного часу. Тому в роботі розглянуто їх модифікації за умови застосування методу просування модельного часу з загальним ресурсом, консервативних і оптимістичних методів паралельного просування мережного часу.

Статистичну оцінку результатів моделювання досліджено в рамках застосування множин показників використання ресурсів, продуктивності та реактивності. Елементи цих множин є показниками, які дозволяють всебічно оцінити ефективність тієї чи іншої реалізації складної дискретної системи.

В рамках аналізу APRO-мереж використано властивості безпеки, обмеженості, консервативності і активності. Безпечність та обмеженість забезпечують функціонування мережі в умовах, які забороняють безконтрольне нарощування інформації в її структурах. Якщо мережа є консервативною, то цей факт символізує незнищуваність ресурсів, які представлені атрибутами міток. Активність мережі визначається правилами виникнення активного стану в її переходах. Сукупність згаданих властивостей формалізує правила можливих еволюцій мережі, які задаються деревом досяжності.

Розглянуто основні аспекти еквівалентності APRO-мереж. В зв'язку з тим, що основні принципи трасової еквівалентності, яка базується на аналізі дерева досяжності, не можуть бути використані при оцінці ієрархічних мереж, запропоновано використання критеріїв строгої взаємної подібності складних дискретних систем. Сформульовано означення строгої взаємної подібності APRO-мереж.

В розділі 4 запропонована нова версія алгебри процесів, яка базується на процесах та активностях. Одна з головних відмінностей даної алгебри від уже відомих засобів формального опису подібного типу, наприклад, CCS-алгебри, полягає у використанні додаткового класу внутрішніх активностей, так званих активностей обробки, які дозволяють використовувати запропонований формальний засіб до опису імітаційних моделей з реальним робочим навантаженням. Алфавіт активностей також включає множину активностей затримки та множину активностей взаємодії, функції яких істотно модифіковані у порівнянні з аналогами, що використовуються, наприклад, у π -алгебрах.

Детально описано синтаксис запропонованої алгебри процесів, який містить означення типів змінних для параметрів, типів змінних для процесів, множин виразів часу, множин виразів для процесів та означення допустимих операцій. Ці операції включають префіксацію, клонування, включення, перейменування, стискання, вибір, паралельну композицію, взаємодію, приховання, рекурсію та цикли. Частину з операцій запропоновано вперше, а ті операції, що вже застосовувались в інших алгебрах, істотно розширені з метою введення таких властивостей, які б дозволили обробляти інформацію, що кваліфікується як реальне робоче навантаження імітаційної моделі.

Операційна семантика даної алгебри задана в термінах класичної нотації Плоткіна, що представляє операції у вигляді дробів, знаменник яких містить запис операцій у загальному вигляді, а чисельник — можливі варіанти їх виконання. Відповідно до даної нотації, операції префіксації не мають чисельника, оскільки існує тільки один варіант їх виконання. Розрізняють п'ять видів префіксації, які визначаються типом використаної активності. Особливість виконання недетермінованої операції вибору полягає в тому, що кількість допустимих варіантів виконання цієї операції дорівнює кількості процесів, які беруть в ній участь. В семантику введено додаткову операцію вибору — керований вибір. Спрощена операція паралельної композиції може застосовуватись для означення паралелізму групи процесів без деталізації характеру зв'язків між ними. Якщо необхідний детальний аналіз процесів обміну інформацією, то застосовують спеціально введені операції взаємодії: синхронну взаємодію, керовану передачу або асинхронну взаємодію. Новою також є операція клонування, яка зводиться до генерації групи процесів на основі шаблону. Для забезпечення функціональної повноти запропонованої алгебри в ній також використано базові операції перейменування й стискання. Функцію формування ієрархічних процесів виділено в окрему операцію включення, що дозволило строго формалізувати побудову ієрархічних

моделей. Базову операцію рекурсії доповнено операцією циклу, яку використовують у випадках, коли заздалегідь відома кількість виконань процесу, що представлений його поточною змінною.

Всі операції алгебри процесів мають графічну інтерпретацію, виконану в термінах APRO-мереж. Такий підхід дозволяє забезпечити трансляцію аналітичного представлення моделі в графічне представлення і, таким чином, поєднати властивості наочності з властивостями компактності і гнучкості.

Модель, побудована за допомогою згаданого синтаксису і семантики, повинна бути подібною до об'єкта моделювання. Для цього потрібно дотримуватись принципів еквівалентності поведінки, які базуються на еквівалентному відношенні і конгруентності. Якщо модель представлена детермінованими операторами, то в якості еквівалентності достатньо застосувати трасову еквівалентність. За наявності недетермінованих властивостей використовують взаємну подібність, яка базується на вимозі не тільки еквівалентності трас, а й досягнення однакових станів після виконання відповідних активностей.

Далі, дано означення строгої взаємної подібності та доведені її властивості як для окремих станів, так і для процесів. Проаналізовано всі операції алгебри процесів на предмет збереження строгої взаємної подібності. Також наведено множину виразів та еквівалентних перетворень алгебри процесів, виконання яких не призводить до втрати строгої еквівалентності моделей. Окремо доведено, що конгруентне перетворення не втрачає властивості строгої взаємної подібності. Дано означення та розглянуто властивості квазістрокої взаємної подібності.

Обґрунтована необхідність застосування слабкої взаємної подібності для оцінки еквівалентності. Розглянуто операції алгебри процесів та еквівалентні перетворення, які не призводять до втрати моделлю спостережуваної еквівалентності. Дано означення квазіслабкої взаємної подібності та розглянуто її властивості.

Наведено прямий алгоритм визначення строгої взаємної подібності та прискорені алгоритми визначення строгої взаємної подібності і слабкої взаємної подібності, які забезпечують практичне застосування принципів оцінки еквівалентності моделей.

Розглянуто напрямки подальшої інтеграції APRO-мереж та алгебр процесів як формальних засобів опису складних дискретних систем.

В розділі 5 розглянуто основні етапи побудови моделей паралельних асинхронних процесів, описано сукупність дій, які необхідно виконати для успішного завершення цих етапів. З використанням запропонованої методики задавання APRO-мереж детально описана

мережна модель узагальненого нейрона, яка характеризується наявністю множин станів та виходів, а також структури зі зворотними зв'язками для реалізації рекурсивних функцій. Узагальнений нейрон має обчислювальні властивості, здатність транзитних пересилок та асинхронної взаємодії з навколишнім середовищем.

На основі узагальненого нейрона запропонована модель двовимірної гомогенної упорядкованої кліткової нейронної мережі, яка розглядається як проблемно-орієнтоване обчислювальне середовище для розв'язування крайових задач математичної фізики. Головна увага приділена розробці локального алгоритму оптимальної маршрутизації в упорядкованій мережі з індексом оточення 4 на основі зв'язків, що характерні для циркулянтних графів.

Наведено опис APRO-мережної моделі тривимірної кліткової нейронної мережі на основі тривимірного пошарового циркулянтного графа. Представлено три варіанти процедур, що реалізують алгоритми оптимальної маршрутизації в тривимірній клітковій мережі. Дано формальний багаторівневий опис даної моделі в термінах алгебри процесів.

Показано, що для забезпечення ефективного використання розподілених ресурсів необхідно застосовувати формальні засоби як проміжну ланку між математичною постановкою задачі та моделлю паралельного обчислювального процесу. Описано структуру та основні елементи інтерфейсу користувача програмного комплексу PROSimul для моделювання складних фізичних процесів. Практично визначено величину завантаження ресурсів кластерної обчислювальної системи програмною системою моделювання зі структурою пошарового циркулянтного графа.

РОЗДІЛ 1

Загальні питання дослідження складних систем і процесів

1.1. Процеси та системи як об'єкт дослідження

Наука як інструмент пізнання навколишнього середовища створила значну кількість різноманітних підходів до його опису. Учені завжди намагалися узагальнити ці підходи з метою створення універсального інструментарію, який був би здатним уніфіковано описувати все розмаїття нашого світу. Так виникла загальна теорія систем — наука, предметом якої є вивчення систем.

Система — сукупність взаємопов'язаних елементів, які впливають один на одного таким чином, що зміна одного спричиняє зміну деякої підмножини елементів або всієї системи.

Це означення системи включає головні ознаки, характерні для будь-якої системи: структурованість (наявність складових частин системи); наявність зв'язків між елементами системи; вплив кожного елемента на систему. Схематичне зображення системи, яка взаємодіє з навколишнім середовищем, показано на рис.1.1.

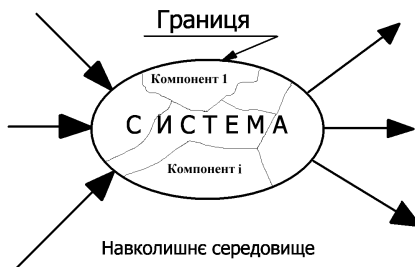


Рис.1.1. Схематичне зображення системи в навколишньому середовищі

Практично в кожній галузі науки накопичились величезні знання про об'єкти дослідження. Тому сучасні системи, що містять ці знання, називають складними системами. *Складна система* — це система, яка утворена сукупністю взаємопов'язаних компонентів, внаслідок чого вона набуває нових властивостей, відсутніх на рівні компонентів. Такі властивості, як правило, не можуть бути утворені за допомогою простої комбінації властивостей компонентів системи. Складні системи характеризуються певним набором властивостей:

1. Складна система має складний характер поведінки. Складна поведінка зумовлена тим, що зміна стану системи є функцією зміни станів її компонентів. Ця властивість найчастіше є об'єктом дослідження з метою прогнозування поведінки системи.

2. Для компонентів системи характерний близький характер взаємодії. Для складних систем типовою є взаємодія між сусідніми елементами. Якщо для функціонування системи потрібна інформація з віддалених компонентів, то вона передається через інші компоненти, і при цьому може підлягати деякій зміні.

3. Компоненти системи взаємодіють переважно нелінійно. В складних системах не завжди більша дія відповідає більшому ефекту. Інколи незначні зміни в одному компоненті можуть мати значні наслідки в сусідніх і навпаки.

4. Між компонентами системи існують зворотні зв'язки. В складних системах, як правило, присутні позитивні та негативні зворотні зв'язки між компонентами. Множина нелінійних зв'язків завжди лежить в основі поведінки системи.

5. Складна система є відкритою системою. Це означає, що енергія і інформація можуть надходити в систему ззовні або бути експортованими за межі системи.

6. Майбутні стани системи визначаються з урахуванням попередніх станів. Невеликі зміни параметрів системи у минулому можуть радикально впливати на її поведінку у майбутньому.

7. Складні системи є ієрархічними системами. Кожен компонент складної системи сам може розглядатися як складна система.

8. Межі складних систем нечітко окреслені. Вони залежать скоріше від потреб зовнішнього спостерігача, ніж визначаються внутрішніми потребами системи.

Базовим критерієм, що формує різновиди сучасних складних систем, є спосіб існування їх в часі та просторі. Просторовий спосіб існування системи визначається станом її компонентів. Якщо компоненти можуть набувати довільних значень станів в деякому допустимому діапазоні, то така система є *неперервною*. Для *дискретної* системи допустимі значення станів її компонентів задаються множиною станів, яка може бути скінченною або нескінченною. Відомі також змішані, гібридні або неперервно-дискретні системи [3], тобто такі системи, для яких характерні властивості як неперервних, так і дискретних систем.

Спосіб існування системи у часі залежить від того, який параметр є двигуном її еволюції. Якщо зміна стану системи відбувається з плином часу, то таку систему називають *таймованою*. Найчастіше таймованими є неперервні системи. Другим параметром, стимулюючим еволюцію системи, вважають механізм подій. Системи з подіями переходять від одного стану до іншого, утворюючи процес їх еволюції.

Процес – це природна або створена послідовність змін атрибутів системи. Він визначає певну траєкторію в просторі допустимих станів системи.

В замкнутих системах, відповідно до законів термодинаміки, еволюційний процес базується на послідовності подій, що збільшують ентропію системи. Таким чином, система намагається перейти до стану з максимальною ентропією, що спричиняє зупинку процесу еволюції. Моделювання процесів у замкнутих системах зводиться до дослідження можливих траєкторій їх еволюції від поточного стану до стану з максимальною ентропією.

Докорінною відмінністю відкритих систем від закритих є можливість обміну енергією з навколишнім середовищем. Якщо відкрита система знаходиться достатньо далеко від стану мак-

симальної ентропії, то процес її еволюції істотно відрізняється від процесу еволюції закритих систем. Такі системи здатні до самоорганізації, тобто переходу від одного стану до іншого, що характеризується зміною структурного складу відкритої системи та складнішим, ніж у попередньому випадку, характером зв'язків між її компонентами. Здатність до самоорганізації у більшості випадків є незворотним процесом. Тому застосування апарату класичної математичної фізики для моделювання відкритих систем викликає істотні труднощі. Цей факт лежить в основі пошуку нових підходів до моделювання складних відкритих систем та процесів їх еволюції.

1.2. Моделі як інструменти дослідження

Існує значна кількість підходів щодо визначення понять *моделювання* та *модель* [4]. У загальному випадку модель можна розглядати як об'єкт, який підлягає вивченню не заради інтересу до його внутрішньої будови, а тільки тому, що він є формалізованим і спрощеним представленням класу явищ, вивчення яких значно спрощується при такому представленні. Моделювання — це процес створення моделі як формального представлення певного реально існуючого об'єкта або такого об'єкта, що знаходиться в стані проектування.

Моделі, що розглядаються в даній роботі, відносять до математичних моделей. Такі моделі у найпростішому випадку можна представити у вигляді деякої функції $y = f(x, d)$, яка забезпечує перетворення вхідної інформації x , що надходить з навколишнього середовища, у вихідну інформацію y за умови наявності збурення d . Саме властивості згаданих елементів визначають велику різноманітність моделей. На рис.1.2 показано схематичне зображення моделі у навколишньому середовищі.

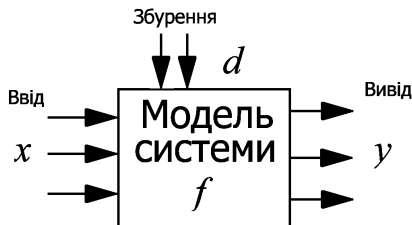


Рис. 1.2. Схематичне зображення моделі у навколишньому середовищі

Головна проблема, яка постає при створенні моделі, полягає у способі задавання функції f . У відносно незначній кількості випадків вона може бути заданою аналітично без втрати подібності об'єкта та його моделі. Складні системи, як правило, включають велику кількість компонент зі складним характером взаємодії. Тому опис таких систем потребує розробки нових засобів формалізації, які включали б нові сутності, що відображають співвідношення порядку дій та станів, а також часові співвідношення зміни цих дій та станів.

Існує два основних підходи до формалізації моделей складних систем. Традиційний підхід полягає у представленні об'єкта моделювання у вигляді системи, заданої за допомогою математичних залежностей, що утворюють його спрощений образ. У цьому випадку модель представляє собою систему рівнянь, розв'язками якої є шукана множина траєкторій станів.

За останні роки набув широкого поширення підхід, що представляє об'єкт моделювання як сукупність процесів. При цьому модель створюють за допомогою спеціальних формальних засобів. Кінцева реалізація таких моделей представлена у вигляді алгоритмів, які утворюють певне середовище моделювання. Моделі можна розглядати як системи, що мають спільну множину параметрів з об'єктом моделювання. Заміщення складної системи моделлю правомірне, якщо досліджувані властивості залежать від спільних параметрів, які пов'язані між собою однаковими залежностями, тобто існує подібність між реальною системою та її моделлю.

Дослідження моделі має сенс у таких випадках:

1. Якщо модель не містить тих властивостей, які перешкоджають безпосередньому дослідженню об'єкта моделювання.
2. Якщо модель дає можливість вимірювання параметрів, які недосяжні при безпосередньому дослідженні об'єкта моделювання.

1.3. Схема класифікації моделей

Моделювання без перебільшення можна вважати одним із атрибутів розвитку людства. Сфери застосування моделей та їх форми весь час змінюються. Тому їх властивості та характеристики не завжди можна чітко сформулювати, незважаючи на те, що поняття моделі має достатньо чітке визначення. Широкий спектр підходів до узагальнення моделей спричинив велику кількість спроб їх систематизації. В роботах [5-7] запропоновано основні критерії оцінки, спираючись на які схему класифікації моделей можна представити у вигляді множини класів, показаних на рис. 1.3.

Абстрактну модель задають у вигляді довільного набору символів. Єдине обмеження полягає у збереженні можливості однозначного представлення моделі. Математична модель описує систему за допомогою математичних символів, і тому є формою абстрактної моделі.

Фізична модель – це фізична копія системи, яку вона представляє. Фізична модель складається із матеріальних об'єктів і призначена для дослідження деяких властивостей системи. Як правило, при створенні фізичної моделі важливо зберегти пропорції між фізичними розмірами, тому такі моделі відтворюють об'єкт моделювання у масштабі.



Рис.1.3. Узагальнена схема класифікації моделей

Створення моделі тільки тоді має сенс, коли чітко визначена мета моделювання. Якщо потрібно задати модель, яка відображає поведінку системи без загострення уваги на якісних характеристиках цієї поведінки, то використовують описові моделі. Найчастіше такі моделі застосовують для підвищення ефективності проектування нових систем, коли ще не конкретизовані якісні показники поведінки. *Описова* модель виступає як гіпотетична система, яка будується з використанням всіх знань у даній галузі і дозволяє висунути ідеї щодо майбутньої поведінки прототипу та впливу на нього збурень навколишнього середовища.

Директивна модель деталізує якісні показники поведінки об'єкта моделювання. Вона містить директиви, якими слід користуватися при формуванні підмоделей компонентів системи та при задаванні порядку взаємодії між ними. Даний клас моделей дозволяє уникнути ускладнень, які неминуче виникають при формулюванні моделі за допомогою конкретного формального апарату. В той же час однозначність таких директив дає можли-

вість використання довільних формальних засобів опису. Проте не всі формальні засоби можуть бути застосовані до описових та директивних моделей з однаковим успіхом. Як результат цього застосування виникають оптимальні, субоптимальні, фізично реалізовані, фізично не реалізовані моделі, тощо.

Ще один важливий критерій класифікації моделей базується на властивостях часу. Якщо модель описує систему без урахування її еволюції у часі, то таку модель називають *статичною*. Як правило, статична модель відображає співвідношення між компонентами на певному етапі еволюції системи, який представляє найбільший інтерес. Наприклад, макет космічного корабля є його статичною фізичною моделлю.

Якщо потрібно дослідити поведінку системи з урахуванням плину часу, то тоді створюють *динамічні* моделі. Динамічні моделі містять залежності, в множині аргументів яких обов'язково входить параметр часу у явному або неявному вигляді. Прикладом динамічної математичної моделі може бути модель траєкторії руху космічного корабля, представлена системою рівнянь.

Одним із ключових критеріїв класифікації моделей є технологія їх створення. Якщо формальним інструментом опису моделей вибрано математичний апарат, який дозволяє представити систему у вигляді сукупності аналітичних залежностей, то такі моделі називають *аналітичними*. *Інформаційні* моделі, на відміну від аналітичних, орієнтовані не на закриту систему розв'язку, а на використання множини обчислювальних процедур, що імітують поведінку компонентів об'єкта. Незалежно від обраної технології реалізації модель може визначатися за допомогою певної скінченної комбінації характеристик.

Якщо взаємодія компонентів описується лінійними залежностями, то такі моделі називають *лінійними*. *Нелінійні* моделі виникають за умови хоча б одного нелінійного зв'язку між компонентами, які задають поведінку об'єкта моделювання.

За реакцією на збурення моделі поділяють на *стабільні* та *нестабільні*. Стабільна модель завжди намагається після зов-

нішнього збурення повернутися до свого початкового стану, тоді як стан нестабільної моделі після збурення може відрізнятися від того стану, що вона мала до збурення.

Ще одна важлива характеристика моделі базується на власливості повторюваності. Якщо реакція моделі на зовнішній чинник є завжди однаковою незалежно від часу її виникнення, то таку модель називають *стійкою*. *Нестійкі* моделі можуть змінювати реакцію на зовнішній чинник в залежності від різних факторів, які можуть мати детерміновану або стохастичну природу.

Стохастичними моделями називають моделі, у яких зміна хоча б одного стану є функцією від однієї або кількох випадкових змінних. Стани *детермінованих* моделей визначаються виключно детермінованими функціями.

Характеристика, яка визначає зв'язок моделей з навколишнім середовищем, дозволяє поділити їх на дві групи: *автономні* та *неавтономні*. *Автономними* називають моделі, функціонування яких відбувається без вводу даних з навколишнього середовища. Такі моделі є деякою мірою незалежними від навколишнього середовища. Логіка їх еволюціонування залежить тільки від початкового стану та стану випадкових змінних (у разі застосування стохастичних функцій). *Неавтономні* моделі призначені переважно для дослідження реакції системи на стан навколишнього середовища. В таких моделях чільне місце займають алгоритми обробки інформації, яка надходить з навколишнього середовища у вигляді вхідного потоку даних або збурень.

1.4. Життєвий цикл моделі

Складні системи, які підлягають дослідженню за допомогою моделей, завжди характеризуються *життєвим циклом*. Для кожної з таких систем виділяють свої етапи життєвого циклу, які відповідають етапам їх еволюції. Оскільки модель має відповідати об'єкту дослідження з певним ступенем подібності, то природно припустити наявність життєвого циклу і у моделі.

Такий життєвий цикл називають *життєвим циклом моделі*. У загальному випадку він містить: постановку задачі, створення моделі, застосування моделі та інтерпретацію результатів (рис.1.4).

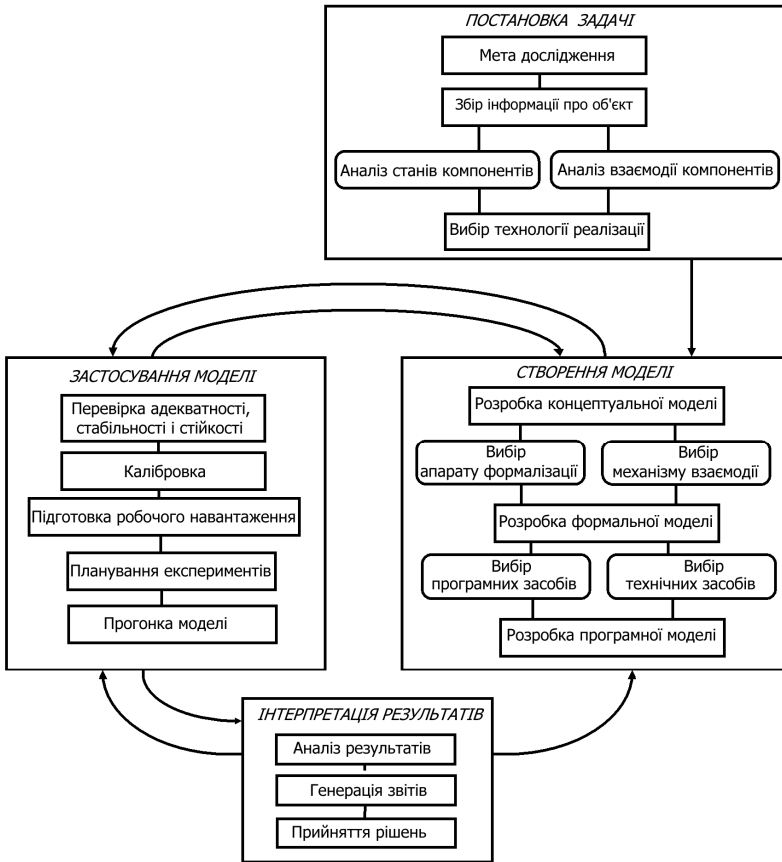


Рис.1.4. Життєвий цикл моделі

Життєвий цикл моделі завжди починається з етапу постановки задачі, ключовим питанням якого є мета моделювання. Точне визначення мети дозволяє чітко сформулювати

критерії оптимальності моделі на всіх наступних етапах її життєвого циклу.

Слід відмітити, що потреба в чіткому формулюванні мети не повинна розглядатися як звуження діапазону результатів, які потенційно можуть бути досягнуті в результаті моделювання. Мета може виступати у вигляді набору ієрархічно пов'язаних та несуперечливих тверджень, які спрямовані на оптимізацію синтезу або на проведення аналізу об'єкта моделювання. Групу цілей синтезу застосовують у випадку, коли модель виступає одним з інструментів проектування складної системи, а результати моделювання використовуються для прийняття оптимальних технічних рішень та оцінки правильності функціонування окремих компонентів.

Використання таких моделей дозволяє активно впливати на процес розробки системи і стимулювати прийняття компромісних рішень, які не є інтуїтивно очевидними [8].

Група цілей аналізу призначена для дослідження вже існуючих систем. Якщо відомі правила функціонування компонентів системи, то завдання моделювання з метою аналізу полягає у визначенні характеристик системи в цілому, як сукупності елементів.

Для успішної побудови моделі необхідна максимально повна інформація про об'єкт моделювання. Джерело цієї інформації та її вигляд може значно відрізнятись в залежності від вибраних цілей. При моделюванні з метою синтезу інформація про об'єкт моделювання є синтезованою в результаті проведення попередніх проектних розробок, а моделювання з метою аналізу базується на інформації, яка може бути одержаною в результаті дослідження системи.

Для того, щоб згадана інформація про об'єкт моделювання була корисною під час реалізації процедур створення моделі, вона повинна бути певним чином структурованою. Перш за все, така інформація повинна включати дані про множину допустимих станів компонентів системи та умови переходу від одного стану до іншого. Ці умови можуть залежати як від внутрішнього стану компонента, так і від станів інших ком-

понентів, що входять в систему. Тому актуальним є також аналіз взаємодії компонентів. Такий аналіз є відправним пунктом під час вибору архітектури моделі.

Зібрана та структурована інформація дозволяє зробити головний крок етапу постановки задачі — вибір технології реалізації моделі. Як випливає з узагальненої схеми класифікації (рис.1.3), необхідно вибрати аналітичну модель. Оскільки етап постановки задачі не є формалізованою процедурою, то вибір технології реалізації моделі неоднозначний. На нього можуть впливати як об'єктивні фактори (кількість компонентів та їх допустимих станів, аналітичність функцій переходів між станами), так і ряд суб'єктивних факторів (рівень кваліфікації розробника моделі, повнота інформації про модель тощо).

Етап створення моделі містить розробку концептуальної, формальної та програмної моделей. Концептуальна модель — це абстрактна модель, яка фіксує рівень деталізації станів компонентів системи та принципи взаємодії між ними. Підходи до побудови концептуальних моделей істотно відрізняються в залежності від вибраної технології реалізації. За умови формування аналітичної моделі її концептуальну модель можна розглядати як початковий етап, який дозволяє сформувати субстрат системи з позицій досягнення мети моделювання. Створення концептуальної моделі при виборі технології реалізації чисельних моделей дещо більш формалізоване і залежить від вибору відповідної концептуальної структури, яка забезпечує розробника способами конструювання ментальної картини моделі. Традиційні концептуальні структури забезпечують перспективу на представлення системи через множину характерних властивостей, які базуються на таких механізмах задавання поведінки моделі: механізм розвитку подій, механізм сканування станів і механізм взаємодії процесів. При розробці концептуальної моделі необхідно головну увагу приділяти вибору концептуальної структури, яка б відповідала структурі об'єкта та принципам його функціонування [9].

Перехід від концептуальної до формальної моделі супроводжується вибором апарату формалізації та механізму взаємодії між компонентами. Очевидно, що апаратом формалізації аналітичних моделей може виступати довільний математичний апарат, який задовольняє вимоги, що сформовані на етапі розробки концептуальної моделі. За необхідності врахування структурної організації об'єкта моделювання для формалізації моделей застосовують широкий набір підходів, які, як правило, є варіаціями алгебри процесів. Алгебри процесів поділяються відповідно до напрямів розвитку концептуальних структур на алгебру комунікаційних послідовних процесів (CSP — Communicating Sequential Processes) [10], числення комунікаційних систем (CCS — Calculus of Communicating Systems) [11] та алгебру комунікаційних процесів (ACP — Algebra of Communicating Processes) [12]. Вибір відповідної алгебри процесів має вирішальне значення при формуванні оптимального механізму взаємодії між компонентами моделі. Причина такого впливу полягає в тому, що узгоджена з концептуальною структурою алгебра процесів задає спосіб просування процесів моделювання. Сучасні моделі переважно орієнтовані на використання асинхронних паралельних процесів, що дало потужний поштовх до розвитку формальних засобів їх опису. Найбільш поширеними серед численної кількості формальних інструментів опису асинхронних паралельних процесів є мережі Петрі з різними модифікаціями [13, 14].

Мета моделювання, що постає перед розробниками реальних моделей, часто вимагає введення додаткових абстракцій, які виходять за межі згаданих класичних варіантів алгебри процесів. Тому розробка формальних моделей, як правило, включає створення розширень формального апарату одного з уже відомих підходів [15]. Формальна модель зберігає внутрішню логіку, що була сформована на рівні концептуальної моделі, але представлена у вигляді набору символів, які відображають цю логіку за допомогою синтаксису та семантичних правил формального засобу.

Сформована формальна модель для реалізації функціональних і логічних залежностей та одержання конкретних результатів потребує втілення в конкретний комп'ютерний код. Для цього необхідно вибрати відповідні технічні та програмні засоби. Щодо програмних засобів завжди постає питання вибору між універсальними алгоритмічними мовами та мовами, які спеціально орієнтовані на побудову моделей. Той факт, що в основу спеціалізованої алгоритмічної мови, орієнтованої на побудову моделей, покладена та чи інша концептуальна структура, дає їй переваги перед універсальними алгоритмічними мовами. Але застосування підходу, що базується на використанні концептуальних структур, також має ряд недоліків. Існує тенденція до використання тієї мови програмування, якою найкраще володіє розробник моделі, а це зумовлює підгонку природного опису моделі до форми, яка зручна для даної мови. В літературі навіть відмічається деякий відхід від програмноцентричної точки зору на процес моделювання у напрямку зростання уваги до формального опису моделі [16]. Мотивація такого відходу полягає у тому, що представлення моделі за допомогою спеціалізованої мови програмування обов'язково містить специфічну для даної мови інформацію, яка дещо спотворює визначення поведінки моделі. Крім того, використання конкретної спеціалізованої мови часто неявно спотворює початкову концептуальну структуру моделі, що може призвести до неадекватної поведінки моделі. Вибір апаратних засобів моделювання в основному базується на визначенні наявного обчислювального ресурсу, необхідного для реалізації моделі.

Розробка програмних засобів має на меті наблизити інструментарій побудови моделей до користувача та постановника задачі. В рамках даного підходу, починаючи з 60-х років, було реалізовано ряд успішних проєктів. Серед найбільш відомих — системи імітаційного моделювання СЛЕНГ, НЕДІС, СТАМ та ін. [17-19]. Згадані системи, в результаті об'єднання методів системної оптимізації з мовами моделювання, дозволили досліджувати різні варіанти складних систем на етапі їх проєктування. У 80-х роках набули широкого застосування спе-

ціалізовані алгоритмічні мови GPSS [20]. Сучасна стратегія створення систем моделювання полягає в охопленні максимально можливої величини життєвого циклу моделі. Як приклад можна навести середовище SMDE (Simulation Model Development Environment) (рис.1.5) [21].



Рис.1.5. Архітектура SMDE

Архітектура цього середовища включає кілька вкладених шарів: технічні засоби та операційні системи, ядро інтерфейсу, базовий і додатковий набір сервісних програм, які забезпечують автоматизацію під час створення, аналізу трансляції, верифікації, при архівному зберіганні та управлінні імітаційними моделями з дискретними подіями. На кожному з етапів створення моделі обов'язково повинна відбуватися перевірка параметрів її подібності, стійкості та стабільності. Поширеними способами перевірки подібності є організація контрольних прогонів на заданому наборі параметрів та експертна оцінка кожного з кроків створення моделі [22].

В залежності від того, на якому етапі створення моделі виявлено порушення подібності, розрізняють глобальну,

локальну та параметричну калібровку [8]. Глобальна калібровка потребує введення істотних змін концептуальної моделі і може бути реалізована при виявленні помилок на етапі постановки задачі. Локальна калібровка відбувається при відсутності подібності окремих компонентів або способів їх взаємодії. Вона проявляється у модифікації апарату формалізації та структури формальної моделі. Параметрична калібровка потребує мінімальних змін формального опису моделі і зводиться до вибору параметрів, які характеризують специфіку функціонування компонентів моделі.

Один із важливих кроків життєвого циклу неавтономної імітаційної моделі полягає у правильному виборі робочого навантаження. Як уже згадувалося, робоче навантаження імітаційної моделі задає вплив навколишнього середовища на поведінку моделі і виступає у вигляді вхідного потоку даних та збурень. Можливий поділ робочих навантажень на *реальні* та *модельні* робочі навантаження. Реальне робоче навантаження будують на базі „сирої” вхідної інформації, яка дійсно може впливати на поведінку об’єкта дослідження. Перевага використання реального робочого навантаження полягає у відсутності необхідності вирішення проблеми його адекватності. Але реальне робоче навантаження завжди містить значний відсоток неважливих даних, які відволікають обчислювальні ресурси і потребують ускладнення компонентів моделі. Модельне робоче навантаження, навпаки, містить тільки ті дані, які необхідні для перевірки цілей моделювання. Тому обсяг обробки вхідної інформації моделлю у даному випадку можна істотно зменшити. Але при цьому гостро постає проблема перевірки адекватності робочого навантаження. Якщо на етапі постановки задачі чітко встановлено гранично допустимі зміни параметрів навколишнього середовища, то модельне робоче навантаження може бути сформоване на базі реального шляхом відбору типових фрагментів. Одержану таким чином модель робочого навантаження будемо називати *активним* робочим навантаженням, щоб підкреслити, що його обробка подібна до обробки реального робочого навантаження.

Модель *синтетичного* робочого навантаження принципово відрізняється від активного тим, що при її створенні відсутня умова збереження внутрішньої логіки поведінки навколишнього середовища. Критеріями побудови такої моделі є імовірнісні показники тих параметрів, які впливають на продуктивність об'єкта моделювання. Тому неавтономні моделі з синтетичним робочим навантаженням частіше застосовують з метою визначення екстремальних показників продуктивності.

Планування експериментів на моделі необхідне для того, щоб досягти цілей моделювання за умови мінімального використання ресурсів. Мінімізація ресурсів досягається шляхом оптимального поєднання стратегічного і тактичного планування експериментів.

Стратегічне планування найбільш актуальне для категорії чисельних моделей. Як правило, в таких моделях неможливо отримати результати моделювання для всіх допустимих наборів параметрів. Тому в рамках стратегічного планування вибирають методику, яка б дозволяла істотно знизити кількість параметрів, що впливають на якість функціонування системи, а також звужити діапазон їх кількісних змін [23, 24]. Після вибору досліджуваних параметрів та діапазону їх зміни застосовують методи тактичного планування, які мають на меті зменшення тривалості машинного експерименту при забезпеченні статистичної достовірності результатів [7, 8, 25, 26].

Процес одержання результатів моделювання за допомогою технічних засобів будемо називати прогонкою моделі. Для прогонки сучасних аналітичних та чисельних моделей найчастіше використовують різноманітні комп'ютери, починаючи від персональних і закінчуючи надпотужними мультипроцесорними, кластерними та грид-системами. Засоби реалізації моделей поділяють на послідовні та паралельні. Послідовні засоби містять єдиний центр обробки інформації (процесор). Тому паралельний розвиток процесів у різних компонентах об'єкта моделювання можна імітувати тільки за допомогою допоміжних програмних засобів. Реального паралелізму розвитку процесів

досягають на паралельних технічних засобах, які містять деяку множину процесорів. Але при цьому кожного разу необхідно вирішувати нетривіальну задачу адаптації структури моделі до структури паралельного обчислювального середовища.

Одержані результати моделювання підлягають інтерпретації. Основне завдання етапу інтерпретації полягає у виявленні факту досягнення мети моделювання. Якщо мета моделювання досягнута, то результати оформляються у вигляді звіту, який допомагає прийняти те чи інше рішення. У випадку, коли мета моделювання не досягнута, аналіз результатів забезпечує виявлення етапів життєвого циклу моделі, які потребують коригування.

1.5. Основні проблеми побудови сучасних математичних моделей

Складність проблем, які підлягають розв'язанню за допомогою сучасних математичних моделей, призводить до необхідності використання значних програмних та технічних ресурсів на етапі їх кінцевої реалізації. Тому розробка життєвого циклу конкретної моделі, як правило, виконується на базі однієї з відомих систем моделювання. У зв'язку з цим постає питання вибору системи критеріїв, яким повинна задовольняти така система моделювання. Розробку нового підходу до побудови життєвого циклу виконують у випадку, коли жоден з існуючих підходів не задовольняє задані критерії. Узагальнюючи відомі підходи [27, 28], сформулюємо вимоги до сучасних систем моделювання, які повинні:

1. Розв'язувати широкий діапазон проблем в різних областях.
2. Забезпечувати ієрархічний опис моделі з достатньою деталізацією як на найвищому, так і на найнижчому рівні.
3. Бути незалежними від архітектури комп'ютера і допускати реалізацію на послідовних, паралельних та розподілених обчислювальних структурах.

4. Мати чітку структурну організацію, щоб забезпечити прозорий алгоритм створення моделі.
5. Забезпечувати ефективну трансляцію формальної моделі в програмну модель.
6. Мати можливості візуального відображення ходу моделювання та його результатів.
7. Включати широкий спектр механізмів калібровки та перевірки адекватності моделі.
8. Містити компоненти планування експерименту.
9. Забезпечувати зв'язок з навколишнім середовищем для підтримки процесів моделювання.
10. Мати стандартизовані форми документування результатів моделювання.

Процес створення моделі має деревовидну структуру, яка умовно може бути розбита на три рівні (рис.1.6).

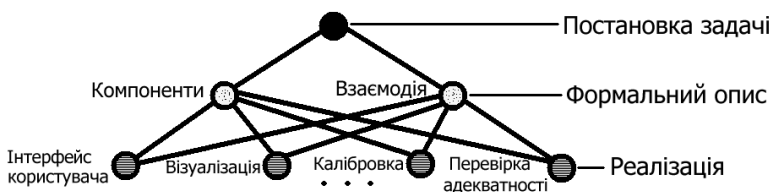


Рис.1.6. Ієрархічні рівні створення моделі

Перший рівень, відповідно до життєвого циклу, містить етап постановки задачі. Суть його полягає у накопиченні певним чином структурованої інформації про об'єкт моделювання, формулюванні мети його подальшого дослідження та вибору технології, необхідної для досягнення поставленої мети.

Другий рівень містить формальний опис моделі, сформований в результаті використання інструментів формалізації. Для формального опису моделі може використовуватися один або кілька інструментів формалізації одночасно. У випадку використання кількох засобів формалізації найчастіше один з них опи-

сує структуру компонентів моделі, а інший — механізми взаємодії між ними.

Останній рівень визначає реалізацію формальної моделі. Він містить обов'язкові елементи, що забезпечують керування моделлю, відображення результатів моделювання, механізми калібровки та перевірки адекватності. На цьому рівні можуть існувати безліч додаткових елементів, які забезпечують сервіс при керуванні моделлю і враховують особливості її технічної реалізації.

РОЗДІЛ 2

Огляд формальних засобів опису моделей

2.1. Системи з дискретними подіями

За останні роки як в промисловій сфері, так і в науці стрімко зріс інтерес до технології моделювання, аналізу та управління складними системами. Типовими прикладами таких систем є гнучкі виробничі комплекси, телекомунікаційні мережі, системи паралельної обробки інформації, логістичні системи тощо. Всі вони найчастіше мають штучне походження і складаються зі скінченної кількості ресурсів (наприклад: машин, комунікаційних каналів, процесорів), які спільно використовуються кількома користувачами (наприклад: типами матеріалів, інформаційними пакетами, завданнями), що роблять внесок в досягнення спільної мети (наприклад: виготовлення продукції, передача набору інформаційних пакетів від передавача до приймача, розв'язання задачі за допомогою паралельних обчислень).

Спільним для згаданих систем є те, що їх еволюція відбувається завдяки здійсненню низки подій, пов'язаних з активностями ресурсів (наприклад: виготовлення елемента продукції, успішна передача інформаційного пакета, обробка завдання процесором). Події відбуваються в дискретні моменти часу, а тривалість інтервалів між подіями може бути детермінованою або носити імовірнісний характер. Звідси походить назва таких систем: *системи з дискретними подіями*.

Подією будемо називати деяку дію або спонтанне явище. Події відбуваються миттєво і, в залежності від внутрішніх умов у системі, можуть викликати або не викликати зміну станів її компонентів.

Формально дискретні системи з подіями задають кортежем з двох елементів: (S, \rightarrow) , де S — множина станів, $\rightarrow \subseteq S \times S$ — бінарне відношення, яке задано на S і означає

множину переходів. Елементом множини станів S є множина $p = \{p_i\}_{i=1}^N$, де p_i — стан i -го компонента, N — кількість компонентів системи. Двома різними станами системи $p, q \in S$ будемо називати такі стани, які відрізняються станом хоча б одного компонента: $p_i \neq q_i$ при $p_i \in p, q_i \in q, 1 \leq i \leq N$. Говорять, що існує перехід (p, q) між станом p і станом q , якщо існує подія, звершення якої спричиняє зміну стану системи зі стану p на стан q . Якщо $p, q \in S$, а $(p, q) \in \rightarrow$, то факт існування переходу зі стану p до стану q може бути представлений у вигляді: $p \rightarrow q$. Системи з даним формальним описом називають *немаркованими системами з переходами*. Динаміка таких систем базується на послідовному звершенні подій. В кожний фіксований момент часу система характеризується певним глобальним станом, який складається зі станів її компонентів. Перехід до іншого глобального стану відбувається під дією події, що входить до множини допустимих подій для даного глобального стану.

Для представлення систем з паралельною обробкою подій застосовують формальний опис у вигляді кортежу: $(S, M, \{ \xrightarrow{\mu} \mid \mu \in M \})$, де S — множина станів, M — множина міток, $\xrightarrow{\mu} \subseteq S \times M \times S$ — тернарне відношення, яке задає переходи між станами моделі під дією міток. Якщо $p, q \in S, \mu \in M$ і $(p, q) \in \xrightarrow{\mu}$, то перехід зі стану p до стану q під дією мітки μ позначають $p \xrightarrow{\mu} q$. Такі системи називають *маркованими системами з переходами*. Маркована система з переходами може бути зведеною до немаркованої системи з переходами за умови, що множина M складається лише з одного елемента.

Оскільки однією з важливих властивостей складних систем є їх ієрархічність, то доцільно також звернути увагу на особливості формального опису *агрегативних* систем, тобто таких систем, компоненти яких підпадають під означення системи. Компонент агрегативної складної системи включає переходи, а тому його стан не може бути зведеним до статичного набору параметрів.

Якщо динаміка агрегативної моделі полягає в еволюції від однієї події до наступної, то такі моделі називають орієнтованими на *активності*. Кожна активність визначається своєю стартовою подією, фінальною подією та умовою. Стартова подія активності може відбутися тільки тоді, коли справджується умова активності. Активність вважається запущеною до звершення фінальної події. Послідовність активностей називають процесом.

Процес — це послідовність природних явищ або штучно спроектованих змін, які мають на меті досягнення певного результату. Згадані зміни також можуть носити ієрархічний характер, що дозволяє говорити про їх ієрархічну структуру, в основі якої лежать тривіальні зміни, які мають назву *активностей*. Отже, стан агрегативного компонента складної системи з подіями представляють процесом p , який складається з послідовності активностей, що активуються та деактивуються під впливом подій.

Агрегативну марковану систему з переходами задамо кортежем $\left(\mathbf{P}, M, \left\{ \xrightarrow{\mu} \mid \mu \in M \right\} \right)$, де \mathbf{P} — множина станів компонентів, виражених множиною процесів, M — множина міток, $\xrightarrow{\mu} \subseteq \mathbf{P} \times M \times \mathbf{P}$ — тернарне відношення, яке задає переходи між станами компонент під дією міток.

Динаміка систем з дискретними подіями характеризується також синхронізацією та паралелізмом. Синхронізація проявляється через узгодження в часі деякої сукупності подій, які зумовлюють перехід системи від попереднього до наступного

етапу її еволюції. Наприклад, машина може знаходитися в бездіяльному стані до того моменту, поки не стануть доступними всі матеріали, які необхідні для випуску елемента продукції. Про наявність паралелізму в системі говорять у випадку, коли в ній існує певна множина вільних ресурсів одного типу. Наприклад, в мультипроцесорній системі чергове завдання може бути призначене одному з вільних на даний момент процесорів.

2.2. Проблеми подібності моделей

Під моделюванням будемо розуміти процес адекватного відображення найбільш істотних сторін дискретної системи з подіями з деякою заданою точністю і проведення експериментів з моделлю для одержання інформації про об'єкт дослідження [5]. Процес адекватного відображення полягає в застосуванні процедур побудови моделі, які дозволяють відкинути неістотні компоненти системи та спростити ті компоненти, що відіграють важливу роль. При цьому необхідно зберегти *подібність* моделі до об'єкта моделювання. Будемо розрізняти пряму, обернену та взаємну подібність між моделлю та системою, яку вона описує.

Означення 2.1. *Між даною маркованою системою з переходами (S, M, \rightarrow) і моделлю (S', M', \rightarrow) існує пряма подібність, якщо для деякого бінарного відношення $R \subseteq S \times S$ на S існує відповідне бінарне відношення $R' \subseteq S' \times S'$ на S' , в якому для кожної пари станів $p, q \in S$, $(p, q) \in R \quad \forall \mu \in M$ існує така пара станів $p', q' \in S'$, $(p', q') \in R' \quad \forall \mu' \in M'$, що для кожного переходу $p \xrightarrow{\mu} q$ існує відповідний перехід $p' \xrightarrow{\mu'} q'$.*

Властивість прямої подібності, як правило, є достатньою за умови використання моделі для аналізу характеристик системи як об'єкта моделювання. Якщо ж результати моделювання використовуються на етапі проектування системи, то потрібна

впевненість у тому, що між моделлю та системою існує обернена подібність.

Означення 2.2. Між даною моделлю (S', M', \rightarrow) і маркованою системою з переходами (S, M, \rightarrow) існує обернена подібність, якщо для деякого бінарного відношення $R' \subseteq S' \times S'$ на S' існує відповідне бінарне відношення $R \subseteq S \times S$ на S , в якому для кожної пари станів $p', q' \in S'$, $(p', q') \in R' \forall \mu' \in M'$ існує така пара станів $p, q \in S$, $(p, q) \in R \forall \mu \in M$, що для кожного переходу $p' \xrightarrow{\mu'} q'$ існує відповідний перехід $p \xrightarrow{\mu} q$.

Маркована система з переходами (S, M, \rightarrow) і модель (S', M', \rightarrow) взаємно подібні, якщо існують пряма та обернена подібності між даною системою і даною моделлю. Поняття взаємної подібності лежить в основі побудови та застосування моделей і охоплює всі класи еквівалентності, що виникають в даному випадку. Основними з цих класів є еквівалентність типу модель — система, яку означають за допомогою прямої та оберненої подібності, а також еквівалентність типу модель — модель та еквівалентність типу стан — стан. Еквівалентність, у загальному випадку, може розглядатися як взаємна подібність поведінки об'єктів спостереження. Тому означення взаємної подібності процесів агрегативної системи $(P, M, \{ \xrightarrow{\mu} \mid \mu \in M \})$ є загальним випадком взаємної подібності.

Означення 2.3. Два процеси, $p, q \in P$, будемо називати взаємно подібними тоді і тільки тоді, коли існує деяке відношення $R \subseteq P \times P$ таке, що за умови $p, q \in R$ для всіх $\mu \in M$:

1. Кожному переходу $p \xrightarrow{\mu} p'$ відповідає деякий процес q' , для якого існує перехід $q \xrightarrow{\mu} q'$ і $(p', q') \in R$.

2. Кожному переходу $q \xrightarrow{\mu} q'$ відповідає деякий процес p' , для якого існує перехід $p \xrightarrow{\mu} p'$ і $(p', q') \in R$.

Згадані означення подібності базуються на встановленні відповідності між тими компонентами та переходами, які належать певному відношенню R .

2.3. Особливості життєвого циклу моделі з дискретними подіями

Згідно з означеннями подібності 2.1–2.3, модель завжди відображає структуру та поведінку об'єкта моделювання, але є простішою, ніж він. Це основна властивість моделі, що дозволяє ефективно аналізувати та впливати на поведінку об'єкта моделювання. Виходячи з даної точки зору, модифікуємо основні етапи типового життєвого циклу моделі, представленого в розділі 1. Як видно з рис.2.1, життєвий цикл моделі з дискретними подіями є відкритим циклом, що включає стан об'єкта моделювання.



Рис.2.1. Життєвий цикл моделі з дискретними подіями

Стан об'єкта моделювання задають за допомогою формального представлення у вигляді кортежу $(\mathbf{P}, M, \{ \xrightarrow{\mu} \mid \mu \in M \})$, який базується на множині процесів \mathbf{P} та множині міток M . Створення моделі об'єкта, що має таке формальне представлення, адекватне побудові взаємно подібного формального представлення $(\mathbf{P}', M', \{ \xrightarrow{\mu'} \mid \mu' \in M' \})$, де $\mathbf{P}' \subset \mathbf{P}$, $M' \subset M$. Вхідну інформацію, яка надходить із зовнішнього середовища, використовують як робоче навантаження моделі за умови $X \subseteq X'$. Наявність моделі та робочого навантаження дозволяє провести експеримент. Суть його найчастіше зводиться до проведення серії тестів для визначення змін станів компонентів моделі, що відбуваються під впливом робочого навантаження X' . Одержані результати аналізують за допомогою спеціальних методик аналізу. Створення таких методик має на меті визначити причинно-наслідкові зв'язки при взаємодії компонентів моделі в процесі обробки робочого навантаження. Основна проблема цього етапу полягає у забезпеченні ефективного виокремлення тих змін компонентів моделі, що виникають під впливом випадкових флуктуацій змінних робочого навантаження. В результаті аналізу мають бути сформовані основні рекомендації, які відповідають загальній меті моделювання. Рекомендації дозволяють зробити висновки про такі необхідні зміни об'єкта моделювання, які здатні забезпечити досягнення мети моделювання. Так завершується один життєвий цикл моделі з дискретними подіями. Якщо загальна стратегія досліджень припускає ітераційне досягнення мети, то життєвий цикл повторюється з урахуванням попередніх змін об'єкта моделювання.

2.4. Формальні засоби опису моделей з дискретними подіями

Модель з дискретними подіями $(P', M', \{ \xrightarrow{\mu'} | \mu' \in M' \})$ для подальшого її дослідження повинна бути певним чином формалізована. Такий підхід дозволяє застосувати методи, за допомогою яких можливо зрозуміти або усвідомити сутність об'єкта моделювання. Формалізація забезпечує опис структури заданих класів об'єктів та задає однозначні правила взаємодії між ними. Структура інструментарію формального опису, в свою чергу, слугує основою для визначення складності класу об'єктів, що підлягають опису. Більшість сучасних формальних засобів опису моделей з дискретними подіями базуються на положенні загальної теорії систем, яке полягає у тому, що реальні системи можуть функціонувати за однаковими правилами і демонструвати однакову поведінку, навіть якщо вони фізично неподібні.

2.4.1. Скінченний автомат

Скінченний автомат [29] базується на припущенні, що кожний наступний стан складної системи однозначно задається її теперішнім станом та набором зовнішніх впливів, дія яких регламентована правилами переходу системи до наступного стану. Його абстрактний опис задають кортежем:

$$A = (S, I, O, \sigma, \lambda), \quad (2.1)$$

де $S = \{s_n\}_{n=1}^N$ — непуста множина доступних станів;

$I = \{i_m\}_{m=1}^M$ — множина вхідних сигналів;

$O = \{o_l\}_{l=1}^L$ — множина вихідних сигналів;

$\sigma : I \times S \rightarrow S$ — функція стану, яка відображає закон формування станів автомата;

$\lambda : I \times S \rightarrow O$ — функція виходу, яка відображає закон формування вихідних сигналів автомата.

Практичне поширення одержали два класи автоматів: автомат Мілі та автомат Мура, які відрізняються один від одного способом визначення функцій виходів. Автомат Мілі при формуванні вихідного сигналу враховує свій стан та вхідні сигнали. Закон його функціонування визначається залежностями:

$$S(t+1) = \sigma(S(t), i(t)), O(t) = \lambda(S(t), i(t)), t = 0, 1, \dots$$

В автоматі Мура формування вихідних сигналів відбувається тільки на базі внутрішнього стану:

$$S(t+1) = \sigma(S(t), i(t)), O(t) = \lambda(S(t)), t = 0, 1, \dots$$

Функції σ та λ найчастіше задають у вигляді графа або таблиці. Опис складних систем за допомогою формального апарата теорії скінченних автоматів доволі громіздкий. Цю проблему намагаються вирішувати за допомогою застосування технології декомпозиції [30], яка дозволяє описувати складну систему як сукупність взаємозв'язаних компонентів, що певним чином взаємодіють між собою.

Перевагою моделей, які представлені у вигляді скінченних автоматів, є їх висока адекватність. Недоліки випливають з відсутності механізму побудови вертикально стратифікованих моделей, який давав би можливість узагальнити правила функціонування деякої підмножини компонентів складної системи. Важливим кроком у напрямку подолання згаданого недоліку стала розробка класу агрегативних моделей [31], основним елементом яких є кусково-лінійний агрегат.

2.4.2. Кусково-лінійний агрегат

Використання на початку 60-х років кусково-лінійного агрегату (КЛА) [31] в якості основного елемента складних моделей започаткувало бурхливий розвиток формальних засобів опису моделей. Цей підхід дав поштовх до розвитку як графічних, так і аналітичних засобів формалізації, оскільки для його створення була використана ідея функціонального перетворення. Графічно КЛА зображається у вигляді об'єкта (рис.2.2), який здатний сприймати входні сигнали x з множини входніх сигналів X , видавати вихідні сигнали y з множини вихідних сигналів Y . КЛА також характеризується допустимою множиною станів Z , в одному з яких z він знаходиться в кожний момент часу. На стан агрегату також впливає набір зовнішніх параметрів B . Використання КЛА для опису моделей з дискретними подіями допускає наявність двох видів подій: внутрішніх та зовнішніх. Звершення внутрішньої події спричиняє перехід КЛА з одного стану до іншого, а зовнішня подія завжди пов'язана з надходженням відповідного входного сигналу. Якщо моменти звершення входної та внутрішньої подій співпадають, то пріоритет завжди має зовнішня подія.

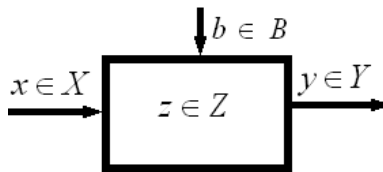


Рис.2.2. Кусково-лінійний агрегат

Аналітично КЛА задають у вигляді кортежу:

$$K = (T, X, Z, Y, B), \quad (2.2)$$

де X — множина входніх сигналів; Y — множина вихідних сигналів; Z — множина внутрішніх станів; T — множина моментів часу; B — множина параметрів.

Таким чином, процес функціонування КЛА визначається змінами, які відбуваються в певні моменти часу — моменти звернення подій. Між цими моментами стан КЛА змінюється детерміновано.

Агрегативна формалізація складної системи базується на абстрактних поняттях події, стану та структури, які є основою побудови вертикально стратифікованих моделей. Тому застосування КЛА дозволяє описувати моделі складних об'єктів. Недоліком даного підходу можна вважати те, що він орієнтований на послідовний розвиток подій, який представлений єдиним процесом моделювання. Цей факт може призвести до зниження адекватності моделі, оскільки сучасні складні системи, як правило, представлені сукупністю компонентів, процеси в яких розвиваються паралельно.

2.4.3. Специфікації систем з дискретними подіями

Серед засобів формального опису складних систем широко відомою є специфікація систем з дискретними подіями (DEVS — Discrete Event System Specification), вперше запропонована в роботі [32], та структура системних компонентів (SES — System Entity Structures) [33]. Ці два підходи можна вважати такими, що знаходяться в ряду піонерських розробок. Тому зупинимося на них більш детально. DEVS визначає три основні елементи, які забезпечують процес моделювання з дискретними подіями:

1. Об'єкт моделювання або система.
2. Модель системи.
3. Засіб реалізації моделі або комп'ютер.

Об'єкт моделювання або система розглядається як частина реального світу, що викликає певний інтерес. Система може бути природною, штучно створеною або знаходитися в стадії проектування. Результатом функціонування системи є структури даних, що відображають послідовну зміну її станів; модель системи представлена у вигляді множини інструкцій для гене-

рації структур даних, подібних до тих, що є результатом функціонування системи; а комп'ютер виступає як технічний засіб, здатний до виконання інструкцій моделі та продукування відповідних даних.

Формальний опис моделі базується на таких припущеннях:

1. Довільна система з дискретними подіями може бути промодельованою в середовищі моделювання з використанням списку подій, кожна з яких характеризується часом звернення, що визначає момент зміни станів компонентів.

2. Час звернення визначається, виходячи з поточної ситуації, яка складається в результаті звернення тих чи інших подій. Якщо час звернення події може бути визначений, то він признається даній події за умови, що вона знаходиться в списку подій.

3. Якщо час звернення події не може бути визначений наперед, то відповідна подія не може спричинити зміну станів компонентів до того часу, поки не виникнуть умови однозначного визначення часу звернення.

Далі, базову модель задають кортежем:

$$M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_a \rangle, \quad (2.3)$$

де X — множина вхідних подій; S — множина послідовних станів; Y — множина вихідних подій; $\delta_{\text{int}} : S \rightarrow S$ — функція внутрішніх переходів; $\delta_{\text{ext}} : Q \times X \rightarrow S$ — функція зовнішніх переходів; $Q = \{(s, e) \mid s \in S, 0 \leq e < t_a(s)\}$ — загальна множина станів; $\lambda : S \rightarrow Y$ — вихідна функція; $t_a : S \rightarrow R^+$ — функція просування часу.

Базова DEVS-модель має модульну структуру з інтерфейсами вводу-виводу, які утворюють порти для взаємодії з зовнішнім середовищем. Внутрішній стан моделі означається змінними, які утворюють множину станів S , а динаміка її поведінки описується двома класами подій:

1. **Вхідні події.** Події, які входять у клас вхідних подій, зумовлюють спрацювання функції зовнішніх переходів δ_{ext} , в результаті чого модель M переходить до наступного стану.

2. **Таймовані внутрішні події.** Для кожного поточного стану моделі функція просування часу t_a задає часовий інтервал до виникнення наступної внутрішньої події. Після закінчення цього інтервалу виникає наступна внутрішня подія, система генерує вихідну подію $y \in Y$, а перехід до наступного стану визначається за допомогою функції внутрішніх переходів δ_{int} .

Агрегативна DEVS–модель або SES–модель може бути утворена шляхом об'єднання базових моделей в ієрархічні структури з метою опису складних систем. В [33] запропоновано підхід до формального опису ієрархічних структур та взаємодії між окремими об'єктами, що входять в такі структури. Представлення SES має вигляд маркованого дерева, яке однозначно визначає декомпозицію складної агрегативної DEVS–моделі. У загальному випадку SES задає формалізм у термінах декомпозиції, таксономії та міжрівневої взаємодії [34]. Методологія SES представлена як технологія розробки процесу, який має вигляд послідовності активностей з об'єднанням специфікацій різних рівнів (декомпозиція). Вона також включає класифікацію різних варіантів використання системних компонент (спеціалізація) і процедуру вибору компонент на основі механізмів декомпозиції та спеціалізації. SES містить також традиційні етапи: розробку моделі, проведення експериментів на моделі, оцінку результатів моделювання та прийняття рішення.

Аналіз показує, що підходи до формального опису, які впливають із загальної теорії систем, є найбільш логічними та виваженими. Вони можуть бути використані не тільки для побудови моделей з дискретними подіями, а й у значно ширшій сфері [34]. Створені в рамках даного підходу DEVS– та SES–формалізми дозволяють будувати добре структуровані моделі, в яких ефективно вирішуються проблеми адекватності та калібровки.

Специфікації систем з дискретними подіями можна розглядати як черговий крок у розвитку формальних засобів опису складних систем. До недоліків DEVS– та SES–формалізмів слід віднести недостатню наочність представлення даних. Тому такі підходи частіше використовують для опису динаміки моделей на низькому рівні.

2.4.4. Графічні формальні засоби

Для підвищення наочності представлення моделей з дискретними подіями найчастіше використовують графи. Поширеними графічними засобами згаданого типу є графи, які відображають взаємодію між подіями [35]. Для побудови такого графа необхідно визначити множину подій і співвіднести елементи цієї множини з вершинами графа. Вершини графа з'єднують за допомогою направлених ребер, які задають характер впливу однієї події на іншу (рис.2.3).

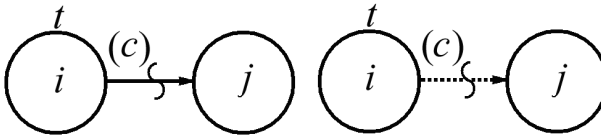


Рис.2.3. Види ребер графа, орієнтованого на події

Розрізняють два види ребер. Направлене ребро, яке зображено у вигляді суцільної лінії, показує, що через t одиниць модельного часу після звершення події i відбудеться подія j за виконання умови c . Якщо ребро графа зображено у вигляді переривчастої лінії, то воно показує, що через t одиниць модельного часу після звершення події i буде скасовано подію j за виконання умови c .

Додатковими умовами графа, орієнтованого на події, можуть бути такі:

1. Дві вершини графа з'єднуються довільною кількістю ребер.

2. Допустима наявність циклічних ребер, тобто таких ребер, які є одночасно вхідними і вихідними для даної вершини.

3. Наявність безумовних ребер, тобто таких ребер, для яких відсутні умови спрацювання.

4. Змінні часу t та змінні умови c мають детермінований або імовірнісний характер.

5. Події та умови можуть залежати від параметрів з метою узагальнення моделі.

6. Граф моделі не обов'язково повинен бути повністю зв'язним.

Очевидно, що для повної реалізації моделі на основі графа необхідно додати механізми просування модельного часу та процедури обробки черги подій.

Прикладом практичного застосування графів, орієнтованих на події, є середовище SIGMA (Simulation Graphical Modeling and Analysis). В цьому середовищі допускається пряме виконання моделі або автоматичне трансформування її в стандартний код на алгоритмічній мові C або Pascal [36]. В роботі [37] наведено узагальнення описаних підходів за допомогою побудови імітаційних графів.

Формально імітаційний граф визначається кортежем:

$$G = (V(G), E_s(G), E_c(G), \Psi_G), \quad (2.4)$$

де $V(G)$ — множина вершин; $E_s(G)$ — множина ребер активації; $E_c(G)$ — множина ребер скасування; Ψ_G — функція інцидентності.

Структуру даних на графі задають множинами:

$F = \{ f_v : S \rightarrow S \mid v \in V(G) \}$ — множина функцій переходів, асоційованих з вершинами з $V(G)$.

$C = \{ c_e : S \rightarrow \{ 0, 1 \} \mid e \in E_s(G) \cup E_c(G) \}$ — множина умов для ребер.

$T = \{ t_e : S \rightarrow R^+ \mid e \in E_s(G) \}$ — множина затримок для ребер.

$\Gamma = \{ \gamma_e : S \rightarrow R^+ \mid e \in E_s(G) \}$ — множина процедур обробки інформації.

Сукупність даних об'єктів утворює імітаційну модель:

$$I = \{ F, C, T, \Gamma, G \}. \quad (2.5)$$

Множини F, C, T, Γ є елементами моделі, а граф G забезпечує просторово-часову організацію цих елементів.

Недоліки моделей з графами, орієнтованими на події, характерні для всіх моделей, які використовують графи. Вони впливають з обмеженої кількості графічних елементів, що викликає необхідність суміщення кількох елементів моделі з одним графічним елементом шляхом введення, наприклад, різних видів ребер. Особливо важливою проблемою є відображення поточного стану моделі у випадку представлення об'єкта моделювання за допомогою кількох паралельних процесів. Для розв'язання цієї проблеми створено дводольні графи, що є основою мережних засобів опису моделей.

2.5. Мережні засоби опису моделей

Мережні засоби опису моделей беруть початок від мереж Петрі, які стали потужним формальним засобом опису паралельних процесів і систем. Широке застосування мереж Петрі, разом з їх безперечними перевагами, виявило також і ряд недоліків. Серед них, наприклад, обмежені можливості формального опису та громіздкість. Ці недоліки стали внутрішнім чинником розвитку мережних засобів. В ході еволюції було створено велику кількість модифікацій мереж Петрі, а також нові мережі: E-мережі, PRO-мережі тощо.

2.5.1. Мережі Петрі

Мережа Петрі є сучасним формальним інструментом дослідження складних систем шляхом моделювання [38]. Модель в даному випадку представляє у формалізованому вигляді основні властивості тих систем, які підлягають вивченню. Маніпулюючи параметрами такої моделі, можна одержати нові знання про систему як об'єкт дослідження та про характер її зв'язків з навколишнім середовищем.

Рівень деталізації моделі визначає складність мережі Петрі. При скороченні обмежень на опис властивостей системи зростає адекватність моделі, але при цьому зростає також і її складність та складність аналізу результатів моделювання. Така залежність характерна і для інших традиційних математичних моделей. Особливістю мереж Петрі є те, що вони базуються на певній технології, яка дозволяє органічно поєднувати аналітичні розрахунки для окремих елементів системи з моделюванням міжелементної взаємодії, і таким чином, одержувати нові знання про поведінку всієї сукупності елементів, що утворюють систему. Саме ця властивість мереж Петрі дозволяє застосовувати їх як засіб моделювання паралельних структур.

Означення 2.4. Мережу Петрі для розв'язування задач моделювання паралельних структур задають у вигляді кортежу:

$$\Phi = \left(P, T, F, W, \mu^{(0)} \right), \quad (2.6)$$

де $P = \{ p_1, p_2, \dots, p_n \}$ — скінченна множина позицій, $n \geq 0$;

$T = \{ t_1, t_2, \dots, t_m \}$ — скінченна множина переходів, $m \geq 0$;

$F \subseteq \{ P \times T \} \cup \{ T \times P \}$ — скінченна множина ребер, що з'єднують переходи та позиції;

$W = \{ w_{i,j} (p_i, t_j) \mid i \in \{ 1, \dots, n \}, j \in \{ 1, \dots, m \} \}$ — скінченна множина вагових коефіцієнтів, які поставлено у відповідність до елементів множини F ;

μ — вектор міток, що генерується функцією $M : P \rightarrow N$, де N — множина натуральних чисел.

Як впливає з означення мережі Петрі, множина ребер $F \subseteq \{P \times T\} \cup \{T \times P\}$ складається з підмножин вхідних ребер $I \subset \{P \times T\}$ та підмножини вихідних ребер $O \subset \{T \times P\}$. Множину вхідних ребер переходу $t \in T$ позначають як $I(t)$, відповідно множину вихідних ребер для даного переходу як $O(t)$.

Динаміка мереж Петрі базується на двох основних правилах:

1. **Правило активації.** Перехід $t \in T$ активується відповідним маркуванням μ його вхідних позицій тоді і тільки тоді, коли вони містять задану кількість міток. Множина активованих переходів $E(\mu)$ в загальному випадку визначається умовами: $\forall p \in I(t); \mu(p) \geq w(p, t)$. Активація переходів створює передумови їх спрацювання, але не кожний активований перехід може спрацювати.

2. **Правило спрацювання.** Спрацювання переходу t , активованого маркуванням $\mu^{(i)}$, має наслідком формування нового маркування мережі $\mu^{(i+1)}$ шляхом вилучення міток з вхідних позицій і розміщення деякої кількості міток на вихідних позиціях. Загальні правила формування маркування шляхом спрацювання переходів визначаються такими умовами:

$$\forall p \in I(t) \cup O(t), \mu^{(i+1)}(p) = \mu^{(i)}(p) - w((p, t)) + w((t, p)).$$

Позначимо через $w_{pt} = w((p, t))$ і $w_{tp} = w((t, p))$ вагові коефіцієнти відповідно вхідних та вихідних ребер. Тоді зміну в маркуванні мережі, що є результатом спрацювання переходу t , визначають як різницю $a_{tp} = w_{tp} - w_{pt}$. Загальна зміна при спрацюванні всіх переходів представлена матрицею інцидент-

ності $A = [a_{tp}]$, де $t \in T, p \in P$. Тепер умова активації переходу $t \in T$ має вигляд: $\mu^{(i)}(p) \geq w_{pt} \quad \forall p \in I(t)$, а зміна маркування позицій може бути задана в такий спосіб:

$$\mu^{(i)} = \mu^{(0)} + A^T \sum_{j=1}^i e_j,$$

де $\mu^{(i)} = (\mu_1^{(i)}, \dots, \mu_k^{(i)}, \dots, \mu_n^{(i)})$ при $\mu_k^{(i)} = \mu^{(i)}(p_k)$;

A^T — транспонована матриця A розміру $m \times n$;

e_j — m -вимірний вектор спрацювання з елементами 0, за виключенням одного елемента, що дорівнює 1.

Нехай елемент k вектора e_j дорівнює 1. Тоді з матриці A буде вибраний лише рядок k , що відповідає зміні маркування від спрацювання переходу t_k . Спрацювання переходів мережі фіксує лічильник спрацювань. Такий лічильник представлений вектором τ , що формується за правилом:

$$\tau = \sum_{j=1}^i e_j.$$

Зміна маркувань мережі, що відбулась при переході між станами i та l , може бути визначена з виразу:

$$\mu^{(l)} - \mu^{(i)} = A^T \tau.$$

Означення 2.5. Вектор τ називають T -інваріантним у випадку, коли він є розв'язком рівняння $A^T \tau = 0$, і елементами його є додатні цілі числа. Очевидно, що застосування T -інваріантного вектора не викликає зміни маркування мережі.

Аналогічно визначають P -інваріантний вектор.

Означення 2.6. Вектор y називають P -інваріантним, якщо він складається з елементів y_j , що відповідають зміним вагових коефіцієнтів при спрацюванні переходу $t_j \in T$ мережі Петрі, і задовольняє умову: $Ay = 0$.

Таким чином, елементами вектора y повинні бути натуральні числа, які виражають середньозважену кількість міток $\bar{\mu} = \sum_{j=1}^m \mu_j^{(i)} y_j$ на відповідних позиціях p_j , інваріантну для довільного маркування $\mu^{(i)}$. Властивість інваріантності є дуже корисною при дослідженні мереж Петрі, орієнтованих на моделювання паралельних структур. На практиці вектор y шукають шляхом розв'язування рівняння:

$$\begin{bmatrix} A_{11}^{|r \times m - r|} & \bar{A}^{|r \times r|} \\ A_{21}^{|n - r \times m - r|} & A_{22}^{|n - r \times r|} \end{bmatrix} \cdot \begin{bmatrix} Y_1^{|m - r \times 1|} \\ Y_2^{|r \times 1|} \end{bmatrix} = 0, \quad (2.7)$$

де \bar{A} — субматриця матриці A з повним рангом r , $[Y_1, Y_2]^T$ — відповідна декомпозиція вектора Y .

Використавши лінійно залежну частину $A_{11}Y_1 + A_{12}Y_2 = 0$ рівняння (2.7), одержимо: $Y_2 = -\bar{A}^{-1}A_{12}Y_1$. Звідси

$$\begin{bmatrix} Y_1^{|m - r \times 1|} \\ Y_2^{|r \times 1|} \end{bmatrix} = \begin{bmatrix} I^{|m - r \times m - r|} \\ -\frac{A_{11}^{|r \times m - r|}}{\bar{A}^{|r \times r|}} \end{bmatrix} \cdot Y_1^{|m - r \times 1|}, \quad (2.8)$$

де I — тотожна матриця розміру $(m - r \times m - r)$, що задовольняє умову $Y_1 = IY_1$.

Рівняння (2.8) дозволяє знайти всі можливі P -інваріантні вектори, як вектори Y з натуральними елементами.

Означення 2.7. Маркування $\mu^{(l)}$ називають досяжним для маркування $\mu^{(i)}$ у випадку, якщо існує послідовність переходів $\lambda = (t_a, t_b, \dots, t_z)$, спрацювання яких викликає зміну маркувань

$$\mu^{(i)} \xrightarrow{t_a} \mu^{(i+1)} \xrightarrow{t_b} \mu^{(i+2)} \xrightarrow{t_c} \dots \xrightarrow{t_z} \mu^{(l)},$$

або скорочено: $\mu^{(i)} \xrightarrow{\lambda} \mu^{(l)}$.

Необхідною умовою досяжності маркування $\mu^{(i)}$ відносно початкового маркування $\mu^{(0)}$, тобто для $\mu^{(0)} \xrightarrow{\lambda} \mu^{(i)}$, є існування вектора лічильника спрацювань τ як розв'язку рівняння $A^T \tau = \Delta \mu$, де $\Delta \mu = \mu^{(i)} - \mu^{(0)}$. Множину всіх маркувань, досяжних відносно довільного маркування $\mu^{(i)}$, задають функцією $R(\mu^{(i)})$, що залежить від початкового маркування та загальної кількості позицій в мережі. Її використовують для поділу мереж Петрі на безпечні, k -обмежені та необмежені.

Означення 2.8. Мережу Петрі називатимемо k -обмеженою, якщо для довільного маркування $\mu \in R(\mu^{(0)})$ кількість міток на довільній позиції $p_i, i \in \overline{1, n}$; не перевищує k . 1-обмежена мережа є безпечною, а ∞ -обмежену мережу Петрі називають необмеженою. Більш жорсткою є властивість структурної обмеженості.

Означення 2.9. Мережу Петрі називають структурно обмеженою з коефіцієнтом обмеження k , якщо має місце нерівність $\mu^{(i)} \leq k$ для довільного обмеженого початкового маркування $\mu^{(0)}$. Структурна обмеженість існує тоді і тільки

тоді, коли існує вектор натуральних чисел y такий, що $Ay \leq 0$.

Означення 2.10. *Мережу Петрі називають консервативною на підмножині $P' \subseteq P$, якщо число міток на P' не зростає, незважаючи на спрацювання переходів. Необхідною та достатньою умовою консервативності є існування P -інваріантного вектора y з натуральними елементами, що відповідають позиціям $p \in P'$. Таке твердження випливає з (2.7).*

Означення 2.11. *Мережу Петрі називають стійкою на підмножині переходів $T' \subseteq T$ за умови, що довільний перехід $t \in T'$ може втратити можливість активації тільки завдяки власному спрацюванню, а не завдяки спрацюванню інших переходів даної підмножини.*

Означення 2.12. *Мережу Петрі називають живою на підмножині переходів $T' \subseteq T$, якщо для кожного переходу $t \in T'$ і довільного досяжного маркування $\mu \in R\left(\mu^{(0)}\right)$ існує маркування $\mu' \in R(\mu)$ таке, що задовольняє умову активації переходу t .*

Жива мережа Петрі завжди складається з потенційно живих переходів.

На практиці існує значна кількість означень живої мережі Петрі [39], кожне з яких є корисним при моделюванні того чи іншого об'єкта.

Означення 2.13. *Маркування $\mu^{(i)}$ називають домашнім станом, якщо для кожного маркування $\mu^{(j)} \in R\left(\mu^{(i)}\right)$ існує*

послідовність спрацювань переходів λ , яка забезпечує зміну маркувань $\mu^{(j)} \xrightarrow{\lambda} \mu^{(i)}$.

2.5.1.1. Класи мереж Петрі

Класи мереж Петрі визначаються обмеженнями, які накладаються на характер зв'язків, заданих елементами множини F . На рис.2.4 показано деякі типи структурних обмежень, що формують такі класи.

Мережі з обмеженнями $|I(t)| = |O(t)| = 1 \forall t \in T$

називають машинами станів (рис.2.4.а), оскільки переходи не можуть ні вилучати мітки із вхідних позицій, ні створювати їх на вихідних позиціях. Функціонування моделей, побудованих з даними обмеженнями, відбувається за правилами автоматів з кінцевими станами.

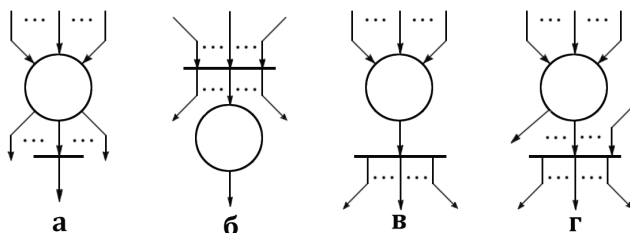


Рис.2.4. Типи структурних обмежень в мережах Петрі

Основною сферою застосування таких мереж є моделювання паралельних процесів та конфліктів, що виникають при доступі процесів до спільних ресурсів.

Мережі Петрі, що базуються на структурному обмеженні

$$|I(p)| = |O(p)| = 1 \forall p \in P,$$

мають назву маркованих графів (рис.2.4.б). У цьому випадку надлишковими елементами є позиції. За допомогою маркованих графів моделюють синхронізацію паралельних процесів.

Структурне обмеження, показане на рис.2.4.в, можна розглядати як розширення маркованих графів. Таке розширення полягає в знятті обмеження на кількість вхідних ребер позиції:

$$|I(p)| \geq 1, |O(p)| = 1 \quad \forall p \in P.$$

Мережі Петрі, які мають дане обмеження, звичайно називають безконфліктними мережами.

Об'єднавши обмеження машин станів та маркованих графів за умов:

$$O(p_i) \cap O(p_j) = \emptyset \Rightarrow |O(p_i)| = |O(p_j)| = 1,$$

$$\forall p_i, p_j \in P, i \neq j,$$

одержують ще один клас, що має назву мереж Петрі з вільним вибором. Цей клас характеризується наявністю конфлікту, який виникає у випадку, що зображений на рис.2.5.

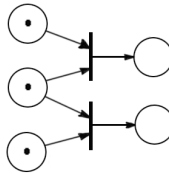


Рис.2.5. Конфлікт у мережі з вільним вибором

Оскільки маркування не містить додаткових відомостей про активацію того чи іншого переходу, то в даному випадку відбувається незалежна активація обох переходів. Але при цьому настає невизначеність умов їх спрацювання.

На рис.2.4.г зображена мережа, яка вільна від будь-яких обмежень на кількість вхідних та вихідних ребер позицій і переходів. Тому її називають узагальненою мережею Петрі.

Незалежно від приналежності до того чи іншого класу, мережі Петрі називають строго зв'язаними за умови:

$$|I(t)| > 1, |O(t)| > 1 \quad \forall t \in T; \quad |I(p)| > 1, |O(p)| > 1 \quad \forall p \in P.$$

2.5.1.2. Мережі Петрі з урахуванням часу

В класичних мережах Петрі спрацювання переходів відбувається миттєво, тобто не враховують час, що витрачається на перехід мережі від одного маркування до іншого. Моделювання реальних систем за допомогою таких мереж дозволяє провести тільки їх якісну оцінку, а саме: визначити причинно-наслідкові зв'язки та характер синхронізації елементів системи. З метою проведення кількісного аналізу вводять часові параметри, які можуть бути асоційовані з переходами, позиціями, ребрами або мітками.

Послідовність спрацювань η в маркованій мережі Петрі – це послідовність маркувань $\mu^{(0)} \xrightarrow{\lambda} \mu^{(j)}$, яку одержують в результаті спрацювання послідовності активованих переходів $\lambda = (t^{(0)}, \dots, t^{(j)})$:

$$\eta = \left\{ \left(\mu^{(0)}, t^{(0)} \right); \left(\mu^{(1)}, t^{(1)} \right); \dots; \left(\mu^{(j)}, t^{(j)} \right); \dots \right\}.$$

Послідовність спрацювань з урахуванням часу:

$$\eta_{\tau} = \left\{ \left(\mu^{(0)}, t^{(0)}, \tau_0 \right); \left(\mu^{(1)}, t^{(1)}, \tau_1 \right); \dots; \left(\mu^{(j)}, t^{(j)}, \tau_j \right); \dots \right\},$$

$$\text{де } \tau_0 \leq \tau_1 \leq \dots \leq \tau_j,$$

період $\tau_{j+1} - \tau_j$ дорівнює періоду життя маркування $\mu^{(j)}$.

Означення 2.14. *Мережі Петрі з урахуванням часу — це марковані мережі Петрі зі встановленою множиною описів та визначеною множиною правил таких, що для кожної легальної послідовності спрацювань η існує відповідна послідовність з урахуванням часу η_{τ} .*

Мережі Петрі з часовими параметрами переходів моделюють процеси, розвиток яких істотно залежить від часу їх існування [40]. Часові параметри позицій, як правило, відповідають проміжкам часу між подіями, що відбуваються в ході моделювання [41].

Розглянемо спочатку алгоритм спрацювання переходу мережі Петрі з часовим параметром позицій (рис.2.6).

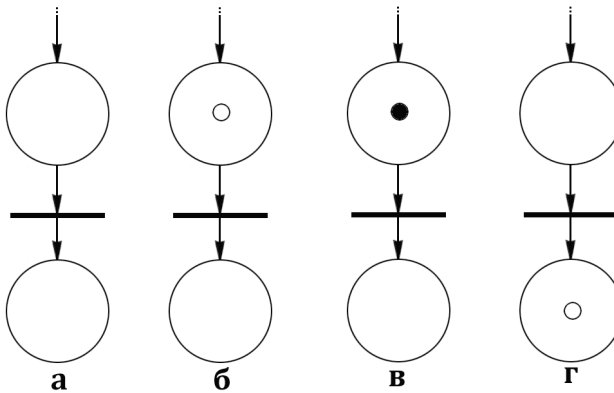


Рис.2.6. Алгоритм спрацювання переходу в мережі Петрі з часовим параметром позицій

Цей алгоритм складається з таких етапів:

1. Поява мітки на вхідній позиції в момент часу τ_i (рис.2.6.б).
2. „Визрівання” мітки протягом періоду $\Delta\tau$. Активація переходу готовою міткою в момент часу $\tau_{i+1} = \tau_i + \Delta\tau$ (рис.2.6.в).
3. Миттєве спрацювання переходу в момент часу τ_{i+1} (рис.2.6.г).

Алгоритм спрацювання переходу мережі Петрі у випадку, коли часовий параметр асоціюється з самим переходом, показано на рис.2.7.

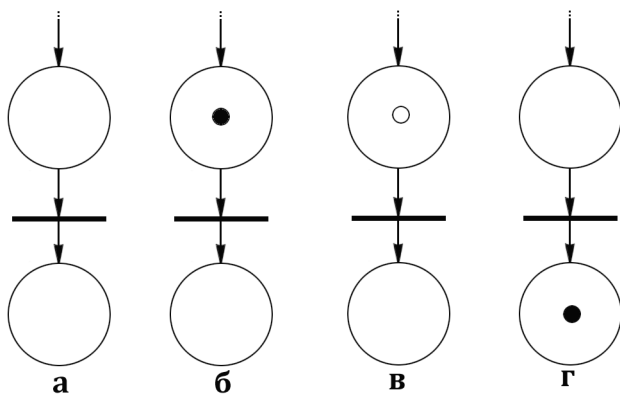


Рис.2.7. Алгоритм спрацювання переходу в мережі Петрі з часовим параметром переходів

При цьому час відводиться на роботу переходу, а активація його відбувається миттєво після виникнення умов активації.

Етапи алгоритму:

1. Поява мітки на входній позиції і активація переходу в момент часу τ_i (рис.2.7.б).
2. Блокування мітки на входній позиції на період $\tau_{i+1} = \tau_i + \Delta\tau$ і запуск роботи переходу.
3. Закінчення роботи переходу в момент часу τ_{i+1} .

В [42] доведено повну алгоритмічну еквівалентність згаданих алгоритмів.

У випадку, коли величина $\Delta\tau$ є фіксованою для кожного об'єкта мережі, говорять про детерміновані мережі Петрі. Якщо ж вибір $\Delta\tau$ змінюється випадково за деяким законом розподілу, то такі мережі називають стохастичними мережами Петрі з урахуванням часу. Ці мережі набули особливо широкого поширення при дослідженні проблем надійності та живучості.

Важливу роль для процесів моделювання та аналізу систем за допомогою мереж Петрі з урахуванням часу відіграють модифіковані правила активації та спрацювання переходів.

Якщо перехід в результаті активації обробляє лише одну мітку, то його називають одноактивним. Такі переходи розглядалися нами до цього часу.

Означення 2.15. *Якщо перехід t активується за умови, коли $\forall p \in I(t), \mu(p) \geq cw((p, t)), c > 1$, то говорять, що такий перехід є мультиактивним зі ступенем активації c .*

Для одноактивного переходу час обробки мітки:

$$\tau_i = \tau_{i-1} + \Delta\tau_i,$$

де τ_{i-1} — час закінчення обробки попередньої мітки, $\Delta\tau_i$ — період часу, відведений на обробку мітки (i).

Мультиактивний перехід здійснює обробку c міток за період $\Delta\tau_i$, що свідчить про паралельну обробку в рамках даної активації.

Параметр часу може асоціюватися як з періодом активації, так і з періодом спрацювання переходу. У випадку врахування часу на активацію для деяких розглянутих класів мереж виникає проблема встановлення додаткових правил активації, які задавалися б пріоритетом міток. У випадку безпріоритетної активації застосовують політику попереднього вибору та елементарного спрацювання.

При активації переходу відбувається аналіз умов спрацювання. Якщо ці умови виконуються, то мітки знімаються з вхідних позицій $I(t)$ і розміщуються на вихідних позиціях $O(t)$. Таку політику називають політикою попереднього вибору, оскільки у випадку конфлікту між двома переходами вибирають тільки один як такий, що повинен спрацювати. При виборі політики елементарного спрацювання запуск переходу відкладається на певний період $\Delta\tau$ після активації. Перехід спрацьовує, якщо він залишається активним до кінця даного періоду. У протилежному випадку вирішується задача повторної активації.

При попередньому виборі конфлікти, які виникають між переходами в ході моделювання, вирішуються за допомогою встановлених правил або стохастичних перемикачів, політика елементарного спрацювання реалізує вирішення конфліктів, виходячи з внутрішньої логіки обробки даних.

Як уже згадувалося раніше, недоліки представлення моделей за допомогою мереж Петрі постають через значну громіздкість, яка зумовлена обмеженістю правил спрацювання переходів. З метою подолання цих недоліків було створено нові типи мереж, зокрема, Е-мережі, що характеризуються розширенням згаданих правил.

2.5.2. Е-мережі

Перші результати застосування Е-мереж було наведено в роботі [43]. Формально Е-мережу задають у вигляді кортежу:

$$A = (P, T, F, M, G), \quad (2.9)$$

де P — скінченна множина позицій,

T — множина переходів,

$F : P \times T \cup T \times P \rightarrow N$ — множина направлених ребер, які з'єднують позиції та переходи,

$M : P \rightarrow W$ — маркування з діапазоном значень W ,

G — правила переключення.

Позиції Е-мережі характеризуються трьома станами:

$l(p)$ — пуста позиція $p \in P$,

$h(p)$ — заповнена позиція $p \in P$,

$r(p)$ — невизначений стан позиції $p \in P$.

Позиції заповнюються мітками, атрибути яких є впорядкованою множиною параметрів з певним діапазоном зміни.

Нехай діапазон для j -го атрибута набуває значення $w_j \in W$. Тоді загальне маркування мережі визначається виразом:

$$M(P) \in \{l(P), r(P)\} \cup h(P),$$

де $h(P) = \{(w_1, \dots, w_j, \dots, w_k) \mid w_j \in W\}$.

Існує особливий вид позицій — це позиції, які містять процедуру обробки:

$$[V_1 \rightarrow m(\nu) = i; V_2 \rightarrow m(\nu) = 1 - i], \quad i \in \{0, 1\},$$

де V_1, V_2 — логічні вирази, що можуть входити у вигляді атрибутів в довільні позиції мережі.

Процедура обробки забезпечує маркування позиції ν величиною i за умови, що вираз $V_1 = \text{false}$, а $V_2 = \text{true}$. Позиція ν маркується величиною $(1 - i)$ і залишається невизначеною у всіх інших випадках. Переходи Е-мережі задають таким набором параметрів:

$$(s, f, \tau, e),$$

де s — тип переходу; f — кількість вхідних та вихідних ребер переходу; τ — час спрацювання; e — процедура переходу.

В [43] розглянуто п'ять основних типів переходів:

J (від *JOIN* – з'єднання) — містить кілька вхідних ребер і одне вихідне;

F (від *FORK* – розгалуження) — відповідає конструкції з одним вхідним ребром і кількома вихідними ребрами;

T (від *TRANSITION* – передача) — схема з одним вхідним і одним вихідним ребром;

X (*CROSSING* – перетин) — містить 2 вхідних і 2 вихідних ребра;

Y (*COMMUTATOR* – перемикач) — містить 3 вхідних ребра і одне вихідне ребро.

Одне вхідне ребро в переходах типу X і Y завжди виходить з керуючої позиції, яка містить процедуру обробки. Для цієї позиції завжди $k(\nu) = 0$, тому вона може мати тільки таке маркування:

$$l(\nu) \rightarrow 0, h(\nu) \rightarrow 1, r(\nu) \rightarrow \beta,$$

де $\beta = 0$, якщо ν — внутрішня позиція; $\beta = \emptyset$, якщо ν — периферійна позиція.

На графі Е-мережі керуючі позиції зображують шестикутниками, а позиції, які задають структуру даних, — кружками. Позицію називають внутрішньою позицією, якщо вона має зв'язок з двома різними переходами, і периферійною позицією, якщо вона з'єднана тільки з одним переходом. Для п'ятірки основних переходів множину f означають через можливість з'єднання з допустимими типами позицій:

$$f = \{ \nu, a, b; c, d \},$$

де ν — керуюча позиція; a, b — вхідні інформаційні позиції; c, d — вихідні інформаційні позиції.

Для однозначності вводять обмеження циклів, тобто позиція не може бути одночасно вхідною і вихідною для одного і того ж ребра.

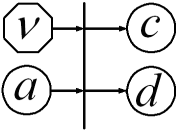
Правила спрацювання переходів задають функцією $S(f)$, яка визначає розміщення міток на позиціях до спрацювання переходу і після спрацювання.

На рис.2.8 наведено зображення основних переходів Е-мережі та правила їх функціонування. Час спрацювання може визначатися в залежності від того, чи має даний перехід керуючу позицію. Переходи типу J, F, T характеризуються єдиною змінною τ , а переходи типу X, Y співвідносяться з парою часових характеристик (τ_0, τ_1) . Вибір змінної τ_0 чи τ_1 відбувається в залежності від маркування керуючої позиції $m(\nu)$, одержаного в результаті процедури обробки:

$$\tau = \tau_i, \quad i \in \{ 0, 1 \},$$

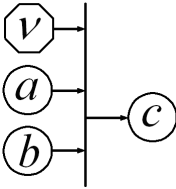
де i — значення керуючої мітки.

X — перехід.



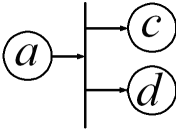
$$\begin{aligned}
 X(\nu, a, b; c, d) : \\
 (0, 1, -; 0, 0) &\rightarrow (\beta, 0, -; 1, 0) \\
 (0, 1, -; 0, 1) &\rightarrow (\beta, 0, -; 1, 1) \\
 (1, 1, -; 0, 0) &\rightarrow (\beta, 0, -; 0, 1) \\
 (1, 1, -; 1, 0) &\rightarrow (\beta, 0, -; 1, 1)
 \end{aligned}$$

Y — перехід



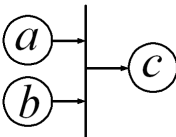
$$\begin{aligned}
 Y(\nu, a, b; c, d) : \\
 (0, 1, 1; 0, -) &\rightarrow (\beta, 0, 1; 1, -) \\
 (0, 1, 0; 0, -) &\rightarrow (\beta, 0, 0; 1, -) \\
 (0, 0, 1; 0, -) &\rightarrow (\beta, 0, 0; 1, -) \\
 (1, 1, 1; 0, -) &\rightarrow (\beta, 1, 0; 1, -) \\
 (1, 1, 0; 0, -) &\rightarrow (\beta, 0, 0; 1, -) \\
 (1, 0, 1; 0, -) &\rightarrow (\beta, 0, 0; 1, -)
 \end{aligned}$$

F — перехід



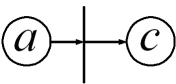
$$\begin{aligned}
 F(\nu, a, b; c, d) : \\
 (-, 1, -; 0, 0) &\rightarrow (-, 0, -; 1, -)
 \end{aligned}$$

J — перехід



$$\begin{aligned}
 J(\nu, a, b; c, d) : \\
 (-, 1, 1; 0, -) &\rightarrow (-, 0, 1; 1, -)
 \end{aligned}$$

T — перехід



$$\begin{aligned}
 T(\nu, a, b; c, d) : \\
 (-, 1, -; 0, -) &\rightarrow (-, 0, -; 1, -)
 \end{aligned}$$

Рис.2.8. Правила спрацювання основних типів переходів Е-мережі

На практиці існує кілька способів задавання змінної τ . Вона може приймати значення константи або бути функцією одного чи кількох атрибутів, що належать міткам на вхідних позиціях. Якщо мережа допускає стохастичні властивості об'єкта дослідження, то τ задають як випадкову змінну. Для збереження фізичного змісту змінна τ завжди повинна бути невід'ємною.

Якщо мітки переходів задають з певною кількістю атрибутів, то відповідні переходи повинні мати процедури типу e . Процедура e може мати пусту множину операцій перетворення значень атрибутів. В цьому випадку значення атрибутів без змін передають із вхідних позицій на вихідні позиції при спрацюванні переходу відповідно до правил, які задають функцією $S(f)$.

При спрацюванні переходу типу T мітка на позиції c одержує атрибути мітки, що повторюють атрибути мітки з позиції a . Спрацювання переходу F веде до дублювання атрибутів мітки з вхідної позиції a на вихідні позиції c і d . Для керованих переходів X і Y також можлива передача атрибутів в незмінному вигляді на вихідні позиції відповідно до правил спрацювання. Перехід типу J не може мати пусту процедуру e . При спрацюванні цього переходу формуються нові атрибути відповідно до правила:

$$\left(\pi_1 \rightarrow \left(a_{11}; a_{12}; \dots; a_{1n(1)} \right), \dots, \pi_k \rightarrow \left(a_{k1}; a_{k2}; \dots; a_{kn(k)} \right) \right),$$

де $k, n(i)$ — натуральні числа, $i = 1, 2, \dots, k$;

π_i — предикат;

a_{ij} — арифметичний оператор присвоєння вигляду:

$$m(c(j)) := g(x_1, x_2, \dots, x_q),$$

де $m(c(j))$ — маркування, яке відповідає значенню j -го атрибуту на позиції c ;

g — довільна арифметична функція з аргументами $x_i, i = \overline{1, q}$.

При $q = 0$ значення g дорівнює константі або випадковій величині.

Функціонування процедури переходу схоже на роботу процедури обробки. Послідовність операторів

$(a_{i1}; a_{i2}; \dots; a_{in(i)})$ завжди виконується за умови:

$$\begin{cases} \pi_i = \text{true} \\ \pi_j = \text{false} \end{cases} \text{ для всіх } i < j, 1 \leq i \leq k.$$

В протилежному випадку атрибуту нових міток не можуть бути визначені.

Згадані зміни відбуваються на елементах мережі шляхом спрацювання різних типів переходів. Динаміка цих змін реалізується за допомогою наступних етапів:

1. *Псевдоініціалізація* (тільки для переходів X і Y). Стан маркування всіх інформаційних позицій (a, b) відповідає лівій стороні правил переключення, які задає функція $S(f)$ (рис.2.8). Стан керуючої позиції ν не визначений. Суть даного етапу полягає у запуску процедури обробки з метою визначення параметрів позиції ν .

2. *Ініціалізація* (для всіх типів переходів). В цій фазі перехід може бути активізованим, якщо стан всіх вхідних і вихідних позицій задовольняє умови спрацювання, які задаються відповідними правилами.

3. *Робота* (для всіх типів переходів). Тривалість цього етапу задається значенням змінної τ , яке відповідає величині затримки. Правила роботи мережі потребують попереднього визначення цього параметра до початку фази роботи. Починаючи з першого етапу і закінчуючи етапом 3 маркування переходу залишається незмінним.

4. *Завершення* (для всіх типів переходів). На даному етапі відбувається знищення міток на вхідних позиціях і розміщення їх на вихідних позиціях відповідно до правил спрацювання. Робота процедури e спричиняє при цьому перетворення атрибутів.

Розглянута версія Е-мереж та численні її модифікації також мають ряд серйозних недоліків. Перш за все — це обмеження, які пов'язані з правилами задавання процедур переходу та керування. Проблема полягає у тому, що правила спрацювання переходів та умови формування атрибутів міток залежать тільки від маркування вхідних позицій. Такі обмеження спричиняють труднощі при розв'язанні задач синхронізації.

Значна кількість типів переходів, яка в даній версії Е-мереж фактично відповідає кількості правил спрацювання, також може значно ускладнити опис моделі. Цей факт істотно впливає на адекватність моделі, заданої Е-мережним описом властивостей об'єкта.

2.5.3. PRO-мережі

В одній з перших публікацій [44] PRO-мережі представлені у вигляді кортежу:

$$\Phi = (S, T, G, V), \quad (2.10)$$

де S — скінченна множина позицій; T — скінченна множина переходів; G — множина ребер; V — множина глобальних змінних.

Множина S містить набір бінарних елементів:

$$S = \left\{ (s_i, m_i, b_i) \right\}_{i=1}^{N_S},$$

де b_i — кількість атрибутів на позиції i ; s_i — номер позиції; m_i — максимально допустима кількість міток; N_S — кількість елементів множини S .

Множина переходів T задає параметри переходів PRO-мережі:

$$T = \left\{ (A_i, B_i, C_i, D_i, \rho_i, \tau_i, d_{\min_i}, d_{\max_i}, \pi_i) \right\}_{i=1}^{N_T},$$

де A_i — множина вхідних ребер з логікою спрацювання „AND”;

B_i — множина вхідних ребер з логікою спрацювання „SELECTOR”; C_i — множина вихідних ребер з логікою спрацювання „AND”; D_i — множина вихідних ребер з логікою спрацювання „SELECTOR”; ρ_i — процедура обробки переходу; τ_i — час спрацювання переходу; d_{\min_i} — мінімальний час затримки між запуском та активацією переходу; d_{\max_i} — максимальний час затримки між запуском та активацією переходу; π_i — процедура переходу.

Глобальні множини вхідних ребер $A = \bigcup_{i=1}^{N_T} A_i$ та

$B = \bigcup_{i=1}^{N_T} B_i$ зв'язані залежністю $A \cap B = \emptyset$, яка гарантує наяв-

ність хоча б одного ребра для кожного переходу. Логіка функціонування типу „SELECTOR” потребує наявності у переходу кількох вхідних ребер, згрупованих у підмножини $\{b_i\} \subseteq B$.

Одержані підмножини повинні задовольняти умову $\{b_i\} \cap \{b_j\} = \emptyset, i \neq j$. Вихідні ребра типу „SELECTOR”

також можуть групуватися у підмножини $\{d_i\} \subseteq D$ за умови

$$\{d_i\} \cap \{d_j\} = \emptyset, i \neq j.$$

Процедура переходу ρ починає працювати на етапі запуску переходу і може виконувати такі дії:

1. Доступ без модифікації до глобальних змінних та атрибутів міток, які розміщені на вхідних позиціях переходу.

2. Вибір однієї з підмножин вхідних ребер $\{b_i\} \subseteq B$.

3. Блокування міток на позиціях, що пов'язані з $\{b_i\}$.

4. Вибір процедури переходу $\pi \in \Pi$.

5. Запуск тесту активації, який визначає стан маркування вхідних позицій.

6. Запуск тесту деактивації, який визначає затримку спрацювання переходу до моменту виникнення сприятливих умов маркування вихідних позицій.

Процедура переходу π допускає виконання наступних дій:

1. Доступ до глобальних змінних та їх модифікація.
2. Зчитування атрибутів міток, які розміщуються на вхідних позиціях.
3. Модифікація атрибутів міток, які розміщуються на вихідних позиціях.
4. Вибір однієї з підмножин вихідних ребер $\{d_i\} \subseteq D$.
5. Обчислення виразу, який визначає час спрацювання переходу.

Структура шляхів переміщення інформації в моделі фіксується множиною ребер G , яка задає структуру зв'язків між позиціями та переходами:

$$G : S^\infty \times T^\infty \cup T^\infty \times S^\infty .$$

PRO-мережа використовує направлені ребра, які відповідають напрямку переміщення інформації.

Множина глобальних змінних $V = \{v_1, v_2, v_3\}$ містить три види підмножин, що відрізняються за функціональним призначенням. Підмножина v_1 містить глобальні змінні, які описують тимчасові характеристики моделі (стан глобального лічильника часу, час реакції різних ресурсів, доступних всім елементам мережної моделі тощо). Підмножина v_2 містить змінні службового характеру. Змінні, що входять в останню множину, визначають поточний стан тих елементів мережі, аналіз яких необхідно здійснювати в ході функціонування моделі. Підмножина v_3 включає глобальні змінні статистичного характеру, що базуються на показниках продуктивності:

$$v_3 = \{L, R, K, W\},$$

де L — множина ознак продуктивності; R — множина ознак реактивності; K — множина ознак використання; W — множина протоколів процесів.

Спрацювання переходу PRO-мережі складається з кількох етапів:

1. *Запуск*. Етап починається за таких умов:

- кожному із вхідних ребер типу A поставлена у відповідність одна мітка;
- для кожної підмножини ребер $\{b_i\} \subseteq B$ повинно існувати хоча б одне ребро з міткою на вхідній позиції;
- для кожного вихідного ребра типу C існує позиція, яка має місце для розташування мітки;
- хоча б для одного вихідного ребра з кожної підмножини $\{d_i\} \subseteq D$ існує місце для розташування вихідної мітки.

Аналіз згаданих умов реалізує процедура ρ , в процесі роботи якої можуть враховуватись стани внутрішніх локальних змінних переходу.

2. *Активация*. Етап починається в результаті виконання умов, які визначаються на етапі запуску. Спочатку відбувається затримка спрацювання переходу на величину Δ , яка знаходиться в межах допустимих значень: $d_{\min} \leq \Delta \leq d_{\max}$. Після закінчення даної затримки відбувається повторний контроль умов запуску. Якщо умови запуску збереглися, то запускається процедура переходу π . Ця процедура може також виконувати блокування станів вхідних і вихідних позицій, якщо зміна маркування в ході роботи процедури істотно впливає на хід обчислень.

3. *Деактивация*. Етап деактивації починається після закінчення періоду затримки τ , який задає процедура переходу. На цьому етапі виконуються такі основні дії:

- пересилання міток з новими атрибутами на вихідні позиції;
- зняття блокування з вхідних та вихідних позицій, якщо воно було введено процедурою переходу;
- модифікація локальних та глобальних змінних.

Створення PRO-мереж стало важливим внеском в теорію формальних мережних засобів. Основний продуктивний задум їх створення впливав із прагнення відійти від жорстких правил спрацювання переходів з метою одержання гнучких механізмів опису об'єктів моделювання. Але реалізація цієї ідеї не була доведена до логічного завершення. Фіксована логіка вхідних та вихідних ребер істотно знижувала гнучкість структури моделі, а недостатньо чітко виписані механізми взаємодії між об'єктами мережі потребували від розробника значних зусиль по їх доопрацюванню для кожної конкретної моделі. В результаті процес побудови PRO-мережної моделі залишався громіздким і не був універсальним. Саме тому зусилля дослідників були спрямовані на пошук засобів формального опису моделей, які могли б описувати структуру об'єкта моделювання у вигляді рівнянь, розв'язками яких були б варіанти поведінки моделі. З цією метою і було створено алгебри процесів.

2.6. Алгебри процесів

Алгебра процесів („процесне числення” або „теорія процесів”) — це метод формального опису складних паралельних систем, що базується на фундаментальних роботах [10-12] і забезпечує аналіз структури та поведінки системи. За останні роки досягнуто значного прогресу у застосуванні його до формального опису об'єктів при побудові стохастичних моделей та моделей з дискретними подіями. Популярність алгебр процесів полягає у вдалому поєднанні формальних підходів з можливістю їх опису за допомогою алгоритмічних мов високого рівня. Об'єкт моделювання розглядається як система з активними

компонентами, які взаємодіють між собою. Активні компоненти моделі називають агентами або процесами. Їх поведінка регулюється активностями. Будь-яка активність може бути внутрішньою відносно процесу або задавати взаємодію між сусідніми процесами. Не існує обмежень на одночасне виконання внутрішніх активностей різними процесами одночасно, а їх взаємодія регулюється семантикою розшарування. Операції, що входять в означення алгебри процесів, дозволяють конструювати процес, у якому визначена перша активність (префікс); альтернативний вибір (вибір); паралельне виконання операцій (композиція). Як правило, семантику операцій для алгебр процесів задають у стилі Плоткіна [45] з використанням системи типу $\left(S, M, \left\{ \xrightarrow{\mu} \mid \mu \in M \right\} \right)$, що враховує стани, переходи та маркування.

Алгебри процесів використовують для опису коректної поведінки систем як в аспекті відповідності до заданих специфікацій, так і в більш абстрактному розумінні. Можуть бути досліджені характеристики роботи окремих компонентів системи, а також параметри їх взаємодії з метою уникнення гонок та клінчових ситуацій.

2.6.1. Стохастичні алгебри процесів

Стохастичні алгебри процесів використовують у випадках, коли існує певна невизначеність щодо поведінки компонентів системи. Така невизначеність набуває кількісної оцінки при задаванні імовірності того чи іншого результату розвитку подій. Семантика операцій для стохастичних алгебр процесів дана в термінах імовірнісних маркованих систем переходів, тобто в таких маркованих системах, для яких імовірності асоційовані з переходами. Такі системи поділяють на *реактивні* та *генеруючі*. В реактивних системах імовірності переходів процесу можуть залежати від середовища, в якому він розвивається. В генерую-

чих системах імовірності не залежать від зовнішнього середовища. По суті, для реактивних систем розподіл імовірності визначений для тих нащадків даного процесу, які виникають при спрацюванні активності, а у випадку генеруючої системи розподіл імовірності визначений для можливих активностей процесу.

Розглянемо основні принципи організації стохастичних алгебр процесів на прикладі алгебри Performance Evaluation Process Algebra [46].

Модель P складної системи представлена у вигляді множини компонентів $P = \{X_i\}_{i=1}^N$. Довільний компонент X_i задають процесом, який може виконувати активність $a = \text{Act}(X_i)$, $a = (\alpha, r)$, де α — тип активності; $r \in R^+ \cup \rho$ — щільність імовірності довільного переходу, яка складається з множини додатних дійсних чисел R^+ та щільності імовірності спрацювання невизначеного переходу ρ . Життєвий цикл моделі, в залежності від розвитку подій, може включати різні варіанти процесу X_i , які об'єднують у множину $A(X_i)$. Компоненти моделі взаємодіють між собою за допомогою такого набору операцій:

$$(\alpha, r).X_i, X_i + X_j, X_i \triangleright_L \triangleleft X_j, X_i/H, \underline{\underline{\text{def}}}, \quad (2.12)$$

де L — множина типів взаємодії між компонентами моделі;

$H \subseteq A(X_i)$ — множина типів, які можуть бути замінені

в компоненті X_i за наявності невизначеного типу τ .

Означення 2.16. *Щільність імовірності $r_\alpha(X_i)$ спрацювання активності типу α в компоненті X_i для операцій (2.12) визначається залежностями:*

$$\text{Префіксація: } r_\alpha((\beta, r).X_i) = \begin{cases} r, & \alpha = \beta, \\ 0, & \alpha \neq \beta. \end{cases}$$

$$\text{Вибір: } r_\alpha(X_i + X_j) = r_\alpha(X_i) + r_\alpha(X_j).$$

Кооперація:

$$r_\alpha\left(X_i \underset{L}{\triangleright\triangleleft} X_j\right) = \begin{cases} r_\alpha(X_i) + r_\alpha(X_j), & \alpha \notin L, \\ \min(r_\alpha(X_i), r_\alpha(X_j)), & \alpha \in L. \end{cases}$$

$$\text{Приховання: } r_\alpha(X_i/H) = \begin{cases} r_\alpha(X_i), & \alpha \notin H, \\ 0, & \alpha \in H. \end{cases}$$

Означення 2.17. Якщо $X_i \xrightarrow{(\alpha, r)} X_{i+1}$, то X_{i+1} називають нащадком у першому поколінні від X_i . У загальному випадку, якщо існує послідовність $X_i \xrightarrow{(\alpha_i, r_i)} X_{i+1} \dots \xrightarrow{(\alpha_{n-1}, r_{n-1})} X_n$, то X_n називають нащадком X_i .

Означення 2.18. Множину нащадків компонента X_i позначають через $\text{ds}(X_i)$ і визначають як мінімальну множину компонентів, які задовольняють такі умови:

1. Якщо $X_j \stackrel{\text{def}}{=} X_i$, то $X_i \in \text{ds}(X_j)$,
2. Якщо $X_i \in \text{ds}(X)$ і існує активність $a \in \text{Act}(X_i)$

така, що $X_i \xrightarrow{a} X_j$, то $X_j \in \text{ds}(X)$.

Відповідно до синтаксичних правил компоненти поділяють на дві групи: модельні компоненти M і послідовні компоненти S :

$$S ::= (\alpha, r).S \mid S + S \mid X,$$

$$M:: = S \mid M \triangleright \triangleleft_L M \mid M/H,$$

де X — константа, яка відноситься до послідовних компонентів.

Означення 2.19. Компонент P називають реверсивним (здатним включати реверсивну пару $(\alpha, -\alpha)$), якщо $(\alpha, r) \in \text{Act}(P)$ і для кожного (α, r) -нащадка P' $(P \xrightarrow{(\alpha, r)} P')$ існує активність $(-\alpha, s) \in \text{Act}(P')$ така, що $P \in (-\alpha, s)$ -нащадком P' , де $r, s \in \mathbb{R}^+$.

Означення 2.20. Послідовний компонент P з початковим компонентом P_0 називають компонентом вводу – виводу, якщо

1. Для всіх $\alpha \in A(P_0)$ таких, що $P_0 \xrightarrow{(\alpha, r)} P'_0$ для деякого r , α формує частину реверсної пари $(\alpha, -\alpha)$;
2. Для всіх $P_i \in \text{ds}(P_0)$, для всіх $\alpha_i \in A(P_i)$ таких, що $P_i \xrightarrow{(\alpha_i, r)} P'_i$ для деякого r , α_i формує частину реверсної пари $(\alpha_i, -\alpha_i)$.

Відомі застосування стохастичних алгебр процесів для аналізу специфікацій [47], при проектуванні нових систем [48] та визначенні показників продуктивності [49]. Окремо варто розглянути застосування алгебр процесів для задавання моделей з дискретними подіями.

2.6.2. Алгебри процесів з дискретними подіями

Проблеми побудови моделей складних систем потребують нових підходів до створення формальних засобів опису об'єктів моделювання. Традиційний підхід, який полягає у використанні однієї з проблемно-орієнтованих мов програмування високого

рівня, не дає бажаного результату. Основна причина, на думку більшості розробників, полягає у тому, що моделі є функцією синтаксису та семантики вибраного інструменту моделювання. Як результат, модель іноді набуває властивостей, які не притаманні об'єкту моделювання. Застосування алгебри процесів — це один зі шляхів, який веде до підвищення рівня адекватності моделей. Ефект більшої гнучкості системи моделювання на основі алгебри процесів досягається за рахунок введення семантики на низькому рівні опису об'єкта.

В [50] запропоновано формальний опис алгебри процесів з дискретними подіями. Моделювання за допомогою даної алгебри базується на поняттях процесу та часу, що представлені змінними процесів $Pvar$, змінними часу $Tvar$, виразами процесів $Pexp$ та виразами часу $Texp$.

Означення 2.21. Множина виразів процесів, $Pexp$ — це мінімальна множина, яка включає

$$\begin{aligned} X \text{ nil} \quad (t).P \quad \alpha.P \quad \alpha[s \leftarrow t].P \quad \mathfrak{R}[s \leftarrow f].P \\ \sum_{i \in I} [q_i] P_i \quad P + Q \quad P|Q \quad P[S] \quad P \setminus A \quad rec(X = P) \end{aligned}$$

де $X \in Pvar$, $P, Q \in Pexp$, $s \in Tvar$, $t \in Texp$.

Множина активностей $Act = \Lambda \cup \{ \tau \}$ складається з елементів процесів $\alpha \in Act$, які відображають взаємодію з зовнішнім середовищем Λ та внутрішні дії $\{ \tau \}$; I — скінченна множина індексів; функція перемаркування $S : \Lambda \rightarrow \Lambda$, $A \subset \Lambda$;

q_i — імовірності в діапазоні $[0, 1]$, $\sum q_i = 1$, f — густина імовірності, $f \in R^+$.

Константа nil позначає процес, який завершився або такий, що знаходиться в стані клінчу. Часовий префіксний запис $(t).P$ відображає ситуацію, яка відповідає попередній затримці

на t одиниць модельного часу перед запуском процесу P . Операція $\alpha.P$ представляє процес з безпосереднім виконанням активності α . Якщо дана активність виконана, то в подальшому процес веде себе як процес P . Операція $\alpha[s \leftarrow t].P$ додатково задає заміну змінної s величиною t , яка вказує на величину попередньої затримки. Якщо значення t залежить тільки від внутрішніх параметрів процесу, то запис операції $\alpha[\] .P$ може бути спрощений. За необхідності параметричного виконання згаданої операції задають її функціональний варіант $\mathfrak{R}[s \leftarrow f].P$, який забезпечує повторний запуск процесу після спрацювання затримки на величину часу, яка є функцією f робочого навантаження.

Просування модельного часу відбувається за допомогою глобального годинника, значення поточного часу від якого передають всім процесам, які знаходяться в стані очікування синхронізації. Спрацювання активності веде до установки в локальну змінну величини глобального часу. Внутрішні активності процесу можуть коригувати це значення в залежності від поточного стану процесу.

Операція $P + Q$ задає недетермінований вибір між процесами P і Q у випадку, коли виключена можливість незалежного розвитку даних процесів. З використанням скінченної множини індексів операція набуває вигляду $\sum_{i \in I} P_i$. Якщо існує можливість задати ймовірнісні параметри вибору процесу з певної індексованої множини процесів, то операція вибору набуває вигляду $\sum_{i \in I} [q_i] P_i$, де q_i — ймовірність вибору процесу P_i в момент синхронізації.

Операцію $P|Q$ використовують для задавання паралельного розвитку процесів, які час від часу взаємодіють між собою.

Просування процесів у даному випадку можливе тільки після спільної для обох процесів активності, що забезпечує акт синхронізації.

$P[S]$ — операція перемаркування виконує зміну параметрів процесу в залежності від значення S .

$P \setminus A$ — дозволяє виключити з розвитку певну кількість активностей, які входять в множину виключення A .

Остання з наведеного переліку операція рекурсії $\text{rec}(X = P)$ застосовується для циклічного виконання процесів.

Алгебри процесів характеризуються високою компактністю представлення моделей. Адекватність моделей, у даному випадку, може бути строго доведена шляхом застосування теорем загальної теорії систем про строгу взаємну подібність. Недоліком такого підходу є низька наочність процесу моделювання, оскільки алгебра процесів не має прямої графічної інтерпретації виразів. Крім того, сучасні алгебри процесів загалом зорієнтовані на дослідження комунікаційних властивостей системи і не дають можливості забезпечити комплексний аналіз об'єкта моделювання, який враховує функціональні властивості компонентів.

РОЗДІЛ 3

APRO-мережі

Огляд формальних засобів опису моделей складних систем, що представлений у попередньому розділі, не є вичерпним. Мета його — показати основні тенденції і сучасні підходи до вирішування проблем формалізації. Існує безліч модифікацій згаданих підходів, кількість яких з кожним днем збільшується. Постає питання: навіщо потрібне таке велике розмаїття формальних засобів, кожен з яких претендує на певну універсальність? Єдина правильна відповідь на це питання полягає у тому, що така необхідність є наслідком необмеженої кількості складних систем, які неможливо укласти в „прокрустове ложе” обмеженого кола формальних засобів. В рамках такого підходу створено APRO-мережу (асинхронну PRO-мережу). Основні властивості APRO-мереж, які вирізняють їх з-поміж інших мереж:

- гнучкі правила визначення процедур переходів та атрибутів міток;
- широкі можливості побудови ієрархічних моделей (стратифікація);
- можливість моделювання обчислювальних структур, що працюють під управлінням операційних систем;
- модернізовані механізми моделювання асинхронної взаємодії.

Задамо структуру елементів даної мережі та принципи їх роботи.

3.1. Структура APRO-мережі

Представимо APRO-мережу у вигляді кортежу:







$$\Phi = (P, T, F, M, V), \quad (3.1)$$

де $P = \{p_i\}_{i=1}^n$ — скінченна множина позицій, $n \in N$;
 $T = \{t_j\}_{j=1}^m$ — скінченна множина переходів, $m \in N$;
 $F = (P \times T) \cup (T \times P)$ — множина ребер між переходами та позиціями;
 $M = \left\{ \left(p_k, \{ \mu_l \}_{l=1}^{\text{Max}_- p_k} \right) \right\}_{k=1}^n$ — скінченна множина маркувань;
 $V = (\Delta, \Psi, \Lambda)$ — множина глобальних змінних;
 N — простір натуральних чисел.

Графічне представлення APRO-мережі складається з графічних елементів, представлених в таблиці 3.1.

Таблиця 3.1

Графічні елементи APRO-мережі

Позначення		Графічне зображення	Назва
$p \in P$			Позиція
θ	τ		Простий перехід
	θ		Операторний перехід
	x_θ		Вхід операторного переходу
	x_θ		Вихід операторного переходу
$(p, t); (t, p) \in F$			Рebro
$\mu \in M$			Мітка

Позиції APRO-мережі:

$$p_i = \{d_i, q_i\}, \quad (3.2)$$

де $d_i = \{ \text{Id}_- p_i, \varphi_- p_i, \text{Cur}_- p_i, \text{Max}_- p_i, \delta_- p_i \}$ — параметри позиції, що включають:

Id_p_i — ідентифікатор позиції, φ_p_i — множину допустимих типів міток, Cur_p_i — поточну кількість міток на позиції, Max_p_i — максимально допустиму кількість міток, δ_p_i — локальний лічильник часу позиції, q_i — множина міток, розміщених на даній позиції.

На рис.3.1 показано APRO-мережу Φ_0 , що ілюструє параметри позицій. У загальному випадку Id_p_i — довільна алфавітно-цифрова послідовність, яка однозначно задає позицію p_i в рамках мережі Φ .

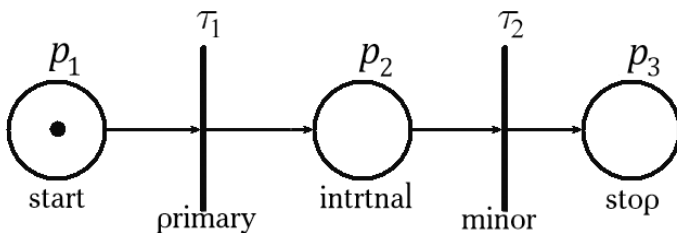


Рис.3.1. Приклад APRO-мережі Φ_0

На рис. 3.1 параметри

$$Id_p_1 := start, Id_p_2 := internal, Id_p_3 := stop$$

описують позиції за ознакою розміщення в мережі. Приклади формального задавання позицій $P = \{p_1, p_2, p_3\}$ в

рамках даної мережі: $p_1 = \{\{start, int, 1, 2, \delta_ \mu_1\}, \{\mu_1\}\}$,

$p_2 = \{\{internal, int, 0, 2, 0\}, \emptyset\}$, $p_3 = \{\{stop, int, 0, 2, 0\}, \emptyset\}$.

Крім ідентифікатора „start”, позиція p_1 характеризується також тим, що допускає розміщення міток типу „int”, тобто міток з атрибутами, представленими за допомогою цілих додатних чисел. Вона містить одну мітку μ_1 при максимально допус-

тимій кількості 2. Позиції p_2 та p_3 не містять міток. Тому параметри $\text{Cur_}p_2 := 0$, $q_2 := \emptyset$ і $\text{Cur_}p_3 := 0$, $q_3 := \emptyset$.

Локальний лічильник часу позиції дорівнює часу створення найстарішої мітки з тих, що розміщені на даній позиції. Тому $\delta_p_1 = \delta_p_1$, $\delta_p_2 = 0$ і $\delta_p_3 = 0$.

Переходи APRO-мережі включають два класи переходів: прості переходи τ та операторні переходи θ , сукупність яких утворює множину $\{t_j\}_{j=1}^m$, де $t = (\tau, \theta)$.

Простий перехід описує така сукупність параметрів:

$$\tau_j = \{ \chi_j, N_j \}, \quad (3.3)$$

де $\chi_j = \{ \text{Id_}\tau_j, \delta_ \tau_j, \Delta_ \tau_j, \text{Level_}\tau_j \}$ — параметри переходу, що включають: $\text{Id_}\tau_j$ — ідентифікатор переходу, $\delta_ \tau_j$ — локальний лічильник часу переходу, $\Delta_ \tau_j$ — період простою, $\text{Level_}\tau_j$ — рівень переходу;

N_j — функціональне ядро:

$$N_j = \{ \rho_j, \pi_j, \gamma_j, \omega_j, A_j, O_j \},$$

де ρ_j — процедура активації переходу;

π_j — процедура обслуговування переходу;

γ_j — процедура деактивації переходу;

ω_j — процедура очікування;

A_j — частково упорядкована послідовність активностей;

O_j — частково упорядкована послідовність вихідних міток.

Для наведення прикладів формального задавання простих переходів знову звернемося до мережі Φ_0 , що показана на

рис.3.1. Ця мережа містить прості переходи $T = \{\tau_1, \tau_2\}$, де $\tau_1 = \{\{\text{primary}, 0, 0, 0\}, N_1\}$, $\tau_2 = \{\{\text{minor}, 0, 0, 0\}, N_2\}$.

Ідентифікатори переходів $\text{Id}_{\tau_1} := \text{primary}$ і $\text{Id}_{\tau_2} := \text{minor}$ однозначно визначають їх в межах мережі Φ_0 . Локальні лічильники $\delta_{\tau_1} := 0$ і $\delta_{\tau_2} := 0$ свідчать, що мережа знаходиться в початковому стані. З цієї ж причини $\Delta_{\tau_1} := 0$ і $\Delta_{\tau_2} := 0$. Рівень переходів даної мережі 0, тобто мережа не містить переходів нижчих рівнів. Ядро N_1 містить процедури активації ρ_1 , обслуговування π_1 , деактивації γ_1 і очікування ω_1 , що реалізують алгоритми аналізу міток та обробки їх атрибутів. Ребра APRO-мережі (p_i, t_j) задають матрицю інцидентності H з елементами:

$$H(p_i, t_j) = \begin{cases} -1, & (p_i, t_j) \in \mathbf{F}, \\ +1, & (p_i, t_j) \in \mathbf{F}^{-1}, \\ 0, & (p_i, t_j) \notin \mathbf{F}, (p_i, t_j) \notin \mathbf{F}^{-1}, \end{cases} \quad \begin{matrix} 1 \leq i \leq n, \\ 1 \leq j \leq m. \end{matrix} \quad (3.4)$$

Матриця інцидентності $H_0(P, T)$ мережі Φ_0 , представленої на рис. 3.1, має вигляд:

$$H_0(P, T) = \begin{pmatrix} & p_1 & p_2 & p_3 \\ t_1 & -1 & 1 & 0 \\ t_2 & 0 & -1 & 1 \end{pmatrix}.$$

Для позиції p_i , використовуємо $\bullet p_i$ для позначення рге-множини, тобто такої підмножини переходів $\{t_j\}_{j=1}^m$, для якої завжди існують ребра від t_j до p_i при $H(p_i, t_j) > 0 \Big|_{j=1, m}$. Post-множина p_i^\bullet визначається подібним чином як підмножина

переходів $\{t_j\}_{j=1}^m$, для якої завжди існують ребра від p_i до t_j ,

$$H(p_i, t_j) < 0 \Big|_{j=1, \overline{m}}.$$

$$\bullet p_i = \{t_j \mid H(p_i, t_j) > 0, 1 \leq j \leq m\},$$

$$p_i^\bullet = \{t_j \mid H(p_i, t_j) < 0, 1 \leq j \leq m\}.$$

Для переходу t_j пре-множина визначається як підмножина множини $\{p_i\}_{i=1}^n$ при $H(p_i, t_j) < 0 \Big|_{i=1, \overline{n}}$ і відповідно пост-множина є підмножиною $\{p_i\}_{i=1}^n$ при $H(p_i, t_j) > 0 \Big|_{j=1, \overline{n}}$.

$$\bullet t_j = \{p_i \mid H(p_i, t_j) < 0, 1 \leq i \leq n\},$$

$$t_j^\bullet = \{p_i \mid H(p_i, t_j) > 0, 1 \leq i \leq n\}.$$

Для Φ_0 згадані множини мають вигляд:

$$\bullet p_1 = \emptyset, p_1^\bullet = \{t_1\} = \bullet p_2, p_2^\bullet = \{t_2\} = \bullet p_3, p_3^\bullet = \emptyset;$$

$$\bullet t_1 = \{p_1\}, t_1^\bullet = \{p_2\} = \bullet t_2, t_2^\bullet = \{p_3\}.$$

Клас операторних переходів θ є структурною підмножиною АPRO-мереж, яка забезпечує композиційне представлення:

$$\theta = (P_\theta, T_\theta, E_\theta, X_\theta, F_\theta), \quad (3.5)$$

де $P_\theta = \{(p_\theta)_1, \dots, (p_\theta)_a\}$, $a \in N$ — множина позицій;

$T_\theta = \{(\tau_\theta)_1, \dots, (\tau_\theta)_b\}$, $b \in N$, $T_\theta \neq \emptyset$ — непуста множина переходів;

$E_\theta = \{(e_\theta)_1, \dots, (e_\theta)_c\}$, $c \in N$, $E_\theta \neq \emptyset$ — непуста множина входів;

$X_\theta = \{ (x_\theta)_1, \dots, (x_\theta)_d \}$, $d \in N$, $X_\theta \neq \emptyset$ — непуста множина виходів;

$F_\theta = (P_\theta \times T_\theta) \cup (T_\theta \times P_\theta) \cup (E_\theta \times T_\theta) \cup (T_\theta \times X_\theta)$ — множина ребер.

Таким чином, операторний перехід θ APRO-мережі Φ вміщує APRO-мережу Φ_θ , що кваліфікується як APRO-мережа нижчого рівня. Переходи APRO-мережі Φ_θ , в свою чергу, також можуть бути операторними переходами, що дозволяє будувати агрегативні мережі з необмеженою кількістю рівнів. Для уникнення плутанини елементи операторного переходу будемо позначати індексом з його іменем.

Формальний опис внутрішніх позицій $(p_\theta)_i \in \Phi_\theta$ операторного переходу θ співпадає з формальним описом позицій верхнього рівня $p_i \in \Phi$. Внутрішні переходи $(\tau_\theta)_j \in \Phi_\theta$ також мають ідентичний формальний опис з простими переходами верхнього рівня $\tau_j \in \Phi$.

Входи операторного переходу:

$$(e_\theta)_j = \{ (\eta_\theta)_j, (N_\theta)_j, (q_\theta)_j \}, \quad (3.6)$$

де $(\eta_\theta)_j = \{ (\text{Id}_-e_\theta)_j, (\delta_-e_\theta)_j, (\Delta_-e_\theta)_j, (\text{Cur}_-e_\theta)_j,$

$(\text{Max}_-e_\theta)_j, (\text{Level}_-e_\theta)_j \}$ — параметри входу переходу θ ,

що включають: $(\text{Id}_-e_\theta)_j$ — ідентифікатор входу,

$(\delta_-e_\theta)_j$ — локальний лічильник часу, $(\Delta_-e_\theta)_j$ — період

простою, $(\text{Cur}_-e_\theta)_j$ — поточну кількість міток,

$(\text{Max}_-e_\theta)_j$ — максимально допустиму кількість міток,

$(\text{Level}_-e_\theta)_j$ — рівень входу;

ядро $(N_\theta)_j = \{ (\rho_\theta)_j, (w_\theta)_j \}$, де $(\rho_\theta)_j$ — процедура активації входу, $(w_\theta)_j$ — процедура очікування;

$(q_\theta)_i$ — множина міток, розміщених на вході операторного переходу θ .

Виходи операторного переходу:

$$(x_\theta)_i = \{ (O_\theta)_i, (N_\theta)_i, (q_\theta)_i \}, \quad (3.7)$$

де $(O_\theta)_i = \{ (\text{Id}_-x_\theta)_i, (\varphi_-x_\theta)_i, (\text{Cur}_-x_\theta)_i, (\text{Max}_-x_\theta)_i \}$ — параметри виходу переходу θ , що включають:

$(\text{Id}_-x_\theta)_i$ — ідентифікатор виходу, $(\varphi_-x_\theta)_i$ — множину допустимих типів міток, $(\text{Cur}_-x_\theta)_i$ — поточну кількість міток, $(\text{Max}_-x_\theta)_i$ — максимально допустиму кількість міток;

$(q_\theta)_i$ — множину міток, розміщених на виході операторного переходу θ ; ядро $(N_\theta)_i = \{ (\gamma_\theta)_i \}$, де $(\gamma_\theta)_i$ — процедура деактивації виходу.

Механізм композиційного представлення агрегативних APRO-мереж, які складаються з мереж на різних рівнях, покажемо на прикладі дворівневої APRO-мережі Φ_1 , зображеної на рис.3.2.

Мережа Φ_1 може розглядатись як імітаційна модель процесу заправки автомобіля на АЗС. Перехід *check* реалізує дії водія в залежності від оцінки стану паливного бака. Якщо автомобіль потребує заправки, то система переходить до стану *tank*, що свідчить про необхідність заїзду на АЗС. Вхідний елемент *input* відповідає процесу пошуку найближчої АЗС. Перехід *pay* описує дії водія по оплаті палива. Після цього система переходить до стану очікування вільної паливної коло-

нки. Перехід fill описує процес заправки автомобіля, який завершується станом output . Стан ready свідчить про готовність продовження поїздки. Перехід final завершує процес моделювання і переводить систему в стан stop . Якщо в результаті спрацювання переходу check приймається рішення про те, що автомобіль не потребує заправки, то система переходить в стан stright , який ініціює спрацювання переходу final .

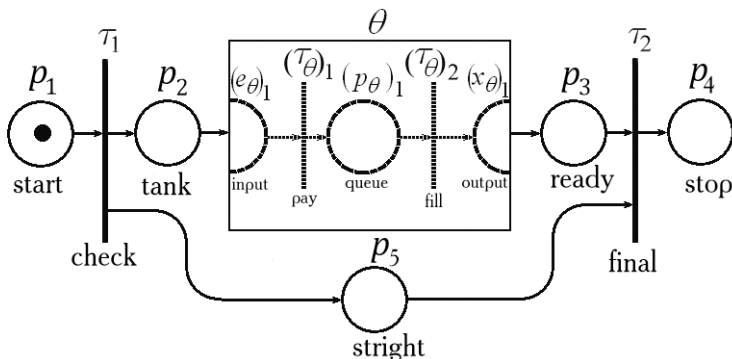


Рис.3.2. Приклад агрегативної APRO-мережі Φ_1

Структурно мережа Φ_1 містить такі елементи:

$$P = \{ p_1, p_2, p_3, p_4, p_5 \}, \quad T = \{ \tau_1, \tau_2, \theta \},$$

$$F = \begin{pmatrix} & p_1 & p_2 & p_3 & p_4 & p_5 \\ \tau_1 & -1 & 1 & 0 & 0 & 1 \\ \tau_2 & 0 & 0 & -1 & 1 & -1 \\ \theta & 0 & -1 & 1 & 0 & 0 \end{pmatrix},$$

$$P_\theta = \{ (p_\theta)_1 \}, \quad T_\theta = \{ (\tau_\theta)_1, (\tau_\theta)_2 \}, \quad E_\theta = \{ (e_\theta)_1 \},$$

$$X_\theta = \{ (x_\theta)_1 \}.$$

Елементи операторного переходу:

$$(e_\theta)_1 = \{ \{ \text{input}, 0, 0, 1 \}, \text{search}, \emptyset \},$$

$$(x_\theta)_1 = \{ \{ \text{output}, \text{car} \}, \emptyset \}.$$

$$(\tau_\theta)_1 = \left\{ \left\{ \text{pay}, 0, 0, 1 \right\}, N_1 \right\}, (\tau_\theta)_2 = \left\{ \left\{ \text{fill}, 0, 0, 1 \right\}, N_2 \right\}.$$

$$(p_\theta)_1 = \left\{ \left\{ \text{queue}, \text{car}, 0, 2, 0 \right\}, \emptyset \right\}.$$

Розглянемо правила формування рге-множин та post-множин на нижчих рівнях мережі, оскільки операторний перехід θ , відповідно до (3.5), визначається кортежем, який містить додатково компоненти E_θ і X_θ .

Кожний елемент e_θ множини входів E_θ поєднує властивості переходу по відношенню до вхідних даних з властивостями позиції по відношенню до вихідних даних. Тому рге-множина переходу t_j , представленого операторним переходом θ_j , є об'єднанням рге-множин входів e_{θ_j} :

$$\bullet t_j = \bullet \theta_j = \bullet E_{\theta_j} = \bigcup_{\theta_j} \bullet e_{\theta_j}. \quad (3.8)$$

В рамках мережі Φ_θ входи e_θ характеризуються post-множинами, елементи яких визначаються з умови:

$$(e_\theta)_i^\bullet = \left\{ (t_\theta)_j \mid H\left((e_\theta)_i, (t_\theta)_j\right) < 0, 1 \leq j \leq b \right\}. \quad (3.9)$$

Кожний елемент x_θ множини виходів X_θ поєднує властивості позиції по відношенню до вхідних даних з властивостями переходу по відношенню до вихідних даних. Тому рге-множина довільного виходу x_θ об'єднує всі з'єднані з ним переходи:

$$\bullet (x_\theta)_j = \left\{ (t_\theta)_j \mid H\left((x_\theta)_i, (t_\theta)_j\right) > 0, 1 \leq j \leq b \right\}. \quad (3.10)$$

Відповідно post-множина операторного переходу θ_j є підмножиною позицій $\{p_i\}_{i=1}^n$ при $H(p_i, \theta_j) > 0 \Big|_{k=1, m}$. Звідси випливає еквівалентність записів:

$$t_j^\bullet = \theta_j^\bullet = X_{\theta_j}^\bullet = \bigcup_{\theta_j} x_{\theta_j}^\bullet. \quad (3.11)$$

Означення 3.1. APRO-мережа називається *T-невиродженою*, якщо для всіх $t_j \in T$, $j = \overline{1, m}$, відповідні pre-множини і post-множини є непустими.

Означення 3.2. APRO-мережа називається *простою*, якщо для кожної позиції $p_i \in P$, $i = \overline{1, n}$ виконуються нерівності для потужностей множин: $|\bullet p_i| \leq 1$ і $|p_i \bullet| \leq 1$.

Означення 3.3. APRO-мережа називається *чистою*, якщо для всіх $t_j \in T$, $j = \overline{1, m}$ справедливе співвідношення $\bullet t_j \cap t_j \bullet = \emptyset$.

Мітки APRO-мережі:

$$\mu_k = \{ \lambda_k, \alpha_k \}, \tag{3.12}$$

де $\lambda_k = \{ \text{Id} _ \mu_k, \delta _ \mu_k, \varphi _ \mu_k \}$ — параметри мітки, що включають: $\text{Id} _ \mu_k$ — ідентифікатор мітки, $\delta _ \mu_k$ — час створення мітки, $\varphi _ \mu_k$ — тип мітки; α_k — множину атрибутів мітки.

Множина міток формує глобальне поточне маркування

$$M_\Phi = \left\{ \left(p_k, \left\{ \mu_l \right\}_{l=1}^{\text{Cur} _ p_k} \right) \right\}_{k=1}^n \quad \text{з поточною кількістю міток}$$

$M(p_i)$ на позиції p_i мережі Φ .

Множина глобальних змінних:

$$V = (\Delta, \Psi, \Lambda), \tag{3.13}$$

Δ — підмножина показників продуктивності, Ψ — підмножина показників реактивності, Λ — підмножина показників використання.

3.2. Принципи функціонування APRO-мережі

APRO-мережа може знаходитися в одному з наступних станів:

1. Немаркований стан.
2. Стан початкового маркування.
3. Стан послідовних кроків.
4. Стан кінцевого маркування.

Перехід APRO-мережі в немаркований стан відбувається в результаті виконання глобальної операції повного очищення: $\underline{\Phi} = \left[(P, T, F, 0, V) \right]$. Операція повного очищення застосовується до мережі в стані поточного маркування при $M_{\Phi} \neq \emptyset$. В результаті її виконання відбувається повне очищення міток з усіх позицій: $\underline{\Phi} : (P, T, F, M_{\Phi}, V) \rightarrow (P, T, F, 0, V)$, що супроводжується наступними діями:

1. Очищення параметра поточної кількості міток:

$$\text{Cur } _p_i := 0 \quad p_i \in P, 1 \leq i \leq n.$$

2. Очищення списків міток: $q_i := \text{nil}, 1 \leq i \leq n$.

3. Очищення ідентифікаторів процесів:

$$\text{Cur } _\tau_j = \text{'none'}, 1 \leq j \leq m.$$

Виконання операції повного очищення приводить APRO-мережу в статичний немаркований стан.

Позначивши початкове маркування і кінцеве глобальне маркування як ${}^{\circ}M$ і M° , задамо операції, які переводять APRO-мережу в стан відповідно початкового і кінцевого маркування:

$\overline{\Phi} : (P, T, F, M_{\Phi}, V) \rightarrow \overline{(P, T, F, {}^{\circ}M, V)}$ — початкове маркування; $\underline{\Phi} : (P, T, F, M_{\Phi}, V) \rightarrow \underline{(P, T, F, M^{\circ}, V)}$ — кінцеве маркування.

Операція початкового маркування $\overline{\Phi}$ забезпечує формування множини початкового маркування ${}^\circ M$, яка створюється на основі множини початкових позицій ${}^\circ \Phi$, тобто таких позицій, які визначені як позиції початкового стану моделі.

У випадку необхідності дострокового переривання процесу моделювання виконують операцію кінцевого маркування $\underline{\Phi}$. Ця операція полягає у формуванні множини кінцевого маркування M° на основі множини кінцевих позицій Φ° .

Розглянемо функціонування APRO-мережі, адаптувавши основні положення семантики послідовних кроків [51] до запропонованої версії мереж. Згадана семантика базується на використанні поняття *мультимножини*, яку формально визначають як пару (X, β) , де X — деяка множина; $\beta : X \rightarrow N$ — функція, яка задає відображення X на простір натуральних чисел N . Множину X називають *базовою множиною елементів*. Для кожного $x \in X$, *різноманіття*, тобто кількість входжень x , позначають $\beta(x)$, $\beta(x) \in N$.

Означення 3.4. *Кроком* будемо називати мультимножину простих переходів $U = (X, \beta)$ при $X \subseteq T$, $X = \{ \tau_j \}$, які активуються за умови, які для кожної позиції $p_i \in P$ виконується нерівність

$$M(p_i) \geq \left| \sum_{t_j \in X} H(p_i, \tau_j) \cdot \beta(\tau_j) \right|. \quad (3.14)$$

Pre-мультимножина кроку U об'єднує всі вхідні позиції простих переходів, що входять в даний крок:

$${}^\circ U = \left\{ (p_i, \varphi_i) \mid p_i \in \bullet \tau_j, \tau_j \in X, \varphi_i = \sum_{\tau_j \in X} \beta(\tau_j), \right. \\ \left. 1 \leq i \leq n, 1 \leq j \leq m \right\} \quad (3.15)$$

Post-мультимножина кроку U об'єднує всі вихідні позиції простих переходів, що входять в даний крок:

$$U \circ = \left\{ (p_i, \varphi_i) \mid p_i \in \tau_j^\bullet, \tau_j \in X, \varphi_i = \sum_{\tau_j \in X} \beta(\tau_j), \right. \\ \left. 1 \leq i \leq n, 1 \leq j \leq m \right\} \quad (3.16)$$

За умови активації і виконання умов спрацювання перехід τ_j може спрацювати, результатом чого є зміна кількості міток на пов'язаній з даним переходом позиції:

$$M'(p_i) := M(p_i) + H(p_i, \tau_j). \quad (3.17)$$

Відповідно до (3.4), якщо p_i є вхідною позицією переходу τ_j , то $H(p_i, \tau_j) = -1$, а якщо вихідною — то $H(p_i, \tau_j) = 1$. Отже, в результаті спрацювання переходу τ_j кількість міток на вхідній позиції зменшується на одиницю, а на вихідній позиції збільшується на одиницю.

В результаті спрацювання кроку U одержимо зміну маркування з $M(U)$ на $M'(U)$ відповідно до виразу:

$$M'(U) := M(U) + \sum_{p_i \in \circ U} H(p_i, \tau_j) + \sum_{p_i \in U \circ} H(p_i, \tau_j) \quad (3.18)$$

Таку зміну маркування в результаті спрацювання кроку U часто позначають як $M|U\rangle M'$. Тоді еволюція маркування σ під дією послідовності кроків $\sigma = U_1 \cdots U_k$ може бути записана:

$$M|\sigma\rangle M' = M|U_1\rangle M_1 \cdots M_{k-1}|U_k\rangle M'. \quad (3.19)$$

Означення 3.5. Для APRO-мережі Φ з маркуванням M маркування M_1 називають *безпосередньо досяжним* з M , якщо існує перехід t_j такий, що $M|t_j\rangle M_1$, і *кроково досяжним* з M при існуванні кроку U_1 такого, що $M|U_1\rangle M_1$.

Якщо існує дана послідовність маркувань, то вважають, що маркування M' *досягне* від маркування M . Пуста послідовність σ утворюється при $k = 0$ і призводить до виродженого випадку $M = M'$.

Позицію p_i мережі Φ з початковим маркуванням M_0 називають k -обмеженою при $k \in N$, $k = \text{Cnr } _ p_i$, якщо $k \leq \text{Max } _ p_i$ для всіх M , досяжних з M_0 . Оскільки дана послідовність маркувань відображає еволюцію APRO-мережі на одному рівні стратифікації, то, за умови використання в мережі k -обмежених позицій, завжди можливо побудувати послідовність спрацювань простих переходів, яка веде від початкового маркування до довільного досяжного маркування.

Означення 3.6. Множиною досяжності $D(\Phi, M)$ мережі Φ з маркуванням M будемо називати множину всіх маркувань, які є досяжними від маркування M і мають такі властивості:

1. $M \in D(\Phi, M)$;
2. Якщо $M_1 \in D(\Phi, M)$ і $M_1 | t_j \rangle M_2$ для деякого $t_j \in T$, то $M_2 \in D(\Phi, M)$.
3. Якщо $M_1 \in D(\Phi, M)$ і $M_1 | U_2 \rangle M_2$ для деякого $U_2 \subset T$, то $M_2 \in D(\Phi, M)$.

При використанні багаторівневих мереж правила формування кроку повинні також враховувати стани мереж нижчих рівнів. Для цього розглянемо складну APRO-мережу, яка утворена шляхом суперпозиції APRO-мереж:

$$\hat{\Phi} = \Phi_0 (\Phi_1 (\dots \Phi_r (\dots \Phi g))), \quad (3.20)$$

де $g \in N$, Φ_r — вкладена мережа, яка містить вкладені мережі, тобто використовує операторні переходи.

Означення 3.7. Складним кроком \hat{U} на мережі $\hat{\Phi}$ будемо називати суперпозицію мультимножин переходів

$$\hat{U} = U_0 \circ U_1 \circ \dots \circ U_r \circ \dots \circ U_g,$$

де $U_r = (T_r, \beta_r)$, $T_r = \left\{ (t_j)_r \right\}$, $j \leq |T_r|$, які активуються за умови, що для кожної послідовності позицій $\langle (p)_0, \dots (p)_r, \dots (p)_g \rangle$ справедлива послідовність нерівностей:

$$\begin{aligned} M((p)_0) &\geq \left| \sum_{(t_j)_0 \in T_0} H((p)_0, (t_j)_0) \cdot \beta((t_j)_0) \right|, \\ &\dots \dots \dots \\ M((p)_r) &\geq \left| \sum_{(t_j)_r \in T_r} H((p)_r, (t_j)_r) \cdot \beta((t_j)_r) \right|, \\ &\dots \dots \dots \\ M((p)_g) &\geq \left| \sum_{(t_j)_g \in T_g} H((p)_g, (t_j)_g) \cdot \beta((t_j)_g) \right|, \end{aligned}$$

де $(p)_r$ — довільна маркована позиція APRO-мережі на рівні r .

Pre-мультимножина складного кроку \hat{U} APRO-мережі $\hat{\Phi}$ включає вхідні позиції переходів різних рівнів, що входять в даний крок:

$$\begin{aligned} \circ\hat{U} &= \left\{ \bigcup_r ((p_i)_r, (\varphi_i)_r) \mid p_i \in \bullet(t_j)_r, (t_j)_r \in T_r, \right. \\ &(\varphi_i)_r = \sum_{(t_j) \in T_r} \beta((t_j)_r), \\ &\left. 0 \leq r \leq q, 1 \leq i \leq |P_r|, 1 \leq j \leq |T_r| \right\} \end{aligned} \quad (3.21)$$

Post-мультимножина складного кроку \hat{U} об'єднує всі вихідні позиції переходів, що входять в даний крок:

$$\begin{aligned} \hat{U}^\circ &= \left\{ \bigcup_r ((p_i)_r, (\varphi_i)_r) \mid p_i \in (t_j)_r^\bullet, (t_j)_r \in T_r, \right. \\ &(\varphi_i)_r = \sum_{(t_j) \in T_r} \beta((t_j)_r), \\ &\left. 0 \leq r \leq q, 1 \leq i \leq |P_r|, 1 \leq j \leq |T_r| \right\} \end{aligned} \quad (3.22)$$

Якщо спрацюють всі переходи, що входять до кроку \hat{U} , одержимо зміну маркування з $M(U)$ на $M'(U)$ відповідно до виразу:

$$M'(\hat{U}) := M(\hat{U}) + \sum_{p \in \circ\hat{U}} H(p, t) + \sum_{p \in \hat{U}\circ} H(p, t), \quad (3.23)$$

де $t \in \hat{U}$.

Зміну маркування в результаті спрацювання складного кроку \hat{U} будемо позначати $M|\hat{U}\rangle M'$. Представивши послідовність складних кроків $\hat{\sigma} = \hat{U}_0 \cdots \hat{U}_g$, визначимо еволюцію складної APRO-мережі $\hat{\Phi}$ у вигляді виразу:

$$M|\hat{\sigma}\rangle M' = M|\hat{U}_0\rangle M_1 \cdots M_{k-1}|\hat{U}_g\rangle M'. \quad (3.24)$$

Отже, складна APRO-мережа є формальним засобом опису складних дискретних систем, які характеризуються паралельним функціонуванням компонентів, пов'язаних не тільки однорівневими зв'язками, а й таких, що утворюють багаторівневі деревовидні структури.

3.2.1. Алгоритмічні аспекти функціонування APRO-мережі

Розглянемо основні аспекти динаміки APRO-мережі, які базуються на *активностях, процесах та подіях*. В даному випадку активності представляють сукупність дій, які відповідають типовим діям переходу на заданому рівні опису APRO-мережі. За умови одночасного використання операторних та простих переходів мережа міститиме активності верхнього рівня та внутрішні активності операторних переходів, утворюючи в такий спосіб дворівневу структуру активностей. Якщо операторний перехід представлений мережею, яка також містить операторні переходи, то кількість рівнів активностей збільшується на одиницю. Таким чином, утворюється деревовидна структура активностей. Логічна послідовність активностей об'єднується в процес. Кожний з процесів, які існують в мережі, може входити в процес вищого рівня як підпроцес. Представимо множину активностей як $A = \{ \text{compute, activate, deactivate, idle} \}$.

Активність `compute` описує сукупність дій, які пов'язані з обробкою інформації під час роботи переходу. Активність `activate` активує перехід, а активність `deactivate` деактивує його. Активність `idle` відображає дії переходу в режимі очікування. Упорядкована послідовність даних активностей може утворювати процес.

Означення 3.8. Нехай дана довільна множина A і три її довільних елементи $a_1, a_2, a_3 \in A$. Множину A будемо називати *частково впорядкованою послідовністю*, якщо для її елементів задано відношення " \preceq " (" $a_1 \preceq a_2$ " — a_1 передує або дорівнює a_2), що відповідає властивостям:

- якщо $a_1 \preceq a_2$ і $a_2 \preceq a_3$, то $a_1 \preceq a_3$ (транзитивність);
- якщо $a_1 \preceq a_2$ і $a_2 \preceq a_1$, то $a_1 = a_2$ (асиметричність);
- $a_1 \preceq a_1$ (рефлексивність).

Наприклад, запис

$$\text{process := activate } \preceq \text{ compute } \preceq \text{ deactivate } \preceq \text{ idle}$$

представляє типовий процес роботи переходу, який спочатку активується, виконує обробку інформації, а потім деактивується і переходить в режим очікування.

Народження та зникнення активностей відбувається шляхом звершення подій, кожна з яких представляє собою миттєву зміну стану APRO-мережі.

В таблиці 3.2 наведено структуру абстрактних об'єктів, які описують динаміку APRO-мережі. З таблиці видно, що активація кожної активності відбувається за допомогою події `start`, а деактивацію активності виконує відповідна подія `stop`.

Таблиця 3.2

Абстрактні об'єкти APRO-мережі

ПРОЦЕС	АКТИВНІСТЬ	ПОДІЯ
process	activate	start
		stop
	compute	start
		stop
	deactivate	start
		stop
	idle	start
		stop

Приклад процесу, що представлений вище, може бути записаний на рівні подій у наступному вигляді:

process := idle.stop \preceq activate.start \preceq activate.stop \preceq
 compute.start \preceq compute.stop \preceq deactivate.start \preceq
 deactivate.stop \preceq idle.start.

З множиною переходів $T = \{ t_1, \dots, t_j, \dots, t_m \}$ співставимо множину процесів $\mathbf{Pr} = \{ Pr_1, \dots, Pr_j, \dots, Pr_m \}$, кожен з яких може містити певну кількість підпроцесів:

$$Pr_j = \{ Pr_{j_1} Pr_{j_2}, \dots, Pr_{j_k} \},$$

де k -ступінь розпаралелювання переходу t_j .

Довільний підпроцес Pr_{j_k} представлений *частково упорядкованою послідовністю* тих активностей, що належать даному підпроцесу.

Розглянемо випадок, коли робота переходу t_j описується єдиним процесом Pr_j . Тоді він може бути представлений, на-

приклад, такою частково упорядкованою послідовністю активностей, що належать множині A :

$$\text{Pr}_j = \text{activate}_j \preceq \text{compute}_j \preceq \text{deactivate}_j \preceq \text{idle}_j.$$

Сукупний процес Pr_j завжди однозначно задає поточний стан відповідного переходу t_j . Тому загальною ознакою упорядкування послідовності Pr_j є час виникнення активностей, які входять в дану послідовність.

Функціонування довільного переходу t_j АPRO-мережі представлено списком подій, упорядкованим відповідно до часу створення. Кожна з подій відображає факт виникнення або зникнення тієї чи іншої активності. Очевидно, що такий список відповідає процесу Pr_j , заданому на рівні подій. На рис.3.3 зображено структуру простого переходу t_j та елементів його множин $\bullet\tau_j$ і τ_j^\bullet .

Список подій EvList_j переходу t_j містить назву типу активності, тип події, локальний час її виникнення δ , активну мітку μ та посилання key на найближчий в часі елемент упорядкованого списку.

Наступна подія формується під час дії поточної активності і розміщується в EvList_j відповідно до часу створення.

Після звершення поточної події виконують операцію $k_0 := k_0.\text{key}$, де $k_0.\text{key} = k_1$. Обслуговування голови списку k_0 починається з модифікації локального годинника переходу $\delta - \tau_j$.

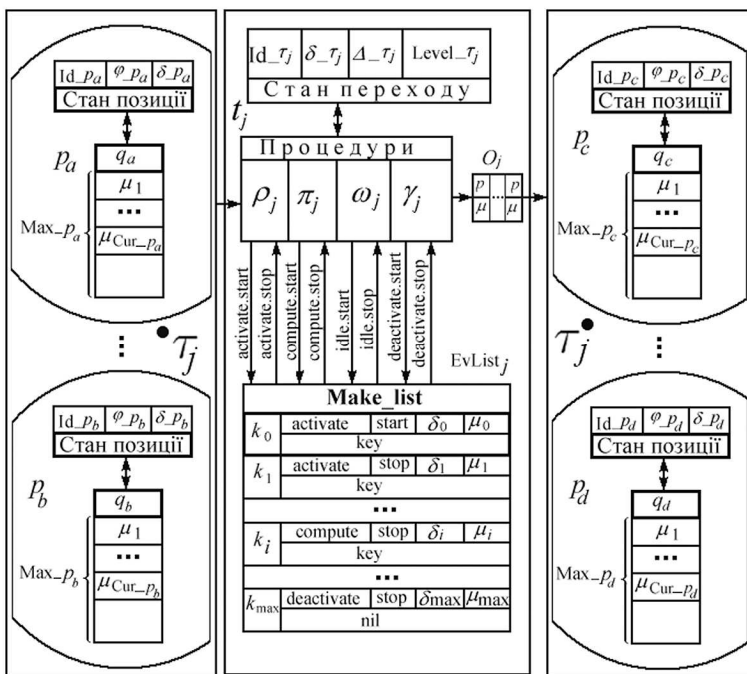


Рис. 3.3. Структура елементів АPRO-мережі

Відповідно до типу активності та типу події відбувається запуск процедури переходу.

Таблиця 3.3 демонструє відповідність між запусками процедур переходу та активностями:

Таблиця 3.3

Відповідність між процедурами переходу та активностями

№ п/п	Активність	Подія	Процедура
1	activate	start	ρ
2	compute	start	π
3	deactivate	start	γ
4	idle	start	ω

Послідовний розвиток процесу Pr_j відбувається за рахунок того, що звершення поточної події супроводжується ство-

ренням нової події з новим часом виконання. Оскільки робота переходу описується єдиним процесом Pr_j , то кожній активності даного переходу поставимо у відповідність його стан.

Позначимо допустимі стани переходу t_j , які формуються під дією вхідної мітки μ_{in} , через $\left(\lfloor t_j(\mu_{in}) \rfloor, \lceil t_j(\mu_{in}) \rceil \right)$, де

$\lfloor t_j(\mu_{in}) \rfloor$ — перехід в стані активації (активність activate);
 $\lceil t_j(\mu_{in}) \rceil$ — перехід в стані обробки (активність compute).

Стану деактивації переходу $\lfloor t_j(\mu_{out}) \rfloor$, що формується під дією вихідної мітки μ_{out} , відповідає активність deactivate, а стан очікування $\lceil t_j \rceil$ і відповідна активність idle не пов'язані з активними мітками переходу. Залежність станів переходу від активних міток вказує на той факт, що кожен зі згаданих станів розглядається відносно відповідної активної мітки. Структура логічних зв'язків процедур $\rho_j(\mu_{in})$, $\pi_j(\mu_{in})$, $\gamma_j(\mu_{out})$ і ω_j показана на рис.3.4.

Процедура активації $\rho_j(\mu_{in})$ забезпечує виконання комплексу операцій, які відповідають стану $\lfloor t_j(\mu_{in}) \rfloor$ переходу t_j , і складається з процедур **Choose**, **Select** і **Timing**. Процедура **Choose** призначена для виконання дій, пов'язаних з аналізом міток, які розміщені на вхідних позиціях переходу t_j .

Процедура **Select** реалізує алгоритм перевірки тієї мітки, яка вибрана за допомогою процедури **Choose**. Однією з важливих умов перевірки є збереження часової цілісності моделі шляхом порівняння локального часу переходу та часу створення мітки. Цю функцію виконує окрема процедура **Timing**.

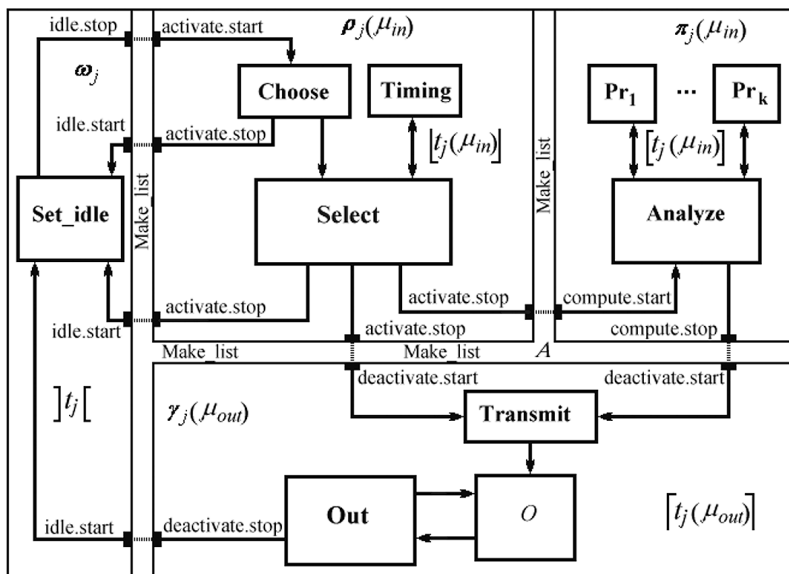


Рис.3.4. Структура процедур $\rho_j(\mu_{in})$, $\pi_j(\mu_{in})$, $\gamma_j(\mu_{out})$ і ω_j

Запуск процедури $\rho_j(\mu_{in})$ відбувається в момент, коли перехід t_j змінює свій стан з $]t_j[$ на $]t_j(\mu_{in})]$. Механізм цієї зміни полягає у тому, що завершальна подія `idle.stop` активності `idle` породжує стартову подію `activate.start` активності `activate`.

Існує чотири можливих завершення процедури $\rho_j(\mu_{in})$ в залежності від результатів аналізу мітки в стані $]t_j(\mu_{in})]$:

1. Якщо пошук активної мітки на вхідних позиціях завершився безрезультатно, то подія завершення активності `activate` породжує подію `idle.start`.
2. Якщо активна мітка μ_{in} знайдена процедурою **Choose**, але значення її атрибутів не дозволяють запустити процедуру

обробки $\pi(\mu_{in})$, то подія завершення активності `activate` також породжує подію `idle.start`.

3. Якщо активна мітка μ_{in} , яка знайдена процедурою **Choose**, має коректні параметри, то подія завершення активності `activate` породжує подію `compute.start`.

4. Якщо активна мітка μ_{in} , яка знайдена процедурою **Choose**, є транзитною міткою, то подія завершення активності `activate` породжує подію `deactivate.start`.

Процедура $\pi_j(\mu_{in})$ реалізує сукупність операцій, що відповідають стану переходу $[t_j(\mu_{in})]$, і складається з процедури **Analyze** та множини процедур $\{\mathbf{Pr}_i(\alpha) \mid \alpha \in \mu\}_{i=1}^k$. Запуск цієї процедури відбувається у випадку, коли мітка є коректною і повинна бути оброблена в даному переході, що підтверджується створенням події `compute.start`. Процедура **Analyze** забезпечує оцінку параметра $\varphi_{\mu_{in}}$ поточної мітки μ_{in} з метою вибору відповідної процедури $\mathbf{Pr}_k(\alpha) \mid \alpha \in \mu_{in}$. Результатом роботи процедури є створення події `compute.stop`, яка під час завершення породжує подію `deactivate.start`.

Процедура $\gamma_j(\mu_{out})$ включає операції, які супроводжують стан переходу $[t_j(\mu_{out})]$, і складається з процедур **Transmit** та **Out**. Існують два варіанти запуску процедури $\gamma_j(\mu_{out})$, кожен з яких ініціюється відповідною подією `deactivate.start`.

Процедура **Transmit** формує параметри та атрибути вихідної мітки і розміщує їх в черзі вихідних міток за допомогою процедури **Set_{O_j}**. Процедура **Get_{O_j}** вибирає першу за чергою вихідну мітку μ_{out} , а процедура **Out** намагається вста-

новити її на відповідну вихідну позицію. Якщо мітка з тих чи інших причин не встановлена на вказаній вихідній позиції, то вона знову повертається в чергу за допомогою процедури **Set_** O_j . За умови пустоти черги вихідних міток O_j робота процедури $\gamma_j(\mu_{out})$ закінчується, про що свідчить подія `deactivate.stop`, звершення якої породжує подію `idle.start`.

Відповідно до рис.3.4 існує три можливих причини виникнення події `idle.start`, яка запускає процедуру очікування ω_j , що встановлює стан очікування t_j . У цьому стані спрацьовує процедура **Set_idle**, яка визначає дії переходу після завершення повного циклу обробки інформації.

З короткого опису роботи даних процедур видно, що в APRO-мережі кожній процедурі переходу відповідає активність. Керування частково упорядкованою послідовністю активностей в рамках переходу виконує процедура обслуговування послідовності A_j **Make_list**, яка є головною процедурою ядра N_j переходу t_j .

Procedure Make_list(new_action);

begin

Case new_action of

 idle:

Case old_action of

 activate : **begin**

 Set_event (δ_{τ_j} , activate.stop);

 Set_event ($\delta_{\tau_j} + \Delta_{activate}$, idle.start);

end;

 deactivate : **begin**

 Set_event (δ_{τ_j} , deactivate.stop);

```

        Set_event ( $\delta\_ \tau_j + \Delta_{\text{deactivate}}$ , idle.start);
    end;
end;
activate : begin
    Set_event ( $\delta\_ \tau_j$ , idle.stop);
    Set_event ( $\delta\_ \tau_j + \Delta_{\text{idle}}$ , activate.start);
end;
compute : begin
    Set_event ( $\delta\_ \tau_j$ , activate.stop);
    Set_event ( $\delta\_ \tau_j + \Delta_{\text{activate}}$ , compute.start);
end;
deactivate:
Case old_action of
    activate : begin
        Set_event ( $\delta\_ \tau_j$ , activate.stop);
        Set_event ( $\delta\_ \tau_j + \Delta_{\text{activate}}$ , deactivate.start);
    end;
    compute : begin
        Set_event ( $\delta\_ \tau_j$ , compute.stop);
        Set_event ( $\delta\_ \tau_j + \Delta_{\text{comput}}$ , deactivate.start);
    end;
end;
end;
old_action:=new_action;
end.

```

Процедура **Set_event** забезпечує дії по створенню події та вмонтування її в частково впорядковану послідовність A_j .

Procedue Set_event(event_time, event_name);

begin

```

For  $i:=1$  to  $\max$  do
begin
  If  $\delta_{i-1} \leq \text{event\_time} < \delta_i$  then
    Insert( $i$ ,  $\text{event\_name}$ ,  $\text{event\_time}$ ,  $\mu$ );
  If ( $\delta_i = \text{event\_time}$ ) & ( $i < \max$ ) then
    Insert( $i+1$ ,  $\text{event\_name}$ ,  $\text{event\_time}$ ,  $\mu$ );
end;
end.

```

Розглянемо детальніше стани переходу APRO-мережі.

3.2.1.1. Активація переходу

Активація відбувається тільки тоді, коли виконуються умови для запуску активації. Перевірку цих умов для переходу t_j реалізує процедура активації $\rho_j(\mu_{in})$ за допомогою внутрішньої процедури **Choose**. Джерелом виникнення станів активації для переходу t_j служать мітки, що можуть бути розміщені на позиціях, які є елементами рге-множини $\bullet t_j$. Умови активації поділяють на необхідні та достатні. Необхідна умова включає наявність хоча б однієї вхідної позиції і хоча б однієї мітки на цій позиції. Формально кількість вхідних позицій визначимо, як потужність рге-множини $\bullet t_j$. Отже, запис $|\bullet t_j| \geq 1$ вказує на те, що перехід t_j має зв'язок з однією вхідною позицією або більшою кількістю вхідних позицій. Оскільки $p_i = (\bullet t_j)_i$, а одним з елементів множини p_i є поточна кількість міток $\text{Cur_}p_i$, то друга необхідна умова активації простого переходу $\sum_{i=1}^{|\bullet t_j|} \text{Cur_}p_i \geq 1$ означає, що на вхідних позиціях переходу t_j

має бути розміщена хоча б одна мітка. Існують деякі відмінності при задаванні достатніх умов активації в залежності від того, чи є даний перехід простим, чи операторним. Для простого переходу об'єктом аналізу є множина $\bullet\tau_j$, а для операторного — множина $\bullet(e_\theta)_j$. Достатня умова активації переходу τ_j полягає в існуванні вільного ресурсу переходу для обробки мітки. Основним критерієм величини згаданого ресурсу будемо вважати максимально допустиму довжину частково впорядкованої послідовності A_j . Тому достатній критерій спрацювання простого переходу $|A_j| \leq \text{Max}$, де Max — допустима для даної мережі величина потужності множини A_j . Оскільки елемент $(e_\theta)_j$ за своєю природою є вхідним буфером операторного переходу θ і поєднує властивості переходу по відношенню до вхідних позицій з властивостями позицій по відношенню до вихідних переходів, то в такому елементі не існує ресурсів обробки інформації, і достатня умова його активації обмежується перевіркою допустимої ємності.

Для простої АPRO-мережі необхідна умова активації всіх переходів кроку U :

$$|U| = \left| \left\{ \left[\tau_j \right] \mid \left| \bullet\tau_j \right| \geq 1, \sum_{i=1}^{|\bullet\tau_j|} \text{Cur}_- p_i \geq 1 \right\}_{\tau_j \in U} \right|. \quad (3.26)$$

Достатня умова активації переходів кроку U :

$$|U| = \left| \left\{ \left[\tau_j \right] \mid |A_j| \leq \text{Max} \right\}_{\tau_j \in U} \right|. \quad (3.27)$$

Для АPRO-мережі, яка містить операторні переходи, розглянемо умови активації складного кроку \hat{U} . Як уже згадувалось, достатні умови активації простих і операторних переходів співпадають. Тому

$$|\hat{U}| = \left| \left\{ \left[t_j \right] \mid \left| \bullet t_j \right| \geq 1, \sum_{i=1}^{\left[t_j \right]} \text{Cur}_- p_i \geq 1 \right\}_{t_j \in \hat{U}} \right|. \quad (3.28)$$

Достатня умова активації складного кроку об'єднує достатні умови активації простих переходів та відповідних входів операторних переходів:

$$|\hat{U}| = \left| \left\{ \left[\tau_j \right] \mid \left| A_j \right| \leq \text{Max} \right\}_{\tau_j \in \hat{U}} \cup \left\{ \left[(e_\theta)_j \right] \mid (\text{Cur}_- e_\theta)_j < (\text{Max}_- e_\theta)_j \right\} \right| \quad (3.29)$$

Пошук мітки на вхідній позиції виконує процедура **Choose**($\bullet t_j$).

Procedure Choose ($\bullet t_j$);

begin

For $i:=1$ **to** $\left| \bullet t_j \right|$ **do**

begin

$p := \left(\bullet t_j \right)_i$;

If $\text{Cur}_- p > 0$ **then**

begin

$\mu_{in} := \text{Find_min}(q)$;

Break;

end;

end;

If **Assigned**(μ_{in}) **then** **Select**(μ_{in}) **else** **Make_list**(idle);

end.

В ході виконання цієї процедури відбувається почерговий перегляд вхідних позицій $\left(\bullet t_j \right)_i$, які є елементами пре-

мультимножини $\bullet t_j$ переходу t_j . Для кожної поточної позиції $p := \left(\bullet t_j \right)_i$ виконують перевірку параметра Cur_p , який визначає кількість міток, розміщених на даній позиції. Якщо на позиції розміщена хоча б одна мітка, то за допомогою функції $\mu_{in} := \text{Find_min}(q)$ в частково упорядкованій множині q знаходять мітку, яка характеризується мінімальним часом створення. Після цього цикл пошуку переривають за допомогою оператора **Break** і продовжують аналіз коректності параметрів мітки μ_{in} за допомогою процедури **Select** $\left(\mu_{in}\right)$. Якщо результат пошуку мітки μ в процедурі **Choose** $\left(\bullet t_j\right)$ є негативним, то наступним станом переходу t_j буде стан $\left] t_j \left[$. Для переходу в цей стан необхідно спершу створити подію завершення активності активації `activate.stop`. Встановлення стану $\left] t_j \left[$ відбудеться тільки після звершення події `idle.start`. Керування списком подій виконують за допомогою процедури **Make_list**.

Процедура перевірки коректності параметрів мітки **Select** (μ) є складовою процедури активації $\rho_j(\mu_{in})$ і полягає в аналізі параметрів мітки μ_{in} , яка була вибрана процедурою **Choose** $\left(\bullet t_j\right)$.

```

Procedure Select $\left(\mu_{in}\right)$ ;
begin
  If  $\text{Id}_{\mu_{in}} \neq \text{Id}_{\tau_j}$  then
    begin
      Make_list(deactivate);
    
```



```

Erase( $p, \mu_{in}$ );
 $\mu_{min} := \mathbf{Find\_min}(q)$ ;
If Assigned( $\mu_{min}$ ) then  $\delta\_p := \delta\_{\mu_{min}}$ ;
end else
begin
  If Timing( $\mu_{in}$ ) then
    begin
      Make_list(compute);
      Erase( $p, \mu_{in}$ );
      If Assigned( $\mu_{min}$ ) then  $\delta\_p := \delta\_{\mu_{min}}$ ;
      end else Make_list(idle);
    end;
  end.

```

Якщо ідентифікатор мітки $\text{Id}_{\mu_{in}}$ співпадає з ідентифікатором переходу Id_{τ_j} , то цей факт свідчить про те, що мітка μ_{in} призначена для обробки в переході τ_j . В протилежному випадку мітка μ_{in} кваліфікується як транзитна мітка μ_{out} , тобто така, що не потребує обробки в даному переході і повинна бути передана на вихідні позиції переходу t_j без змін. Для виконання цих дій необхідно спочатку завершити активність активації переходу `activate` за допомогою події `activate.stop`. Наступною дією процедури **Make_list** є створення активності деактивації `deactivate` за допомогою події `deactivate.start`, оскільки саме ця активність відповідає стану $[t_j(\mu_{out})]$, який здійснює переміщення міток на вихідні позиції. Після сформування події `deactivate.start` мітку μ_{in} можна вилучити із вхідної позиції p

за допомогою процедури **Erase** (p, μ_{in}) . Умова $\text{Id}_{-\mu} = \text{Id}_{-\tau_j}$ веде до запуску функції аналізу мережного часу **Timing** (μ_{in}) . Якщо перевірка на часову коректність пройшла успішно, то така мітка є повністю коректною, і за наявності відповідного ресурсу ініціює створення події `compute.start` для виконання дій переходом t_j , які відповідають стану $[t_j(\mu_{in})]$. В цьому випадку також необхідно виконати процедуру **Erase** (p, μ_{in}) вилучення відповідної вхідної мітки з позиції p . Якщо перевірка часової коректності мітки дала негативний результат, то наступний стан переходу буде $]t_j[$, що реалізує процедура **Make_list**(idle).

3.2.1.2. Робота переходу

Процедура активації переходу $\rho_j(\mu_{in})$ в результаті аналізу міток на вхідних позиціях визначає коректну мітку і створює активність `compute` за допомогою процедури **Make_list**(`compute`). В момент свого звершення подія `compute.start` запускає процедуру обслуговування переходу $\pi_j(\mu_{in})$. Ця процедура складається з процедури аналізу типів та процедур локальних процесів:

$$\pi_j(\mu_{in}) = (\mathbf{Analyze}(\mu_{in}), \text{pr}_1(\alpha_{in}), \dots, \text{pr}_k(\alpha_{in})).$$

Процедура **Analyze** (μ_{in}) виконує функцію селектора процесів переходу τ_j .

Procedure Analyze (μ_{in}) ;

begin

Case $\varphi_{-\mu_{in}}$ **of**

```

1 : pr1 ( αin );
...
i : pri ( αin );
...
k : prk ( αin );
end ;
Make_list (deactivate);
end.

```

Процедури $pr_1(\alpha_{in}), \dots, pr_i(\alpha_{in}), \dots, pr_k(\alpha_{in})$ реалізують процеси обробки робочого навантаження, представленого атрибутами поточної мітки $\alpha_{in} \in \mu_{in}$. Наприклад, при побудові моделі цифрової обробки сигналів згадані процедури реалізують алгоритм швидкого перетворення Фур'є, а при розв'язуванні крайових задач математичної фізики — ітераційні формули у вузлах сітки дискретизації. Після відпрацювання довільної процедури $pr_i(\alpha_{in})$ завжди виконується процедура **Make_list**(deactivate), що запускає етап деактивації переходу.

3.2.1.3. Деактивація переходу

Деактивація — завершальний етап активності переходу. Цей етап, як правило, супроводжується розміщенням міток на його вихідних позиціях. Розміщення довільної мітки на вихідній позиції потребує виконання таких умов:

1. Мітка повинна бути „дозрілою”, тобто процедура **Analyze**(μ_{in}), яка є керуючим елементом процедури $\pi_j(\mu_{in})$, повинна успішно завершити роботу. Результатом успішного її завершення є формування вихідної мітки μ_{out} з атрибутами α_{out} та параметрами $Id_{\mu_{out}}, \varphi_{\mu_{out}}, \delta_{\mu_{out}}$.

2. Кількість міток на цільовій вихідній позиції p_i не повинна дорівнювати або перевищувати максимально допустиму кількість Max_{p_i} .

3. Тип мітки $\varphi_{\mu_{out}}$ повинен входити в множину допустимих типів міток для даної позиції: $\varphi_{\mu_{out}} \in \varphi_{p_i}$.

Ці умови є базовими для побудови алгоритму роботи процедури переходу $\gamma(\mu_{out})$, яка складається з процедур **Transmit**, **Set $_O_j$** , **Get $_O_j$** та **Out**.

Існує три варіанти роботи алгоритму процедури **Transmit** в залежності від змісту параметра $\text{Id}_{\mu_{out}}$:

1. Параметр $\text{Id}_{\mu_{out}} = 0$, який вказує на той факт, що в ході формування параметрів вихідної мітки не було задано цільовий перехід.

$$2. \text{Параметр } \text{Id}_{\mu_{out}} > 0 \text{ і } |\bullet t_{out_i} \cap t_j^\bullet| > 0,$$

де $\text{Id}_{\mu_{out}} = \text{Id}_{\tau_{out_i}}$. В цьому випадку існує хоча б одна спільна позиція між переходом t_j та цільовим переходом t_{out_i} для мітки μ_{out} .

$$3. \text{Параметр } \text{Id}_{\mu_{out}} > 0 \text{ і } |\bullet t_{out_i} \cap t_j^\bullet| = 0,$$

де $\text{Id}_{\mu_{out}} = \text{Id}_{\tau_{out_i}}$. В цьому випадку не існує спільної позиції між переходом t_j та цільовим переходом t_{out_i} для мітки μ_{out} .

Позначимо множину вихідних позицій переходу t_j через P_j . Тоді

$$P_j = (\bullet t_{out_1} \cap t_j^\bullet) \cup \dots \cup (\bullet t_{out_i} \cap t_j^\bullet) \cup \dots \cup (\bullet t_{out_k} \cap t_j^\bullet),$$

де $T_{out} = \{ t_{out_1}, \dots, t_{out_i}, \dots, t_{out_k} \}$ — множина переходів, для яких множина вхідних позицій є спільною з множиною вихідних позицій P_j переходу t_j .

Якщо $\text{Id}_{\mu_{out}} = 0$, то вихідна мітка вважається безадресною і процедура **Transmit** створює екземпляри вихідних міток μ_{out} , які призначені для розміщення на всіх вихідних позиціях даного переходу. Оскільки таке розміщення не може бути гарантовано виконане в довільний момент часу через можливу зайнятість позиції, то вихідні мітки попередньо розміщуються у черзі міток типу FIFO. Елемент цієї черги має формат (p, μ_{out}) , де p — це вихідна позиція переходу t_j , на яку необхідно передати мітку μ_{out} для того, щоб вона була здатна досягти цільового переходу t_{out} за умови $\text{Id}_{t_{out}} = \text{Id}_{\mu_{out}}$. Процедура **Set $_O_j$** забезпечує розміщення чергового елемента в черзі.

Procedure Transmit(μ_{out});

begin

If $\text{Id}_{\mu_{out}} = 0$ **then**

begin

For $i:=1$ **to** $|t_j^\bullet|$ **do**

begin

$p := (t_j^\bullet)_i$;

For $k:=1$ **to** $|p^\bullet|$ **do**

begin

$t_{out} := (p^\bullet)_k$; $\text{Id}_{\mu_{out}} := \text{Id}_{t_{out}}$;

```

    SetOj(p,  $\mu_{out}$ );
  end;
end ;
end ;
end .

```

Другий зі згаданих вище варіантів базується на тому, що АPRO-мережа використовує адресні мітки. У цьому випадку, якщо цільовий перехід t_{out} — сусідній до даного переходу t_j , то процедура **Transmit** матиме наступний вигляд.

```

Procedure Transmit( $\mu_{out}$ );
begin
  If ( $\text{Id}_{\mu_{out}} > 0$ ) then
    begin
      For  $i:=1$  to  $|t_j^\bullet|$  do
        begin
           $p := (t_j^\bullet)_i$ ;
          For  $k:=1$  to  $|p^\bullet|$  do
            begin
               $t_{out} := (p^\bullet)_k$ ;
              If  $\text{Id}_{\mu_{out}} = \text{Id}_{t_{out}}$  then SetOj(p,  $\mu_{out}$ );
            end;
          end ;
        end ;
      end .
    end .
  end .

```

Третій варіант розглядає ситуацію, коли необхідно пере-силати мітки віддаленим переходам АPRO-мережі. Для цього

застосовують механізм транзитних пересилок, що реалізований в даній мережі. В наступному параграфі буде детально розглянуто можливі алгоритми транзитних пересилок.

Дії переходу по розміщенню вихідних міток забезпечує процедура **Out**.

Procedure Out;

begin

While $\text{Get_}O_j$ **do**

begin

If $\text{Cur_}p < \text{Max_}p$ **then**

begin

If $\varphi_{\mu_{out}}$ **in** φ_p **then**

begin

$\text{Cur_}p := \text{Cur_}p + 1;$

$q := \text{Add}(\mu_{out});$

$\mu_{\min} := \text{Find_min}(q);$

$\delta_p := \delta_{\mu_{\min}};$

end;

end else $\text{Set_}O_j(p, \mu_{out});$

end;

Make_list(idle);

end.

Дана процедура циклічно опитує чергу вихідних міток за допомогою процедури $\text{Get_}O_j$. Якщо черга не містить вихідних міток, то процедура $\text{Get_}O_j$ повертає значення false, і цикл опитування черги міток припиняється. Завершення роботи процедури деактивації супроводжується встановленням стану

$\left. \right] t_j \left[\right.$ шляхом запуску процедури **Make_list**(idle). Якщо процедура **Get** $_O_j$ повертає значення true, то цей факт свідчить, що пересилці на вихідну позицію p підлягає вихідна мітка μ . Спочатку виконують перевірку поточної кількості міток на позиції p . Якщо ця кількість допускає розміщення додатково ще хоча б однієї мітки, то виконуємо перевірку, чи тип мітки-кандидата входить в множину допустимих типів міток на даній позиції. У випадку позитивної відповіді на це питання вихідна мітка за допомогою функції **Add** розміщується в частково упорядкованій множині міток q позиції p , а лічильник поточної кількості міток **Cur** $_p$ збільшується на одиницю. Якщо поточна вихідна мітка не може бути розміщена на вихідній позиції через те, що дана позиція заповнена мітками повністю, то мітка μ_{out} знову ставиться в чергу за допомогою процедури **Set** $_O_j$.

3.2.2. Проблема транзитних пересилок міток

В попередньому параграфі розглядався випадок, коли цільовий перехід вихідної мітки μ_{out} не є сусіднім переходом. Тоді виникає потреба транзитної пересилки даної мітки через проміжні переходи APRO-мережі. Якщо структура згаданої мережі допускає альтернативні варіанти такої пересилки, то актуальним стає задання вибору маршруту пересилок, який відповідав би деяким критеріям оптимальності. В рамках формування алгоритмів маршрутизації існують два підходи до конструювання APRO-мереж, які відрізняються способом задавання ідентифікаторів переходів. У випадку їх довільного задавання APRO-мережі називають неупорядкованими. Упорядковані APRO-мережі характеризуються просторово упорядкованими переходами. Така упорядкованість може досягатися, наприклад, довільною однозначною системою нумерації

переходів. Характер роботи процедури **Transmit**(μ_{out}) повністю залежить від властивості упорядкованості APRO-мережі.

Для неупорядкованих APRO-мереж може бути застосована тривіальна процедура **Transmit**(μ_{out}), яка забезпечує дублювання транзитної мітки на всіх вихідних позиціях переходу. Така процедура безумовно забезпечує потрапляння мітки на заданий перехід, але породжує значне завантаження мережі, що може викликати зниження показників продуктивності і навіть клінч.

Procedure Transmit(μ_{out});

begin

For $i:=1$ **to** $|t_j^\bullet|$ **do**

begin

$p := (t_j^\bullet)_i$;

Set $O_j(p, \mu_{out})$;

end;

end.

Незважаючи на вказані недоліки, цю процедуру можна успішно застосовувати для невеликих за кількістю переходів мереж або для мереж, які містять переходи з невеликою кількістю вихідних позицій.

3.2.3. Просування мережного часу

Після виконання послідовності активностей, які приводять перехід в стан $]t_j[$, постає питання про формування наступного циклу активності. Для цього необхідно вибрати спосіб просу-

вання мережного часу. Цей спосіб повинен бути узгодженим з обумовленим механізмом імітаційного моделювання мережі. Тому розв'язання проблеми просування мережного часу лежить в площині практичної реалізації алгоритмів функціонування APRO-мережі.

3.2.3.1. Методи просування мережного часу з загальним ресурсом

Спочатку розглянемо адаптований підхід, який базується на просуванні мережного часу за допомогою глобального годинника з фіксованим кроком. Мета адаптації полягає в об'єднанні відомих підходів для створення універсальної стратегії просування модельного часу, яка б дозволила без істотних змін забезпечувати функціонування APRO-мережі як на однопроцесорних системах, так і в багатопроцесорних обчислювальних середовищах.

Нагадаємо, що кожний перехід τ_j характеризується множиною параметрів χ_j , яка включає локальний лічильник часу δ_{τ_j} . Запропонований механізм просування глобального часу базується на деревовидній структурі лічильників часу, у вершині якої знаходиться глобальний лічильник часу G . Локальний час просувається в δ_{τ_j} асинхронно на кожному з переходів, а часова цілісність мережі забезпечується локальними алгоритмами просування мережного часу та корекцією локального лічильника часу δ_{τ_j} в стані t_j за допомогою процедури ω_j . На рис.3.5 наведено приклад просування мережного часу з фіксованим кроком для однорівневої мережі.

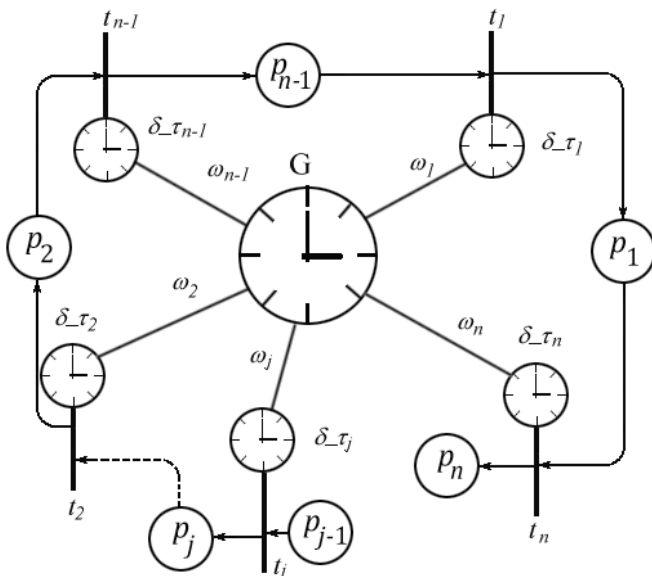


Рис.3.5. Приклад просування мережного часу з фіксованим кроком

Цикл роботи довільного переходу t_j закінчується станом

$] t_j [$. В цьому стані виконується процедура очікування ω_j , основне завдання якої полягає у зчитуванні глобального мережного часу з лічильника G та запису його в локальний лічильник δ_tau_j . Якщо перехід входить до складу операторного переходу, то такий перехід не має безпосереднього доступу до глобального лічильника часу, оскільки не взаємодіє безпосередньо з переходами вищого рівня.

У випадку багаторівневої мережі еталоном часу для всіх переходів нижчого рівня є віртуальний локальний лічильник операторного переходу θ , який модифікується процедурами $(\omega_\theta)_k$ його входів. Процедура **Set_idle** демонструє можливий алгоритм такої модифікації.

Procedure Set_idle;

begin

If $\text{Level}_{\tau_j} = 0$ **then** $\delta_{\tau_j} := G$ **else**

$(\delta_{\tau_\theta})_j := \max_k((\delta_{e_\theta})_k);$

end.

Якщо перехід t_j знаходиться на найвищому рівні мережі, тобто $\text{Level}_{\tau_j} = 0$, то при роботі процедури **Set_idle** його локальний лічильник δ_{τ_j} встановлюється рівним глобальному лічильнику G . У випадку, коли перехід знаходиться на нижчих рівнях ієрархії, тобто коли $\text{Level}_{\tau_j} > 0$, локальний лічильник даного переходу δ_{τ_j} коригується локальним лічильником того входу операторного переходу, що має найбільше значення.

Значення лічильника в станах активації, обробки та деактивації дорівнює часу стартової події поточної активності, обробкою якої займається перехід. Нехай такою активністю буде активність *activate*, яка запускає процедуру $\rho_j(\mu_{in})$, що включає процедуру аналізу міток на вхідних позиціях **Choose**. Якщо відповідна мітка знайдена, то наступною дією процедури $\rho_j(\mu_{in})$ є запуск процедури **Select** (μ_{in}) , яка забезпечує перевірку мітки на коректність. Якщо час створення мітки $\delta_{\mu_{in}}$ перевищує значення локального лічильника часу δ_{τ_j} , то це свідчить про те, що перехід τ_j ще не готовий до обробки мітки μ_{in} . Таку мітку залишають очікувати належного зростання локального лічильника часу, що може статися в результаті просування глобального мережного часу.

Function Timing(μ_{in}): *Boolean*;

*/*Common resource algorithm*

Begin

If $\delta_{\mu_{in}} \leq \delta_{\tau_j}$ **then**

begin

Result:=true;

$\Delta_{\tau_j} = 0$;

end else

begin

Result:=false;

$\Delta_{\tau_j} := (\delta_{\mu_{in}} - \delta_{\tau_j})$;

end ;

end .

Ще однією важливою властивістю функції **Timing**(μ_{in}) є формування поточного періоду простою Δ_{τ_j} переходу τ_j . Якщо параметр $\Delta_{\tau_j} > 0$, то він вказує на величину простою переходу до досягнення глобального часу, достатнього для обробки поточної мітки μ_{in} .

Використання даного підходу до просування мережного часу дозволяє реалізувати модель APRO-мережі як на послідовній, так і на паралельній обчислювальній системі. Максимальне розпаралелювання її функціонування зумовлене використанням мінімального спільного ресурсу, який представлений лише глобальним лічильником часу.

Недоліки просування мережного часу з фіксованим кроком:

1. Наявність періодів простою переходів Δ_{τ_j} .
2. Проблеми, пов'язані з доступом до глобального лічильника у випадку великої кількості переходів APRO-мережі.

Мінімізувати час простою переходів можна шляхом оптимізації часового кроку глобального годинника. Але такий підхід буде ефективним лише за умови невеликої різниці між величинами періодів роботи різних переходів мережі. Проблеми завантаження каналу доступу до глобального лічильника також частково можуть бути розв'язані за ієрархічної організації мережі.

Повністю позбутись простоїв переходів мережі дозволяє підхід, який базується на просуванні мережного часу в рамках єдиного списку, керованого мітками.

Якщо всі мітки мережі знаходяться в єдиному списку, сортованому за часом їх створення, то такий підхід виключає виникнення колізій і забезпечує однозначне просування мережного часу. Для реалізації цього механізму необхідно забезпечити існування глобального ресурсу, доступного для всіх переходів мережі. Таким ресурсом є структура даних, яка упорядкована за часом створення міток. На рис.3.6 наведено приклад формування згаданої структури.

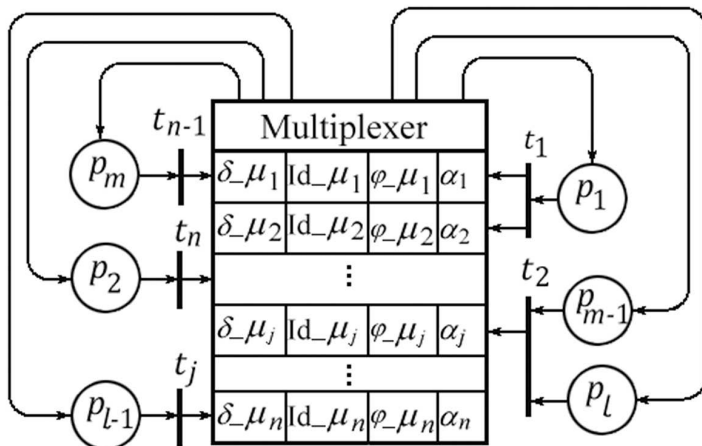


Рис.3.6. Приклад керування послідовністю міток

Вихідні мітки представлених на рис.3.6 переходів утворюють частково упорядковану послідовність міток, яка нале-

жить спільному ресурсу обчислювальної структури. Мітка з найменшим часом створення завжди знаходиться на початку цієї послідовності і є кандидатом на виконання. Програма *Multiplexer* дозволяє реалізувати один з розглянутих вище алгоритмів транзитних пересилок, результатом роботи якого є вибір позиції, на якій необхідно розмістити мітку, що є першою в черзі на виконання. Розміщення мітки на вхідній позиції може активувати даний перехід. Результатом роботи активованого переходу є мітка або кілька міток, які розміщуються відповідно до часу створення у спільному ресурсі.

Очевидно, що просування мережного часу за допомогою керування частково впорядкованою послідовністю міток є варіантом підходу, який базується на просуванні мережного часу, керованого списком подій. Запропонований варіант цього підходу, показаного на рис. 3.6, може бути використаний для організації обчислювального процесу в багатопроесорній обчислювальній структурі. Особливістю такої структури є наявність керуючого центру, який зміг би забезпечувати підтримку процедур роботи з єдиною частково упорядкованою послідовністю міток. Слід відмітити, що реалізація даного підходу для ієрархічних мереж викликає низку проблем, пов'язаних з ускладненнями обслуговування загальної послідовності міток для таких мереж. Використання єдиного списку значно збільшує його розмір та кількість супровідної інформації. Для розв'язання цієї проблеми необхідно забезпечити формування ієрархічної послідовності міток, але такий підхід ускладнює процеси керування мережею. Даний алгоритм просування мережного часу також може викликати зациклювання у випадку, коли на мережі існують циклічні зв'язки між переходами, і час проходження таких циклів менший, ніж час спрацювання інших переходів мережі. З метою подолання згаданих недоліків в останнє десятиліття активно розвиваються підходи до побудови локальних послідовностей міток, які могли б функціонувати без використання глобальних ресурсів при реалізації мережі на мультипроцесорній системі.

3.2.3.2. Консервативні методи паралельного просування мережного часу

Для розгляду загального підходу до паралельного просування мережного часу представимо APRO-мережу як сукупність переходів, які взаємодіють між собою, використовуючи мітки. З кожним переходом пов'язаний локальний годинник, список активностей, а також набір параметрів, які характеризують стан переходу. Від переходу до переходу мітки передаються через позиції, які можуть розглядатися як канали зв'язку. Кожна мітка включає інформацію про час її створення і ініціює послідовність активностей в переході-приймачі з початковим часом, що дорівнює часу створення даної мітки. Таким чином підтримується часова цілісність мережі, що є фундаментальним при розгляді даного підходу. Очевидно, що головною проблемою паралельного просування мережного часу є синхронізація, порушення якої може призвести до помилок у функціонуванні мережі. Наприклад, розглянемо фрагмент APRO-мережі, показаний на рис.3.7.

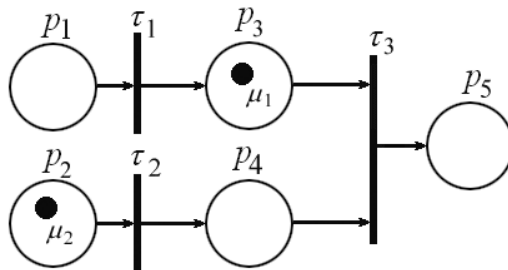


Рис.3.7. Приклад часової колізії

Поточне маркування цієї мережі $M = \{(p_3, \mu_1), (p_2, \mu_2)\}$, а еволюція $M | \sigma \rangle M''$ складається з кроків $\sigma = \{U_1, U_2\}$, де $U_1 = \{\tau_2, \tau_3\}$ і $U_2 = \{\tau_3\}$. Виконання першого кроку

$M|U_1\rangle M'$ формує маркування $M' = \{(p_5, \mu'_1), (p_4, \mu'_2)\}$, при якому мітка μ'_1 з часом створення $\delta_{-\mu'_1} = \delta_{-\mu_1} + \delta_{-\tau_3}$ розміщується на позиції p_5 , а мітка μ'_2 з часом створення $\delta_{-\mu'_2} = \delta_{-\mu_2} + \delta_{-\tau_2}$ займатиме позицію p_4 .

При паралельному просуванні мережного часу локальний лічильник переходу τ_3 дорівнюватиме $\delta_{-\tau_3} = \delta_{-\mu_2}$. Помилка синхронізації виникає за умови $\delta_{-\mu_1} > \delta_{-\mu'_2}$, тобто коли перехід τ_3 першою обробляє мітку з пізнішим часом створення.

Існують два напрямки розв'язання проблеми синхронізації, які зумовлені консервативною [52] та оптимістичною [53] стратегіями побудови алгоритму просування часу в паралельних моделях. В адаптації до APRO-мереж консервативна стратегія полягає у тому, що спрацювання переходу під дією мітки відбувається тільки у випадку, коли існує гарантія, що дана мітка є міткою з мінімальним часом, тобто коли точно відомо, що молодші мітки не будуть з'являтися на вхідних позиціях даного переходу.

Розглянемо застосування консервативного алгоритму до паралельного просування мережного часу на переході t_j APRO-мережі. Відомо, що довільна позиція $p_i \in \bullet t_j$ може містити множину міток $q_i = \{\mu_1, \dots, \mu_n\}$, які накопичуються на даній позиції внаслідок роботи того переходу APRO-мережі, для якого ця позиція є вихідною. Тому для виконання головної умови згаданого консервативного алгоритму необхідно перевірити вік найстаршої мітки на кожній із вхідних позицій. Таку перевірку виконують шляхом модифікації процедури **Timing**.

```

Function Timing( $\mu_{in}$ ): Boolean;
/* Conservative algorithm
/* simple APRO-net
begin
     $p_1 := (\bullet t_j)_1$ ;
     $\delta_{min} := \delta\_p_1$ ;
For  $i:=1$  to  $|\bullet t_j|$  do
    begin
         $p := (\bullet t_j)_i$ ;
        If  $\delta\_p \leq \delta_{min}$  then  $\delta_{min} := \delta\_p$ ;
    end;
If ( $\delta\_p = \delta\_p_{in}$ ) then
    begin
         $\delta\_p := \delta\_p$ ;
        Result:=true;
         $\Delta\_p = 0$ ;
    end else
    begin
        Result:=false;
         $\Delta\_p := (\delta\_p_{in} - \delta\_p)$  ;
    end ;
end.

```

В процедурі **Timing** аналізують знайдену за допомогою процедури **Choose** мітку μ_{in} , яка характеризується мінімальним для даної позиції часом створення і відповідає поточному значенню локального лічильника часу на даній позиції. Але у

випадку, коли на деякій позиції p мітки відсутні, то локальний лічильник вказує на час створення останньої мітки, яка знаходилася на цій позиції. Тому для попередження часової колізії в умовах консервативного алгоритму необхідно знайти вхідну позицію, яка характеризується мінімальним значенням локального лічильника. Якщо час створення мітки δ_{μ_m} співпадає з мінімальним значенням лічильника δ_p , то така мітка вважається безпечною і може бути оброблена переходом t_j . У протилежному випадку події `activate.stop` і `idle.start` переводять перехід t_j в стан $\left] t_j \left[$. Розглянутий підхід до уникнення часової колізії в умовах консервативного алгоритму може бути реалізованим за умови, що APRO-мережа є простою, тобто коли для потужностей множин мають місце нерівності $\left| \bullet p \right| \leq 1$ і $\left| p \bullet \right| \leq 1$. У цьому випадку виконується правило безпеки консервативного підходу, яке полягає у тому, що мітки повинні надходити на вхідну позицію з неспадним часом створення.

Слід відмітити, що головною проблемою консервативних методів просування мережного часу є тупикові ситуації. Класичний приклад тупикової ситуації за умови використання консервативного методу для простих мереж показаний на рис.3.8. У даному випадку мережа постійно буде знаходитися в стані очікування, оскільки як для переходу τ_1 , так і для переходу τ_2 не виконуються умови активації. Причиною тупикової ситуації є той факт, що локальні лічильники часу пустих позицій $\delta_{p_1} = 2$ і $\delta_{p_3} = 5$ мають менші значення, ніж час створення мітки $\delta_{\mu_2} = 20$, розміщеної на позиції p_2 , і час створення мітки $\delta_{\mu_4} = 10$, розміщеної на позиції p_4 .

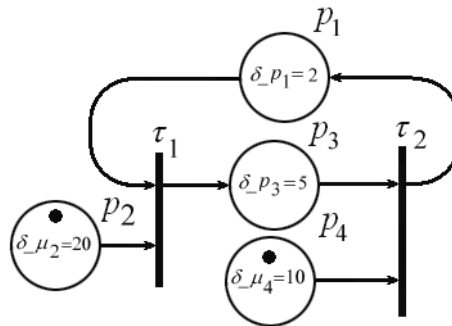


Рис.3.8. Тупикова ситуація

Пошук підходів, які дозволили б уникнути або зменшити кількість колізій та тупикових ситуацій, є основною рушійною силою розвитку консервативних алгоритмів просування мережного часу.

Якщо APRO-мережа не є простою або допускає транзитні пересилки, неспадний характер потрапляння міток на позицію забезпечити значно важче, що утруднює запобігання часовій колізії. Для розв'язання цієї проблеми, а також проблеми тупикових ситуацій, введемо деякий віковий параметр *Safe*, за участю якого модифікуємо процедуру **Timing**.

Function $\text{Timing}(\mu_{in}) : \text{Boolean};$

/ Conservative algorithm*

/ not simple APRO-net*

begin

$p_1 := (\bullet t_j)_1;$

$\delta_{\min} := \delta_{-p_1};$

For $i:=1$ **to** $|\bullet t_j|$ **do**

begin

```

 $p := (\bullet t_j)_i;$ 
If  $\delta\_p \leq \delta_{\min}$  then
  begin
     $\delta_{\min} := \delta\_p;$ 
     $p_{\min} := p;$ 
  end;
end;
 $\delta_{\max} := 0$ 
For  $k := 1$  to  $|q_{\min}|$  do
  begin
     $\mu_k := (q_{\min})_k;$ 
    If  $\delta\_{\mu_k} \geq \delta_{\max}$  then  $\delta_{\max} := \delta\_{\mu_k};$ 
  end;
If  $(\delta\_p = \delta\_{\mu})$  AND  $((\delta_{\max} - \delta\_{\mu}) \leq Safe)$  then
  begin
     $\delta\_{\tau_j} := \delta\_p;$ 
    Result:=true;
     $\Delta\_{\tau_j} = 0;$ 
  end else
  begin
    Result:=true;
     $\Delta\_{\tau_j} := (\delta\_{\mu} - \delta\_p);$ 
  end;
end.

```

Величину критичного віку *Safe* вибирають для кожного конкретного випадку, виходячи з того, що на позиції p_{\min} не

з'явиться мітка з іще меншим часом створення після досягнення найстаршою міткою віку *Safe* .

Консервативний метод з віковим параметром має значний недолік, який полягає у тому, що мережний час передачі інформації від одного переходу до іншого не може бути меншим від вікового параметру *Safe* .

Прискорення взаємодії переходів за умови, що АPRO-мережа не є простою, можна досягти шляхом застосування принципу синхронізуючих пересилок [54]. Цей принцип полягає у тому, що переходи АPRO-мережі, крім традиційних інформаційних міток, передають також спрощені синхронізуючі мітки, які містять інформацію про час створення. Формально синхронізуючі мітки представимо скороченим набором елементів $\mu^s = \{ Id_ \mu, \delta_ \mu, \phi_ \mu \equiv 0 \}$, а для їх обробки модифікуємо процедуру **Timing**.

Function Timing(μ):*Boolean*;

/ Conservative algorithm*

/ with synchronizing tokens*

begin

$p_1 := (\bullet t_j)_1$;

$\delta_{\min} := \delta_ p_1$;

For $i:=1$ **to** $|\bullet t_j|$ **do**

begin

$p := (\bullet t_j)_i$;

If $\delta_ p \leq \delta_{\min}$ **then** $\delta_{\min} := \delta_ p$;

end;

If $(\delta_ p = \delta_ \mu)$ **then**

begin

$\delta_ \tau_j := \delta_ p$;

```
If  $\varphi_{\mu} = 0$  then Result:=false else Result:=true;  
   $\Delta_{\tau_j} = 0$ ;  
end else  
begin  
  Result:=false;  
   $\Delta_{\tau_j} := (\delta_{\mu} - \delta_p)$  ;  
end ;  
end.
```

Вибір стратегії генерації синхронізуючих міток є джерелом створення нових консервативних методів паралельного просування мережного часу. Найпростіший підхід полягає у формуванні чергової синхронізуючої мітки після кожної зміни локального лічильника часу переходу. За цієї умови максимально зберігається часова цілісність мережі, але істотним недоліком є те, що продуктивність такої мережі падає в зв'язку з тим, що значні ресурси відволікаються на обробку синхронізуючих міток. З метою подолання цього недоліку в [54] запропонована ідея передачі синхронізуючої інформації від передавача тільки за запитом приймача. У даному випадку значно зменшується кількість синхронізуючих міток, але при цьому збільшується час очікування відповіді.

Підсумовуючи описані консервативні підходи, можна зробити висновок, що їх ефективність залежить від того, наскільки передбачуваною є поведінка переходів мережі. Тільки ці знання дають можливість застосовувати консервативні методи просування мережного часу, уникаючи колізій та тупикових ситуацій.

3.2.3.3. Оптимістичні методи паралельного просування мережного часу

Переважає більшість реальних паралельних систем, які підлягають формалізації за допомогою мереж, не мають строго передбачуваної поведінки. Тому застосування до них консервативних методів паралельного просування мережного часу викликає значні труднощі, що призводить до зниження продуктивності. В якості альтернативного підходу в [55] запропоновано оптимістичну стратегію просування часу, яка була розроблена для застосування в паралельних імітаційних моделях. Головна ідея цього підходу полягає в тому, щоб відмовитися попереднього визначення безпечної послідовності подій, які зумовлюють роботу кожного компонента моделі. Нова стратегія допускає виконання довільної послідовності подій, і основна увага тепер приділяється визначенню часових колізій, які вже відбулися, з метою подальшого виправлення помилок. Найважливіша перевага даного підходу полягає у безперешкодному виконанні імітаційною моделлю тих дій, які потенційно можуть призвести до часової колізії, але реально цієї колізії не відбувається.

Якщо на вхід переходу t_j надходить мітка μ з часом $\delta_{\mu} < \delta_{\tau_j}$, тобто така мітка, яка випадає з хронологічного порядку обслуговування міток даним переходом, то необхідно виконати послідовність дій, які повернуть APRO-мережу до базового стану, коректного для обробки даної мітки. Базовим будемо вважати стан, який відповідає умові $\delta_{\mu} \geq \delta_{\tau_j}$ і є початком функціонування мережі з урахуванням мітки μ . Для забезпечення можливості переходу t_j до базового стану необхідно зберігати всі мітки, що пройшли через вхідні ребра переходу t_j впродовж певного часу, а також історію частково

впорядкованої множини A_j , стани χ_j та мітки, які зберігаються у вихідній черзі O_j . Структура елементів АПРО-мережі, які забезпечують реалізацію оптимістичної стратегії паралельного просування мережного часу, показана на рис.3.9.

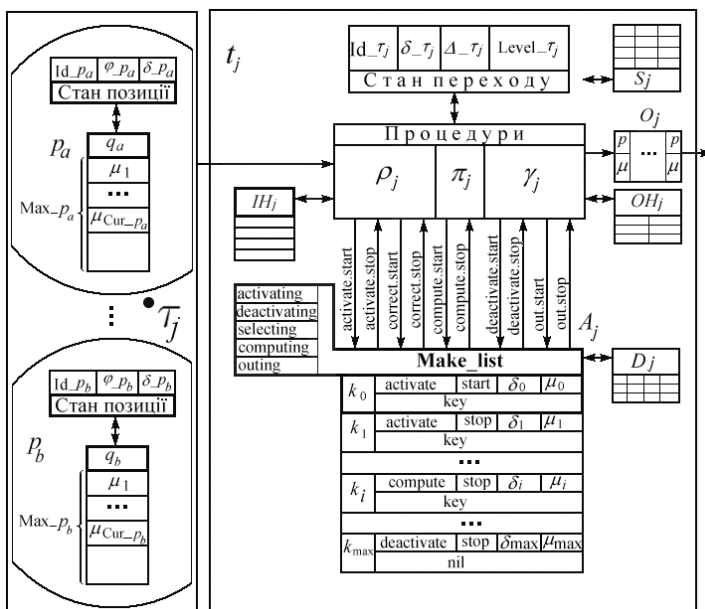


Рис. 3.9. Структура АПРО-мережі для оптимістичних методів паралельного просування мережного часу

Дана структура представляє зв'язки у модифікованому ядрі переходу АПРО-мережі (3.1). Нагадаємо, що $T = \{t_j\}_{j=1}^m, \tau_j = \{\chi_j, N_j\}$, χ_j — параметри переходу, N_j — функціональне ядро. Модифіковане функціональне ядро представимо у вигляді:

$$N_j = \{\rho_j, \pi_j, \gamma_j, S_j, D_j, O_j, IH_j, OH_j\},$$

де ρ_j — процедура активації, π_j — процедура обслуговування, γ_j — процедура деактивації, S_j — частково впорядкована послідовність минулих станів, D_j — частково впорядкована послідовність минулих подій, O_j — частково впорядкована послідовність вихідних міток, IH_j — частково впорядкована послідовність минулих вхідних міток, OH_j — частково впорядкована послідовність минулих вихідних міток.

Для реалізації методу оптимістичного паралельного просування мережного часу також необхідно модифікувати структуру міток, ввівши додатковий кваліфікаційний поділ на два види. Структура модифікованої мітки має вигляд:

$$\mu_k = \{ \lambda_k, \alpha_k, \sigma_k \},$$

де λ_k — параметри мітки, α_k — множина атрибутів мітки,

$\sigma_k = \{1, 0\}$ — вид мітки.

При $\sigma_k = 1$ мітка μ_k несе інформацію, яка забезпечує просування мережного часу вперед, а при $\sigma_k = 0$ мітка μ_k використовується для виконання зворотних операцій, що повертають мережу до базового стану. Принципова різниця між консервативною та оптимістичною стратегією полягає у тому, що при застосуванні оптимістичної стратегії знімається важливе обмеження хронологічної впорядкованості обслуговування вхідних міток.

Кожному елементу множини переходів $T = \{ t_j \}_{j=1}^m$ відповідає множина процесів $\mathbf{Pr} = \{ \text{Pr}_j \}_{j=1}^m$. На відміну від розглянутої раніше версії, APRO-мережа для реалізації оптимістичного методу просування мережного часу представлена переходами, які допускають одночасне існування кількох

підпроцесів в одному переході. Таким чином, процес довільного переходу має вигляд:

$$Pr_j = (Pr_{j1}, Pr_{j2}, Pr_{j3}, Pr_{j4}, Pr_{j5}) ,$$

де $Pr_{j1} := \mathfrak{R}(\text{activate})$ — підпроцес активації, який складається з рекурсивної активності activate ; $Pr_{j2} := \mathfrak{R}(\text{select})$ — рекурсивний підпроцес з активності select ; $Pr_{j3} := \text{compute}$ — підпроцес обробки переходу; $Pr_{j4} := \text{deactivate}$ — підпроцес деактивації; $Pr_{j5} := \mathfrak{R}(\text{out})$ — рекурсивний підпроцес виводу.

Граф зв'язків між даними підпроцесами показаний на рис.3.10.

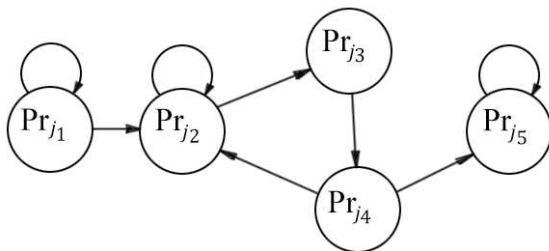


Рис.3.10. Граф процесу переходу Pr_j

Як видно з рис.3.10, Pr_{j1} є першим підпроцесом в ланцюжку обробки інформації переходом t_j . Цей підпроцес рекурсивно прокручує активність activate , яка забезпечує дії переходу по аналізу вхідних міток та формуванню вхідної послідовності міток IH_j . Наступна ланка обробки інформації — підпроцес Pr_{j2} . Активність select , яка є рекурсивною активністю даного підпроцесу, включає дії з аналізу параметрів чергової мітки з вхідної черги міток. Якщо чергова мітка призначена для

обробки на даному переході, то запускають підпроцес Pr_{j_3} , в результаті роботи якого активність `comput` формує вихідну мітку, маршрут подальшої передачі якої задає активність `deactivate` підпроцесу Pr_{j_4} . Рекурсивний процес Pr_{j_5} обслуговує чергу вихідних міток O_j та забезпечує розміщення міток на вихідних позиціях відповідно до маршруту передачі. Реалізація даного графа процесу для оптимістичного методу паралельного просування мережного часу показана на рис.3.11.

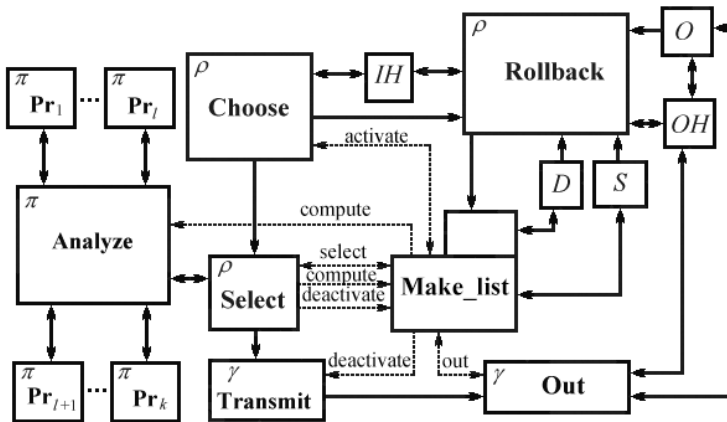


Рис.3.11. Структура процедур переходу для оптимістичного методу паралельного просування мережного часу

```

Procedure Make_list(new_action);
begin
  Case (new_action) of
    activate: begin
      If activating then
        Set_event( $\delta_{\tau_j}$ , activate.stop) else

```

```

Set_event( $\delta_{\tau_j} + \Delta_{\text{activate}}$ , activate.start);
    end;
select : begin
    If selecting then Set_event( $\delta_{\tau_j}$ , select.stop) else
        Set_event( $\delta_{\tau_j} + \Delta_{\text{correct}}$ , select.start);
        end;
deactivate : begin
    If deactivating then
Set_event( $\delta_{\tau_j}$ , deactivate.stop) else
Set_event( $\delta_{\tau_j} + \Delta_{\text{deactivate}}$ , deactivate.start);
        end;
compute : begin
    If computing then
Set_event( $\delta_{\tau_j}$ , compute.stop) then
Set_event( $\delta_{\tau_j} + \Delta_{\text{compute}}$ , compute.start);
        end;
end;
end.

```

Як і в попередньому випадку, процедура обслуговує список подій, які створюють або знищують відповідні активності. Активність activate створюється подією activate.start і знищується подією activate.stop. Вона забезпечує роботу процедури **Choose** ($\bullet t_j$) із виконання послідовності дій переходу, які відповідають стану активації. Основним результатом функціонування процедури **Choose** ($\bullet t_j$) є заповнення вхідної послідовності міток *IH* в хронологічному порядку. Процедура також

аналізує умови запуску процедури **Rollback**(μ_{in}), яка забезпечує повернення переходу до базового стану у випадку виникнення часової колізії. Активність **select** керує роботою процедури **Select**(IH_j). Завдання цієї процедури полягає в аналізі хронологічної послідовності вхідних міток IH_j , сформованої процедурою **Choose**($\bullet t_j$). В результаті такого аналізу мітки поділяються на транзитні (такі, що призначені іншим переходам) та коректні (такі, що потребують обробки у даному переході). Якщо чергова мітка є транзитною, то в послідовність активностей A_j вставляють активність **deactivate**, яка забезпечує дії переходу з передачі цієї мітки за допомогою процедури **Transmit**(μ_{out}). Ця процедура формує вихідну чергу міток із вказівкою вихідної позиції, на яку ту чи іншу мітку необхідно передати. Якщо чергова мітка з вхідної послідовності IH_j кваліфікується процедурою **Select**(IH_j) як коректна, то результатом є створення активності **compute**, яка керує роботою комплексу процедур, об'єднаних в процедурі обробки переходу π_j . Некоректна мітка спричиняє простий перезапуск процедури **Select**(IH_j) шляхом створення чергової активності **select**. Дії з безпосереднього розміщення міток на вихідних позиціях та ведення історії роботи переходу забезпечує процедура **Out**.

Розглянемо детальніше принципи функціонування згаданих процедур. Збереження часової цілісності мережі досягають за рахунок введення черги вхідних міток IH_j та попереднього аналізу вхідних міток за допомогою модифікованої процедури **Choose**.

Procedure Choose($\bullet t_j$);

begin

For $i:=1$ **to** $|\bullet t_j|$ **do**

begin

$p := (\bullet t_j)_i$;

If $\text{Cur}_p > 0$ **then**

begin

$\mu_{in} := \text{Find_min}(q)$;

$\mu_{IH} := \text{Find}(IH_j, \mu_{in})$;

If $\delta_{\mu_{in}} < \delta_{\tau_j}$ **then**

begin

If $\text{Assigned}(\mu_{IH})$ **then**

begin

If $((\sigma_{in} = 0) \text{ and } (\sigma_I = 1))$ **then** **Rollback**(μ_{in});

end else **If** $(\sigma_{in} = 1)$ **then** **Rollback**(μ_{in});

end;

If $\text{Assigned}(\mu_{IH})$ **then** **Erase**(IH_j, μ_{IH}) **else**

Set $_{IH_j}(\mu_{in})$;

end;

end;

Make_list(activate);

end.

Спочатку вибирають мітку μ_{in} з вхідної позиції p переходу t_j . Для мітки μ_{in} у частково впорядкованій послідовності

вхідних міток IH_j виконують пошук дуальної мітки μ_{IH} . Дуальною міткою μ_{IH} до мітки μ_{in} будемо називати таку мітку, параметри якої відповідають умові $\lambda_{in} = \lambda_{IH}$. Нерівність $\delta_{\mu_{in}} < \delta_{\tau_j}$ свідчить про те, що мітка μ_{in} потенційно несе загрозу часовій цілісності моделі. Для уникнення цієї загрози аналізують результати пошуку дуальної мітки μ_{IH} . Якщо така мітка була знайдена у вхідній послідовності IH_j , то функція $\text{Assigned}(\mu_{IH})$ повертає значення true, тобто змінна μ_{IH} містить коректне посилання на дуальну мітку. Подальші дії процедури **Choose** залежать як від виду мітки μ_{in} , так і від виду мітки μ_{IH} . За умови $((\sigma_{in} = 0) \text{ and } (\sigma_I = 1))$ виконують повернення переходу t_j до базового стану за допомогою процедури **Rollback** (μ_{in}) . Запуск цієї процедури також відбувається при $((\sigma_{in} = 1) \text{ and } (\text{not Assigned}(\mu_{IH})))$. Якщо для мітки μ_{in} знайдена дуальна мітка μ_{IH} , то вона завжди вилучається з послідовності IH_j за допомогою процедури **Erase** (IH_j, μ_{IH}) , а у протилежному випадку мітка μ_{in} розміщується в послідовності IH_j відповідно до значення параметра $\delta_{\mu_{in}}$.

Процедура **Rollback** (μ_{in}) виконує стандартну послідовність дій, які забезпечують повернення переходу t_j до базового стану.

Procedure Rollback(μ_{in});

begin

$\chi_j := \mathbf{Find}(S_j, \delta_{-}\mu_{in})$;

For $k := 1$ **to** $\text{Max}_{-}D_j$ **do**

begin

$(\text{Event}_k, \delta_k, \mu_k) := \mathbf{Get}_{-}D_j(k)$;

If $\delta_{-}\tau_j \leq \delta_k \leq \delta_{-}\mu_{in}$ **then**

begin

$\mathbf{Set}_{-}A_j(\text{Event}_k, \delta_k, \mu_k)$;

$\mathbf{Erase}(D_j, (\text{Event}_k, \delta_k, \mu_k))$;

end;

end;

For $k := 1$ **to** $\text{Max}_{-}IH_j$ **do**

begin

$\mu_k := \mathbf{Get}_{-}IH_j(k)$;

If $\delta_{-}\mu_{in} > \delta_{-}\mu_k$ **then Erase**(IH_j, μ_k) ;

end;

For $k := 1$ **to** $\text{Max}_{-}O_j$ **do**

begin

$(p_{out}, \mu_{out}) := \mathbf{Get}_{-}OH_j(k)$;

If $\delta_{-}\mu_{in} > \delta_{-}\mu_{out}$ **then**

begin

If $\sigma_{out} = 0$ **then** $\sigma_{out} := 1$ **else** $\sigma_{out} := 0$;

$\mathbf{Set}_{-}O_j(p_{out}, \mu_{out})$;

$\mathbf{Erase}(OH_j, (p_{out}, \mu_{out}))$;

end;
end;
end.

Згадана послідовність дій включає:

1. Відновлення того стану переходу, який був на час створення мітки μ_{in} , за допомогою функції

$$\chi_j := \mathbf{Find}(S_j, \delta_{\mu_{in}}).$$

2. Відновлення послідовності подій A_j , що мала місце на час створення мітки μ_{in} , шляхом пошуку в частково впорядкованій послідовності D_j .

3. Очищення послідовності минулих вхідних міток IH_j від тих міток, час створення яких перевищує $\delta_{\mu_{in}}$.

4. Створення послідовності дуальних міток на базі послідовності OH_j .

Процедура перевірки коректності параметрів мітки **Select** запускається подією `select.start` і обслуговує частково впорядковану послідовність вхідних міток IH_j .

Procedure Select(IH_j);

begin

$\mu_1 := \mathbf{Get_IH}_j(1);$

If Assigned(μ_1) **then**

begin

If $Id_{\mu_1} \neq Id_{\tau_j}$ **then** **Make_list**(deactivate) **else**

begin

Make_list(compute);

```

    end;
  end else Make_list(select);
end.

```

За допомогою функції $\mu_1 := \mathbf{Get_IH}_j(1)$ вибираємо з голови частково впорядкованої послідовності вхідних міток IH_j мітку μ_1 , яка характеризується мінімальним часом створення. Якщо $Id_ \mu_1 \neq Id_ \tau_j$, то таку мітку слід вважати транзитною міткою, подальша обробка якої виконується під керуванням активності deactivate. За умови $Id_ \mu_1 = Id_ \tau_j$ процедура створює активність compute, яка керує роботою процедури обробки переходу π_j .

Процедура $\mathbf{Transmit}(\mu_{out})$ входить до складу процедури деактивації γ_j . Головні завдання цієї процедури полягають у виконанні дій, пов'язаних з вибором вихідних позицій для передачі міток, формуванням частково впорядкованої послідовності вихідних міток O_j та формуванням послідовності минулих вихідних міток OH_j . Нагадаємо, що структура процедури $\mathbf{Transmit}(\mu_{out})$ залежить від способу транзитних пересилок, прийнятих в APRO-мережі. Принципи формування таких пересилок розглядалися в попередніх параграфах. Як приклад, наведемо один з варіантів реалізації процедури $\mathbf{Transmit}(\mu_{out})$.

```

Procedure  $\mathbf{Transmit}(\mu_{out})$ ;
begin

```

```

  For  $i:=1$  to  $|t_j^\bullet|$  do

```

```

begin
   $p := \left( t_j^\bullet \right)_i$ ;
  For  $k := 1$  to  $|p^\bullet|$  do
    begin
       $t_{out} := \left( p^\bullet \right)_k$ ;
      If  $\text{Id}_{\tau_{out}} = \text{Id}_{\mu_{out}}$  then  $p_{out} := p$ ;
    end;
  end;
  If Assigned $(p_{out})$  then
    begin
      Set $_O_j(p_{out}, \mu_{out})$ ;
      Set $_OH_j(p_{out}, \mu_{out})$ ;
    end else
      begin
        For  $i := 1$  to  $|t_j^\bullet|$  do
          begin
             $p := \left( t_j^\bullet \right)_i$ ;
            Set $_O_j(p, \mu_{out})$ ;
            Set $_OH_j(p, \mu_{out})$ ;
          end;
        end;
        Make_list(deactivate);
      end .

```

У даному випадку процедура реалізує алгоритм пошуку сусіднього переходу t_{out} , пов'язаного з переходом t_j за допомогою позиції p . Якщо знайдений таким чином перехід t_{out} є

цільовим переходом мітки μ_{out} , то згадана мітка призначається для передачі на позицію p . Якщо жоден із знайдених сусідніх переходів t_{out} не відповідає цільовому переходу мітки μ_{out} , то передача мітки відбувається на всі вихідні позиції переходу t_j .

Процедура **Transmit**(μ_{out}) розміщує вихідні мітки разом з маршрутом їх передачі в черзі вихідних міток O_j за допомогою процедури **Set** $_O_j$. Симетрична процедура **Get** $_O_j$ застосовується процедурою **Out** і виконує сукупність дій по добуванню чергової мітки μ_{out} та її цільової позиції p_{out} з черги O_j . В ході виконання процедури **Out** проводять попередній аналіз згаданої цільової позиції з метою визначення величини її заповнення та допустимого типу мітки. Якщо результати перевірки є позитивними, то виконується сукупність дій по розміщенню мітки, які були описані при розгляді попередньої версії процедури **Out**. При реалізації оптимістичної стратегії паралельного просування мережного часу процедура **Out** виконує також функції збереження історії вихідних міток OH_j , які пройшли через чергу вихідних міток O_j .

Procedure Out;

begin

While Get $_O_j$ **do**

begin

If $Cur_p < Max_p$ **then**

begin

If $\varphi_ \mu_{out}$ **in** φ_p **then**

$Cur_p := Cur_p + 1;$

```

    q := Add( $\mu_{out}$ );
     $\mu_{min} := \mathbf{Find\_min}(q)$ ;
     $\delta\_p := \delta\_ \mu_{min}$ ;
    Set_OHj;
end;
end else Set_Oj(p,  $\mu_{out}$ );
end;
Make_list(idle);
end.

```

3.2.4. Збір статистичної інформації

Відомо, що одним з головних критеріїв для оцінки та коригування адекватності моделі є ступінь її калібрування. Існує велика кількість методик калібрування, які визначаються метою моделювання та деталізацією компонентів моделі. Важливу роль також відіграє характер робочого навантаження моделі. Слід відмітити, що незалежно від вибраної методики основним інформаційним джерелом для аналізу моделі залишається статистична інформація, зібрана в ході функціонування моделі.

Найпростішою мірою адекватності деякого модельного параметра y^* та ідеального параметра y може служити абсолютне відхилення $\Delta y = |y^* - y|$ або відносна похибка

$\delta y = \frac{|y^* - y|}{y}$. Модель можна вважати адекватною за умов:

$\Delta y < \varepsilon$, $\delta y < \alpha$, де $\varepsilon, \alpha \rightarrow 0$. На жаль, практичне застосування даного критерію адекватності найчастіше неможливе з ряду ускладнень, які випливають з того, що неможливо визначити ідеальне значення y . Адекватність може бути функцією від сукупності параметрів, а не кожного параметра окремо. Тому в

даному формальному описі APRO-мереж запропоновано використання статистичних показників (Δ, Ψ, Λ) , що мають ієрархічну структуру.

Множина показників продуктивності $\Delta = (\delta_1, \delta_2, \delta_3)$, наприклад, може включати пропускну здатність δ_1 , швидкість обробки заданих фрагментів алгоритму δ_2 , швидкість виконання тривіальних операцій δ_3 .

Множина показників реактивності $\Psi = (\psi_1, \psi_2, \psi_3)$ також не є жорстко заданою і включає три ієрархічно зв'язаних показники: ψ_1 — загальний час реакції моделі, ψ_2 — час реакції фрагменту моделі, ψ_3 — час реакції елемента моделі на заданий вплив.

Множина показників використання $\Lambda = (\lambda_1, \lambda_2, \lambda_3)$ визначає вузькі місця, що виникають при використанні ресурсів. Застосовуючи ієрархічний підхід до формування показників, задамо: λ_1 — інтегральний показник використання ресурсів, λ_2 — показник використання операторного переходу, λ_3 — показник використання простого переходу.

3.3. Аналіз APRO-мереж

В рамках аналізу мереж найчастіше використовують наступні властивості: безпека, обмеженість, консервативність і активність.

Означення 3.8. APRO-мережа Φ є *безпечною*, якщо для кожної її позиції $p_i \in P$, $1 \leq i \leq n$, з початковим маркуванням $M(p_i)$ справедлива нерівність $M'(p_i) \leq 1$ за умови, що $M'(p_i) \in D(\Phi, M(p_i))$.

Безпека мережі відіграє важливу роль при побудові детальних мережних об'єктів. В цьому випадку допускається відсутність міток на позиції, а максимальна їх кількість дорівнює 1. Безпечні мережі найчастіше застосовують на найнижчих рівнях багаторівневих мереж. Оскільки максимально допустима кількість міток на позиції p_i визначається параметром $\text{Max}_{-}p_i$, то APRO-мережа безпечна при $\text{Max}_{-}p_i = 1$.

Узагальненням безпеки є властивість обмеженості. Обмеженою вважають мережу, кількість міток на позиціях якої обмежена.

Означення 3.9. APRO-мережа Φ є k -безпечною або k -обмеженою, якщо для кожної її позиції $p_i \in P$, $1 \leq i \leq n$, з початковим маркуванням $M(p_i)$ справедлива нерівність $M'(p_i) \leq k$ за умови, що $M'(p_i) \in D(\Phi, M(p_i))$.

Оскільки k -обмежена мережа допускає наявність одночасно кількох міток на позиції, то означення позицій APRO-мережі включає параметр $\varphi_{-}p_i$, що визначає множину допустимих типів міток та структуровану множину q_i , яка дозволяє полегшити доступ до структур даних мітки.

При представленні за допомогою APRO-мереж опису функціонування технічних ресурсів важливу роль відіграє властивість консервативності.

Означення 3.10. APRO-мережа Φ з початковим маркуванням M є консервативною, якщо для всіх маркувань $M' \in D(\Phi, M)$

$$\sum_{p_i \in P} M'(p_i) = \sum_{p_i \in P} M(p_i).$$

Якщо мережа є консервативною, то цей факт символізує незнищуваність ресурсів, які представлені атрибутами міток.

Аналіз APRO-мережі включає також розгляд активності мережі як результату активності її переходів. Відповідно до [14] визначимо кілька рівнів активності переходів.

Рівень 0: Перехід t_j має активність рівня 0, якщо ніколи не можуть виникнути умови його запуску.

Рівень 1: Перехід t_j має активність рівня 1, якщо він потенційно може бути запущеним, тобто якщо існує маркування $M' \in D(\Phi, M)$, яке включає спрацювання переходу t_j .

Рівень 2: Перехід t_j має активність рівня 2, якщо він фактично спрацьовує хоча б один раз за період роботи мережі.

Рівень 3: Перехід t_j має активність рівня 3, якщо він фактично спрацьовує нескінченну кількість разів у випадку нескінченної кількості запусків мережі.

Рівень 4: Перехід t_j має активність рівня 4, якщо для довільного маркування $M' \in D(\Phi, M)$ існує послідовність спрацювань σ , в яку завжди входить перехід t_j .

Якщо перехід має рівень активності 0, то його називають пасивним переходом. У випадку, коли перехід має рівень активності 4, його називають активним. APRO-мережа Φ має рівень активності i , якщо всі її переходи мають саме такий рівень активності.

Основою аналізу APRO-мережі є дерево досяжності, яке відповідає впорядкованій множині досяжності $D(\Phi, M)$. Для побудови дерева досяжності адаптуємо алгоритм траверсу дерева [56]. Розглянемо два різновиди цього алгоритму (префіксний і постфіксний) в залежності від способу вибору вершин графа.

Дерево задамо у вигляді впорядкованої послідовності з початковим елементом Node, що містить структуру даних вузла:

Type

Tmarking: array [1..n] of **Integer**;

Tnode = **record**

M: Tmarking;

Up: Tnode;

Down: Tnode;

end.

Префіксний алгоритм траверсу дерева базується на застосуванні рекурсивної процедури:

Var Node:Tnode;

Procedure Visit(Node);

begin

Print Node.M;

If Node.Down \neq nil **then**

begin

Visit(Node.Down);

Node.Down:=Node.Down.Down;

end else

begin

Visit(Node.Up);

Node.Up:=Node.Up.Up;

end;

end.

Постфіксний алгоритм відрізняється від префіксного порядком обходу дерева:

Var Node:Tnode;

Procedure Visit(Node);

begin

```

if Node.Down  $\neq$  nil then
  begin
    Visit(Node.Down);
    Node.Down := Node.Down.Down;
  end else
  begin
    Visit(Node.Up);
    Node.Up := Node.Up.Up;
  end;
  print Node.M;
end.

```

Представимо дерево досяжності у вигляді графа, вершинами якого є маркування, а ребра — активні переходи APRO-мережі. Розглянемо приклад APRO-мережі, що показана на рис. 3.12.

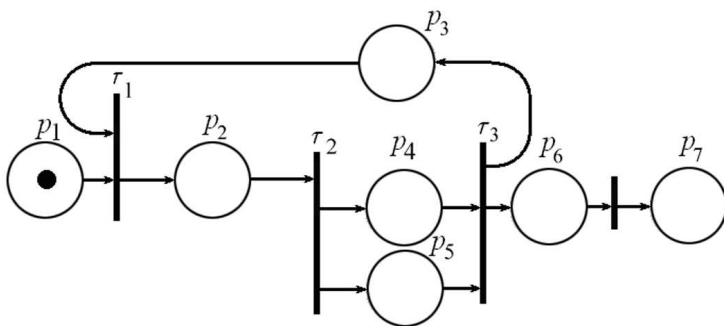


Рис.3.12. Приклад APRO-мережі

Вузол дерева досяжності Node містить параметр Node.M, який зберігає поточне маркування APRO-мережі. Наприклад, дерево досяжності (рис.3.13) відповідає випадку безпечної мережі.

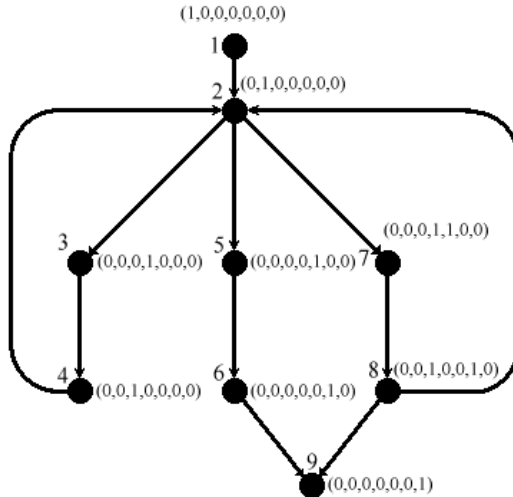


Рис.3.13. Дерево досяжності

Тому максимальна кількість міток на кожній позиції задана множиною $\{ p_i \mid \text{Max}_{-} p_i = 1 \}_{i=1}^n$. Параметри Node.Down і Node.Up зберігають посилання на відповідно нижчі та вищі за ієрархією вузли дерева досяжності. Наведемо в таблиці 3.4 структури даних вузлів дерева досяжності.

Таблиця 3.4

Структура даних вузлів дерева досяжності

Node	M	Down	Up
1	1,0,0,0,0,0,0	2	nil
2	0,1,0,0,0,0,0	3,5,7	1
3	0,0,0,1,0,0,0	4	2
4	0,0,1,0,0,0,0	2	3
5	0,0,0,0,1,0,0	6	2
6	0,0,0,0,0,1,0	5	9
7	0,0,0,1,1,0,0	2	8
8	0,0,1,0,0,1,0	2,9	7
9	0,0,0,0,0,0,1	nil	6,8

З таблиці 3.4 видно, що посилання на верхній вузол для вузла 1 дорівнює nil. Цей факт свідчить про те, що вузол 1 є *початковим* вузлом APRO-мережі. Якщо посилання на нижній вузол для деякого вузла i дорівнює nil, то вузол i називають кінцевим вузлом. Відповідно до таблиці 3.4 кінцевим вузлом мережі є вузол 9. Якщо параметр Node.Down містить послідовність, яка складається з кількох посилань, то це свідчить про розгалуження дерева досяжності. Прикладами такого розгалуження є вузли 2 і 8. Навпаки, якщо параметр Node.Up містить послідовність, яка складається з кількох посилань, то цей факт свідчить про об'єднання кількох гілок дерева досяжності. Прикладом такого об'єднання є вузол 9.

Порядок побудови дерева базується на використанні правил спрацювання переходів. Нагадаємо, що спрацювання переходу t_j APRO-мережі залежить від змісту процедур ρ_j , π_j і γ_j , а також від атрибутів тієї мітки, яка ініціює спрацювання цих процедур. В даному випадку актуальною є лише та частина процесу спрацювання, яка пов'язана зі зміною вхідного та вихідного маркування переходу t_j . Правила обробки вхідних міток впливають з того факту, що в APRO-мережах мітки не є засобами синхронізації ресурсів (як в більшості версій мереж Петрі), а виконують функцію носіїв інформації. Тому кожна мітка, що з'являється на вхідній позиції переходу, є ініціатором його активації. В загальному випадку APRO-мережа не є консервативною. Якщо інформація, яка передається з атрибутами мітки, призначена для переходу t_j і немає необхідності в зовнішній реакції на прийом цієї інформації, то мітка поглинається переходом. Якщо зовнішнє середовище переходу t_j очікує від нього результатів обчислень, ініційованих вхідними даними, то тоді, в результаті спрацювання переходу, на вихідні позиції видаються мітки з атрибутами очікуваних результатів. Згадаємо також те, що в APRO-мережах існують транзитні мітки, тобто такі мітки, які передаються із вхідних позицій на вихідні без

зміни атрибутів. У випадку, коли перехід має кілька вихідних позицій, процедура формування вихідних міток залежить від того, чи є дана APRO-мережа впорядкованою. У випадку впорядкованості APRO-мережі завжди відомий напрямок передачі мітки. Тому в результаті спрацювання переходу відбувається зміна маркування, що допускає зміну станів тільки деякої підмножини вихідних позицій. Для невпорядкованих APRO-мереж перехід реалізує дублювання міток на всі свої вихідні позиції. Якщо збільшити критерій обмеженості мережі до

$$\left\{ p_i \mid \text{Max} _ p_i \leq k, k = \text{const} \right\}_{i=1}^n,$$

то тоді виникає питання, чи буде дерево досяжності скінченним. В [14, 57] наведено ряд теорем, які задають умови скінченності дерева досяжності для довільної k -обмеженої мережі. В зв'язку з цим дерево досяжності стало одним з базових інструментів оцінки еквівалентності мереж.

3.4. Еквівалентність APRO-мереж

Розв'язання проблем еквівалентності мереж є актуальним з початку виникнення мереж Петрі в 60-х роках минулого століття. За цей час сформовано ряд підходів, більшість з яких базувалася на застосуванні методики визначення досяжності. Такі види еквівалентності дістали назву еквівалентності маркування та трасової еквівалентності. В [57] наведено ґрунтовне дослідження обмежень застосування цих видів еквівалентності для мереж Петрі. Результатом дослідження став ключовий висновок, що обмеження застосування еквівалентності впливають з фіксованого набору правил роботи мереж. Важливим результатом став також висновок про неможливість гарантовано застосовувати критерії еквівалентного маркування до класичних мереж Петрі, які мають багаторівневу структуру. Так звана „проблема включення” полягає у тому, що включення досяжної мережі N_1 в досяжну мережу N_0 може призвести до недосяж-

ності мережі N_0 . Для того, щоб довести можливість застосування критеріїв еквівалентного маркування, необхідно для мережі N_0 побудувати дерево досяжності після включення в неї мережі N_1 .

Означення 3.7. Дві APRO-мережі Φ_1 і Φ_2 будемо вважати пов'язаними еквівалентністю маркування, тобто $Mr(\Phi_1) = Mr(\Phi_2)$, якщо $P_1 = P_2$, і для кожної позиції p_i , $1 \leq i \leq n$, існують однакові досяжні маркування $M_1(p_i) = M_2(p_i)$.

Даний критерій еквівалентності оперує тільки досяжними маркуваннями. Тому його частіше використовують для безпечних чистих мереж, що характеризуються відсутністю циклів. Для складніших мереж поряд з досяжністю маркування важливо також визначити проміжну послідовність зміни станів, яка веде від початкового маркування до досяжного.

Означення 3.8. Нехай Φ — APRO-мережа, у якій активність кожного переходу $t_j \in T$, $1 \leq j \leq n$, представлена відповідною функцією активації ρ_j . Множиною трас $\text{Tr}(\Phi)$ мережі Φ будемо називати множину послідовностей $\langle \rho_1, \dots, \rho_k \rangle$, якщо для кожної такої послідовності існує послідовність маркувань $\langle M_1 \dots M_k \rangle$ така, що

$$M_0 \xrightarrow{\rho_1} M_1 \xrightarrow{\rho_2} \dots \xrightarrow{\rho_k} M_k$$

Означення 3.9. Дві APRO-мережі Φ_1 і Φ_2 будемо вважати пов'язаними трасовою еквівалентністю, якщо їх множини трас співпадають, тобто $\text{Tr}(\Phi_1) = \text{Tr}(\Phi_2)$.

Як згадувалося, застосування трасової еквівалентності не є універсальним і може бути застосованим тільки у випадку впевненості в досяжності кінцевого маркування M_k для обох мереж, що підлягають порівнянню.

Якщо кінцеві маркування мереж є досяжними, то існують відповідні траси, які складаються з послідовностей спрацювань переходів, що приводять до виникнення даних кінцевих маркувань. У цьому випадку виникає можливість застосування поведінкової еквівалентності [58, 59], яка базується на вимозі не тільки еквівалентності трас, а й досягнення однакових маркувань після виконання відповідних функцій активації.

Означення 3.10 Відношення R між множинами маркувань двох APRO-мереж є відношенням *строкої взаємної подібності*, якщо для кожної пари $(M_i, M_j) \in R$ і для довільної функції активації ρ :

- для спрацювання $M_i \xrightarrow{\rho} M'_i$ існує маркування M'_j таке, що $M_j \xrightarrow{\rho} M'_j$ при $(M'_i, M'_j) \in R$,
- для спрацювання $M_j \xrightarrow{\rho} M'_j$ існує маркування M'_i таке, що $M_i \xrightarrow{\rho} M'_i$ при $(M'_i, M'_j) \in R$.

Означення 3.11. Дві APRO-мережі Φ_1 і Φ_2 є *строго взаємно подібними*, якщо існує відношення строкої взаємної подібності R , що містить пару (M_{01}, M_{02}) початкових маркувань мереж Φ_1 і Φ_2 .

Слід відмітити, що доведення даних критеріїв еквівалентності, на жаль, веде до виникнення NP -повних алгоритмів. Причиною цього є той факт, що доведення трасової еквівалентності потребує розв'язання проблеми досяжності. Але на сього-

дні дана проблема не має узагальненого розв'язку навіть для базових мереж Петрі.

Оскільки APRO-мережі не обмежені жорсткими правилами спрацювання переходів, то існування траси визначається за наявності зв'язку між відповідними позиціями та алгоритмічним навантаженням переходів. Останнє значно утруднює формалізацію проблеми досяжності і робить її залежною від інформаційного наповнення атрибутів міток. Тому єдиним підходом до встановлення строгої взаємної подібності APRO-мереж є алгоритмічний підхід.

Суть алгоритму встановлення строгої взаємної подібності APRO-мереж полягає у пошуку хоча б однієї пари маркувань $(M_i)_1 \neq (M_i)_2$, яка б заперечувала строгу взаємну подібність мереж Φ_1 і Φ_2 . Якщо цей факт не доведено, то наступний крок полягає в доведенні $(M_i)_1 \sim (M_i)_2$.

Нехай $((M_i)_1, (M_i)_2)$ — довільна пара маркувань мереж Φ_1 і Φ_2 . Тоді алгоритм визначення строгої взаємної подібності цих маркувань складається з наступних кроків:

1. Виконати спробу переходу $(M_i)_1 \xrightarrow{\rho} (M'_i)_1$ від маркування $(M_i)_1$ до довільного маркування $(M'_i)_1$.
2. За умови успішного виконання переходу $(M_i)_1 \xrightarrow{\rho} (M'_i)_1$ виконати перехід $(M_j)_2 \xrightarrow{\rho} (M'_j)_2$.
3. Якщо переходи $(M_i)_1 \xrightarrow{\rho} (M'_i)_1$ і $(M_j)_2 \xrightarrow{\rho} (M'_j)_2$ виконані успішно, встановити нову початкову пару маркувань шляхом переприсвоювань: $(M_i)_1 := (M'_i)_1$ і $(M_j)_2 := (M'_j)_2$.

Алгоритм встановлення строгої взаємної подібності завжди починається з початкової пари маркувань $((M)_1, (M)_2)$ і може розвиватися в кількох напрямках в залежності від результату першого кроку. Другий крок є завжди залежним від першого, оскільки на першому і другому кроці переходи відбуваються під дією тієї ж процедури активації. Функціонування алгоритму відбувається до досягнення кінцевої пари маркувань. Зупинка алгоритму відбувається за умови неможливості спрацювання переходу на першому або на другому кроці обробки поточної пари маркувань. Якщо алгоритм зупиняється на першому етапі обробки поточної пари маркувань, то вважають, що $(M_i)_1 \sim (M_i)_2$, а у випадку зупинки на другому кроці — $(M_i)_1 \not\sim (M_i)_2$. У випадку, коли завжди можливо виконати перехід як на першому, так і на другому кроці, вважають, що $(M_i)_1 \sim (M_1)_2$.

РОЗДІЛ 4

Алгебра процесів

4.1. Неформальний опис алгебри процесів

Алгебра процесів для моделювання складних дискретних систем і процесів використовує деякі властивості класичних базових алгебр процесів [10-12], а також нові тенденції розвитку, що виникли при застосуванні алгебр процесів до розв'язування проблем імітаційного моделювання складних систем [60-62]. Парадигми алгебри процесів припускають наявність у системі активних компонентів і взаємодію між ними. У [10, 11] активні компоненти називаються процесами, і вони здійснюють активності, що відображають дискретні дії системи.

Будь-яка активність в даній алгебрі процесів може належати множині взаємодії Λ або множині внутрішніх активностей Ψ . Множина взаємодії містить підмножину активностей вводу Δ та підмножину активностей виводу $\bar{\Delta}$: $\Lambda = \Delta \cup \bar{\Delta}$.

Власне акт взаємодії відбувається тільки за наявності джерела та приймача інформації, що зумовлює його бінарний характер. Тому кожна активність $a(x) \in \Delta$ завжди має сполучену активність $\bar{a}(x) \in \bar{\Delta}$, і це саме та сполучена активність, з якою готова синхронізуватися активність $a(x)$. Дужки (\cdot) містять значення змінної, що підлягає вводу або виводу, і можуть випускатись у випадках, коли важливим є тільки факт взаємодії.

Множина внутрішніх активностей Ψ складається з підмножини обробки Φ та підмножини затримок T : $\Psi = \Phi \cup T$.

Підмножина обробки включає активності обробки типу $\phi \in \Phi$. Кожна активність такого типу представляє певні дії процесу по перетворенню інформації. Виходячи з даного означення робимо висновок, що активність обробки може також

представляти процеси, і тим самим забезпечує агрегативність структур опису процесів. В алгебрі процесів можна виключити внутрішню поведінку процесу і розглядати його як „чорну шухляду”. У цьому випадку виникає можливість побудувати складну імітаційну модель, виключивши з опису більш прості частини, поведінка яких не представляє інтересу при дослідженні. Така невидима поведінка моделюється за допомогою активностей $\tau \in T$, що заміняють внутрішні дії затримкою, яка виражена в одиницях модельного часу. Активності τ також використовують для явного задавання модельного часу при виконанні активностей обробки.

Таким чином, повний алфавіт активностей $\text{Act} = \Lambda \cup \Psi$ об'єднує множину активностей взаємодії Λ і множину внутрішніх активностей Ψ .

Процес в алгебрі процесів задають виразом, у який входять активності в якості незалежних чи залежних змінних. Найпростіший константний процес, що не може виконувати жодної активності і призводить до звільнення всіх ресурсів, задається активністю nil , $\text{nil} \in \Phi$. За наявності у процесі більш ніж однієї активності послідовність їх здійснення регулюється префіксними записами $\varphi.P$, $\langle \tau \rangle.P$, $a(x).P$ і $\bar{a}(x).P$. Зміст префіксного запису полягає в тому, що він представляє процес, який може виконати спочатку активність φ , $\langle \tau \rangle$, $a(x)$ або $\bar{a}(x)$, а потім функціонувати, як P . Прикладом елементарного процесу P_1 є вираз: $P_1 := \text{begin}.\text{end}.\text{nil}$, де $\{\text{begin}, \text{end}, \text{nil}\} \subset \Phi$.

Символ $:=$ означає присвоєння значення виразу в правій частині процесу P_1 . Дії процесу відповідно до даного виразу складаються з прямої послідовності трьох активностей. Активність begin виконує дії, пов'язані з початком функціонування процесу P_1 , end виконує дії, пов'язані з завершенням процесу P_1 , nil звільняє ресурси і виключає процес P_1 з моделі.

Для активностей, що здійснюють взаємодію з іншими процесами, важливу роль відіграє узгодження їх дій у часі. З цією метою введемо префіксну конструкцію $\langle \tau \rangle.P$. Вона представляє собою процес, що повинен бути затриманий на τ одиниць часу, а потім виконуватися як процес P . Прикладом процесу з такою префіксною конструкцією може служити процес $P_2 := \text{begin}.\langle 2 \rangle.\text{end}.\text{nil}$, подібний у своїх діях до процесу P_1 за винятком того, що здійснення активності end відбувається з затримкою, яка дорівнює двом одиницям модельного часу.

Для реалізації механізму зовнішньої синхронізації роботи процесу використовують дві синтаксичні конструкції. Перша з них, $P[t/\tau]$, одержала назву синтаксичної підстановки. Зміст її полягає в тому, що в процесі P всі входження змінної τ замінюються виразом t . Друга синтаксична конструкція представлена операцією префіксації $\langle \tau \leftarrow t \rangle.P$, яка регулює момент введення заміни. Наприклад, розглянемо процес $P_3 := \text{begin}.\langle \tau \leftarrow 2 \rangle.\text{work}.\langle \tau \rangle.\text{end}.\text{nil}$.

У даному процесі після здійснення активності begin відбувається операція заміни $P[t/\tau]$, в результаті якої процес P_3 набуває вигляду $\text{work}.\langle 2 \rangle.\text{end}.\text{nil}$. Після виконання активності work функціонування процесу P_3 буде затримано на 2 одиниці модельного часу. Відмінність процесу P_3 від процесу P_2 полягає у тому, що величина затримки τ невідома до моменту виконання операції заміни.

Нехай, наприклад, активність $\text{work} := \varphi = \sqrt{|x|}$, де $x \in R$. Тоді процес $P_4 := \text{begin}.a(x).\text{work}.\text{end}.\text{nil}$ забезпечує обчислення φ для довільного дійсного числа, яке вводиться за допомогою активності $a(x)$.

Якщо виникає потреба виводу результату обчислень, то необхідно включити в процес активність виводу $\bar{b}(\varphi)$:

$$P_5 := \text{begin} .a(x) .\text{work} .\bar{b}(\varphi) .\text{end} .\text{nil} .$$

Розглянуті конструкції дозволяють будувати залежні послідовності активностей і затримок, за допомогою яких можна моделювати складні алгоритми впливу одних процесів на інші.

Для моделювання систем з паралельним функціонуванням компонентів синтаксис алгебри доповнюють операціями вибору. Операція $P + Q$ представляє процес, що чекає готовності процесів P або Q взяти участь у деякій активності. Після цього відбувається *недетермінований вибір* між ними. Застосувавши префіксний запис, наведемо приклад процесу P_6 , який використовує цю операцію: $P_6 := \text{begin}_1 .P_1 + \text{begin}_2 .P_2$.

Цей процес пропонує вибір між двома альтернативами, контрольований початковими активностями складових процесів і незалежний від зовнішнього впливу.

Для того, щоб конструювати складні процеси, необхідно також мати можливість паралельного запуску кількох підпроцесів, забезпечуючи механізми взаємодії і синхронізації. З цією метою використовують паралельну конструкцію $P \mid Q$, яка представляє процеси з паралельним розвитком. Процесам P і Q дозволено виконувати будь-які індивідуальні дії, проте при наявності в них сполучених активностей виконується синхронізація шляхом здійснення цих активностей.

До загальних операцій, які використовують у різних алгебрах процесів, відносять перейменування $P[K]$ і стискання $P \setminus A$. Процес $P[K]$ поводитья так, як P , але у всіх своїх активностях перейменовується функцією K . Найпростішим

видом перейменування є заміна $P[t/\tau]$. Процес $P \setminus A$ поводитьсь так, як P , але не дозволяє виконувати жодних активностей з A . Остання конструкція може бути використана для організації примусової синхронізації. Виключивши з процесу P за допомогою операції $P \setminus \{a\}$ всі послідовності, що починаються з a чи \bar{a} , дозволяють розвиватися паралельно і незалежно тільки тим частинам процесу, які не вимагають попередньої синхронізації.

Розглянутий неформальний опис основних синтаксичних операцій дає загальне уявлення про цей інструмент формалізації. Висока ефективність даного підходу стимулює розробників застосовувати алгебри процесів до різних моделей. На сьогоднішній день добре розроблені підходи до використання алгебри процесів для побудови стохастичних моделей паралельних структур [63] і для імітаційних моделей із синтетичними робочими навантаженнями, що базуються на імовірнісному розподілі параметрів вхідних завдань [64]. Проте такі підходи вимагають попередніх знань про систему і її робоче навантаження для одержання адекватних результатів моделювання. При використанні моделей на етапі розробки складної системи такі знання, як правило, відсутні. Тому становить значний інтерес створення алгебри процесів для імітаційного моделювання систем з реальним робочим навантаженням.

4.2. Синтаксис

Синтаксис алгебри процесів для моделювання складних систем з реальним робочим навантаженням базується на використанні таких типів змінних: Var — тип змінних для параметрів, Pvar — тип змінних для процесів.

Вирази для процесів об'єднані в множину Pexp , а вирази для представлення функціональних залежностей та часу складають множину виразів Exp .

Означення 4.1. Множина змінних типу Var — це множина змінних для тимчасового збереження і обробки параметрів, яка включає:

прості змінні: x, y, z, \dots ;

індексовані змінні: $x \{ i \}, y \{ i \}, z \{ i \}, \dots$,

де i — індекс;

посилання: $\hat{x}, \hat{y}, \hat{z}, \dots$

Означення 4.2. Множина змінних типу Pvar — це множина змінних для тимчасового збереження і обробки екземплярів процесів, яка включає:

прості змінні: X, Y, Z, \dots ;

індексовані змінні: $X \{ i \}, Y \{ i \}, Z \{ i \}, \dots$, де i — індекс;

посилання: $\hat{X}, \hat{Y}, \hat{Z}, \dots$

Означення 4.3. Множина виразів часу Expr — це множина виразів, заданих на множині елементарних математичних операцій.

Означення 4.4. Множина виразів процесів Pexpr — це множина, що включає такі операції:

— префіксація: $\varphi.P, \langle \tau \rangle.P, \langle \tau \leftarrow t \rangle.P, a(x).P, \bar{a}(x).P$;

— вибір: $\sum_{i \in I} P_i, \sum_{i \in I} v_i! P_i$;

— паралельна композиція та взаємодія: $P \mid Q \mid \dots \mid C$,

$P \xleftrightarrow{L} Q \mid \dots \mid C, P \triangleright_L Q \mid \dots \mid C, P \triangleleft_L Q \mid \dots \mid C$;

— перейменування та рестрикція: $P[K], P \setminus A$;

— клонування, включення та приховання:

$k \times P, C \{ P \}, \phi_P(C)$;

— рекурсія та цикл: $\mathfrak{R}(X = P)$, $\text{cycle}[x](X = P)$, де
 $x, \tau, v, w, k \in \text{Var}, X \in \text{Pvar}, P, Q, C \in \text{Pexp}$,
 $a(x), \bar{a}(x), \varphi, \langle \tau \rangle \in \text{Act}$,

$\text{Act} = \Lambda \cup \Psi, L \subseteq \Lambda, I$ — скінченна множина індексів,
 K — функція перейменування $K : \text{Act} \rightarrow \text{Act}, A \subset \text{Act}$ —
 підмножина активностей.

Змінні процесу: $X, X \{i\}, \hat{X}$.

Формальний опис алгебри процесів допускає наявність скінченної множини змінних, значення яких визначаються за допомогою операції $X := P$. Даний вираз надає змінній X поведінку процесу P . X позначає просту змінну процесу. Змінні складного типу можуть відображати індексні структури та посилання. Індексна змінна процесу має вигляд $X \{i\}$, де i — значення індексу. Посилання \hat{X} завжди містить адресу розміщення в пам'яті першої активності процесу X .

Префіксація: $\varphi.P, \langle \tau \rangle.P, \langle \tau \leftarrow t \rangle.P, a(x).P, \bar{a}(x).P$

Префіксація — це базовий механізм, за допомогою якого конструюється поведінка процесу. Задамо такі різновидності префіксації.

Означення 4.5. Тривіальною префіксацією $\varphi.P$ будемо називати операцію, яка представляє процес як послідовність, що складається з активності φ і процесу P . Першою завжди виконується активність φ , а після цього процес поводить себе так, як процес P .

Означення 4.6. Префіксна затримка $\langle \tau \rangle.P$ — це операція, яка задає процес з початковою затримкою на τ одиниць модельного часу.

Означення 4.7. Параметрична префіксна затримка $\langle \tau \leftarrow t \rangle.P$ описує дії процесу, для якого спочатку визначається значення часового параметру τ шляхом присвоєння йому результату обчислення виразу t . Після затримки процес поводитья як процес P .

Означення 4.8. Префіксний ввід $a(x).P$ задає ввід деякого значення k з наступною заміною $P[k/x]$.

Означення 4.9. Префіксний вивід $\bar{a}(x).P$ задає дії процесу по виводу того значення змінної x , яке вона набула на момент здійснення активності виводу $\bar{a}(x)$.

Здійснення префіксацій $\varphi.P$ та $\langle \tau \rangle.P$ не залежить від стану зовнішнього середовища, а визначається виключно внутрішніми параметрами процесу. Тому їх використовують для опису незалежних процесів. Префіксації $\langle \tau \leftarrow t \rangle.P$, $a(x).P$ та $\bar{a}(x).P$ відображають характер взаємодії процесів із зовнішнім середовищем. Така взаємодія може бути виражена у вигляді часової або функціональної синхронізації. Синхронізацію в часі задають шляхом параметричної заміни величини затримки початку розвитку в одному з паралельних процесів. Префіксація вводу $a(x).P$ та префіксація виводу $\bar{a}(x).Q$ утворюють дуальні пари, які належать різним процесам. Розвиток процесу, що містить активність вводу $a(x)$ або активність виводу $\bar{a}(x)$, припиняється до моменту встановлення факту готовності дуальної пари. Після цього відбувається передача інформації, і розвиток обох процесів продовжується.

Вибір $\sum_{i \in I} P_i$, $\sum_{i \in I} v_i! P_i$.

Операції вибору призначені для вибору одного процесу з множини процесів з метою його подальшого розвитку.

Означення 4.10. Недетермінований вибір $\sum_{i \in I} P_i$ забезпечує визначення одного з процесів $P_1 + \dots + P_i + \dots + P_I$ в момент часу, який називається моментом вибору.

Означення 4.11. Керований вибір $\sum_{i \in I} v_i! P_i$ забезпечує вибір того процесу P_i , для якого відповідна логічна змінна v_i набуває істинного значення.

Множина активностей процесу вибору формується шляхом об'єднання множин складових процесів.

$$\text{Act} \left(\sum_{i \in I} P_i \right) := \text{Act}(P_1) \cup \dots \cup \text{Act}(P_i) \cup \dots \cup \text{Act}(P_I).$$

Умова виконання активності $\alpha \in \text{Act} \left(\sum_{i \in I} P_i \right)$ вимагає приналежності даної активності до множини активностей одного з процесів $\alpha \in \text{Act}(P_k)$, $k \in I$.

Суть операції вибору полягає у відображенні конкуренції елементів системи за володіння прихованим ресурсом. Тому у випадку наявності однакових активностей в множинах різних процесів спрацьовує тільки та активність, яка першою одержала доступ до прихованого ресурсу.

Паралельна композиція та взаємодія:

$$P \mid Q \mid \dots \mid C, P \leftrightarrow_L Q \mid \dots \mid C, P \triangleright_L Q \mid \dots \mid C, P \triangleleft_L Q \mid \dots \mid C .$$

Операції паралельної композиції та взаємодії означають взаємну поведінку процесів, що розвиваються паралельно. Операцію $P \mid Q \mid \dots \mid C$ застосовують у випадках, коли немає необхідності акцентувати увагу на характері взаємодії між паралельними процесами або коли такої взаємодії не існує. Операція може застосовуватись для означення паралелізму групи процесів без деталізації характеру зв'язків між ними. Операція взаємодії визначає поведінку сукупного процесу, який складається з двох або більшого числа процесів, що розвиваються паралельно.

Означення 4.12. Паралельною композицією називають систему $P \mid Q \mid \dots \mid C$, яка відображає одночасне існування групи процесів, що можуть розвиватися незалежно або взаємодіяти один з одним.

Означення 4.13. Синхронною взаємодією $P \leftrightarrow_L Q \mid \dots \mid C$ процесу P з паралельною композицією процесів $Q \mid \dots \mid C$ називають систему, що складається з процесу-передавача P і множини процесів-приймачів $Q \mid \dots \mid C$, в якій передача відбувається від передавача до приймачів у випадку одночасного звернення комплементарних подій з множини $L \subseteq \Lambda$.

На відміну від операції вибору при взаємодії розглядають процеси, активності яких користуються власними прихованими ресурсами. Це дозволяє їм розвиватися незалежно і використовувати свої ресурси у випадку, якщо події, що їх спричиняють, не входять до множини подій взаємодії. При виникненні події, що входить до множини подій взаємодії, відбувається синхронізація розвитку паралельних процесів з метою виконання спільної активності.

Означення 4.14. Керованою передачею $P \triangleright_L Q | \dots | C$ від процесу P до паралельної композиції $Q | \dots | C$ називають систему, що складається з процесу-передавача P і множини процесів-приймачів $Q | \dots | C$, які паралельно розвиваються і взаємодіють в момент звершення в процесі P активності, що входить до спільної множини подій взаємодії $L \in \Lambda$.

Головна відмінність цієї операції від операції синхронної взаємодії полягає в тому, що момент передачі даних відбувається безпосередньо в момент звершення активності виводу $\bar{a}(x)$ в процесі P , а композиція процесів $Q | \dots | C$ завжди готова до прийому інформації.

Означення 4.15. Асинхронною взаємодією $P \triangleleft_L Q | \dots | C$ процесу P і паралельної композиції $Q | \dots | C$ називають систему, що складається з процесу P і множини процесів $Q | \dots | C$, які паралельно розвиваються і здатні передавати або приймати інформацію від процесу P відповідно до спільної множини подій взаємодії $L \subseteq \Lambda$ незалежно один від одного.

При асинхронній взаємодії кожен з процесів, що входить в систему, може ініціювати активність вводу або виводу незалежно від готовності дуальної активності. Тому затримка для синхронізації при такій організації взаємодії процесів не виникає.

Перейменування та стискання: $P[K]$, $P \setminus A$.

Операції перейменування та стискання відносять до операцій групового редагування, оскільки вони забезпечують модифікацію групи активностей процесу одночасно.

Означення 4.16. Операцією перейменування $P[K]$ називають систему, що представлена процесом P , усі активності якого перейменовано за допомогою функції K .

Означення 4.17. Операцією стискання $P \setminus A$ називають систему, яка представлена процесом P , для якого усі активності, що належать множині активностей A , заблоковані від виконання.

Головним призначенням даних операцій являється модифікація поведінки групи об'єктів моделі в ході моделювання. Наприклад, якщо множина взаємодії $A \subset \Lambda$, то за допомогою операції стискання можливо керувати синхронізацією обміну даними між процесами.

Клонування, включення та приховання:

$$k \times P, C\{P\}, \varphi_L(P).$$

Операції клонування, включення та приховання призначені для динамічної модифікації множини компонентів системи в ході моделювання.

Означення 4.18. Операцією клонування $k \times P$ називають систему, яка складається з k ідентичних процесів P , що розвиваються паралельно і можуть взаємодіяти між собою.

Означення 4.19. Операцією включення $C\{P\}$ називають систему, яка складається з процесу C , що включає процес P в якості підпроцесу.

Означення 4.20. Операцією приховання $\varphi_P(C)$ називають систему, що складається з процесу C , у якому всі активності, що входять до підпроцесу P , замінюються внутрішньою активністю φ .

Рекурсія та цикли: $\mathfrak{R}(X = P), \text{cycle}[x](X = P)$.

Операції рекурсії та циклу призначені для багатократного повторення деякої послідовності активностей, які входять в процес P .

Означення 4.21. Операцією рекурсії $\mathfrak{R}(X = P)$ називають систему, що складається з рекурсивно повторюваної послідовності активностей процесу P , присвоєних змінній типу Pvar.

Означення 4.22. Операцією циклу $\text{cycle}[x](X = P)$ називають систему, що складається з циклічно повторюваної x разів послідовності активностей процесу P , присвоєних змінній типу Pvar.

Операція рекурсії може задавати нескінченну або скінченну послідовність процесів. Скінченна послідовність виникає у випадку, коли в ході виконання чергового екземпляра процесу справджується умова завершення рекурсії. Отже, умовою завершення рекурсії завжди є внутрішній стан процесу. Операція циклу відрізняється від операції рекурсії наявністю змінної x , яка обмежує кількість повторювань виконання процесу P .

Для запису алгебраїчних виразів будемо використовувати дужки для того, щоб уникнути неоднозначностей або для покращення сприймання послідовності операцій.

Операції алгебри процесів традиційно мають такий статус:

1. Операції префіксації.
2. Операції клонування та включення.
3. Операції перейменування та стискання.
4. Операції вибору.
5. Операції взаємодії.
6. Операції рекурсії та циклу.

4.3. Семантика алгебри процесів

Операційна семантика процесів моделювання, представлених даною алгеброю, зводиться до означення переходів, які характеризують зміну стану імітаційної моделі. Приймаючи в якості базового представлення марковану систему з переходами, визначатимемо динаміку виконання операцій в термінах нотації Плоткіна [45], що представляє операцію у вигляді дробу, знаменник якої містить загальний запис операції, а чисельник — можливі варіанти її виконання.

Операції префіксації:

$$\overline{\varphi.P \longrightarrow P}, \quad (4.1)$$

де φ — довільна активність обробки;

$$\overline{\langle \tau \rangle . P \longrightarrow P}, \quad (4.2)$$

де $\langle \tau \rangle$ — константа затримки;

$$\overline{\langle \tau \leftarrow t \rangle . P \longrightarrow P \left[t / \tau \right]}, \quad (4.3)$$

де $\langle t \rangle$ — вираз затримки;

$$\overline{a(x).P \xrightarrow{a(k)} P \left[k / x \right]}, \quad (4.4)$$

де k — значення для заміни x ;

$$\overline{\bar{a}(x).P \xrightarrow{\bar{a}(m)} P}, \quad (4.5)$$

де m — значення змінної x .

Відповідно до даної нотації операції префіксації не мають чисельника, оскільки існує єдиний варіант їх виконання, що полягає в продовженні поточного процесу після виконання активності. Відмінності між даними операціями зумовлені різними станами змінних процесу і способами їх заміщення. На рис. 4.1 наведено приклади мережної інтерпретації операцій префіксації:

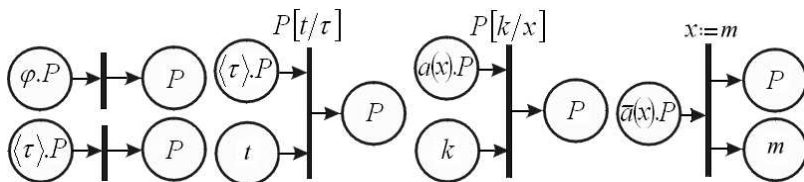


Рис.4.1. Операції префіксації

Розглянемо динаміку виконання операцій вибору, використовуючи нотацію алгебри процесів і її мережне представлення. Кількість варіантів виконання недетермінованої операції вибору дорівнює кількості процесів, що беруть участь у даній операції:

$$\frac{P_1 \xrightarrow{\alpha} P'_1}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'_1}, \dots, \frac{P_k \xrightarrow{\beta} P'_k}{\sum_{i \in I} P_i \xrightarrow{\beta} P'_k}, \dots, \frac{P_l \xrightarrow{\gamma} P'_l}{\sum_{i \in I} P_i \xrightarrow{\gamma} P'_l} \quad (4.6)$$

Мережна інтерпретація цієї операції за допомогою елементів APRO-мережі (рис.4.2) представляє собою комутаційний недетермінований вибір одного з процесів, що претендують на продовження.

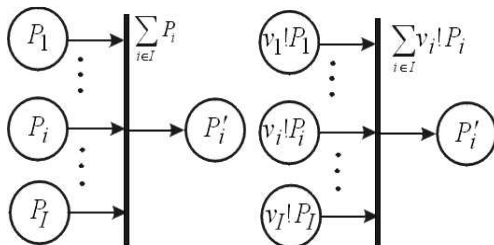


Рис.4.2. Операції вибору

Операція керованого вибору в нотації алгебри процесів у знаменнику містить загальний запис можливого вибору. Чисельник вказує на здійснення вибору того процесу, для якого відповідна логічна змінна v_i набуває істинного значення:

$$\frac{P_1 \xrightarrow{v_1} P'_1, \dots, P_k \xrightarrow{v_k} P'_k, \dots, P_l \xrightarrow{v_l} P'_l}{\sum_{i \in I} v_i ! P_i \xrightarrow{v_i} P'_i} \quad (4.7)$$

Структурний механізм керованого вибору використовує значення логічної змінної як аргумент функції вибору.

Паралельна композиція процесів описується двома видами нотацій. Нотація першого виду дозволяє описати розвиток процесу під дією довільної активності a за умови, що дана активність не ініціює взаємодію з іншими процесами даної групи.

$$\left\{ \begin{array}{l} \frac{P \xrightarrow{a} P'}{P | Q | \dots | C \xrightarrow{a} P' | Q | \dots | C} \\ \frac{Q \xrightarrow{a} Q'}{P | Q | \dots | C \xrightarrow{a} P | Q' | \dots | C} \\ \frac{C \xrightarrow{a} C'}{P | Q | \dots | C \xrightarrow{a} P | Q | \dots | C'} \end{array} \right. \quad (4.8)$$

Нотація другого виду задає загальну формулу взаємодії між двома процесами, що входять в групу паралельних процесів:

$$\frac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\bar{a}(x)} Q'}{P | Q \xrightarrow{\tau} P' | Q'} \quad (4.9)$$

Конкретизація допустимих видів взаємодії реалізована за допомогою операцій взаємодії. Динаміка взаємодії в алгебрі процесів включає операції керованої передачі, синхронної взаємодії та асинхронної взаємодії. Відсутність взаємодії розглядається в даному випадку як паралельне виконання процесів. Нотації цих операцій представлені виразами (4.10)-(4.12).

Керована передача:

$$\frac{P \xrightarrow{\bar{a}(x)} P' \quad Q \xrightarrow{a(x)} Q' \quad \dots \quad C \xrightarrow{a(x)} C'}{P \triangleright_L Q | \dots | C \xrightarrow{\tau} P' \triangleright_L Q' | \dots | C'} \quad (4.10)$$

Синхронна взаємодія:

$$\frac{P \xrightarrow{\bar{a}(x)} P' \quad Q \xrightarrow{a(x)} Q' \quad C \xrightarrow{a(x)} C'}{P \xleftarrow{L} Q \mid \dots \mid C \xrightarrow{\tau} P' \xleftarrow{L} Q' \mid \dots \mid C'} \quad (4.11)$$

Асинхронна взаємодія:

$$\frac{P \xrightarrow{\bar{a}(x)} P' \quad Q \xrightarrow{a(x)} Q' \quad \dots \quad C \xrightarrow{a(x)} C'}{P \triangleleft_L Q \mid \dots \mid C \xrightarrow{\tau} P' \triangleleft_L Q' \mid \dots \mid C'} \quad (4.12)$$

На рис.4.3 показано мережне представлення операцій взаємодії.

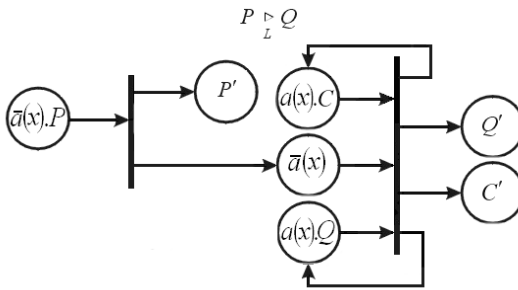


Рис.4.3а. Операція керованої передачі

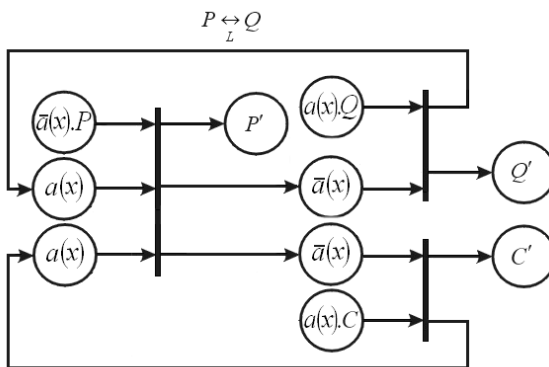


Рис.4.3б. Операція синхронної взаємодії

Керована передача $P \underset{L}{\triangleright} Q | \dots | C$ реалізована у вигляді зв'язку переходів через позицію, що містить активність $\bar{a}(x) \in L$. Тому ініціатором акту передачі є префіксний процес $\bar{a}(x).P$. Процеси паралельної композиції $Q | \dots | C$ виступають в якості постійно готових до прийому приймачів інформації завдяки префіксації $a(x).Q | \dots | a(x).C$. Передача відбувається шляхом одночасного звернення активності $\bar{a}(x)$ в процесі-передавачі та активностей $a(x)$ в процесах-приймачах. Інакше композиція процесів $Q | \dots | C$ буде знаходитися в стані очікування прийому даних.

У випадку синхронної взаємодії переходи мережі містять симетричні зв'язки через сполучені активності. Тому обидва процеси можуть ініціювати взаємодію. Перший із тих процесів, що прибули в точку взаємодії, відсилає процесу-партнеру активність і зупиняється до одержання сполученої активності на вхідну позицію переходу. Прибуття очікуваної активності на вхідну позицію говорить про те, що партнер по взаємодії готовий до обміну даними.

Операція асинхронної взаємодії додатково представлена переходом, що моделює функції розподіленого ресурсу. Активнація переходу відбувається після надходження на одну з його вхідних позицій мітки з атрибутом активності $a(x)$ або $\bar{a}(x)$. Після завершення роботи на вихідні позиції завжди видаються мітки з атрибутами $a(x)$ і $\bar{a}(x)$, що дає можливість продовження розвитку того процесу, який ініціював взаємодію. Передана інформація знаходиться в розподіленому ресурсі і може бути видана процесу-приймачу в момент активації в ньому активності $a(x)$.

Операції перейменування і стискання представлені в алгебрі процесів нотаціями:

$$\frac{P \xrightarrow{\alpha} P'}{P[K] \xrightarrow{K(\alpha)} P'[K]}, \quad (4.13)$$

$$\frac{P \xrightarrow{\alpha} P'}{P \setminus A \xrightarrow{\alpha} P' \setminus A} \left(\{ \alpha, \bar{\alpha} \} \not\subset A \right). \quad (4.14)$$

На відміну від операцій вибору і взаємодії ці операції працюють з одним процесом, але призначені для організації групових модифікацій його активностей. Джерелом модифікацій для операції перейменування є функція K , а для операції стискання – множина активностей A . Очевидно, що в мережному представленні ці джерела можуть виступати у вигляді атрибутів міток, які ініціюють дані операції (рис.4.4).

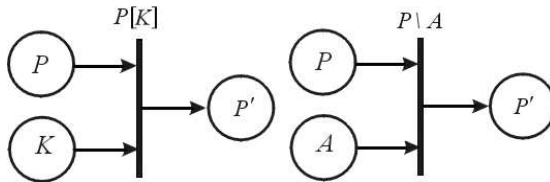


Рис.4.4. Операції перейменування і стискання

Динаміка операції клонування процесів представлена виразом:

$$\frac{P \xrightarrow{\alpha} P'_1 \quad P \xrightarrow{\alpha} P'_2 \quad \dots \quad P \xrightarrow{\alpha} P'_k}{P \xrightarrow{\alpha} P'_1 | P'_2 | \dots | P'_k}. \quad (4.15)$$

Суть операції $k \times P$ зводиться до генерації групи з k процесів на основі шаблону P . Мережне представлення операції клонування показано на рис.4.5.

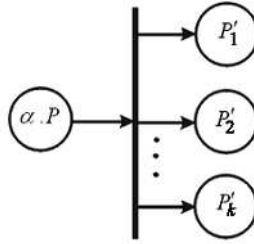


Рис.4.5. Операція клонування

Операції рекурсії і циклу представлені нотаціями, що відображають динаміку їх виконання.

$$\frac{P \{ \Re(X = P) / X \} \xrightarrow{\alpha} P'}{\Re(X = P) \xrightarrow{\alpha} P'} \text{— рекурсія, (4.16)}$$

$$\frac{P \{ \text{cycle}[x-1](X = P) / X \} \xrightarrow{\alpha} P'}{\text{cycle}[x](X = P) \xrightarrow{\alpha} P'} (x \geq 0) \text{— цикл. (4.17)}$$

Кожна з операцій складається з трьох основних етапів: ініціалізації елемента циклу шляхом присвоєння значень процесу змінній X , виконання створеного екземпляра процесу і перевірки умов завершення. На рис.4.6 видно, що операція циклу відрізняється від рекурсії наявністю службової позиції з атрибутом мітки, яка забезпечує ведення обліку кількості ітерацій.

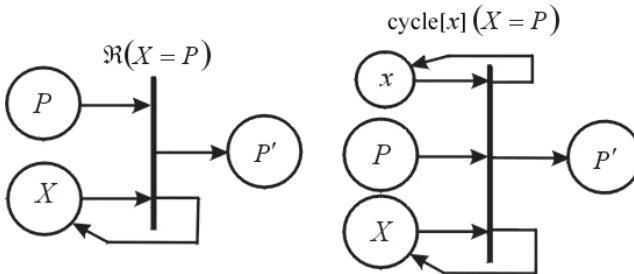


Рис.4.6. Операції рекурсії і циклу

Перевірка умов завершення рекурсивної операції завжди залежить від внутрішнього стану процесу P , а операція циклу для завершення роботи використовує значення лічильника x .

Операція включення $C\{P\}$ забезпечує можливість ієрархічної побудови опису системи шляхом об'єднання процесів, що входять до різних рівнів стратифікації моделі, яка описує дану систему. Динаміка виконання даної операції відбувається відповідно до нотації:

$$\frac{C \xrightarrow{a(X)} C'}{C\{X\} \xrightarrow{a(P)} C'\{P\}}, \quad X \in \text{Pvar}. \quad (4.18)$$

Отже, в ході виконання активності, яка ініціює операцію включення, відбувається присвоєння змінній процесу X значення деякого процесу P , посилання на який містить згадана активність. Подальший розвиток процесу C буде використовувати процес P як підпроцес. Мережне представлення даної операції показано на рис.4.7.

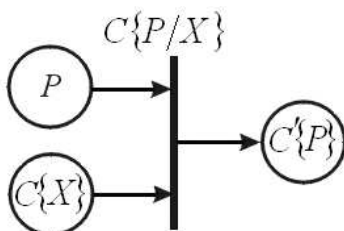


Рис.4.7. Операція включення

Операція приховання в деякому сенсі може розглядатися як зворотна до операції включення. Вона також використовується при побудові та дослідженні стратифікованих моделей. За допомогою цієї операції виникає можливість приховувати деталізацію функціонування деяких компонентів системи з метою спрощення процесу моделювання. Операція приховання представлена нотацією:

$$\frac{C \xrightarrow{\alpha} C'}{\varphi_P(C) \xrightarrow{\varphi} \varphi_P(C')}, \quad (4.19)$$

де $\alpha \in P$.

Виконання останньої операції зводиться до заміни деякої послідовності активностей, які утворюють підпроцес P , однією внутрішньою активністю φ . Мережне представлення даної операції показано на рис.4.8.

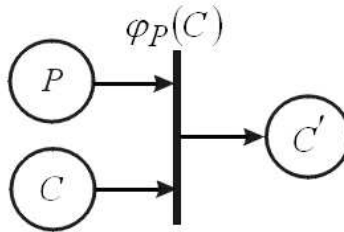


Рис.4.8. Операція приховання

4.4. Еквівалентність поведінки

Синтаксис та семантика алгебри процесів задають спосіб представлення об'єктів моделі. Але для того, щоб модель була адекватною, необхідно дотримуватись відповідних принципів еквівалентності поведінки. Поняття еквівалентності, що використовується для порівняння складних систем, базується на еквівалентному відношенні.

Означення 4.23. Нехай P — процес, а бінарне відношення R на P — це підмножина множини $P \times P$ з елементами $(p, q) \in R$ або pRq . Тоді еквівалентним відношенням R на P будемо називати відношення, яке задовольняє такі умови:

- R рефлексивне, тобто pRp при $p \in P$;
- R симетричне, тобто з pRq випливає, що qRp при $p, q \in P$;

— R транзитивне, тобто з pRq і qRz випливає, що pRz при $p, q, z \in P$.

Умова рефлексивності бінарного відношення є базовою умовою, яка забезпечує внутрішню несуперечливість процесів, що еквівалентно відображають внутрішньо несуперечливі процеси. Умова транзитивності забезпечує збереження еквівалентності під час покрокової реалізації моделі за її специфікацією. Якщо задана специфікація фрагменту системи, то завдяки властивості транзитивності еквівалентне відношення сукупності таких фрагментів буде еквівалентним вихідній специфікації. Для того, щоб об'єкт моделювання та його специфікація були взаємно подібними, очевидна необхідність виконання принципу симетричності.

Для означення еквівалентності поведінки застосовують також конгруентність, тобто збереження еквівалентного відношення на різних рівнях стратифікованих моделей.

Означення 4.24. Нехай P і Q — процеси, пов'язані еквівалентним відношенням $P R Q$. Такі процеси будемо називати конгруентними, якщо еквівалентне відношення зберігається при $C\{P\} R C\{Q\}$, де $C\{ \}$ — процес вищого рівня, і він використовує процес P або Q як підпроцес.

Завдяки конгруентності еквівалентність поведінки може визначатися як видима поведінка процесів, а не через їх структуру, виражену кількістю допустимих станів та переходів між ними. При такому підході еквівалентні процеси розпізнаються спостерігачем як такі, що однаково взаємодіють з навколишнім середовищем, і їх взаємодії ведуть до однакового результату.

Для формальних засобів опису паралельних процесів та структур принципи конгруентної поведінки є основою аналізу складних систем. Будемо розуміти конгруентність як відношення еквівалентності, що відповідає властивостям рефлексивності, симетрії та транзитивності. Конгруентність поведінки у більшо-

сті алгебр процесів [65, 66] визначають, виходячи з відповідності послідовностей активностей, що утворюють процеси. В даній системі позиції однозначно відповідають станам системи: $Q = \{q_0, \dots, q_n\}$, а переходи забезпечують зміну станів шляхом виконання активностей $A = \{a_0, \dots, a_m\}$ за умови наявності відповідних міток: $M = \{\mu_0, \dots, \mu_k\}$. Позначимо через $S(q)$ сукупність послідовностей станів, в яких для кожного стану системи q_i існує така активність a_i , що $q_i \xrightarrow{a_i} q_{i+1}$. Тоді дві системи мають конгруентну поведінку, якщо $S(q) = S(p)$ для відповідних послідовностей станів q і p .

Сьогодні відома велика кількість означень еквівалентності поведінки [65,66], що є актуальними в залежності від мети побудови моделі. Частина з них використовують для аналізу паралельних систем, а інші відображають аспекти поведінки компонентів згаданих систем. Найпростіший вид еквівалентності поведінки може бути зведений до порівняння послідовностей активностей, які входять до складу тих процесів, що підлягають порівнянню. Такі послідовності називають трасами або протоколами активностей [67]. Оскільки алгебра процесів може відображати нескінченну кількість систем, які відповідають її синтаксичним конструкціям, то для конкретного визначення еквівалентної поведінки розглянемо марковану систему з

переходами, задану кортежем $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$, де

Π — множина процесів; A — множина активностей; \xrightarrow{a} — множина переходів: $\xrightarrow{a} \subseteq \Pi \times A \times \Pi$.

Позначимо через U набір послідовностей, що складаються з елементів $a_i \in A$ при $1 \leq i \leq n$, $|A| = n$.

Означення 4.25. Нехай в маркованій системі з переходами $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$ існує послідовність активностей $u = a_1 \cdots a_n \in U$, $U \subseteq \Pi$, яка приводить до існування переходу $q \xrightarrow{u} q'$ за умови, що існує послідовність переходів $q_i \xrightarrow{a_{i+1}} q_{i+1}$, де $0 \leq i \leq n-1$, $q = q_0, q' = q_n$. Тоді u будемо називати трасою для q , якщо існує стан q' такий, що $q \xrightarrow{u} q'$.

Означення 4.26. Нехай $S(p)$ і $S(q)$ — множини всіх допустимих трас для станів p і q . Тоді для даних станів існує:

- строга еквівалентність поведінки $p \equiv q$ за умови, що $S(q) = S(p)$ і $a_i \in \text{Act}$;
- слабка еквівалентність поведінки $p \cong q$ за умови, що $S(q) = S(p)$ і $a_i \in \Lambda$.

Оскільки сукупність станів та переходів характеризує марковану систему з переходами, то поняття еквівалентності поведінки може бути поширене на всю систему.

Згадане означення еквівалентності поведінки може використовуватися тільки для порівняння моделей систем, які представлені детермінованими операторами, тобто такими операторами, що не містять невизначеностей. Проблема полягає у тому, що наявність однакових трас у недетермінованих моделях не гарантує еквівалентної поведінки. Наприклад, нехай $p := q_0 \cdot q_1 \cdot \text{nil} + q_0 \cdot q_2 \cdot \text{nil}$ і $r := q_0 \cdot (q_1 \cdot \text{nil} + q_2 \cdot \text{nil})$. Тоді $p \cong r$, оскільки

$$S(p) = S(r) = \{q_0, q_0q_1, q_0q_2\}.$$

Для p на момент виконання активності q_0 наступна активність чітко визначена, тоді як для r на момент виконання q_0 існує вибір між активностями q_1 і q_2 .

4.5. Строга взаємна подібність

Взаємна подібність, яка базується на вимозі не тільки еквівалентності трас, а й досягнення однакових станів після виконання відповідних активностей, вперше була запропонована в [60]. У випадку взаємної подібності з розгалуженням для еквівалентності відповідних процесів додається умова проходження однакових проміжних станів. Перевага застосування взаємної подібності полягає в можливості посилення на означення подібності, виходячи з загальної теорії систем та побудови доведення на основі цього означення. В залежності від того, чи беруться до уваги приховані активності, взаємну подібність поділяють на строгу та слабку.

Означення 4.27 (строга взаємна подібність).

Нехай в маркованій системі з переходами $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$ існує відношення $R \subseteq \Pi \times \Pi$ і процеси $P, Q \subseteq \Pi$ з допустимими станами $p, p' \in P, q, q' \in Q$ при $(p, q) \in R$. Тоді якщо для довільної активності a :

— для кожного переходу $p \xrightarrow{a} p'$ існує стан q' такий, що $q \xrightarrow{a} q'$ при $(p', q') \in R$,

— для кожного переходу $q \xrightarrow{a} q'$ існує стан p' такий, що $p \xrightarrow{a} p'$ при $(p', q') \in R$,

відношення R будемо називати строгою взаємною подібністю, якщо $a \in \text{Act}$.

Стани p і q будемо називати строго взаємно подібними, якщо існує відношення R , яке містить (p, q) за умови $a \in \text{Act}$. Позначатимемо строго взаємно подібні стани як $p \sim q$.

Доведемо основні властивості строгої взаємної подібності для маркованій системи з переходами.

Теорема 4.1. У маркованій системі з переходами $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$ строга взаємна подібність є еквівалентним відношенням.

Д о в е д е н н я. Нехай множина процесів Π маркованої системи з переходами містить процеси $P, Q \subseteq \Pi$, кожен з яких включає стани $p, p' \in P$ і $q, q' \in Q$. Для того, щоб стани $p \sim q$ відповідали еквівалентному відношенню на $R, (p, q) \in R$, необхідно, відповідно до означення 4.23, довести його рефлексивність, симетричність та транзитивність. Спираючись на означення 4.27, введемо заміну p' на p і q' на q . Тоді одержимо систему, в якій існує активність a , що спричиняє тотожні переходи між станами, а саме: $p \xrightarrow{a} p$ і $q \xrightarrow{a} q$. Очевидно, що в такій системі підмножинами множини R будуть бінарні відношення:

$$I_p = \{(p, p) \mid p \in P\}, I_q = \{(q, q) \mid q \in Q\}.$$

Оскільки $I_p, I_q \subseteq R$ є відношеннями ідентичності, то вони є рефлексивними. Для симетричності $p \sim q$ необхідно довести, що має місце $q \sim p$. Виходячи з умови строгої взаємної подібності можемо стверджувати, що $(p, q) \in R$. Тоді $R^{-1} = \{(q, p) \mid (p, q) \in R\}$. Очевидно, що R^{-1} також є відношенням взаємної подібності. Звідси випливає, що $p \sim q$ і $q \sim p$ є взаємно симетричними.

Якщо строга взаємна подібність є транзитивною, строгі взаємні подібності станів $p \sim q$ і $q \sim z$ зумовлюють $p \sim z$. Оскільки за умовою $p \sim q$, то $(p, q) \in R'$, відповідно, оскільки $q \sim z$, то $(q, z) \in R''$. Розглянемо бінарне відношення

$R = \{(p, z) \mid (p, q) \in R', (q, z) \in R''\}$ для деякого q . Оскільки R' і R'' є відношеннями строгої взаємної подібності, то і R є відношенням строгої взаємної подібності. Звідси випливає, що строга взаємна подібність транзитивна.

Теорема 4.2. *Найбільшою строгою взаємною подібністю будемо називати об'єднання всіх допустимих відношень взаємної подібності. Тоді у маркованій системі з переходами $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$ строга взаємна подібність є найбільшою строгою взаємною подібністю.*

Д о в е д е н н я. Розглянемо, як і в попередньому випадку, процеси $P, Q \subseteq \Pi$, кожен з яких включає стани $p, p' \in P$ і $q, q' \in Q$. Виходячи з умови теореми: $\sim = \cup \{R\}$, де R — відношення взаємної подібності. Отже, необхідно довести, що $\cup \{R\}$ також є відношенням взаємної подібності. Для цього повинні виконуватися такі умови:

— якщо $(p, q) \in \cup \{R\}$ і $p \xrightarrow{a} p'$, то існує стан q' такий, що $q \xrightarrow{a} q'$ при $(p', q') \in \cup \{R\}$,

— якщо $(p, q) \in \cup \{R\}$ і $q \xrightarrow{a} q'$, то існує стан p' такий, що $p \xrightarrow{a} p'$ при $(p', q') \in \cup \{R\}$.

Нехай $(p, q) \in \cup \{R\}$, тоді існує таке відношення взаємної подібності R , що $(p, q) \in R$. Оскільки R — це відношення взаємної подібності, і $p \xrightarrow{a} p'$, то існує стан q' такий, що $q \xrightarrow{a} q'$ при $(p', q') \in R$. Виходячи з властивостей симетричності для $q \xrightarrow{a} q'$ робимо висновок, що існує стан p' такий,

що $p \xrightarrow{a} p'$ при $(p', q') \in R$. Але пара $(p', q') \in \cup\{R\}$, тобто строга взаємна подібність включає всі відношення взаємної подібності і відповідно є найбільшою строгою взаємною подібністю.

Теорема 4.3. У маркованій системі з переходами $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$, що включає процеси $P, Q \subseteq \Pi$ з допустимими станами $p, p' \in P$ і $q, q' \in Q$, для довільної активності a два стани $p \sim q$ тоді і тільки тоді, коли:

— для переходу $p \xrightarrow{a} p'$ існує перехід $q \xrightarrow{a} q'$ такий, що $p' \sim q'$,

— для переходу $q \xrightarrow{a} q'$ існує перехід $p \xrightarrow{a} p'$ такий, що $p' \sim q'$.

Графічне зображення строгої взаємної подібності станів p і q показано на рис.4.9.

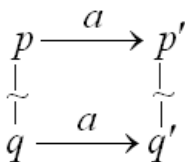


Рис.4.9. Строга взаємна подібність станів p і q

Д о в е д е н н я. Оскільки, відповідно до означення 4.25, для кожного переходу $p \xrightarrow{a} p'$ існує стан q' такий, що $q \xrightarrow{a} q'$ при $(p', q') \in R$, то для довільної активності a два стани $p \sim q$ тоді, коли для переходу $p \xrightarrow{a} p'$ існує перехід $q \xrightarrow{a} q'$ такий, що $p' \sim q'$. Аналогічно, якщо для кожного

переходу $q \xrightarrow{a} q'$ існує стан p' такий, що $p \xrightarrow{a} p'$ при $(p', q') \in R$, то для довільної активності a два стани $p \sim q$ тоді, коли для переходу $q \xrightarrow{a} q'$ існує перехід $p \xrightarrow{a} p'$ такий, що $p' \sim q'$. Отже, пряме твердження теореми безпосередньо впливає з означення строгої взаємної подібності. Для доведення зворотного твердження необхідно показати, що строга взаємна подібність $p \sim q$ містить відношення (p, q) . Доведення згаданого факту базується на конструктивному підході до формування множини R відношень взаємної подібності. Для доведення факту $p \sim q$ необхідно, перш за все, включити в множину R відношення (p, q) , оскільки для того, щоб пряме доведення було коректним, кожному переходу $p \xrightarrow{a} p'$ зі стану p повинен відповідати перехід $q \xrightarrow{a} q'$ зі стану q для деякого стану q' такого, що $(p', q') \in R$. Сформуємо спочатку множину R шляхом об'єднання всіх відношень взаємної подібності в маркованій системі з переходами $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$. Оскільки за теоремою 4.2 строга взаємна подібність є найбільшою взаємною подібністю, то $R = (p, q) \cup R$. Отже, множина відношень R містить елемент (p, q) . Теорему доведено.

Спираючись на теорему 4.2, можна зробити висновок, що в ході еволюції строго взаємно подібні системи залишаються строго взаємно подібними. На цьому факті базується означення строго взаємно подібних процесів.

Означення 4.28. Два процеси P і Q називають строго взаємно подібними і позначають $P \sim Q$ тоді і тільки тоді, коли

для довільної активності a існує відношення строгої взаємної подібності R таке, що $P'RQ'$ (рис. 4.10).

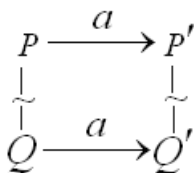


Рис.4.10. Строга взаємна подібність процесів P і Q

Теорема 4.4. Нехай P, Q, G — процеси. Якщо $P \sim Q$, то:

1. $a.P \sim a.Q$ для довільної активності a ;
2. $\langle \tau \rangle.P \sim \langle \tau \rangle.Q$ для довільної префіксної затримки $\langle \tau \rangle$;
3. $\langle \tau \leftarrow t \rangle.P \sim \langle \tau \leftarrow t \rangle.Q$ для довільного значення виразу $\langle t \rangle$;
4. $P + G \sim Q + G$ для довільного процесу G ;
5. $v_1!P + v_2!G \sim v_1!Q + v_2!G$ для довільного процесу G ;
6. $P \mid G \sim Q \mid G$ для довільного процесу G ;
7. $P[K] \sim Q[K]$ для довільної функції перейменування K ;
8. $P \setminus A \sim Q \setminus A$ для довільної множини активностей A ;
9. $k \times P \sim k \times Q$ для довільного коефіцієнта клонування k ;
10. $\mathfrak{R}(X = P) \sim \mathfrak{R}(X = Q)$, де X — змінна процесу;
11. $\text{cycle}[x](X = P) \sim \text{cycle}[x](X = Q)$, де x — індекс циклу.

Д о в е д е н н я.

1. $a.P \sim a.Q$.

Для доведення $a.P \sim a.Q$ використаємо умову рефлексивності строгої взаємної подібності. За цією умовою $a \sim a$ для довільної активності φ . Після виконання переходів

$a.P \xrightarrow{a} P$ та $a.Q \xrightarrow{a} Q$ одержимо $P \sim Q$ за умовою теореми. Отже, $a.P \sim a.Q$.

$$2. \langle \tau \rangle . P \sim \langle \tau \rangle . Q.$$

Аналогічним є доведення префіксної затримки. Виходячи з умови рефлексивності можемо записати $\langle \tau \rangle \sim \langle \tau \rangle$. Після виконання префіксних затримок $\langle \tau \rangle . P \xrightarrow{\langle \tau \rangle} P$ та $\langle \tau \rangle . Q \xrightarrow{\langle \tau \rangle} Q$ одержуємо систему $P \sim Q$. Отже, додавання однакової префіксної затримки не змінює умови еквівалентності двох процесів.

$$3. \langle \tau \leftarrow t \rangle . P \sim \langle \tau \leftarrow t \rangle . Q.$$

В даному випадку також розглянемо умову рефлексивності $\langle \tau \leftarrow t \rangle \sim \langle \tau \leftarrow t \rangle$. Після виконання префіксних затримок $\langle \tau \leftarrow t \rangle . P \xrightarrow{\langle t \rangle} P$ та $\langle \tau \leftarrow t \rangle . Q \xrightarrow{\langle t \rangle} Q$ одержуємо систему $P \sim Q$. Отже, операція параметричної затримки не змінює умови еквівалентності процесів.

$$4. P + G \sim Q + G.$$

Розглянемо тепер збереження властивості строгої взаємної подібності при виконанні операцій вибору $P + G \sim Q + G$. Для доведення даного факту необхідно побудувати відношення строгої взаємної подібності R , яке б містило пару процесів $(P + G, Q + G)$. Припустимо, що існує відношення $R = \{(P + G, Q + G) \mid P \sim Q\}$. Тоді доведення того, що R є відношенням строгої взаємної подібності, зводиться до доведення істинності умови: якщо $(P + G, Q + G) \in R$ і $P + G \xrightarrow{\alpha} M$, то існує процес $Q + G \xrightarrow{\alpha} N$ такий, що $(M, N) \in R$.

Припустимо, що $(P + G, Q + G) \in R$ і існує деяка активність α така, що $\alpha.(P + G) \xrightarrow{\alpha} M$. Тоді, спираючись на правило семантики операції вибору, розглянемо можливі варіанти формування процесу M :

4.1. Варіант $\frac{P \xrightarrow{\alpha} P'}{P + G \xrightarrow{\alpha} P'}$ зумовлює подальший розвиток процесу P ,

тобто $(\alpha.P + G) \xrightarrow{\alpha} M$, оскільки $P + G \xrightarrow{\alpha} P'$ і $M = P'$ для деякого α .

Оскільки $P \xrightarrow{\alpha} P'$ і $P \sim Q$, то, виходячи з означення строгої взаємної подібності можемо стверджувати, що існує перехід $Q \xrightarrow{\alpha} Q'$ і $P' \sim Q'$.

4.2. Варіант $\frac{G \xrightarrow{\alpha} G'}{P + G \xrightarrow{\alpha} G'}$ зумовлює подальший розвиток процесу G ,

тобто $(P + \alpha.G) \xrightarrow{\alpha} M$, оскільки $P + G \xrightarrow{\alpha} G'$ і $M = G'$ для деякого α .

Оскільки $G \xrightarrow{\alpha} G'$ для $P + G$ і $G \xrightarrow{\alpha} G'$ для $Q + G$, то, виходячи з умови рефлексивності строгої взаємної подібності, $G' \sim G'$.

Звідси випливає можливість виконання для варіантів 4.1 і 4.2 операції вибору $Q + G \xrightarrow{\alpha} N$ і $(M, N) \in R$. Отже, включення в операцію вибору одного і того ж процесу не впливає на строгу взаємну подібність одержаних систем.

5. $v_1!P + v_2!G \sim v_1!Q + v_2!G$.

Розглянемо виконання строгої взаємної подібності для операції керованого вибору. Нехай існує відношення:

$$R = \left\{ (v_1!P + v_2!G, v_1!Q + v_2!G) \mid v_1!P \sim v_1!Q \right\}.$$

Тоді якщо $v_1!P + v_2!G \xrightarrow{v_1} M$, то існує процес $v_1!Q + v_2!G \xrightarrow{v_1} N$ такий, що $(M, N) \in R$. Розглянемо спочатку можливі варіанти формування процесу M :

5.1. Якщо $v_1 = \text{true}$ то операція $\frac{v_1!P \xrightarrow{v_1} P'}{v_1!P + v_2!G \xrightarrow{v_1} P'}$ зумовлює подальший розвиток процесу P . Тобто $(v_1!P + v_2!G) \xrightarrow{v_1} M$, оскільки $v_1!P + v_2!G \xrightarrow{v_1} P'$ і $M = P'$ для $v_1 = \text{true}$.

Оскільки $v_1!P \xrightarrow{v_1} P'$ і $P \sim Q$, то, виходячи з означення строгої взаємної подібності, існує перехід $v_1!Q \xrightarrow{v_1} Q'$ і $P' \sim Q'$.

5.2. Якщо $v_2 = \text{true}$ то операція $\frac{v_2!G \xrightarrow{v_2} v_2!G'}{v_1!P + v_2!G \xrightarrow{v_2} G'}$ зумовлює подальший розвиток процесу G , тобто $(v_1!P + v_2!G) \xrightarrow{v_2} M$, оскільки $v_1!P + v_2!G \xrightarrow{v_2} G'$ і $M = G'$ для логічної змінної $v_2 = \text{true}$.

Оскільки $v_2!G \xrightarrow{v_2} G'$ для $v_1!P + v_2!G$ і $v_2!G \xrightarrow{v_2} G'$ для $v_1!Q + v_2!G$, то, виходячи з умови рефлексивності строгої взаємної подібності, $G' \sim G'$.

Звідси випливає можливість виконання для варіанту 4.1 операції вибору $v_1!Q + v_2!G \xrightarrow{v_1} N$ і $(M, N) \in R$, а для варіанту 4.2 — $v_1!Q + v_2!G \xrightarrow{v_2} N$ і $(M, N) \in R$. Отже, включення в операцію вибору процесу з однією і тією ж логічною змінною не впливає на строгу взаємну подібність одержаних систем.

$$6. P \mid G \sim Q \mid G.$$

Розглянемо строгу взаємну подібність при виконанні операцій $P \mid G$ і $Q \mid G$. Для доведення $P \mid G \sim Q \mid G$ необхідно сформулювати відношення строгої взаємної подібності R , яке включає процеси $(P \mid G, Q \mid G)$. Нехай існує відношення $R = \{(P \mid G, Q \mid G) \mid P \sim Q\}$ і активність α , яка зумовлює подальший розвиток хоча б одного процесу в паралельних композиціях процесів $P \mid G$ і $Q \mid G$. Тоді R можна вважати відношенням строгої взаємної подібності за умови, що при $(P \mid G, Q \mid G) \in R$ і $P \mid G \xrightarrow{\alpha} M$ існує процес $Q \mid G \xrightarrow{\alpha} N$ такий, що $(M, N) \in R$.

Відповідно до семантики операції паралельної композиції існує три варіанти розвитку системи $P \mid G$ під дією активності α . Необхідно довести, що відношення R є відношенням строгої взаємної подібності для кожного зі згаданих варіантів розвитку паралельної композиції.

$$6.1. \text{ У випадку } \frac{P \xrightarrow{\alpha} P'}{P \mid G \xrightarrow{\alpha} P' \mid G} \text{ активність } \alpha \text{ не впли-$$

ває на процес G , тобто $P \mid G \xrightarrow{\alpha} M$, оскільки $P \xrightarrow{\alpha} P'$ і $M = P' \mid G$.

Оскільки $P \xrightarrow{\alpha} P'$ і $P \sim Q$, то за означенням строгої взаємної подібності існує перехід $Q \xrightarrow{\alpha} Q'$ і маємо строгу взаємну подібність процесів $P' \sim Q'$. Аналогічно при $Q \xrightarrow{\alpha} Q'$ і $P \sim Q$ існує перехід $P \xrightarrow{\alpha} P'$ і $P' \sim Q'$. Виходячи з означення відношення строгої взаємної подібності,

$(P' \mid G, Q' \mid G) \in R$. Отже, $N = Q' \mid G$, що доводить збереження строгої взаємної подібності у випадку 4.1.

6.2. У випадку $\frac{G \xrightarrow{\alpha} G'}{P \mid G \xrightarrow{\alpha} P \mid G'}$ активність α не впливає на процес P , тобто $P \mid G \xrightarrow{\alpha} M$, оскільки $G \xrightarrow{\alpha} G'$

і $M = P \mid G'$.

Оскільки $G \xrightarrow{\alpha} G'$ при $P \mid G \xrightarrow{\alpha} P \mid G'$ і при $Q \mid G \xrightarrow{\alpha} Q \mid G'$, то, виходячи з властивості рефлексивності, $G' \sim G'$. Звідси $(P \mid G', Q \mid G) \in R$. Отже, $N = Q \mid G'$, тобто властивість строгої взаємної подібності доведено для даного випадку розвитку паралельної композиції.

6.3. У випадку $\frac{P \xrightarrow{a} P' \quad G \xrightarrow{\bar{a}} G'}{P \mid G \xrightarrow{\tau} P' \mid G'}$ активність $\alpha \in$

сукупністю активностей, які включають активності взаємодії a, \bar{a} та ряд прихованих активностей, що задають один із видів взаємодії: $P \triangleright G$, $P \leftrightarrow G$, $P \triangleleft G$. Тоді перехід $P \mid G \xrightarrow{\tau} M$, оскільки $P \xrightarrow{a} P'$, $G \xrightarrow{\bar{a}} G'$ і $M = P' \mid G'$.

Даний варіант паралельної композиції об'єднує умови збереження строгої взаємної подібності, які описані в попередніх двох випадках. З існування переходу $P \xrightarrow{a} P'$ і $P \sim Q$ впливає $Q \xrightarrow{a} Q'$ і $P' \sim Q'$. Перехід $Q \xrightarrow{a} Q'$ і строга взаємна подібність $P \sim Q$ зумовлюють $P \xrightarrow{a} P'$ і $P' \sim Q'$. В той же час, перехід $G \xrightarrow{\bar{a}} G'$ при $P \mid G \xrightarrow{\tau} P' \mid G'$

і при $Q \mid G \xrightarrow{\tau} Q' \mid G'$ не порушує строгої взаємної подібності завдяки умові рефлексивності: $G' \sim G'$. Звідси $(P' \mid G', Q' \mid G') \in R$, а отже, $N = Q' \mid G'$, $(M, N) \in R$ і для третього варіанту паралельної композиції, що свідчить про істинність виразу $P \mid G \sim Q \mid G$. Оскільки операція

$$\frac{P \xrightarrow{a} P' \quad G \xrightarrow{\bar{a}} G'}{P \mid G \xrightarrow{\tau} P' \mid G'}$$
 є узагальненням операцій $P \triangleright G$,

$P \leftrightarrow G$, $P \triangleleft \triangleright G$, то доведення виразів $(P \triangleright G) \sim (Q \triangleright G)$, $(P \leftrightarrow G) \sim (Q \leftrightarrow G)$, $(P \triangleleft \triangleright G) \sim (Q \triangleleft \triangleright G)$ може бути одержане шляхом виконання аналогічних міркувань.

$$7. P[K] \sim Q[K].$$

Доведення, як і в попередньому випадку, базується на побудові відношення строгої взаємної подібності R , яке б містило пару процесів $(P[K], Q[K])$. Нехай існує $R = \{(P[K], Q[K]) \mid P \sim Q\}$. Відношення R є відношенням строгої взаємної подібності, якщо існує активність α , для якої при $(P[K], Q[K]) \in R$ і $P[K] \xrightarrow{\alpha} M$ існує процес $Q[K] \xrightarrow{\alpha} N$ такий, що $(M, N) \in R$.

Якщо α належить до області означення K , то перехід $P[K] \xrightarrow{K(\alpha)} P'[K]$ безпосередньо виводиться з переходу $P \xrightarrow{\alpha} P'$ і $M = P'[K]$. Тоді існує перехід $Q \xrightarrow{\alpha} Q'$ при $P' \sim Q'$. Отже, $Q[K] \xrightarrow{K(\alpha)} Q'[K]$ і $N = Q'[K]$, $(M, N) \in R$. Випадок, коли α не належить до області означення K , є тривіальним і не потребує доведення.

8. $P \setminus A \sim Q \setminus A$.

Для доведення задамо $R = \left\{ (P \setminus A, Q \setminus A) \mid P \sim Q \right\}$ з умовою строгої взаємної подібності, якщо існує активність α , для якої при $(P \setminus A, Q \setminus A) \in R$ і $P \setminus A \xrightarrow{\alpha} M$ існує процес $Q \setminus A \xrightarrow{\alpha} N$ такий, що $(M, N) \in R$.

Якщо $\alpha \notin A$, то $P \setminus A \xrightarrow{\alpha} P' \setminus A$ зводиться до $P \xrightarrow{\alpha} P'$ і $M = P' \setminus A$. Тоді існує перехід $Q \xrightarrow{\alpha} Q'$ при $P' \simeq Q'$. Отже, $Q \setminus A \xrightarrow{\alpha} Q' \setminus A$ і $N = Q' \setminus A$, $(M, N) \in R$. У випадку, коли $\alpha \in A$, дія активності виключається з процесу, і перехід не відбувається.

9. $k \times P \sim k \times Q$.

Розглянемо відношення $R = \left\{ (k \times P, k \times Q), P \sim Q \right\}$ зі строгою взаємною подібністю за умови існування активності α , для якої при $(k \times P, k \times Q) \in R$ і $k \times P \xrightarrow{\alpha} M$ існує процес $k \times Q \xrightarrow{\alpha} N$ такий, що $(M, N) \in R$.

У відповідності до семантичного правила даної операції:

$$\frac{P \xrightarrow{\alpha} P'_1 \quad P \xrightarrow{\alpha} P'_2 \quad \dots \quad P \xrightarrow{\alpha} P'_k}{P \xrightarrow{\alpha} P'_1 \mid P'_2 \mid \dots \mid P'_k},$$

звернення активності α призводить до формування паралельної композиції $P \xrightarrow{\alpha} P'_1 \mid P'_2 \mid \dots \mid P'_k$. Відповідно

$M = P'_1 \mid P'_2 \mid \dots \mid P'_k$. Для кожного клона, який утворюється внаслідок переходу $P \xrightarrow{\alpha} P'_i$ при $1 \leq i \leq k$, існує відповідний клон, який утворюється внаслідок переходу $Q \xrightarrow{\alpha} Q'_i$ при $P'_i \sim Q'_i$. Спираючись на раніше доведений випадок збереження

строгій взаємній подібності операції паралельної композиції для $Q \xrightarrow{\alpha} Q'_1 \mid Q'_2 \mid \dots \mid Q'_k$, можемо стверджувати, що $N = Q'_1 \mid Q'_2 \mid \dots \mid Q'_k$, а отже, $k \times P \sim k \times Q$.

$$10. \mathfrak{R}(X = P) \sim \mathfrak{R}(X = Q).$$

Розглянемо випадок, коли рекурсія є скінченим процесом $S \{ \mathfrak{R}(X = P) / X \}$. Тоді виникає можливість побудувати відношення R , що складається з пар елементів $(S \{ \mathfrak{R}(X = P) / X \}, S \{ \mathfrak{R}(X = Q) / X \})$, де $S, P, Q \in \text{Pexpr}$ – процеси; $X \in \text{Pvar}$ – незалежна змінна для процесу. Покажемо, що R є відношенням строгої взаємної подібності після присвоєння $S = X$, тобто існує активність α , для якої при $(S \{ \mathfrak{R}(X = P) / X \}, S \{ \mathfrak{R}(X = Q) / X \}) \in R$ і

$S \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\alpha} M$ існує процес

$S \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\alpha} N$ такий, що $(M, N) \in R$.

Оскільки процес S є результуючим процесом, попередньо сформованим в змінній X , розглянемо ряд випадків його структури:

$$10.1. S \equiv X.$$

Відповідно до семантичного правила рекурсії:

$$\frac{P \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\alpha} P'}{\mathfrak{R}(X = P) \xrightarrow{\alpha} P'},$$

звершення активності α викликає перехід

$P \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\alpha} P'$. Тому $M = P'$. Оскільки

$P \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\alpha} P'$ і $P \sim Q$, то за означенням

строгій взаємній подібності існує перехід

$P \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\alpha} N$ і $P' \sim Q'$. З $P \sim Q$ випливає

також, що $Q \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\alpha} N$. Отже, $(M, N) \in R$, а відношення R є відношенням строгої взаємної подібності.

$$10.2. S \equiv \alpha.S_1.$$

Виходячи з семантики операції префіксації $S \xrightarrow{\alpha} S'$ і $\alpha.S_1 \xrightarrow{\alpha} S'$. Звідси $(S', S') \in R$ за умовою рефлексивності.

$$10.3. S \equiv \langle \tau \rangle . S_1.$$

Доведення аналогічне доведенню випадку $S \equiv \alpha.S_1$.

$$10.4. S \equiv \langle \tau \leftarrow t \rangle . S_1.$$

Для даного випадку:

$$S \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\langle t \rangle} S_1 [t/\tau] \{ \mathfrak{R}(X = P) / X \}.$$

Виходячи, як і в попередньому випадку, з умови $P \sim Q$:

$$S \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\langle t \rangle} S_1 [t/\tau] \{ \mathfrak{R}(X = Q) / X \}.$$

Звідси випливає

$$(S_1 [t/\tau] \{ \mathfrak{R}(X = P) / X \}, S_1 [t/\tau] \{ \mathfrak{R}(X = Q) / X \}) \in R.$$

$$10.5. S \equiv S_1 + S_2.$$

Нехай в результаті виконання операції вибору продовжить функціонування процес S_1 . Тоді

$$S_1 \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\alpha} M.$$

Оскільки $S_1 \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\alpha} N$ при $(M, N) \in R$, то

$$S_1 \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\alpha} N \text{ при } (M, N) \in R.$$

$$10.6. S \equiv S_1 \mid S_2.$$

Для паралельної композиції існує три варіанти розвитку.

10.6.1. Активність α впливає лише на процес S_1 .

Тоді $S_1 \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\alpha} M$. За означенням строгої взаємної подібності $S_1 \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\alpha} N$ при $(M, N) \in R$. Таким чином, $S \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\alpha} M \mid S_2$ і $S \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\alpha} N \mid S_2$ при $(M, N) \in R$.

Тоді для деякого S' представимо

$$S' \{ \mathfrak{R}(X = P) / X \} \equiv M \text{ і } S' \{ \mathfrak{R}(X = Q) / X \} \equiv N.$$

Представивши $S'' \equiv S' \mid S_2$, одержимо $(M \mid S_2, N \mid S_2) \in R$.

10.6.2. Активність α впливає лише на процес S_2 .

Доведення цього випадку аналогічне попередньому і випливає з умови симетричності.

10.6.3. Активність α впливає на процеси S_1 і S_2 .

Для даного випадку $S_1 \{ \mathfrak{R}(X = P) / X \} \xrightarrow{a} M_1$ і $S_2 \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\bar{a}} M_2$. За означенням строгої взаємної подібності $S_1 \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{a} N_1$ при $(M_1, N_1) \in R$ і $S_2 \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{a} N_2$ при $(M_2, N_2) \in R$.

Виходячи з цього, $((M_1 \mid N_1), (M_2 \mid N_2)) \in R$.

10.7. $S \equiv S_1 \setminus A$.

Якщо $S_1 \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\alpha} M_1$, то за означенням строгої взаємної подібності $S_1 \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\alpha} N_1$ при $(M_1, N_1) \in R$. Виконавши операції $M = M_1 \setminus A$ і $N = N_1 \setminus A$, одержимо $(M, N) \in R$.

$$10.8. S \equiv S_1 [K].$$

Доведення аналогічне доведенню випадку $S \equiv S_1 \setminus A$.

$$10.9. S \equiv \mathfrak{R}(Y = S_1).$$

Для даного випадку

$$\mathfrak{R}(Y = S_1) \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\alpha} M \text{ або}$$

$$S \{ \mathfrak{R}(Y = S_1) / Y \} \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\alpha} M.$$

За означенням строгої взаємної подібності

$$S \{ \mathfrak{R}(Y = S_1) / Y \} \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\alpha} N$$

при $(M, N) \in R$.

$$\text{Звідси } \mathfrak{R}(Y = S_1) \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\alpha} N$$

при $(M, N) \in R$.

$$11. \text{cycle}[x](X = P) \sim \text{cycle}[x](X = Q).$$

Доведення аналогічне доведенню випадку 10.

Теорему доведено.

Ми розглянули умови збереження властивості строгої взаємної подібності процесів під час виконання операцій алгебри процесів. Для опису складних систем використовують дані операції, об'єднані у алгебраїчні вирази. Тому важливо навести основні алгебраїчні властивості таких виразів.

Теорема 4.5. *У таких алгебраїчних виразах з використанням константного процесу nil завжди зберігається строга взаємна подібність:*

$$1. P + \text{nil} \sim P;$$

$$2. P \mid \text{nil} \sim P;$$

$$3. \text{nil} \setminus A \sim \text{nil};$$

$$4. \text{nil}[K] \sim \text{nil}.$$

Д о в е д е н н я.

Розглянемо відношення

$$R = \left\{ (P + \text{nil}, P), (P, P + \text{nil}), (P, P) : \forall P \right\}$$

і доведемо, що воно є відношенням строгої взаємної подібності.

Нехай $P \xrightarrow{\alpha} P'$. Тоді за правилом (4.6) $P + \text{nil} \xrightarrow{\alpha} P'$,

оскільки $\text{nil} \xrightarrow{\alpha} 0$. Виходячи з властивості рефлексивності, можемо стверджувати, що бінарним відношенням є $P'RP'$. І

навпаки, нехай $P + \text{nil} \xrightarrow{\alpha} P'$. Тоді за правилом (4.6)

$P \xrightarrow{\alpha} P'$ і знову отримаємо $P'RP'$. Справедливість виразів

2÷4 випливає безпосередньо з означення відповідних операцій.

Теорему доведено.

Теорема 4.6. У наведених нижче алгебраїчних виразах, які використовують властивість комутативності, завжди зберігається строга взаємна подібність:

1. $P_1 + P_2 \sim P_2 + P_1$;
2. $v_1!P_1 + v_2!P_2 \sim v_2!P_2 + v_1!P_1$;
3. $P_1 \Big| P_2 \sim P_2 \Big| P_1$;
4. $P_1 \leftrightarrow_L P_2 \sim P_2 \leftrightarrow_L P_1$;
5. $P_1 \triangleleft_L P_2 \sim P_2 \triangleleft_L P_1$;
6. $P_1 \triangleright_L P_2 \sim P_2 \triangleleft_L P_1$.

Д о в е д е н н я. Справедливість виразу 1 випливає з умови, що відношення $R = \left\{ (P_1 + P_2, P_2 + P_1), (P_1, P_1) : \forall P_1, P_2 \right\}$ є

відношенням строгої взаємної подібності. Нехай

$P_1 + P_2 \xrightarrow{\alpha} P'_1$. Тоді за семантичним правилом операції

вибору (4.6) $P_1 \xrightarrow{\alpha} P'_1$. Але за правилом (4.6) також $P_2 + P_1 \xrightarrow{\alpha} P'_1$. Звідси випливає, що відношення $P'_1 R P'_1$ є відношенням строгої взаємної подібності за властивістю рефлексивності. Вираз $P_2 + P_1 \sim P_1 + P_2$ справедливий за симетрією.

Аналогічне доведення для виразу 2 відображає комутативність операцій керованого вибору.

Розглянемо відношення $R = \left\{ \left(P_1 \mid P_2, P_2 \mid P_1 \right) : \forall P_1, P_2 \right\}$.

Нехай $P_1 \xrightarrow{\alpha} P'_1$, $P'_2 := P_2$. Тоді

$P_1 \mid P_2 \xrightarrow{\alpha} P'_1 \mid P'_2$. Очевидно, що $P'_1 \mid P'_2 R P'_1 \mid P'_2$ є відношенням строгої взаємної подібності. Оскільки, за теоремою 4.1, відношення строгої взаємної подібності є еквівалентним відношенням, то $P'_1 \mid P'_2 R P'_2 \mid P'_1$. Отже, вираз 4, який відображає комутативність операції паралельного вибору, зберігає строгу взаємну подібність. Аналогічне доведення для виразів 4÷6, що відображають комутативність операцій синхронної взаємодії, асинхронної взаємодії та керованої передачі. Теорему доведено.

Теорема 4.7. *У наведених нижче виразах, які використовують властивість асоціативності, завжди зберігається строга взаємна подібність:*

1. $(P_1 + P_2) + P_3 \sim P_1 + (P_2 + P_3)$;
2. $(v_1!P_1 + v_2!P_2) + v_3!P_3 \sim v_1!P_1 + (v_2!P_2 + v_3!P_3)$;
3. $P_1 \left| \left(P_2 \mid P_3 \right) \sim \left(P_1 \mid P_2 \right) \mid P_3$;
4. $P_1 \xleftrightarrow{L} \left(P_2 \xleftrightarrow{L} P_3 \right) \sim \left(P_1 \xleftrightarrow{L} P_2 \right) \xleftrightarrow{L} P_3$;
5. $P_1 \triangleleft_L \left(P_2 \triangleleft_L P_3 \right) \sim \left(P_1 \triangleleft_L P_2 \right) \triangleleft_L P_3$.

Д о в е д е н н я. Властивість асоціативності виразів 1 і 2 базується на застосуванні правила семантики операцій вибору.

Розглянемо відношення:

$$R = \left\{ \left((P_1 + P_2) + P_3, P_1 + (P_2 + P_3) \right), \right. \\ \left. (P_1, P_1), (P_2, P_2), (P_3, P_3) \mid \forall P_1, P_2, P_3 \right\}.$$

Нехай $(P_1 + P_2) + P_3 \xrightarrow{\alpha} P'_1$.

Тоді $P_2 \xrightarrow{\alpha} 0$ і $P_3 \xrightarrow{\alpha} 0$. Отже, $P'_1 := (P'_1 + 0) + 0$.

При $P_1 + (P_2 + P_3) \xrightarrow{\alpha} P'_1$ відповідно

$P_2 \xrightarrow{\alpha} 0$ і $P_3 \xrightarrow{\alpha} 0$. Звідси $P'_1 := P'_1 + (0 + 0)$. Тому

одержуємо $P'_1 R P'_1$. Аналогічно у випадку

$(P_1 + P_2) + P_3 \xrightarrow{\alpha} P'_2$ можемо отримати відношення

$P'_2 R P'_2$, а у випадку $(P_1 + P_2) + P_3 \xrightarrow{\alpha} P'_3$ — відношення

$P'_3 R P'_3$.

Правила семантики для керованого та імовірнісного вибору також забезпечують вибір одного процесу з деякої множини процесів. Тому схема доведення пункту 2 використовує вже застосований підхід. Якщо $\sum_{i \in I} v_i ! P_i \xrightarrow{v_k} P'_k$ при $v_k = \text{true}$,

то $\sum_{i \in I, i \neq k} v_i ! P_i \xrightarrow{v_i} 0$. Нехай $v_1 = \text{true}$.

Тоді $(v_1 ! P_1 + v_2 ! P_2) + v_3 ! P_3 \xrightarrow{v_1} P'_1$, оскільки

$P'_1 := (P'_1 + 0) + 0$, та $v_1 ! P_1 + (v_2 ! P_2 + v_3 ! P_3) \xrightarrow{v_1} P'_1$,

оскільки $P'_1 := P'_1 + (0 + 0)$. Отже, маємо $P'_1 R P'_1$. Аналогічно у

випадку $(v_1 ! P_1 + v_2 ! P_2) + v_3 ! P_3 \xrightarrow{v_2} P'_2$ одержуємо $P'_2 R P'_2$,

а для $(v_1 ! P_1 + v_2 ! P_2) + v_3 ! P_3 \xrightarrow{v_3} P'_3$ одержуємо $P'_3 R P'_3$.

Група виразів $3 \div 5$ відображає факт збереження строгої взаємної

подібності при застосуванні властивості асоціативності щодо операцій паралельної композиції та взаємодії.

Розглянемо відношення

$R = \{(P_1 \mid (P_2 \mid P_3), (P_1 \mid P_2) \mid P_3) : \forall P_1, P_2, P_3\}$ і доведемо, що воно є відношенням строгої взаємної подібності для виразу 3.

Відповідно до семантичного правила операції паралельної композиції необхідно розглянути випадки розвитку паралельних процесів без взаємодії та зі взаємодією.

Нехай $P_1 \mid (P_2 \mid P_3) \xrightarrow{\alpha} P'_1 \mid (P'_2 \mid P'_3)$. Для випадку застосування активності до процесу P_1 одержимо: $P_1 \xrightarrow{\alpha} P'_1$ і $P'_2 := P_2$, $P'_3 := P_3$. Тоді, відповідно до означення операції паралельної композиції, має місце перехід $P_1 \mid P_2 \xrightarrow{\alpha} P'_1 \mid P'_2$. Звідси випливає, що за умови $P'_3 := P_3$ справедливий перехід $(P_1 \mid P_2) \mid P_3 \xrightarrow{\alpha} (P'_1 \mid P'_2) \mid P'_3$. Аналогічне доведення для випадків $P_2 \xrightarrow{\alpha} P'_2$, $P'_1 := P_1$, $P'_3 := P_3$ і $P_3 \xrightarrow{\alpha} P'_3$, $P'_1 := P_1$, $P'_2 := P_2$.

Розглянемо тепер випадок взаємодії

$$P_1 \mid (P_2 \mid P_3) \xrightarrow{\tau} P'_1 \mid (P'_2 \mid P'_3)$$

при $P_1 \xrightarrow{a} P'_1$, $P_2 \xrightarrow{\bar{a}} P'_2$ і $P'_3 := P_3$. Відповідно до означення операції паралельної композиції справедливий перехід $P_1 \mid P_2 \xrightarrow{\tau} P'_1 \mid P'_2$, а отже, $(P_1 \mid P_2) \mid P_3 \xrightarrow{\tau} (P'_1 \mid P'_2) \mid P'_3$.

Аналогічне доведення для випадків $P_1 \xrightarrow{a} P'_1$, $P_3 \xrightarrow{\bar{a}} P'_3$, $P'_2 := P_2$ і $P_2 \xrightarrow{a} P'_2$, $P_3 \xrightarrow{\bar{a}} P'_3$, $P'_1 := P_1$.

Для доведення властивості асоціативності операції синхронної взаємодії розглянемо відношення

$$R = \left\{ \left(P_1 \leftrightarrow (P_2 \leftrightarrow P_3), (P_1 \leftrightarrow P_2) \leftrightarrow P_3 \right) : \forall P_1, P_2, P_3 \right\}$$

і доведемо, що воно є відношенням строгої взаємної подібності для виразу 4. Для спрощення будемо вважати, що в рамках даних перетворень множина спільних подій взаємодії L є незмінною, і тому може бути представлена неявно. Семантичне правило синхронної взаємодії нагадує правило, що задає випадок взаємодії для паралельної композиції, тому використаємо відповідну схему доведення. Нехай

$$P_1 \leftrightarrow (P_2 \leftrightarrow P_3) \xrightarrow{\tau} P'_1 \leftrightarrow (P'_2 \leftrightarrow P'_3) \quad \text{при} \quad P_1 \xrightarrow{\bar{a}} P'_1,$$

$$P_2 \xrightarrow{a} P'_2, \quad P_3 \xrightarrow{a} P'_3.$$

Тоді відповідно до семантичного правила синхронної взаємодії мають місце переходи

$$P_1 \leftrightarrow P_2 \xrightarrow{\tau} P'_1 \leftrightarrow P'_2 \quad \text{і} \quad P_1 \leftrightarrow P_3 \xrightarrow{\tau} P'_1 \leftrightarrow P'_3, \quad \text{а отже,}$$

$$(P_1 \leftrightarrow P_2) \leftrightarrow P_3 \xrightarrow{\tau} (P'_1 \leftrightarrow P'_2) \leftrightarrow P'_3.$$

Аналогічне доведення для випадків $P_1 \xrightarrow{a} P'_1, P_2 \xrightarrow{\bar{a}} P'_2, P_3 \xrightarrow{a} P'_3$ і

$$P_1 \xrightarrow{a} P'_1, P_2 \xrightarrow{a} P'_2, P_3 \xrightarrow{\bar{a}} P'_3.$$

Для доведення властивості асоціативності операції асинхронної взаємодії розглянемо відношення

$$R = \left\{ \left(P_1 \triangleleft\triangleright (P_2 \triangleleft\triangleright P_3), (P_1 \triangleleft\triangleright P_2) \triangleleft\triangleright P_3 \right) : \forall P_1, P_2, P_3 \right\} \quad \text{і}$$

використаємо попередню схему доведення при

$$P_1 \triangleleft\triangleright (P_2 \triangleleft\triangleright P_3) \xrightarrow{\tau} P'_1 \triangleleft\triangleright (P'_2 \triangleleft\triangleright P'_3) \quad \text{і} \quad P_1 \xrightarrow{\bar{a}} P'_1,$$

$$P_2 \xrightarrow{a} P'_2, \quad P_3 \xrightarrow{a} P'_3.$$

За семантичним правилом асинхронної взаємодії може бути виконаний перехід

$$P_1 \triangleleft\triangleright P_2 \xrightarrow{\tau} P'_1 \triangleleft\triangleright P'_2, \quad \text{а отже,}$$

$$(P_1 \triangleleft\triangleright P_2) \triangleleft\triangleright P_3 \xrightarrow{\tau} (P'_1 \triangleleft\triangleright P'_2) \triangleleft\triangleright P'_3.$$

Аналогічне доведення для випадків $P_1 \xrightarrow{a} P'_1, P_2 \xrightarrow{\bar{a}} P'_2, P_3 \xrightarrow{a} P'_3$ і

$$P_1 \xrightarrow{a} P'_1, P_2 \xrightarrow{a} P'_2, P_3 \xrightarrow{\bar{a}} P'_3. \quad \text{Теорему доведено.}$$

Теорема 4.8. У наведених нижче виразах, які використовують властивість дистрибутивності, завжди зберігається строга взаємна подібність:

1. $\alpha.(P \setminus A) \sim (\alpha.P) \setminus A$ при $\alpha \notin A$;
2. $\alpha.(P \setminus A) \sim 0$ при $\alpha \in A$;
3. $\langle \tau \rangle.(P \setminus A) \sim (\langle \tau \rangle.P \setminus A)$;
4. $\langle \tau \leftarrow t \rangle.(P \setminus A) \sim (\langle \tau \leftarrow t \rangle.P \setminus A)$;
5. $(P_1 + P_2) \setminus A \sim (P_1 \setminus A) + (P_2 \setminus A)$;
6. $(\alpha.P)[K] \sim K(\alpha).P[K]$;
7. $(\langle \tau \rangle.P)[K] \sim \langle \tau \rangle.(P[K])$;
8. $(\langle \tau \leftarrow t \rangle.P[K]) \sim \langle \tau \leftarrow t \rangle(P[K])$;
9. $(P_1 + P_2)[K] \sim P_1[K] + P_2[K]$;
10. $(v_1!P_1 + v_2!P_2)[K] \sim v_1!P_1[K] + v_2!P_2[K]$;
11. $k \times (P_1 + P_2) \sim k \times P_1 + k \times P_2$.

Д о в е д е н н я. Вирази 1÷4 є тривіальними. Справедливість їх базується на означенні операції стискування. Розглянемо відношення $R = \{(\alpha.(P \setminus A), (\alpha.P) \setminus A), (P \setminus A, P \setminus A)\}$.

Переходи $\alpha.(P \setminus A) \xrightarrow{\alpha} P' \setminus A$ і $(\alpha.P) \setminus A \xrightarrow{\alpha} P' \setminus A$ утворюють ідентичні процеси, які розвиваються однаково за умови $\alpha \notin A$, а при $\alpha \in A$ має місце перехід $\alpha.(P \setminus A) \xrightarrow{\alpha} 0$. Якщо довільну активність α замінити константою затримки $\langle \tau \rangle$ або змінною затримки $\langle \tau \leftarrow t \rangle$, то одержимо переходи $\langle \tau \rangle.(P \setminus A) \xrightarrow{\langle \tau \rangle} P' \setminus A$ і

$\langle \tau \leftarrow t \rangle . (P \setminus A) \xrightarrow{\langle t \rangle} P' [t/\tau] \setminus A$, що мають місце при довільному $\langle \tau \rangle$, оскільки за означенням $\langle \tau \rangle \notin A$. Виходячи з останнього твердження, приходимо до висновку, що мають місце також переходи $(\langle \tau \rangle . P \setminus A) \xrightarrow{\langle \tau \rangle} P' \setminus A$ і $(\langle \tau \leftarrow t \rangle . P \setminus A) \xrightarrow{\langle t \rangle} P' [t/\tau] \setminus A$.

Для доведення властивості дистрибутивності виразу 5 задамо бінарне відношення:

$$R = \left\{ (P_1 + P_2) \setminus A, (P_1 \setminus A) + (P_2 \setminus A), (P_1 \setminus A, P_1 \setminus A), (P_2 \setminus A, P_2 \setminus A) \mid \forall P_1, P_2 \right\}.$$

Нехай $(P_1 + P_2) \setminus A \xrightarrow{\alpha} P'_1 \setminus A$. Тоді $P_2 \xrightarrow{\alpha} 0$.

Звідси випливає $P'_1 \setminus A := (P'_1 + 0) \setminus A$. При $P_2 \xrightarrow{\alpha} 0$ одержуємо $(P_2 \setminus A) \xrightarrow{\alpha} 0$, а отже,

$$(P_1 \setminus A) + (P_2 \setminus A) \xrightarrow{\alpha} P'_1 \setminus A.$$

Тому справедливе відношення $(P_1 \setminus A) R (P'_1 \setminus A)$.

І навпаки, при $(P_1 + P_2) \setminus A \xrightarrow{\alpha} P'_2 \setminus A$ — відповідно $P_1 \xrightarrow{\alpha} 0$. Звідси $P'_2 \setminus A := (0 + P'_2) \setminus A$. В той же час при $P_1 \xrightarrow{\alpha} 0$ одержуємо $(P_1 \setminus A) \xrightarrow{\alpha} 0$, а отже, $(P_1 \setminus A) + (P_2 \setminus A) \xrightarrow{\alpha} P'_2 \setminus A$ і, відповідно,

$$(P_2 \setminus A) R (P'_2 \setminus A).$$

Доведення властивості дистрибутивності виразів 6÷10 аналогічне тому, що використовувалось для операції стискання. Введемо відношення

$$R = \left\{ ((\alpha.P)[K], K(\alpha).P[K]), (P[K], P[K]): \forall P \right\}.$$

Тоді існування переходів $(\alpha.P)[K] \xrightarrow{\alpha} P'[K]$ і $K(\alpha).P[K] \xrightarrow{\alpha} P'[K]$ випливає з означення семантики операції перейменування. Звідси $(P[K])R(P[K])$. Ввівши відношення

$$R = \{((\langle \tau \rangle.P)[K], (\langle \tau \rangle.P)[K]), (P[K], P[K]) : \forall P\},$$

розглянемо переходи $(\langle \tau \rangle.P)[K] \xrightarrow{\langle \tau \rangle} P'[K]$ і

$(\langle \tau \rangle.P)[K] \xrightarrow{\langle \tau \rangle} P'[K]$. Оскільки $(P'[K], P'[K]) \in R$, то операція префіксної затримки має властивість дистрибутивності. Аналогічно доводимо властивість дистрибутивності параметричної префіксної затримки, заданої на відношенні

$$R = \{((\langle \tau \leftarrow t \rangle.P)[K], (\langle \tau \leftarrow t \rangle.P)[K]), (P[K], P[K]) : \forall P\},$$

шляхом виконання переходів

$$\begin{aligned} (\langle \tau \leftarrow t \rangle.P)[K] &\xrightarrow{\langle t \rangle} P'[K] \text{ і} \\ (\langle \tau \leftarrow t \rangle.P)[K] &\xrightarrow{\langle t \rangle} P'[K]. \end{aligned}$$

Як результат також одержуємо $(P'[K])R(P'[K])$.

Властивість дистрибутивності виразу 9 розглянемо на бінарному відношенні:

$$\begin{aligned} R = \{ & (P_1 + P_2)[K], ((P_1[K]) + (P_2[K])), \\ & (P_1[K], P_1[K]), (P_2[K], P_2[K]) \mid \forall P_1, P \}. \end{aligned}$$

Нехай $(P_1 + P_2)[K] \xrightarrow{\alpha} P'_1[K]$. Тоді $P_2 \xrightarrow{\alpha} 0$.

Отже, $P'_1[K] := (P'_1 + 0)[K]$. У згаданому випадку

$(P_2[K]) \xrightarrow{\alpha} 0$, а тому, $(P_1[K]) + (P_2[K]) \xrightarrow{\alpha} P'_1[K]$.

Звідси випливає відношення $(P_1[K])R(P_1[K])$. І також при

$(P_1 + P_2)[K] \xrightarrow{\alpha} P'_2[K]$ — відповідно $P_1 \xrightarrow{\alpha} 0$. А тому

$P'_2[K] := (0 + P'_2)[K]$. В той же час при $P_1 \xrightarrow{\alpha} 0$ одержуємо $(P_1[K]) \xrightarrow{\alpha} 0$, а отже,

$$(P_1[K]) + (P_2[K]) \xrightarrow{\alpha} P'_2[K]$$

і відповідно $(P_2[K]) R (P_2[K])$.

Доведення справедливості виразу 10 проводимо за тією ж схемою, що і виразу 9.

Введемо бінарне відношення:

$$R = \left\{ (v_1!P_1 + v_2!P_2)[K], (v_1!P_1[K] + v_2!P_2[K]), \right. \\ \left. (P_1[K], P_1[K]), (P_2[K], P_2[K]) \mid \forall P_1, P_2 \right\}.$$

У випадку $v_1 = \text{true}$ можливе виконання переходу $(v_1!P_1 + v_2!P_2)[K] \xrightarrow{v_1} P'_1[K]$. Тоді $P_2 \xrightarrow{v_2} 0$. Отже,

$P'_1[K] := (P'_1 + 0)[K]$. При $P_2 \xrightarrow{v_2} 0$ отримуємо

$v_2!P_2[K] \xrightarrow{v_2} 0$, а отже при $v_1 = \text{true}$,

$v_1!P_1[K] + v_2!P_2[K] \xrightarrow{v_2} P'_1[K]$. Тому справедливе відно-

шення $(P_1[K]) R (P_1[K])$. Також при $v_2 = \text{true}$

$(v_1!P_1 + v_2!P_2)[K] \xrightarrow{v_2} P'_2[K]$ — відповідно $P_1 \xrightarrow{v_1} 0$.

Звідси $P'_2[K] := (0 + P'_2)[K]$. В той же час при $P_1 \xrightarrow{v_1} 0$

одержуємо $v_1!P_1 \xrightarrow{v_1} 0$, а тому при $v_2 = \text{true}$

$v_1!P_1[K] + v_2!P_2[K] \xrightarrow{v_2} P'_2[K]$ і відповідно

$$(P_2[K]) R (P_2[K]).$$

Для доведення дистрибутивності операції клонування розглянемо бінарне відношення:

$$R = \left\{ k \times (P_1 + P_2), (k \times P_1 + k \times P_2), \left(\overbrace{P_1 | \dots | P_1}^k, \overbrace{P_1 | \dots | P_1}^k \right), \left(\overbrace{P_2 | \dots | P_2}^k, \overbrace{P_2 | \dots | P_2}^k \right) \right\} \Big| \forall P_1, P_2 \}.$$

Якщо в результаті виконання операції вибору продовжується розвиток процесу P_1 , то таке продовження може бути

представлене переходом $k \times (P_1 + P_2) \xrightarrow{\alpha} \overbrace{P_1' | \dots | P_1'}^k$. Тоді за правилом семантики даної операції $P_2 \xrightarrow{\alpha} 0$. Звідси

$$\overbrace{P_1' | \dots | P_1'}^k := \left(\overbrace{P_1' | \dots | P_1'}^k + 0 \right). \text{ Але } k \times P_1 \xrightarrow{\alpha} \overbrace{P_1' | \dots | P_1'}^k. \text{ Тому}$$

для згаданого випадку вибору $k \times P_1 + k \times P_2 \xrightarrow{\alpha} \overbrace{P_1' | \dots | P_1'}^k$.

$$\text{А отже, } \left(\overbrace{P_1 | \dots | P_1}^k \right) R \left(\overbrace{P_1 | \dots | P_1}^k \right).$$

При $k \times (P_1 + P_2) \xrightarrow{\alpha} \overbrace{P_2' | \dots | P_2'}^k$ процес P_1 не має розвитку,

тобто $P_1 \xrightarrow{\alpha} 0$. Тому $\overbrace{P_2' | \dots | P_2'}^k := \left(0 + \overbrace{P_2' | \dots | P_2'}^k + \right)$. В той

же час при $P_1 \xrightarrow{\alpha} 0$ одержуємо $k \times P_1 \xrightarrow{\alpha} 0$ і

$$k \times P_1 + k \times P_2 \xrightarrow{\alpha} \overbrace{P_2' | \dots | P_2'}^k.$$

Отже, $\left(\overbrace{P_2 | \dots | P_2}^k \right) R \left(\overbrace{P_2 | \dots | P_2}^k \right)$. Теорему доведено.

Теорема 4.9. Для довільного процесу $C \{ \}$ і для процесів $P \sim Q$ операція включення є конгруентною: $C \{ P \} \sim C \{ Q \}$.

Д о в е д е н н я. Представимо процес високого рівня $C \{ \}$ у вигляді послідовності активностей α_{c_i} , $1 \leq i \leq N$, з можливими варіантами включення послідовності активностей процесів низького рівня $P: = \alpha_{p_1} \cdot \alpha_{p_2} \cdots \alpha_{p_L}$ або

$$Q: = \alpha_{q_1} \cdot \alpha_{q_2} \cdots \alpha_{q_K} :$$

- a) $C: = \{ \} \cdot \alpha_{c_1} \cdot \alpha_{c_2} \cdots \alpha_{c_N}$ — початкове включення;
 b) $C: = \alpha_{c_1} \cdot \alpha_{c_2} \cdots \alpha_{c_N} \cdot \{ \}$ — прикінцеве включення;
 c) $C: = \alpha_{c_1} \cdot \alpha_{c_2} \cdot \alpha_{c_i} \cdot \{ \} \cdot \alpha_{c_{i+1}} \cdots \alpha_{c_N}$ — внутрішнє включення.

Розглянемо переходи $C \{ P \} \xrightarrow{\alpha} S_1$ і $C \{ Q \} \xrightarrow{\alpha} S_2$, де α — довільна активність системи $C \{ P \} \cup C \{ Q \}$.

a) Задамо бінарне відношення:

$$R = \left\{ \begin{array}{l} (p_i, q_j) \mid p_i \in P, q_j \in Q; \\ (p_L, c_1), (q_K, c_1) \mid p_L \in P, q_K \in Q, c_1 \in C; \\ (c_m, c_m) \mid c_m \in C \end{array} \right\}.$$

Якщо поточна активність α задовольняє умови $p_i \xrightarrow{\alpha} p_{i+1}$ і $q_j \xrightarrow{\alpha} q_{j+1}$, одержуємо $S_1 R S_2$, оскільки $P \sim Q$. При переході до станів процесу високого рівня $p_L \xrightarrow{\alpha} c_1$ і $q_K \xrightarrow{\alpha} c_1$ також існує єдина активність α , оскільки ця активність є префіксною активністю $\alpha.C$. Тому також справедливе відношення $S_1 R S_2$. При $c_i \xrightarrow{\alpha} c_{i+1}$, $i < N$, відповідно $S_1 \equiv S_2$ з очевидних причин.

b) Для прикінцевого включення задамо бінарне відношення:

$$R = \left\{ \begin{array}{l} (c_m, c_m) \mid c_m \in C; \\ (c_N, p_1), (c_N, q_1) \mid p_1 \in P, q_1 \in Q, c_N \in C; \\ (p_i, q_j) \mid p_i \in P, q_j \in Q \end{array} \right\}$$

Спочатку виконуються активності процесу високого рівня $c_i \xrightarrow{\alpha} c_{i+1}$, $i < N$. Тому $S_1 \equiv S_2$ з очевидних причин. При $c_N \xrightarrow{\alpha} p_1$ і $c_N \xrightarrow{\alpha} q_1$ активність α є одночасно префіксною активністю процесів $\alpha.P$ і $\alpha.Q$. Оскільки $P \sim Q$, то за теоремою 4.4 $\alpha.P \sim \alpha.Q$. Звідси S_1RS_2 . При $p_i \xrightarrow{\alpha} p_{i+1}$ і $q_j \xrightarrow{\alpha} q_{j+1}$ також S_1RS_2 , що випливає з умови теореми.

с) Для внутрішнього включення задамо бінарне відношення:

$$R = \left\{ \begin{array}{l} (c_m, c_m) \mid c_m \in C; \\ (c_z, p_1), (c_z, q_1) \mid p_1 \in P, q_1 \in Q, c_z \in C; \\ (p_L, c_{z+1}), (q_K, c_{z+1}) \mid p_L \in P, q_K \in Q, c_{z+1} \in C; \\ (p_i, q_j) \mid p_i \in P, q_j \in Q \end{array} \right\}.$$

Доведення внутрішнього включення базується на двох попередніх варіантах.

Як і у випадку б), спочатку виконуються активності процесу високого рівня $c_i \xrightarrow{\alpha} c_{i+1}$, $i < z$. Отже, $S_1 \equiv S_2$ і відповідно S_1RS_2 . При $c_z \xrightarrow{\alpha} p_1$ і $c_z \xrightarrow{\alpha} q_1$ активність α є одночасно префіксною активністю процесів $\alpha.P$ і $\alpha.Q$. Оскільки $P \sim Q$, то за теоремою 4.4 $\alpha.P \sim \alpha.Q$. Звідси S_1RS_2 . Якщо активність α задовольняє умови $p_i \xrightarrow{\alpha} p_{i+1}$ і $q_j \xrightarrow{\alpha} q_{j+1}$, то S_1RS_2 , оскільки $P \sim Q$. При переходах $p_L \xrightarrow{\alpha} c_{z+1}$ і $q_K \xrightarrow{\alpha} c_{z+1}$ існує єдина активність α , оскільки ця активність є префіксною активністю $\alpha.C$. Тому

S_1RS_2 . Для переходів $c_i \xrightarrow{\alpha} c_{i+1}$, $i > z$, як і в попередньому випадку, $S_1 \equiv S_2$ і відповідно S_1RS_2 . Теорему доведено.

Теорема 4.10. Для довільного процесу $C \{ \}$ і процесів $P \sim Q$ завжди існує строга взаємна подібність: $\varphi_P(C\{P\}) \sim \varphi_Q(C\{Q\})$.

Д о в е д е н н я. Розглянемо задані на бінарному відношенні $R = \{ \varphi_P(C\{P\}), \varphi_Q(C\{Q\}) \}$ переходи $\varphi_P(C\{P\}) \xrightarrow{\alpha} S_1$ і $\varphi_Q(C\{Q\}) \xrightarrow{\alpha} S_2$, для яких існує три можливих варіанти розміщення:

- a) $C = \{ \} . \alpha_{c_1} . \alpha_{c_2} \dots \alpha_{c_N}$ — початкове розміщення;
- b) $C = \alpha_{c_1} . \alpha_{c_2} \dots \alpha_{c_N} . \{ \}$ — прикінцеве розміщення;
- c) $C = \alpha_{c_1} . \alpha_{c_2} \dots \alpha_{c_i} . \{ \} . \alpha_{c_{i+1}} \dots \alpha_{c_N}$ — внутрішнє розміщення.

a) Для випадку початкового розміщення виділимо підмножину $R_a \subseteq R$:

$$R_a = \left\{ \begin{array}{l} (p_i, c_1), (q_j, c_1) \mid p_i \in P, q_j \in Q, c_1 \in C, i = \overline{1, L}, j = \overline{1, K}; \\ (c_m, c_m) \mid c_m \in C, m = \overline{1, N} \end{array} \right\}.$$

Для довільних переходів $p_i \xrightarrow{\alpha} c_1$ і $q_j \xrightarrow{\alpha} c_1$ при $i = \overline{1, L}, j = \overline{1, K}$ активності α заміщають активність φ відповідно до семантичного правила виконання операції приховання, і ця активність є префіксною активністю $\varphi.C$. Отже, $S_1R_aS_2$. При $c_i \xrightarrow{\alpha} c_{i+1}$, $i < N$, завжди $S_1 \equiv S_2$ з очевидних причин.

b) Для прикінцевого розміщення виділимо підмножину $R_b \subseteq R$:

$$R_b = \left\{ \begin{array}{l} (c_m, c_m) \mid c_m \in C, m = \overline{1, N}; \\ (c_N, p_i), (c_N, q_j) \mid p_i \in P, q_j \in Q, c_N \in C, i = \overline{1, L}, j = \overline{1, K} \end{array} \right\}$$

Для активностей процесу високого рівня $c_i \xrightarrow{\alpha} c_{i+1}$, $i < N$, очевидно $S_1 \equiv S_2$. Переходи $c_N \xrightarrow{\alpha} p_i$ і $c_N \xrightarrow{\alpha} q_j$ заміщаються за семантичним правилом операції приховання переходом $c_N \xrightarrow{\varphi} \text{nil}$. Оскільки $P \sim Q$, то $\varphi.P \sim \varphi.Q$. Звідси $S_1 R_b S_2$.

с) Для внутрішнього розміщення виділимо підмножину $R_c \subseteq R$:

$$R_c = \left\{ \begin{array}{l} (c_m, c_m) \mid c_m \in C; \\ (c_z, p_i), (c_z, q_j), (p_i, c_{z+1}), (q_j, c_{z+1}) \mid p_i \in P, q_j \in Q, c_z \in C, \\ i = \overline{1, L}, j = \overline{1, K} \end{array} \right\}$$

Доведення використовує варіанти б) і с). При $c_i \xrightarrow{\alpha} c_{i+1}$, $i < z$, отримуємо $S_1 \equiv S_2$ і, відповідно, $S_1 R S_2$. За семантичним правилом операції приховання переходи $c_z \xrightarrow{\alpha} p_i$ і $c_z \xrightarrow{\alpha} q_i$, а також переходи $p_i \xrightarrow{\alpha} c_{z+1}$ і $q_i \xrightarrow{\alpha} c_{z+1}$ заміняють переходом $c_z \xrightarrow{\varphi} c_{z+1}$. Звідси $S_1 R S_2$. Для переходів $c_i \xrightarrow{\alpha} c_{i+1}$, $i > z$, як і в попередньому випадку, $S_1 \equiv S_2$ і, відповідно, $S_1 R S_2$. Теорему доведено.

Ми розглянули означення класичної строгої взаємної подібності та її властивості. Відповідно до означення 4.27 особливість даного виду еквівалентності полягає у тому, що вона справджується для всіх без винятку станів, які підлягають порівнянню. Але на практиці іноді важливо розглядати строгу взаєм-

ну подібність тільки підмножин процесів, якщо, наприклад, відомо, що решта процесів порівнюваних систем відповідають умовам строгої взаємної подібності.

Означення 4.29 (квазістрога взаємна подібність). Нехай в маркованій системі з переходами $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$ існує відношення $S \subseteq \Pi \times \Pi$ і процеси $P, Q \subseteq \Pi$ з допустимими станами $p, p' \in P$, $q, q' \in Q$ при $(p, q) \in S$. Тоді якщо для довільної активності a :

— для кожного переходу $p \xrightarrow{a} p'$ існує стан q' такий, що $q \xrightarrow{a} q'$ при $p' \sim S \sim q'$,

— для кожного переходу $q \xrightarrow{a} q'$ існує стан p' такий, що $p \xrightarrow{a} p'$ при $p' \sim S \sim q'$,

то відношення S будемо називати квазістрогою взаємною подібністю, якщо $a \in \text{Act}$.

Графічне зображення квазістрогої взаємної подібності станів p і q показано на рис. 4.11.

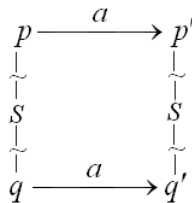


Рис.4.11. Квазістрога взаємна подібність станів p і q

Співвідношення між строгою взаємною подібністю та квазістрогою взаємною подібністю визначимо наступною теоремою.

Теорема 4.11. *Якщо у маркованій системі з переходами $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$ відношення S — це відношення квазістрокої взаємної подібності, то повне об'єднання всіх можливих відношень квазістрокої взаємної подібності зі строгою взаємною подібністю $(\sim \cup S)^*$ є відношенням строгої взаємної подібності і $S \subseteq \sim$.*

Д о в е д е н н я. Нехай в маркованій системі з переходами існують процеси $P, Q \subseteq \Pi$ з допустимими станами $p, p' \in P$ і $q, q' \in Q$. Позначимо $K = (\sim \cup S)^*$ і розглянемо pKq , де $(p, q) \in K$. З умови pKq очевидно, що $p \sim q$ і має місце pSq . Відношення $p \sim q$ є еквівалентним відношенням за умовою. Для доведення еквівалентності pSq необхідно показати, що воно, відповідно до означення 4.23, є рефлексивним, симетричним і транзитивним.

За означенням 4.29 введемо заміну p' на p і q' на q . Якщо існує спільна активність a , то вона породжує тотожні переходи $p \xrightarrow{a} p$ і $q \xrightarrow{a} q$. Тому в заданій системі підмножина I множини S містить бінарні відношення $I = \{(p, p), (q, q)\}$. Оскільки $I \subseteq S$ містить елементи відношення ідентичності, то вони є рефлексивними.

Симетричність квазістрокої взаємної подібності, тобто еквівалентність відношень pSq і qSp випливає з симетричності означення 4.29. Строге доведення даного факту аналогічне використаному в теоремі 4.1.

Транзитивність строгої взаємної подібності станів pSq припускає наявність стану z такого, що $pSzSq$. Нехай існує перехід $p \xrightarrow{a} p'$. Тоді за означенням 4.29 для деякого z

існує перехід $z \xrightarrow{a} z'$ і $p'Sz'$, оскільки pSz . Аналогічно, якщо існує перехід $z \xrightarrow{a} z'$, то для деякого q справедливий перехід $q \xrightarrow{a} q'$ і $z'Sq'$ при умові zSq . Звідси $p'Sq'$ при pSq . Отже, K є відношенням строгої взаємної подібності. Але $S \subseteq K$, тому $S \subseteq \sim$. Теорему доведено.

4.6. Слабка взаємна подібність

Розглянута вище строга взаємна подібність та її підмножина — квазістрога взаємна подібність є основними видами еквівалентності, які дозволяють розробляти сценарії порівняння складних дискретних систем. Спираючись на теорему 4.2, можна стверджувати, що строга взаємна подібність об'єднує всі допустимі відношення взаємної подібності і тому є найбільшою взаємною подібністю. Ті чи інші властивості строгої взаємної подібності мають всі оператори запропонованої алгебри процесів. Але чи є застосування даного виду еквівалентності вичерпним для дослідження складних систем? Відповідь на це питання впливає з того факту, що відношення строгої взаємної подібності будується з урахуванням всіх без винятку активностей, які входять в алфавіт системи. Очевидно, що такий підхід не дозволяє виконувати узагальнюючі оцінки, які враховували б тільки деяку наперед задану підмножину властивостей. Відомий механізм узагальнення базується на застосуванні методик вертикальної та горизонтальної стратифікації. В рамках розглянутих підходів згадані методики використовують операції включення та приховання для стратифікації на рівні процесів. З метою абстрагування від внутрішньої поведінки системи на рівні окремих активностей, наприклад, у класичній алгебрі процесів CCS [60], використовують спеціальний тип активності — „невидима” або „тиха” активність. Використання такого підходу потребує удосконалення принципів еквівалентності з метою поширення їх

на процеси, у яких існує прихована внутрішня поведінка. Нагадаємо, що у даній версії алгебри процесів множина внутрішніх активностей Ψ складається з підмножин обробки Φ та підмножини затримок T : $\Psi = \Phi \cup T$. Внутрішня поведінка може моделюватися за допомогою активностей $\{\varphi\} \subseteq \Phi$. Розширення поняття „невидимої” активності спричинило необхідність пошуку нових підходів до аналізу поняття еквівалентності.

Основна проблема при введенні відношення еквівалентності полягає в тому, що просте виключення прихованих активностей є некоректним, оскільки вони можуть ініціювати видимі активності. При цьому виникає тупикова ситуація, яка порушує інформаційну цілісність системи. Для розв’язання цієї проблеми вводять новий тип переходів.

Означення 4.30. В маркованій системі з переходами $\left(\Pi, A, \left\{ \Rightarrow^a \mid a \in A \right\} \right)$ задамо перехід $\Rightarrow^a \subseteq \Pi \times \Pi$ для довіль-

них активностей $a \in A$ таким чином, що \Rightarrow^a є переходом \xrightarrow{a} , перед яким і після якого можуть бути розміщені внутрішні активності φ :

$$\Rightarrow^a := \begin{cases} \left(\xrightarrow{\varphi} \right)^* \circ \xrightarrow{a} \circ \left(\xrightarrow{\varphi} \right)^*, & a \neq \varphi, \\ \left(\xrightarrow{\varphi} \right)^*, & a = \varphi, \end{cases}$$

де символ \circ вказує на допустимість вставки в цьому місці композиції бінарних відношень, а $\left(\xrightarrow{\varphi} \right)^*$ — рефлексивне і транзитивне замикання бінарного відношення $\xrightarrow{\varphi}$.

Позначимо $\left(\overset{\varepsilon}{\longrightarrow}\right)^*$ через $\overset{\varepsilon}{\Rightarrow}$. Тоді для довільної

активності a існуватиме перехід $P \overset{a}{\Rightarrow} Q$ за умови існування проміжних процесів P' і Q' , які формуються послідовністю:

$$P \overset{\varepsilon}{\Rightarrow} P' \xrightarrow{a} Q' \overset{\varepsilon}{\Rightarrow} Q.$$

Отже, перехід $\overset{a}{\Rightarrow}$ допускає наявність внутрішніх переходів до і після звернення переходу \xrightarrow{a} .

Використання такого типу переходу дозволяє представляти відношення взаємної подібності, що спираються на процеси, які демонструють еквівалентну спостережувану поведінку без відображення іноді дуже великої кількості внутрішніх переходів.

Означення 4.31 (слабка взаємна подібність). Нехай в маркованій системі з переходами

$\left(\Pi, A, \left\{ \overset{a}{\Rightarrow} \mid a \in A \right\} \right)$ існує відношення $R \subseteq \Pi \times \Pi$ і процеси $P, Q \subseteq \Pi$ з допустимими станами $p, p' \in P$, $q, q' \in Q$ при $(p, q) \in R$. Тоді якщо для довільної активності a :

— для кожного переходу $p \xrightarrow{a} p'$ існує стан q' такий, що $q \overset{a}{\Rightarrow} q'$ і $(p', q') \in R$;

— для кожного переходу $q \xrightarrow{a} q'$ існує стан p' такий, що $p \overset{a}{\Rightarrow} p'$ і $(p', q') \in R$,

то відношення R будемо називати слабкою взаємною подібністю або спостережуваною еквівалентністю, а стани p і q — слабо взаємно подібними, якщо існує спостережувана

еквівалентність R , яка містить (p, q) . Позначатимемо слабко взаємно подібні стани як $p \approx q$.

Основні властивості слабкої взаємної подібності співпадають з властивостями строгої взаємної подібності для маркової системи з переходами. Доведемо ці властивості.

Теорема 4.12. У маркованій системі з переходами

$\left(\Pi, A, \left\{ \begin{array}{c} a \\ \Rightarrow \\ a \in A \end{array} \right\} \right)$ слабка взаємна подібність є еквівалент-

ним відношенням.

Д о в е д е н н я. Нехай множина процесів Π маркованої системи з переходами містить процеси $P, Q \subseteq \Pi$, кожен з яких включає стани $p, p' \in P$ і $q, q' \in Q$. Для доведення еквівалентності станів $p \approx q$, $p, q \in P$, необхідно, у відповідності до означення 4.23, довести рефлексивність, симетричність та транзитивність відношення R , $(p, q) \in R$. Спираючись на означення слабкої взаємної подібності, введемо заміну p' на p і q' на q . Одержана система міститиме активність a , яка спричиняє

тотожні переходи $p \xrightarrow{a} p$ і $q \xrightarrow{a} q$. Очевидно, що така система міститиме бінарні відношення:

$$I_p = \{(p, p) \mid p \in P\}, I_q = \{(q, q) \mid q \in Q\}.$$

Оскільки $I_p, I_q \subseteq R$ є відношеннями ідентичності, то вони є рефлексивними.

Симетричність $p \approx q$ випливає з умови слабкої взаємної подібності, оскільки $R^{-1} = \{(q, p) \mid (p, q) \in R\}$. За означенням 4.31 відношення R^{-1} також є відношенням взаємної подібності.

Звідси $p \approx q$ і $q \approx p$ є взаємно симетричними.

Транзитивність станів $p \approx q$ і $q \approx z$ зумовлює $p \approx z$.

Оскільки $p \approx q$, то $(p, q) \in R'$, відповідно якщо $q \approx z$, то $(q, z) \in R''$. Розглянемо бінарне відношення $R = \{(p, z) \mid (p, q) \in R', (q, z) \in R''\}$ для деякого q . Оскільки R' і R'' є відношеннями слабкої взаємної подібності, то і R є відношенням слабкої взаємної подібності. Отже, слабка взаємна подібність транзитивна.

Теорема 4.13. У маркованій системі з переходами

$\left(\Pi, A, \left\{ \begin{array}{c} a \\ \Rightarrow \mid a \in A \end{array} \right\} \right)$ слабка взаємна подібність є найбільшою

слабкою взаємною подібністю.

Д о в е д е н н я. Розглянемо, як і в попередньому випадку, процеси $P, Q \subseteq \Pi$, кожен з яких включає стани $p, p' \in P$ і $q, q' \in Q$. Доведення цієї теореми аналогічне доведенню теореми 4.2 про строгу взаємну подібність. В даному випадку також необхідно довести, що $\cup\{R\}$ є відношенням взаємної подібності. Для цього повинні виконуватися такі умови:

якщо $(p, q) \in \cup\{R\}$ і $p \xrightarrow{a} p'$, то існує стан q' такий,

що $q \xrightarrow{a} q'$ при $(p', q') \in \cup\{R\}$,

якщо $(p, q) \in \cup\{R\}$ і $q \xrightarrow{a} q'$, то існує стан p' такий,

що $p \xrightarrow{a} p'$ при $(p', q') \in \cup\{R\}$.

Нехай $(p, q) \in \cup\{R\}$, тоді існує таке відношення взаємної подібності R , що $(p, q) \in R$. Оскільки R — це відношення

взаємної подібності і $p \xrightarrow{a} p'$, то існує стан q' такий, що $q \xRightarrow{a} q'$ при $(p', q') \in R$. Виходячи з властивостей симетричності для $q \xrightarrow{a} q'$ приходимо до висновку, що існує стан p' такий, що $p \xRightarrow{a} p'$ при $(p', q') \in R$. Але пара $(p', q') \in \cup\{R\}$, тобто слабка взаємна подібність включає всі відношення взаємної подібності і, відповідно, є найбільшою слабкою взаємною подібністю.

Теорема 4.14. У маркованій системі з переходами $\left(\Pi, A, \left\{ \xRightarrow{a} \mid a \in A \right\} \right)$, що включає процеси $P, Q \subseteq \Pi$ з допустимими станами $p, p' \in P$ і $q, q' \in Q$, для довільної активності a два стани $p \approx q$ тоді і тільки тоді, коли:

— для переходу $p \xrightarrow{a} p'$ існує перехід $q \xRightarrow{a} q'$ такий, що $p' \approx q'$,

— для переходу $q \xrightarrow{a} q'$ існує перехід $p \xRightarrow{a} p'$ такий, що $p' \approx q'$.

Графічне зображення слабкої взаємної подібності станів p і q показане на рис.4.12.

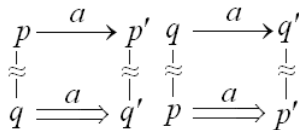


Рис.4.12. Слабка взаємна подібність станів p і q

Д о в е д е н н я. Оскільки, за означенням 4.31, для довільного переходу $p \xrightarrow{a} p'$ існує відповідний стан q' такий, що $q \xRightarrow{a} q'$ при $(p', q') \in R$, то для довільної активності a стани p і q слабко взаємно подібні тоді, коли для переходу $p \xrightarrow{a} p'$ існує перехід $q \xRightarrow{a} q'$ такий, що $p' \approx q'$. Аналогічно, якщо для кожного переходу $q \xrightarrow{a} q'$ існує стан p' такий, що $p \xRightarrow{a} p'$ при $(p', q') \in R$, то для довільної активності a два стани $p \approx q$ тоді, коли для переходу $q \xrightarrow{a} q'$ існує перехід $p \xRightarrow{a} p'$ такий, що $p' \approx q'$. Отже, пряме твердження даної теореми безпосередньо випливає з означення слабкої взаємної подібності. Для доведення зворотного твердження необхідно показати, що слабка взаємна подібність $p \approx q$ містить відношення (p, q) . Доведення згаданого факту базується на конструктивному підході до формування множини R відношень взаємної подібності. Для $p \sim q$ перш за все необхідно включити в множину R відношення (p, q) , оскільки для того, щоб пряме доведення було коректним, кожному переходу $p \xrightarrow{a} p'$ зі стану p повинен відповідати перехід $q \xRightarrow{a} q'$ зі стану q для деякого стану q' такого, що $(p', q') \in R$. Сформуємо спочатку множину R шляхом об'єднання всіх відношень взаємної подібності в маркованій системі з переходами $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$.

Оскільки за теоремою 4.13 слабка взаємна подібність є найбільшою взаємною подібністю, то $R = (p, q) \cup R$.

Отже, множина відношень R містить елемент (p, q) .
Теорему доведено.

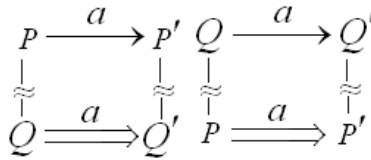


Рис.4.13. Слабка взаємна подібність процесів P і Q

Означення 4.32. Два процеси P і Q називають слабо взаємно подібними і позначають $P \approx Q$ тоді і тільки тоді, коли для довільної активності a існує відношення слабкої взаємної подібності R таке, що $P'RQ'$ (рис. 4.13).

Теорема 4.15. Якщо в маркованій системі з переходами $\left(\Pi, A, \left\{ \Rightarrow^a \mid a \in A \right\} \right)$ задане відношення слабкої взаємної подібності $R \subseteq \Pi \times \Pi$, і процеси $P, Q \subseteq \Pi$ при $(P, Q) \in R$, то строга взаємна подібність $P \sim Q$ є підмножиною слабкої взаємної подібності $P \approx Q$ на множині допустимих активностей A , тобто $\sim \subseteq \approx$.

Д о в е д е н н я. Нехай $P \sim Q$, і для деякої активності a існує перехід $P \xrightarrow{a} P'$. Тоді існує процес Q' такий, що $Q \xrightarrow{a} Q'$ при $(P', Q') \in R$. Але перехід \xrightarrow{a} можна розглядати як перехід \Rightarrow^a за умови відсутності в системі переходів $\xrightarrow{\varphi}$. Тому для кожного переходу $P \xrightarrow{a} P'$ існує перехід $Q \Rightarrow^a Q'$ при $P' \approx Q'$. Отже, $\sim \subseteq \approx$. Теорему доведено.

Теорема 4.16. Для P, Q і активності a справедливі такі спостережувані еквівалентності при $P \approx Q$:

1. $\langle \tau \rangle . \varphi . P \approx \langle \tau \rangle . P$;
2. $a . \varphi . P \approx a . P$;
3. $P + \varphi . P \approx \varphi . P$;
4. $a . (P + \varphi . Q) \approx a . (P + \varphi . Q) + a . Q$.

Д о в е д е н н я .

1. $\langle \tau \rangle . \varphi . P \approx \langle \tau \rangle . P$. Розглянемо еволюцію лівої і правої частини виразу, представивши процес P у префіксному вигляді $a . P'$, де a — стартова активність процесу P . Тоді еволюція

лівої частини матиме вигляд $\langle \tau \rangle . \varphi . a . P' \xrightarrow{\langle \tau \rangle} \varphi . a . P' \xrightarrow{\phi \ a} P'$,

а еволюція правої частини $\langle \tau \rangle . a . P' \xrightarrow{\langle \tau \rangle} a . P' \xrightarrow{a} P'$. За означенням слабкої взаємної подібності введемо заміну переходів у лівій частині $\xrightarrow{\varphi \ a} \text{на} \Rightarrow$ і в правій частині $\xrightarrow{a} \text{на} \Rightarrow$.

Одержимо $\langle \tau \rangle . \varphi . a . P' \xrightarrow{\langle \tau \rangle} \varphi . a . P' \Rightarrow P'$ і

$\langle \tau \rangle . a . P' \xrightarrow{\langle \tau \rangle} a . P' \Rightarrow P'$ і

$\langle \tau \rangle . \varphi . P \approx \langle \tau \rangle . P$. Отже, $\langle \tau \rangle . \varphi . P \approx \langle \tau \rangle . P$.

2. $a . \varphi . P \approx a . P$. Доведення аналогічне попередньому доведенню.

3. $P + \varphi . P \approx \varphi . P$. Розглянемо два варіанти в залежності від результату виконання операції вибору під дією деякої довільної активності a :

3.1. У лівій частині варіант $\frac{P \xrightarrow{a} P'}{P + \varphi . P \xrightarrow{a} P'}$ представ-

ляє вибір шляхом виконання переходу $(a . P' + \varphi . P) \xrightarrow{a} P'$.

Права частина $\varphi.P$ еволюціонує під дією активності a :

$$\varphi.a.P' \xRightarrow{a} P'.$$

3.2. У лівій частині варіант $\frac{\varphi.P \xrightarrow{\varphi} P}{P + \varphi.P \xrightarrow{\varphi} P}$ представ-

ляє вибір шляхом виконання переходу $(P + \varphi.P) \xrightarrow{\varphi} P$.

Права частина $\varphi.P$ еволюціонує під дією активності φ :

$$\varphi.P \xRightarrow{\varphi} P.$$

Отже, для обох випадків $P + \varphi.P \approx \varphi.P$.

$$4. a.(P + \varphi.Q) \approx a.(P + \varphi.Q) + a.Q.$$

Для доведення задамо відношення

$$R = \left\{ (a.(P + \varphi.Q), a.(P + \varphi.Q) + a.Q) \mid \forall P, Q \right\}$$

і розглянемо варіанти виконання операцій вибору.

Еволюція правої частини:

$$а). a.(P + \varphi.Q) + a.Q \xrightarrow{a} P.$$

$$б). a.(P + \varphi.Q) + a.Q \xrightarrow{a} \varphi.Q \xrightarrow{\varphi} Q.$$

$$в). a.(P + \varphi.Q) + a.Q \xrightarrow{a} Q.$$

Еволюція лівої частини:

$$а). a.(P + \varphi.Q) \xrightarrow{a} P.$$

$$б). a.(P + \varphi.Q) \xrightarrow{a} \varphi.Q \xrightarrow{\phi} Q.$$

Отже, вираз $a.(P + \varphi.Q) \approx a.(P + \varphi.Q) + a.Q$ еволюціонує до одного зі співвідношень: $P \approx Q$, $Q \approx P$, $Q \approx Q$ і $P \approx P$. Співвідношення $P \approx Q$ входить до умови теореми, а $P \approx Q, Q \approx Q$ та $P \approx P$ випливають з теореми 4.12. Теорему доведено.

Слабка взаємна подібність характеризується рядом властивостей. Основні з них доведемо в теоремі 4.17.

Теорема 4.17. *Нехай P, Q, G — процеси.*

Якщо $P \approx Q$, то:

1. $a.P \approx a.Q$ для довільної активності a ;
2. $\langle \tau \rangle.P \approx \langle \tau \rangle.Q$ для довільної префіксної затримки $\langle \tau \rangle$;
3. $\langle \tau \leftarrow t \rangle.P \approx \langle \tau \leftarrow t \rangle.Q$ для довільного виразу $\langle t \rangle$;
4. $P + G \approx Q + G$ для довільного процесу G ;
5. $v_1!P + v_2!G \approx v_1!Q + v_2!G$ для довільного процесу G ;
6. $P \mid G \approx Q \mid G$ для довільного процесу G ;
7. $P[K] \approx Q[K]$ для довільної функції перейменування K ;
8. $P \setminus A \approx Q \setminus A$ для довільної множини активностей A ;
9. $k \times P \approx k \times Q$ для довільного коефіцієнта клонування k ;
10. $\mathfrak{R}(X = P) \approx \mathfrak{R}(X = Q)$, де X — змінна процесу;
11. $\text{cycle}[x](X = P) \approx \text{cycle}[x](X = Q)$, де x — індекс циклу.

Д о в е д е н н я.

1. $a.P \sim a.Q$. Доведення $a.P \approx a.Q$ базується на умові рефлексивності слабкої взаємної подібності. Оскільки $a \approx a$ для довільної активності a , то після виконання переходів

$\overset{a}{a.P} \Rightarrow P$ та $\overset{a}{a.Q} \Rightarrow Q$ одержимо $P \approx Q$ за умовою теореми. Отже, $a.P \approx a.Q$.

2. $\langle \tau \rangle.P \approx \langle \tau \rangle.Q$. Виконавши префіксні затримки

$\overset{\langle \tau \rangle}{\langle \tau \rangle}.P \Rightarrow P$ та $\overset{\langle \tau \rangle}{\langle \tau \rangle}.Q \Rightarrow Q$ знову одержимо систему $P \approx Q$.

Тому префіксна затримка не змінює умови еквівалентності процесів.

3. $\langle \tau \leftarrow t \rangle . P \approx \langle \tau \leftarrow t \rangle . Q$. Оскільки в лівій і правій частині використовуємо один і той же вираз $\langle t \rangle$, то доведення даного пункту зводиться до доведення пункту 2.

4. $P + G \approx Q + G$. Задамо відношення

$$R = \left\{ (P + G, Q + G) \mid \forall P, Q, G, P \approx Q \right\}.$$

Тоді доведення того, що R є відношенням слабкої взаємної подібності, зводиться до доведення істинності умови: якщо $(P + G, Q + G) \in R$ і $P + G \xrightarrow{a} M$, то існує процес

$Q + G \xRightarrow{a} N$ такий, що $(M, N) \in R$. Нехай

$(P + G, Q + G) \in R$ і для деякої активності a існує перехід $a.(P + G) \xrightarrow{a} M$. Тоді можливі два варіанти формування процесу M :

4.1. Варіант $\frac{P \xrightarrow{a} P'}{P + G \xrightarrow{a} P'}$ представляє вибір процесу

P шляхом виконання переходу $(a.P + G) \xrightarrow{a} M$, оскільки

$$P + G \xRightarrow{a} P' \text{ і } M \equiv P'.$$

Оскільки $P \xrightarrow{a} P'$ і $P \approx Q$, то існує перехід $Q \xRightarrow{a} Q'$ і $P' \approx Q'$.

4.2. Варіант $\frac{G \xRightarrow{a} G'}{P + G \xRightarrow{a} G'}$ представляє вибір процесу G

шляхом виконання переходу $(P + a.G) \xrightarrow{a} M$, тому що

$$P + G \xrightarrow{a} G' \text{ і } M \equiv G'.$$

Оскільки $G \xrightarrow{a} G'$ і за умовою рефлексивності $G \approx G$, то існує перехід $G \xRightarrow{a} G'$ і $G' \approx G'$.

Отже, для першого і другого варіантів виконання операції вибору $Q + G \xRightarrow{a} N$ і $(M, N) \in R$.

5. $v_1!P + v_2!G \approx v_1!Q + v_2!G$. Доведемо, що при

$$(v_1!P + v_2!G, v_1!Q + v_2!G) \in R \text{ і } v_1!P + v_2!G \xrightarrow{v_1} M$$

існує процес $v_1!Q + v_2!G \xRightarrow{v_1} N$, для якого $(M, N) \in R$.

Нехай $(v_1!P + v_2!G, v_1!Q + v_2!G) \in R$ і для деякої змінної $v_1 = \text{true}$ існує перехід $(v_1!P + v_2!G) \xrightarrow{v_1} M$. Тоді можливі два варіанти формування процесу M :

5.1. Варіант $\frac{v_1!P \xrightarrow{v_1} P'}{v_1!P + v_2!G \xrightarrow{a} P'}$ представляє вибір

процесу P шляхом виконання переходу $(v_1!P + v_2!G) \xrightarrow{v_1} M$, через те, що $v_1!P + v_2!G \xRightarrow{v_1} P'$ і $M \equiv P'$.

Оскільки $v_1!P \xrightarrow{v_1} P'$ і $v_1!P \approx v_1!Q$, то існує перехід $v_1!Q \xRightarrow{v_1} Q'$ і $P' \approx Q'$.

5.2. Варіант $\frac{v_2!G \xRightarrow{v_2} G'}{v_1!P + v_2!G \xRightarrow{v_2} G'}$ представляє вибір про-

цесу G шляхом виконання переходу $(v_1!P + v_2!G) \xRightarrow{v_2} M$,

через те, що $v_1!P + v_2!G \Rightarrow^{v_2} G'$ і $M \equiv G'$. Оскільки $v_2!G \Rightarrow^{v_2} G'$ і за умовою рефлексивності $v_2!G \approx v_2!G$, то існує перехід $v_2!G \Rightarrow^{v_2} G'$ і $G' \approx G'$.

Отже, для першого і другого варіантів операція вибору відбувається шляхом виконання переходу $v_1!Q + v_2!G \Rightarrow^{v_2} N$ при $(M, N) \in R$.

6. $P \mid G \approx Q \mid G$. Задамо відношення

$$R = \left\{ (P \mid G, Q \mid G) \mid \forall P, Q, G, P \approx Q \right\}$$

і активність a , яка входить в алфавіт хоча б одного з процесів P, Q або G . Тоді R є відношенням слабкої взаємної подібності за умови, що при $(P \mid G, Q \mid G) \in R$ і $P \mid G \xrightarrow{a} M$ існує процес $Q \mid G \Rightarrow^a N$ такий, що $(M, N) \in R$.

Розглянемо три варіанти розвитку системи $P \mid G$ під дією активності a .

Необхідно довести, що відношення R є відношенням слабкої взаємної подібності для кожного зі згаданих варіантів розвитку паралельної композиції.

6.1. При $\frac{P \xrightarrow{a} P'}{P \mid G \xrightarrow{a} P' \mid G}$ існує перехід

$P \mid G \xrightarrow{a} M$, тому що $P \xrightarrow{a} P'$ і $M = P' \mid G$.

Оскільки $P \xrightarrow{a} P'$ і $P \approx Q$, то за означенням слабкої взаємної подібності існує перехід $Q \xRightarrow{a} Q'$ і $P' \approx Q'$. Аналогічно при $Q \xrightarrow{a} Q'$ і $P \approx Q$ існує перехід $P \xRightarrow{a} P'$ і $P' \approx Q'$. Виходячи з означення відношення слабкої взаємної подібності: $(P' \mid G, Q' \mid G) \in R$. Отже, $N = Q' \mid G$, що доводить збереження слабкої взаємної подібності у першому випадку.

6.2. При $\frac{G \xrightarrow{a} G'}{P \mid G \xrightarrow{a} P \mid G'}$ існує перехід $P \mid G \xrightarrow{a} M$, оскільки $G \xrightarrow{a} G'$ і $M = P \mid G'$.

Через те, що $G \xrightarrow{a} G'$ при $P \mid G \xrightarrow{a} P \mid G'$ і при $Q \mid G \xrightarrow{a} Q \mid G'$, то за властивістю рефлексивності $G' \approx G'$. Звідси $(P \mid G', Q \mid G) \in R$. Отже, $N = Q \mid G'$, тобто властивість слабкої взаємної подібності доведена для другого випадку розвитку паралельної композиції.

6.3. При $\frac{P \xrightarrow{a} P' \quad G \xrightarrow{\bar{a}} G'}{P \mid G \xrightarrow{\tau} P' \mid G'}$ можливий один із видів взаємодії: $P \triangleright G$, $P \leftrightarrow G$, $P \triangleleft \triangleright G$. Тоді існує перехід $P \mid G \xrightarrow{\tau} M$, оскільки $P \xrightarrow{a} P'$, і $G \xrightarrow{\bar{a}} G'$, і $M = P' \mid G'$.

Даний варіант паралельної композиції об'єднує два попередні випадки. З існування переходу $P \xrightarrow{a} P'$ і $P \approx Q$ випливає $Q \xRightarrow{a} Q'$ і $P' \approx Q'$. Аналогічно, при $Q \xrightarrow{a} Q'$

і $P \approx Q$ для слабкої взаємної подібності $P \xRightarrow{a} P'$ і $P' \approx Q'$.
 Перехід $G \xrightarrow{\bar{a}} G'$ при $P \mid G \xrightarrow{\tau} P' \mid G'$ і при $Q \mid G \xrightarrow{\tau} Q' \mid G'$ не порушує слабкої взаємної подібності через рефлексивність $G' \approx G'$. Звідси випливає $(P' \mid G', Q' \mid G') \in R$, тому $N = Q' \mid G'$, $(M, N) \in R$ і для третього варіанту паралельної композиції, що свідчить про істинність виразу $P \mid G \approx Q \mid G$. Операція $\frac{P \xrightarrow{a} P' \mid G \xrightarrow{\bar{a}} G'}{P \mid G \xrightarrow{\tau} P' \mid G'}$ є узагальненням операцій $P \triangleright G$, $P \leftrightarrow G$, $P \triangleleft \triangleright G$. Отже, справедливність виразів $(P \triangleright G) \approx (Q \triangleright G)$, $(P \leftrightarrow G) \approx (Q \leftrightarrow G)$, $(P \triangleleft \triangleright G) \approx (Q \triangleleft \triangleright G)$ можна довести за аналогічною схемою.

7. $P[K] \approx Q[K]$. Задамо

$R = \{(P[K], Q[K]) \mid P \approx Q\}$. Для деякої активності a існує слабка взаємна подібність, якщо при $(P[K], Q[K]) \in R$ і

$P[K] \xrightarrow{a} M$ існує перехід $Q[K] \xRightarrow{a} N$ такий, що $(M, N) \in R$.

Якщо a належить до області означення K , то перехід $P[K] \xrightarrow{K(a)} P'[K]$ безпосередньо одержуємо з переходу $P \xrightarrow{a} P'$ і $M = P'[K]$. Тоді існує перехід $Q \xRightarrow{a} Q'$ при $P' \approx Q'$. Отже, $Q[K] \xrightarrow{K(a)} Q'[K]$ і $N = Q'[K]$,

$(M, N) \in R$. Випадає, коли a не належить до області визначення K , не потребує доведення через свою очевидність.

8. $P \setminus A \approx Q \setminus A$. Задамо $R = \left\{ (P \setminus A, Q \setminus A) \mid P \approx Q \right\}$

з умовою слабкої взаємної подібності, якщо існує активність a , для якої при $(P \setminus A, Q \setminus A) \in R$ і $P \setminus A \xrightarrow{a} M$ існує процес

$Q \setminus A \xRightarrow{a} N$ такий, що $(M, N) \in R$.

Якщо $a \notin A$, то $P \setminus A \xrightarrow{a} P' \setminus A$ зводиться до $P \xrightarrow{a} P'$ і $M = P' \setminus A$. Тоді існує перехід $Q \xrightarrow{a} Q'$ при

$P' \approx Q'$. Отже, $Q \setminus A \xRightarrow{a} Q' \setminus A$ і $N = Q' \setminus A$, $(M, N) \in R$. У випадку $a \in A$ перехід не відбувається.

9. $k \times P \approx k \times Q$. Задамо $R = \left\{ (k \times P, k \times Q), P \approx Q \right\}$

зі слабкою взаємною подібністю за умови існування активності a , для якої при $(k \times P, k \times Q) \in R$ і $k \times P \xrightarrow{a} M$ існує

процес $k \times Q \xRightarrow{a} N$ такий, що $(M, N) \in R$.

Відповідно до семантичного правила даної операції:

$$\frac{P \xrightarrow{a} P'_1 \quad P \xrightarrow{a} P'_2 \quad \dots \quad P \xrightarrow{a} P'_k}{P \xrightarrow{a} P'_1 \mid P'_2 \mid \dots \mid P'_k},$$

звершення активності a призводить до формування паралельної композиції $P \xrightarrow{a} P'_1 \mid P'_2 \mid \dots \mid P'_k$. Відповідно

$M = P'_1 \mid P'_2 \mid \dots \mid P'_k$. Для кожного клона, який утворюється внаслідок переходу $P \xrightarrow{a} P'_i$ при $1 \leq i \leq k$ існує відповідний

клон, який утворюється внаслідок переходу $Q \xRightarrow{a} Q'_i$ при $P'_i \approx Q'_i$. Оскільки $Q \xRightarrow{a} Q'_1 \mid Q'_2 \mid \dots \mid Q'_k$, то $N = Q'_1 \mid Q'_2 \mid \dots \mid Q'_k$, а отже, $k \times P \approx k \times Q$.

10. $\mathfrak{R}(X = P) \approx \mathfrak{R}(X = Q)$. У випадку скінченності рекурсії $S \{ \mathfrak{R}(X = P) / X \}$ задамо відношення

$$R = (S \{ \mathfrak{R}(X = P) / X \}, S \{ \mathfrak{R}(X = Q) / X \}).$$

Покажемо, що R є відношенням слабкої взаємної подібності після присвоєння $S = X$, тобто існує активність a , для якої при $(S \{ \mathfrak{R}(X = P) / X \}, S \{ \mathfrak{R}(X = Q) / X \}) \in R$ і $S \{ \mathfrak{R}(X = P) / X \} \xrightarrow{a} M$ існує процес

$$S \{ \mathfrak{R}(X = Q) / X \} \xRightarrow{a} N \text{ такий, що } (M, N) \in R.$$

Розглянемо ряд випадків структури процесу S .

10.1. $S \equiv X$. Відповідно до семантичного правила рекурсії:

$$\frac{P \{ \mathfrak{R}(X = P) / X \} \xrightarrow{a} P'}{\mathfrak{R}(X = P) \xrightarrow{a} P'},$$

звершення активності a породжує перехід $P \{ \mathfrak{R}(X = P) / X \} \xrightarrow{a} P'$, з чого випливає $M = P'$.

Оскільки $P \{ \mathfrak{R}(X = P) / X \} \xrightarrow{a} P'$ і $P \approx Q$, то за означенням слабкої взаємної подібності існує перехід

$$P \{ \mathfrak{R}(X = Q) / X \} \xRightarrow{a} N \text{ і } P' \approx Q'.$$

Тому $Q \{ \mathfrak{R}(X = Q) / X \} \xRightarrow{a} N$. Отже, $(M, N) \in R$, а відношення R є відношенням слабкої взаємної подібності.

10.2. $S \equiv a.S_1$. Виходячи з семантики операції префіксації, приходимо до висновку, що $S \xrightarrow{a} S'$ і $a.S_1 \xrightarrow{a} S'$. Звідси $(S', S') \in R$ за умовою рефлексивності.

10.3. $S \equiv \langle \tau \rangle . S_1$. Доведення аналогічне доведенню випадку $S \equiv a.S_1$.

10.4. $S \equiv \langle \tau \leftarrow t \rangle . S_1$. Розглянемо перехід:

$$S \{ \mathfrak{R}(X = P) / X \} \xrightarrow{\langle t \rangle} S_1 [t/\tau] \{ \mathfrak{R}(X = P) / X \}.$$

Виходячи з умови $P \approx Q$, маємо

$$S \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{\langle t \rangle} S_1 [t/\tau] \{ \mathfrak{R}(X = Q) / X \}.$$

Звідси

$$(S_1 [t/\tau] \{ \mathfrak{R}(X = P) / X \}, S_1 [t/\tau] \{ \mathfrak{R}(X = Q) / X \}) \in R.$$

10.5. $S \equiv S_1 + S_2$. Нехай

$$S_1 \{ \mathfrak{R}(X = P) / X \} \xrightarrow{a} M. \text{ Оскільки}$$

$$S_1 \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{a} N \text{ при } (M, N) \in R, \text{ то}$$

$$S_1 \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{a} N \text{ при } (M, N) \in R.$$

10.6. $S \equiv S_1 \mid S_2$. Розглянемо три варіанти.

10.6.1. Нехай активність a впливає лише на процес S_1 .

Тоді $S_1 \{ \mathfrak{R}(X = P) / X \} \xrightarrow{a} M$. За означенням слабкої взаємної подібності $S_1 \{ \mathfrak{R}(X = Q) / X \} \xrightarrow{a} N$ при $(M, N) \in R$. Таким чином, $S \{ \mathfrak{R}(X = P) / X \} \xrightarrow{a} M \mid S_2$

і $S \left\{ \mathfrak{R}(X = Q) / X \right\} \stackrel{a}{\Rightarrow} N \mid S_2$ при $(M, N) \in R$. Тоді для деякого S' представимо $S' \left\{ \mathfrak{R}(X = P) / X \right\} \equiv M$ і $S' \left\{ \mathfrak{R}(X = Q) / X \right\} \equiv N$. При $S'' \equiv S' \mid S_2$ одержимо $(M \mid S_2, N \mid S_2) \in R$.

10.6.2. Нехай активність a впливає лише на процес S_2 . Тоді доведення аналогічне попередньому за умови симетричності.

10.6.3. Нехай активність a впливає на процеси S_1 і S_2 . Тоді актуальні переходи: $S_1 \left\{ \mathfrak{R}(X = P) / X \right\} \xrightarrow{a} M_1$ і $S_2 \left\{ \mathfrak{R}(X = P) / X \right\} \xrightarrow{\bar{a}} M_2$. За означенням слабкої взаємної подібності $S_1 \left\{ \mathfrak{R}(X = Q) / X \right\} \stackrel{a}{\Rightarrow} N_1$ при $(M_1, N_1) \in R$ і $S_2 \left\{ \mathfrak{R}(X = Q) / X \right\} \stackrel{a}{\Rightarrow} N_2$ при $(M_2, N_2) \in R$. Виходячи з цього, отримуємо $((M_1 \mid N_1), (M_2 \mid N_2)) \in R$.

10.7. $S \equiv S_1 \setminus A$. Якщо $S_1 \left\{ \mathfrak{R}(X = P) / X \right\} \xrightarrow{a} M_1$, то за означенням слабкої взаємної подібності $S_1 \left\{ \mathfrak{R}(X = Q) / X \right\} \stackrel{a}{\Rightarrow} N_1$ при $(M_1, N_1) \in R$. Виконавши операції $M = M_1 \setminus A$ і $N = N_1 \setminus A$, одержимо $(M, N) \in R$.

10.8. $S \equiv S_1 [K]$. Доведення аналогічне випадку $S \equiv S_1 \setminus A$.

10.9. $S \equiv \mathfrak{R}(Y = S_1)$. Актуальний перехід $\mathfrak{R}(Y = S_1) \left\{ \mathfrak{R}(X = P) / X \right\} \xrightarrow{a} M$ або

$S \{ \mathfrak{R}(Y = S_1) / Y \} \{ \mathfrak{R}(X = P) / X \} \xrightarrow{a} M$. За означенням слабкої взаємної подібності

$$S \{ \mathfrak{R}(Y = S_1) / Y \} \{ \mathfrak{R}(X = Q) / X \} \xRightarrow{a} N \text{ при } (M, N) \in R.$$

Звідси $\mathfrak{R}(Y = S_1) \{ \mathfrak{R}(X = Q) / X \} \xRightarrow{a} N$ при $(M, N) \in R$.

11. $\text{cycle}[x](X = P) \approx \text{cycle}[x](X = Q)$. Для процесу S доведення $S \equiv \text{cycle}[x](Y = S_1)$ аналогічне доведенню випадку $S \equiv \mathfrak{R}(Y = S_1)$. Теорему доведено.

Іноді важливо мати можливість використання властивості слабкої взаємної подібності тільки для підмножини процесів за умови, що решта процесів порівнюваних систем відповідають умовам слабкої взаємної подібності. Такий вид подібності будемо називати квазіслабкою взаємною подібністю.

Означення 4.33 (квазіслабка взаємна подібність).

Нехай в маркованій системі з переходами $\left(\Pi, A, \left\{ \xRightarrow{a} \mid a \in A \right\} \right)$ існує відношення $S \subseteq \Pi \times \Pi$, і процеси

$P, Q \subseteq \Pi$ мають допустимі стани $p, p' \in P$, $q, q' \in Q$ при $(p, q) \in S$. Тоді якщо для довільної активності a :

— для кожного переходу $p \xrightarrow{a} p'$ існує стан q' такий, що $q \xRightarrow{a} q'$ при $p' \approx S \approx q'$,

— для кожного переходу $q \xrightarrow{a} q'$ існує стан p' такий, що $p \xRightarrow{a} p'$ при $p' \approx S \approx q'$,

відношення S будемо називати квазіслабкою взаємною подібністю, якщо $a \in \text{Act}$.

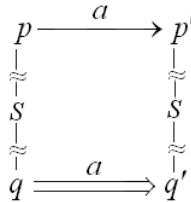


Рис.4.14. Квазіслабка взаємна подібність станів p і q

Теорема 4.18. *Якщо у маркованій системі з переходами*

$\left(\Pi, A, \left\{ \begin{array}{l} a \\ \Rightarrow | a \in A \end{array} \right\} \right)$ *відношення S — це відношення квазіслабкої*

взаємної подібності, то повне об'єднання всіх можливих відношень квазіслабкої взаємної подібності зі слабкою взаємною подібністю $(\approx \cup S)^$ є відношенням слабкої взаємної подібності і $S \subseteq \approx$.*

Д о в е д е н н я. Розглянемо стани марковної системи з переходами p, p' і q, q' за умови існування процесів $P, Q \subseteq \Pi$ таких, що $p, p' \in P$ і $q, q' \in Q$. Позначимо всі можливі об'єднання відношень квазіслабкої та слабкої взаємної подібності через $K = (\approx \cup S)^*$ і розглянемо pKq , де $(p, q) \in K$. З умови pKq випливає, що $p \approx q$ і pSq . Відношення $p \approx q$ є еквівалентним відношенням за теоремою 4.12. Для доведення еквівалентності pSq необхідно показати, що воно, відповідно до означення 4.23, є рефлексивним, симетричним і транзитивним.

Спираючись на означення квазіслабкої взаємної подібності, введемо заміну p' на p і q' на q . Якщо існує спільна актив-

ність a , то вона породжує тотожні переходи $p \xrightarrow{a} p$ і $q \Rightarrow q$. Тому в заданій системі підмножина I множини S містить бінарні відношення $I = \{(p, p), (q, q)\}$. Оскільки $I \subseteq S$ містить елементи відношення ідентичності, то вони є рефлексивними.

Симетричність квазіслабкої взаємної подібності, тобто еквівалентність відношень pSq і qSp , випливає з симетричності означення 4.33. Строге доведення даного факту аналогічне застосованому щодо теореми 4.12.

Транзитивність слабкої взаємної подібності станів pSq припускає наявність стану z такого, що $pS'zS''q$. Тому розглянемо бінарне відношення $S = \{(p, q) \mid (p, z) \in S', (z, q) \in S''\}$ для деякого z . Оскільки S' і S'' є відношеннями слабкої взаємної подібності, то і S є відношенням слабкої взаємної подібності. Звідси випливає, що відношення S також транзитивне. Отже, K є відношенням слабкої взаємної подібності. Але $S \subseteq K$, тому $S \subseteq \approx$. Теорему доведено.

4.7. Алгоритми визначення еквівалентності

4.7.1. Прямий алгоритм визначення строгої взаємної подібності

Прямий алгоритм визначення строгої взаємної подібності моделей полягає в безпосередньому застосуванні означення 4.27 до всіх станів, що утворюють бінарні відношення. Розглянемо застосування цього алгоритму на прикладі маркованого системи з

переходами $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$,

де $\Pi = \{s_1, s_2, s_3\} \cup \{t_1, t_2\}$ — множина станів, $A = \{a, b\}$ — множина активностей,

$\xrightarrow{a} = \{(s_1, s_2), (s_1, s_3)\} \cup \{(t_1, t_2)\}$ — множина переходів під дією активності a ,

$\xrightarrow{b} = \{(s_2, s_3), (s_3, s_3)\} \cup \{(t_2, t_2)\}$ — множина переходів під дією активності b .

Графічне представлення даної маркованій системи показано на рис.4.15.

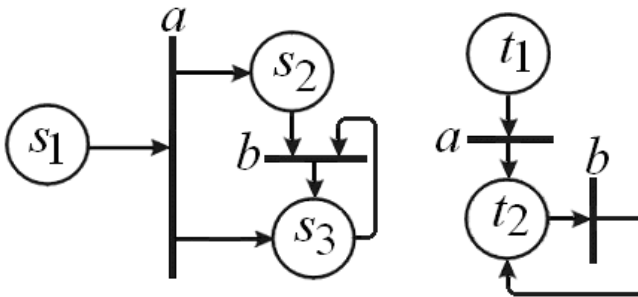


Рис.4.15. Графічне представлення маркованій системи з переходами

Визначимо строгу взаємну подібність станів $s_1 \sim t_1$. Для цього задамо бінарне відношення R таким чином, щоб $(s_1, t_1) \in R$:

$$R = \{(s_1, t_1), (s_2, t_2), (s_3, t_2)\}.$$

Тепер потрібно довести, що $R \in$ відношенням строгої взаємної подібності, тобто відповідає умові із означення 4.27. Для цього для кожної пари станів, що є елементом відношення R , необхідно дослідити всі можливі переходи з першого стану і визначити, чи співпадають вони з відповідними переходами з другого стану. Співпадіння переходів слід розуміти як той факт, що вони відбуваються під дією однієї і тієї ж активності. Подальші кроки алгоритму полягають в послідовній перевірці всіх елементів відношення R .

Елемент $(s_1, t_1) \in R$:

1. Переходи зі стану s_1 : переходу $s_1 \xrightarrow{a} s_2$ відповідає перехід $t_1 \xrightarrow{a} t_2$ при $(s_2, t_2) \in R$; переходу $s_1 \xrightarrow{a} s_3$ відповідає перехід $t_1 \xrightarrow{a} t_2$ при $(s_3, t_2) \in R$.

2. Переходи зі стану t_1 : переходу $t_1 \xrightarrow{a} t_2$ відповідає перехід $s_1 \xrightarrow{a} s_2$ при $(s_2, t_2) \in R$; переходу $t_1 \xrightarrow{a} t_2$ відповідає перехід $s_1 \xrightarrow{a} s_3$ при $(s_3, t_2) \in R$.

Елемент $(s_2, t_2) \in R$:

1. Переходи зі стану s_2 : переходу $s_2 \xrightarrow{b} s_3$ відповідає перехід $t_2 \xrightarrow{b} t_2$ при $(s_3, t_2) \in R$.

2. Переходи зі стану t_2 : переходу $t_2 \xrightarrow{b} t_2$ відповідає перехід $s_2 \xrightarrow{b} s_3$ при $(s_3, t_2) \in R$.

Елемент $(s_3, t_2) \in R$:

1. Переходи зі стану s_3 : переходу $s_3 \xrightarrow{b} s_3$ відповідає перехід $t_2 \xrightarrow{b} t_2$ при $(s_3, t_2) \in R$.

2. Переходи зі стану t_2 : переходу $t_2 \xrightarrow{b} t_2$ відповідає перехід $s_3 \xrightarrow{b} s_3$ при $(s_3, t_2) \in R$.

Отже, R є відношенням строгої взаємної подібності і, відповідно, $s_1 \sim t_1$. Аналогічно може бути доведено строгую взаємну подібність $s_2 \sim s_3$ або $s_3 \sim t_2$. Для цього необхідно визначити відповідні бінарні відношення. У першому випадку $R = \{(s_2, s_3), (s_3, s_3)\}$, а у другому — $R = \{(s_3, s_3), (t_2, t_2)\}$.

Перевірка відповідності переходів дає можливість впевнитися в тому, що згадані стани є строго взаємно подібними.

Такий підхід є громіздким, оскільки кількість перевірок зростає пропорційно 2^{n^2} , де n — кількість станів, що підлягають перевірці. Практичне застосування його можливе тільки на невеликих моделях. Тому актуальним є пошук алгоритмів, які дозволяють скоротити кількість перевірок.

4.7.2. Прискорений алгоритм визначення строгої взаємної подібності

Альтернативним підходом до встановлення строгої взаємної подібності є пошук хоча б однієї пари станів $s \neq t$, яка б заперечувала строгу взаємну подібність моделей. Такий пошук можливо провести у вигляді послідовного алгоритму, за допомогою якого спочатку намагаються довести, що $s \neq t$. Якщо цей факт не доведено, то наступний крок полягає в доведенні $s \sim t$.

Нехай $\left(\Pi, A, \left\{ \xrightarrow{a} \mid a \in A \right\} \right)$ — маркована модель з

переходами, яка складається з таких компонентів:

$\Pi = \{ s_1, s_2, \dots, s_{n_s} \} \cup \{ t_1, t_2, \dots, t_{n_t} \}$, $n_s, n_t \in N$, де N — множина натуральних чисел;

$A = \{ a_k \}_{k=1}^z$, $z \in N$; $\xrightarrow{a} \subseteq \Pi \times \Pi$, $a \in A$.

Прискорений алгоритм визначення строгої взаємної подібності (рис.4.16) дозволяє розглядати пари станів

$$(s, t) \in \left\{ (s_x, t_y) \right\}_{l=1}^{|R|},$$

де $(s_x, t_y)_l \in R$, $x \in \{ 1, \dots, n_s \}$, $y \in \{ 1, \dots, n_t \}$.

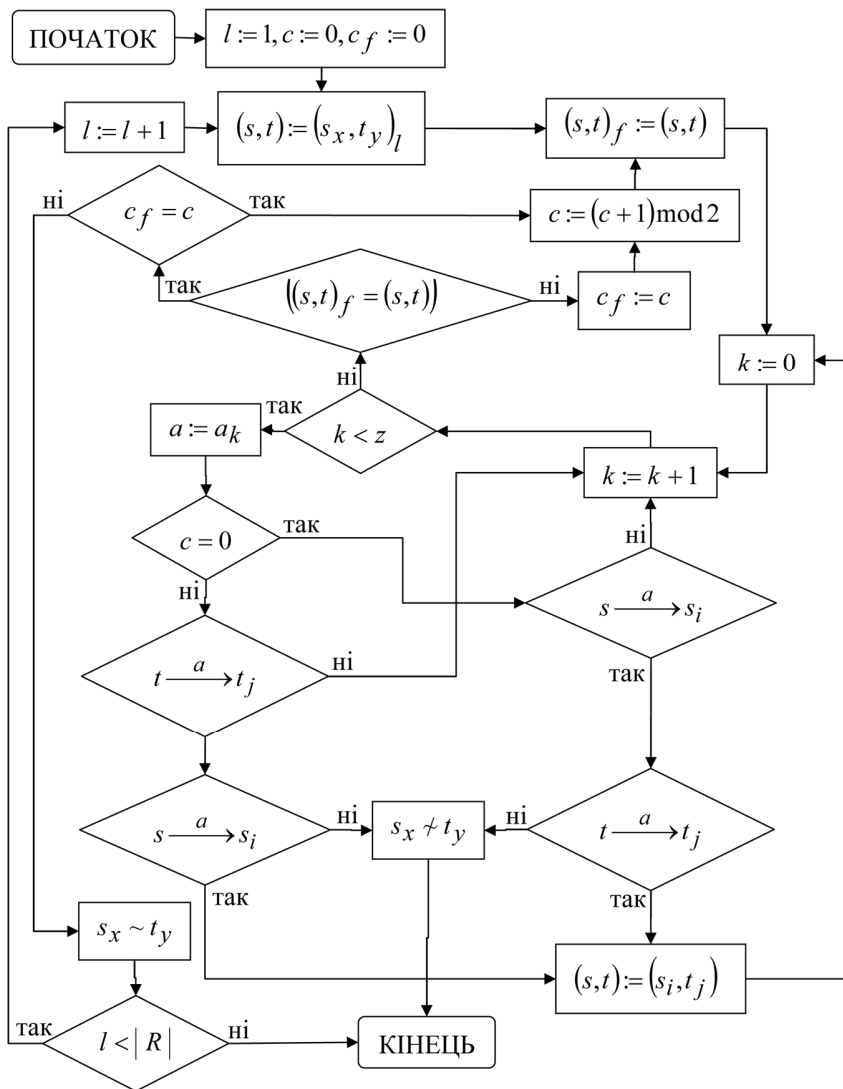


Рис.4.16. Алгоритм визначення строгої взаємної подібності

Ототожнимо порядок розгляду строгої взаємної подібності пар станів (s, t) з порядком розміщення згаданих пар у множині R , який відповідає значенню індексу l . Алгоритм

може мати два варіанти розвитку в залежності від вибору переходу, який відбувається на першому кроці обробки поточної пари станів. Таким чином, другий крок є завжди залежним від першого в зв'язку з тим, що вибраний на другому кроці перехід має відбуватися під дією тієї ж активності, що і перехід на першому кроці.

У випадку $c = 0$ розгортається перший варіант алгоритму, у якому для реалізації переходу $s \xrightarrow{a} s_i$ при $s_i \in \Pi$, $1 \leq i \leq n_s$, виконується підбір поточної активності $a \in \{a_k\}_{k=1}^z$. Тоді другим кроком алгоритму буде перевірка можливості виконання переходу $t \xrightarrow{a} t_j$ під дією вибраної на попередньому кроці активності a .

За умови, що існують переходи $s \xrightarrow{a} s_i$ і $t \xrightarrow{a} t_j$, для продовження аналізу даної послідовності переходів, встановлюють нову поточну пару станів $(s, t) := (s_i, t_j)$. Під час реалізації цих послідовностей переходів може виникнути ситуація, коли черговому переходу $s \xrightarrow{a} s_i$ не відповідає жоден перехід $t \xrightarrow{a} t_j$. У цьому випадку робимо висновок, що не існує строгої взаємної подібності між початковими станами даної послідовності: $s_x \neq t_y$. Якщо послідовність переходів $(s_i, \xrightarrow{a})_i$ приводить до стану s_i , для переходу з якого не існує жодної активності $a \in A$, необхідно проаналізувати можливість переходу $t \xrightarrow{a} t_j$ шляхом підбору відповідної активності a з тієї ж множини A . Ця гілка алгоритму відповідає значенню параметра $c = 1$. Не виключено, що знайдена активність a може спричинити продовження послідовності переходів $(t_j, \xrightarrow{a})_j$, кожному елементу якої відповідатиме перехід $s \xrightarrow{a} s_i$, що відбувається під дією активності, вибраної на

першому кроці алгоритму. Якщо черговому переходу $t \xrightarrow{a} t_j$ не відповідає жодний перехід $s \xrightarrow{a} s_i$, то це буде свідчити про те, що між початковими станами s_x і t_y відсутня строга взаємна подібність, тобто $s_x \not\sim t_y$. Якщо в ході виконання алгоритм вибирає поточний стан (s, t) , для якого після перебору всіх можливих активностей не знайдено ні переходу $s \xrightarrow{a} s_i$, ні переходу $t \xrightarrow{a} t_j$, то такий стан вказує на той факт, що $s_x \sim t_y$.

Алгоритм зупиняється за умови виявлення першої ж пари $s_x \not\sim t_y$ з висновком про те, що система та її модель не мають строгої взаємної подібності станів. Зупинка алгоритму передбачена також у випадку, коли визначена строга взаємна подібність всіх пар станів множини R . При цьому вважають, що система та її модель строго взаємно подібні.

4.7.3. Прискорений алгоритм визначення слабкої взаємної подібності

Прискорений алгоритм визначення слабкої взаємної подібності є модифікацією попереднього алгоритму для визначення строгої взаємної подібності. Тому його роботу будемо розглядати також з використанням маркованої системи з переходами

$$\left(\Pi, A, \left\{ \Rightarrow \mid \alpha \in A \right\} \right), \quad \text{де } \Pi = \{ s_1, s_2, \dots, s_{n_s} \} \cup \{ t_1, t_2, \dots, t_{n_t} \};$$

$$A = \{ \alpha_k \}_{k=1}^z;$$

$$\Rightarrow \subseteq \Pi \times \Pi, \alpha = \alpha \vee \varphi.$$

Прискорений алгоритм визначення слабкої взаємної подібності (рис.4.17) забезпечує послідовний розгляд пар станів

$$(s, t) \in \left\{ (s_x, t_y)_l \right\}_{l=1}^{|R|},$$

де $(s_x, t_y)_l \in R$, $x \in \{1, \dots, n_s\}$, $y \in \{1, \dots, n_t\}$.

Для поточної пари станів (s, t) і параметра $c = 0$ виконаємо перехід $s \xrightarrow{\alpha} s_i$ зі стану s в деякий довільний стан $s_i \in \Pi$, $1 \leq i \leq n_s$. За умови, що перехід $s \xrightarrow{\alpha} s_i$ успішно завершений, виконаємо перехід $t \xRightarrow{a} t_j$ зі стану t в деякий довільний стан $t_j \in \Pi$, $1 \leq j \leq n_t$, при $\alpha = a$ або перехід $t \xRightarrow{\varphi} t$ при $\alpha = \varphi$. Після успішного виконання переходів $s \xrightarrow{a} s_i$ і $t \xRightarrow{\varphi} t_j$ або $t \xRightarrow{\varphi} t$ формують нову поточну пару станів $(s, t) := (s_i, t_j)$. Якщо після успішного виконання переходу $s \xrightarrow{\alpha} s_i$ неможливо виконати жодного переходу $t \xRightarrow{a} t_j$, то це свідчить про те, що початкові стани s_x і t_y не пов'язані слабкою взаємною подібністю, тобто $s_x \not\approx t_y$.

Ситуація, коли неможливо виконати перехід $s \xrightarrow{\alpha} s_i$ при $c = 0$, веде до зміни порядку аналізу станів. У цьому випадку встановлюють значення параметра $c = 1$ і роблять спробу виконати перехід $t \xrightarrow{\alpha} t_j$ зі стану t в деякий довільний стан $t_j \in \Pi$, $1 \leq j \leq n_t$.

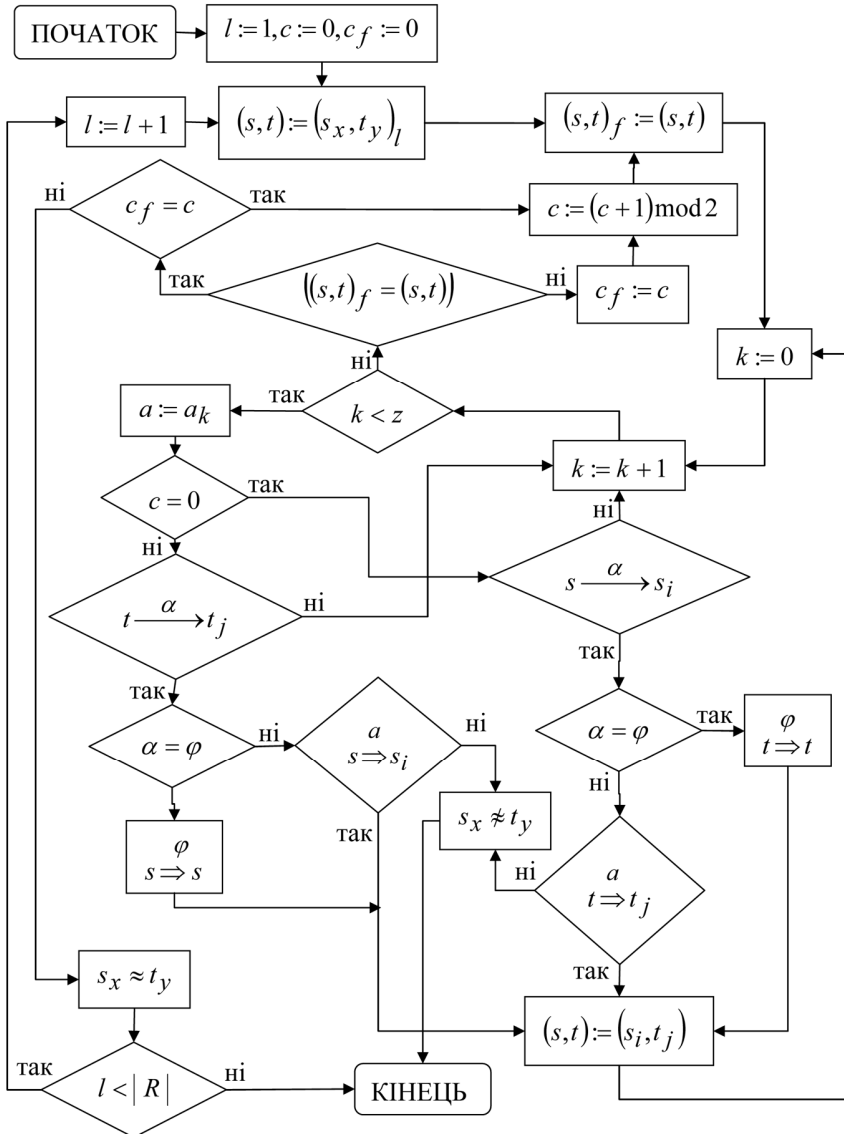


Рис.4.17. Алгоритм визначення слабкої взаємної подібності

Після успішного виконання переходу $t \xrightarrow{\alpha} t_j$, виконують перехід $s \xRightarrow{a} s_i$ зі стану s в деякий довільний стан $s_i \in \Pi, 1 \leq i \leq n_s$, при $\alpha = a$ або перехід $s \xRightarrow{\varphi} s$ при $\alpha = \varphi$. Нова поточна пара станів $(s, t) := (s_i, t_j)$ може бути сформована після успішного виконання переходів $t \xrightarrow{a} t_j$ і $s \xRightarrow{a} s_i$ або $s \xRightarrow{\varphi} s$.

Якщо ж після успішного виконання переходу $t \xrightarrow{\alpha} t_j$ неможливо виконати перехід $s \xRightarrow{a} s_i$, то $s_x \not\approx t_y$. Ця ситуація викликає закінчення алгоритму, оскільки відсутність слабкої взаємної подібності хоча б для однієї пари станів $(s_x, t_y) \in R$ веде до відсутності слабкої взаємної подібності між системою та її моделлю.

Для пари станів (s_x, t_y) існує взаємна подібність у випадку неможливості виконання переходу $s \xrightarrow{\alpha} s_i$ при $c = 0$ та переходу $t \xrightarrow{\alpha} t_j$ при $c = 1$ одночасно. Пара станів $s_x \approx t_y$ також у випадку, коли кожному елементу послідовності $(s_i, \xrightarrow{\alpha}_i)$ відповідає перехід $t \xRightarrow{a} t_j$ або $t \xRightarrow{\varphi} t$ при $c = 0$ і навпаки, коли кожному елементу послідовності $(t_j, \xrightarrow{\alpha}_j)$ відповідає перехід $s \xRightarrow{a} s_i$ або $s \xRightarrow{\varphi} s$ при $c = 1$.

Алгоритм закінчується також і тоді, коли всі пари $(s_x, t_y) \in R$ визначені як слабо взаємно подібні. В цьому випадку результатом роботи алгоритму є підтвердження слабкої взаємної подібності системи та моделі.

4.8. Зв'язок алгебри процесів та APRO-мереж

APRO-мережі орієнтовані на моделювання процесів, що відбуваються в складній системі при взаємодії компонентів. Механізм маркування та представлення системи у вигляді дводольного графа дає можливість опису довільних алгоритмів взаємодії та обробки інформації за допомогою множини про-

цедур активації переходів $\rho = \sum_{j=1}^m \rho_j$ та множини процедур

обслуговування переходів $\pi = \sum_{j=1}^m \pi_j$, де m — загальна кіль-

кість переходів. Оскільки формальний опис APRO-мережі не задає правил формування згаданих алгоритмів, то для їх реалізації допустиме використання довільних середовищ програмування. Такий підхід є свідченням універсальності APRO-мереж, проте не знімає питання оптимального вибору мови програмування для кожного конкретного варіанту застосування. Основною метою імітаційного моделювання, як правило, є визначення показників продуктивності за умови внесення мінімальних похибок від інструментів вимірювання. Такий підхід викликає потребу побудови точної імітаційної моделі, яка б могла бути використана як для аналізу, так і для виконання реального робочого навантаження з метою експериментального визначення

продуктивності. Звідси випливає необхідність застосування такого формального опису алгоритмів функціонування імітаційної моделі, який, з одного боку, базувався б на формалізмах високого рівня, а з іншого — був описаний чітко і однозначно за допомогою формальної семантики. Існуючі мови програмування, у більшості випадків, використовують складні об'єкти типу черг з пріоритетами та списками подій, що не завжди дозволяє будувати моделі з достатнім рівнем адекватності. Огляд літератури [10-12, 49, 60] показує, що сучасним підходом до розв'язання згаданих проблем є створення версії алгебри процесів, властивості якої змогли б максимально ефективно сприяти реалізації цільової функції моделювання.

РОЗДІЛ 5

Формальний опис програмної системи моделювання

Дослідження реальних фізичних процесів, динаміка яких представлена крайовими задачами математичної фізики, є складною математичною проблемою, яка у кожному конкретному випадку потребує детального розгляду. Як правило, для розв'язування таких задач необхідні значні обчислювальні ресурси. Тому важливою рисою розвитку чисельних методів є орієнтація на паралельні обчислення. Для ефективної реалізації обчислень на кластерних системах використовують обчислювальне середовище, побудоване на основі програмної системи моделювання, що забезпечує адаптацію паралельного методу до структури обчислювального засобу. Оскільки важливою рисою такої адаптації є властивість навчання, то доцільно використати кліткові мережі в якості базової концепції організації паралельних обчислень.

Кліткові нейронні мережі (КНМ), публікації про які вперше з'явилися в [68, 69], відразу набули широкого визнання як парадигма штучних нейронних мереж, орієнтованих на паралельні асинхронні обчислювальні процеси. За останні роки створено велику кількість алгоритмів обробки зображень, розпізнавання образів, оцінки динаміки механічних систем та ін., що успішно реалізовані за допомогою кліткових нейронних мереж. Загальною спільною рисою згаданих алгоритмів можна вважати наявність властивості декомпозиції глобальної задачі на множину локальних задач, кожна з яких може виконуватись окремим клітковим нейроном КНМ. Властивість декомпозиції характерна також для методів розв'язування крайових задач, що зумовило застосування КНМ для їх реалізації. Постановка задачі, в даному випадку, зводиться до розбивки області розв'язування на окремі зони, кожна з яких відповідає одному нейрону кліткової нейронної мережі. Локальні зв'язки між нейронами КНМ задають структурування зон, а характер

обчислень визначає динаміку взаємодії. Розвиток КНМ для розв'язування крайових задач характеризується двома основними технологічними напрямками. Перший з них полягає у створенні спеціалізованих пристроїв на базі VLSI-чипів [70], а другий передбачає створення обчислювальних середовищ на базі імітаційних моделей КНМ для реалізації на традиційних комп'ютерах, паралельних системах [71]. Крім згаданої властивості навчання, використання імітаційної моделі КНМ обумовлено рядом переваг, основні з яких полягають у можливості застосування складних багатовимірних структур КНМ і досягнення високої точності результатів.

Кліткова нейронна мережа моделює функціонування однорідної обчислювальної структури з локальними зв'язками, у вузлах якої розташовано процесорні елементи (кліткові нейрони). Структура зв'язків між клітковими нейронами може змінюватись залежно від розв'язуваних на КНМ завдань. При моделюванні складних фізичних процесів, яке зводиться до розв'язування крайових задач математичної фізики, застосовують чисельні методи, які потребують забезпечення прямих зв'язків тільки з певною локальною підмножиною вершин [72]. Тому для таких обчислень більш ефективними є топології, що базуються на регулярних графах [73].

Сучасні підходи до побудови однорідних обчислювальних середовищ мають тенденцію до збільшення показників їх вимірності. У зв'язку зі стрімким розвитком всього спектру технологій, починаючи від VLSI-технологій на мікрорівні до кластерних архітектур на макрорівні, актуальною є проблема побудови та аналізу топологій тривимірних циркулянтних графів.

Застосувавши властивості двовимірних циркулянтних графів до тривимірних гомогенних структур, одержимо графи, які характеризуються такими параметрами:

– всі вершини пронумеровані за допомогою послідовності цілих чисел $0, \dots, N - 1$;

– з кожної вершини $0 \leq i \leq N - 1$ виходять шість ребер, які пов'язують цю вершину з сусідніми вершинами:

$$\left(\left\lfloor \frac{i}{\sqrt[3]{N^2}} \right\rfloor \sqrt[3]{N^2} \pm (i \pm s_1) \bmod (\sqrt[3]{N^2}) \right), \left(\left\lfloor \frac{i}{\sqrt[3]{N^2}} \right\rfloor \sqrt[3]{N^2} \pm (i \pm s_2) \bmod (\sqrt[3]{N^2}) \right), (i \pm s_3) \bmod N \Big\};$$

– співвідношення між елементами множини

$$S = \{ s_1, s_2, s_3 \};$$

$$0 < s_1 < s_2 < s_3, s_1 = \frac{\sqrt[3]{N}}{2} - 1, s_2 = \sqrt[3]{N}, s_3 = \sqrt[3]{N^2}. \quad (5.1)$$

Приклад даного тривимірного графа для $N = 512$ показано на рис.5.1.

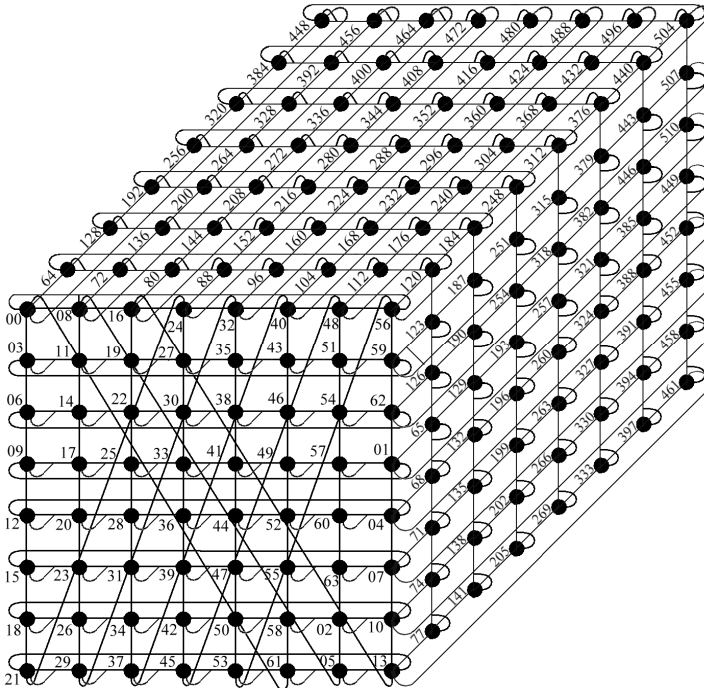


Рис.5.1. Тривимірний пошаровий циркулянтний граф (топология 3D-Circulant)

Функція досяжності $f_N : Z^3 \rightarrow V$ задає номер вершини, у яку можна потрапити з вершини 0, виконуючи послідовно переходи x, y та z , $(x, y, z) \in Z^3$, вздовж ребер графа $G_N(N; s_1, s_2, s_3)$. Розглянемо два можливих варіанти функції, що відповідають модифікаціям тривимірних циркулянтних графів. Перший варіант, який базується на функції досяжності

$$f_N(x, y, z) = (xs_1 + ys_2) \bmod \sqrt[3]{N^2} + zs_3 \bmod N,$$

будемо називати пошаровим циркулянтним графом, оскільки у даному випадку номери вершин розподіляються шарами.

Другий варіант називатимемо однорідним циркулянтним графом. Його функція досяжності включає рівноправні кроки по трьох координатах:

$$f_N(x, y, z) = (xs_1 + ys_2 + zs_3) \bmod N,$$

що забезпечує властивість однорідності. Тому в такому графі неможливо визначити шари, а множина вершин має однорідний характер:

$$D = \{ j \mid j = 0, \dots, N - 1 \}.$$

Всі вершини, які можуть бути досягнуті шляхом послідовного виконання k кроків вздовж ребер x, y і z від вершини 0, утворюють область:

$$D(k) = \{ (x, y) \mid |x| + |y| + |z| \leq k \}.$$

У кожній вершині даного тривимірного циркулянтного графа знаходиться спеціалізована обчислювальна структура — узагальнений нейрон. Згадана структура відрізняється від типової структури формального нейрона, оскільки головна ідея узагальнення формального нейрона полягає у тому, щоб єдину змінну, яка виражає його поточний стан, замінити деякою множиною внутрішніх змінних $S = \{ s_0, s_1, s_2, \dots, s_k \}$ (рис. 5.2).

Множина S є недоступною для інших нейронів, що пов'язані з даним нейроном.

Для обміну інформацією біологічні нейрони використовують сигнали, які можуть бути представлені множиною характеристичних функцій

$$O = \left\{ o_i : R^{|S|} \rightarrow R, i = 1, \dots, l \right\},$$

де l — кількість виходів узагальненого нейрона.

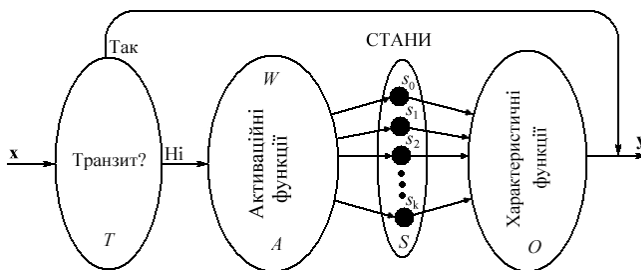


Рис.5.2. Узагальнення формального нейрона

Динаміка зміни внутрішніх змінних визначається активаційними функціями, які утворюють множину

$$A = \left\{ a_j : R^{|x|} \rightarrow R, j = 1, \dots, |S| \right\},$$

де $x = \{ x_0, x_1, \dots, x_n \}$ — множина вхідних сигналів узагальненого нейрона, які є значеннями характеристичних функцій сусідніх нейронів.

Кожному елементу множини $x = \{ x_0, x_1, \dots, x_n \}$ ставиться у відповідність ваговий коефіцієнт w . Всі вагові коефіцієнти узагальненого нейрона об'єднані в множину W .

Спираючись на наведені відомості про структуру вузлового нейрона та характер зв'язків між нейронами, наведемо їх формальний опис з використанням запропонованих вище засобів формалізації.

5.1. АPRO-мережний опис програмної системи моделювання

5.1.1. Структура моделі узагальненого нейрона та принципи його функціонування

Сучасні дослідження в галузі нейробіології [74] показали, що нейрон є складною системою, роботу якої неможливо описати однією змінною. Узагальнений нейрон має дві основні функції: обчислювальну та транзитну. Обчислювальна функція полягає в обробці вхідної інформації, а транзитна функція забезпечує передачу вхідних сигналів на вихід нейрона в незмінному вигляді. Модель такого нейрона розглядається як паралельна система з асинхронною взаємодією компонентів. Тому використаємо для її опису формальний апарат АPRO-мереж. Узагальнений нейрон опишемо у вигляді операторного переходу:

$$Ne_v = (P_v, T_v, E_v, X_v, F_v), \quad (5.2)$$

де $P_v = \{(p_v)_i\}_{i=1}^z$ — скінченна множина позицій узагальненого нейрона; $T_v = \{(\tau_v)_j\}_{j=1}^m$ — скінченна множина переходів узагальненого нейрона;

$E_v = \{(e_v)_c\}_{c=1}^l$ — скінченна множина входів узагальненого нейрона; $X_v = \{(x_v)_d\}_{d=1}^n$ — скінченна множина виходів узагальненого нейрона; $F_v = (P_v \times T_v) \cup (T_v \times P_v) \cup (E_v \times T_v) \cup (T_v \times X_v)$ — множина ребер.

Множина ребер F_v задає мережну структуру моделі узагальненого нейрона, що показано на рис.5.3.

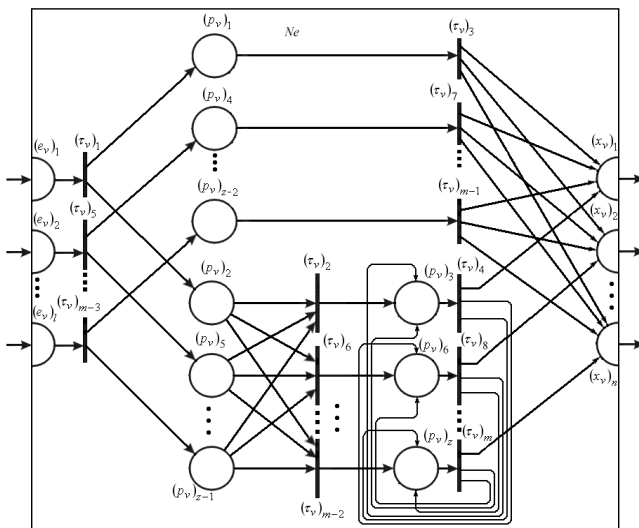


Рис.5.3. Мережна структура моделі узагальненого нейрона

Представимо множину F_v у вигляді матриці інцидентності H_v :

$$H_v = \begin{pmatrix} & (e_v)_1 \cdots (e_v)_l & (p_v)_1 \cdots (p_v)_z & (x_v)_1 \cdots (x_v)_n \\ (\tau_v)_1 & -1 \cdots 0 & 1 \cdots 0 & 0 \cdots 0 \\ (\tau_v)_2 & 0 \cdots 0 & 0 \cdots 0 & 0 \cdots 0 \\ (\tau_v)_3 & 0 \cdots 0 & -1 \cdots 0 & 1 \cdots 1 \\ (\tau_v)_4 & 0 \cdots 0 & 0 \cdots -1 & 1 \cdots 0 \\ \dots & \dots & \dots & \dots \\ (\tau_v)_{m-3} & 0 \cdots -1 & 0 \cdots 0 & 0 \cdots 0 \\ (\tau_v)_{m-2} & 0 \cdots 0 & 0 \cdots 1 & 0 \cdots 0 \\ (\tau_v)_{m-1} & 0 \cdots 0 & 0 \cdots 0 & 1 \cdots 1 \\ (\tau_v)_m & -1 \cdots 0 & 0 \cdots -1 & 0 \cdots 1 \end{pmatrix} .$$

Переходи $\{ (\tau_v)_2, (\tau_v)_6, \dots, (\tau_v)_{m-2} \}$ призначені для реалізації множини функцій активації узагальненого нейрона, множина внутрішніх станів представлена множиною позицій

$$\{ (p_v)_3, (p_v)_6, \dots, (p_v)_z \} ,$$

а множина $\{(\tau_v)_4, (\tau_v)_8, \dots, (\tau_v)_m\}$ формує характеристичні функції.

Функції транзитної передачі міток реалізовані за допомогою позицій $\{(p_v)_1, (p_v)_4, \dots, (p_v)_{z-2}\}$ та переходів $\{(\tau_v)_3, (\tau_v)_7, \dots, (\tau_v)_{m-1}\}$. Решта елементів мережної структури узагальненого нейрона виконують службові функції.

Відповідно до структури операторного переходу APRO-мережі вхідні сигнали узагальненого нейрона надходять на

входи $E_v = \{(e_v)_1, (e_v)_2, \dots, (e_v)_n\}$, які виконують роль вхідних буферів, місткості яких визначаються множиною

$\{(\text{Max} _ e_v)_1, (\text{Max} _ e_v)_2, \dots, (\text{Max} _ e_v)_n\}$. Функціонально

призначення вхідних буферів узагальненого нейрона полягає у забезпеченні асинхронної взаємодії з навколишнім середовищем. Мітки, які надходять з навколишнього середовища у вхідні буфери, є джерелом активації множини

переходів $\{(\tau_v)_1, (\tau_v)_5, \dots, (\tau_v)_{m-3}\}$, у яких відбувається

аналіз ідентифікаторів вибраних міток. Якщо ідентифікатор мітки співпадає з номером даного нейрона, то така мітка призначена для обробки у цьому нейроні і називається своєю міткою. Якщо ідентифікатор мітки не співпадає з номером нейрона, то таку мітку називають транзитною. Свої мітки

розміщують на позиціях $\{(p_v)_2, (p_v)_5, \dots, (p_v)_{z-1}\}$, а

транзитні мітки — на позиціях $\{(p_v)_1, (p_v)_4, \dots, (p_v)_{z-2}\}$.

Атрибути міток, які розміщені на позиціях

$\{(p_v)_2, (p_v)_5, \dots, (p_v)_{z-1}\}$, виступають у ролі значень

відповідних аргументів множини активаційних функцій A

узагальненого нейрона (рис.5.2). Виконання цих функцій відбувається в множині переходів $\{(\tau_v)_2, (\tau_v)_2, \dots, (\tau_v)_{m-2}\}$. Спрацювання даної множини переходів формує внутрішній стан нейрона, представленого атрибутами міток на позиціях $\{(p_v)_3, (p_v)_6, \dots, (p_v)_z\}$. Внутрішні стани безпосередньо не доступні іншим нейронам, які знаходяться в зовнішньому середовищі. Для формування вихідної інформації використовують множину характеристичних функцій O узагальненого нейрона (рис.5.2). Виконання характеристичних функцій реалізують переходи $\{(\tau_v)_4, (\tau_v)_8, \dots, (\tau_v)_m\}$. Результати їх спрацювання розміщуються на вихідних буферах $\{(x_v)_1, (x_v)_2, \dots, (x_v)_l\}$. Транзитні мітки підлягають обробці переходами $\{(\tau_v)_3, (\tau_v)_7, \dots, (\tau_v)_{m-1}\}$. Робота цих переходів забезпечує вибір подальшого маршруту передачі транзитних міток.

Отже, виходячи з даного опису, можна зробити висновок, що функціонування АPRO-мережі, яка представляє узагальнений нейрон, має кроковий характер з послідовностями кроків $\sigma_1 = \langle U_1, U_2, U_3 \rangle$ і $\sigma_2 = \langle U_1, U'_2 \rangle$,

$$\text{де } U_1 = \{(\tau_v)_1, (\tau_v)_5, \dots, (\tau_v)_{m-3}\},$$

$$U_2 = \{(\tau_v)_2, (\tau_v)_6, \dots, (\tau_v)_{m-2}\},$$

$$U'_2 = \{(\tau_v)_3, (\tau_v)_7, \dots, (\tau_v)_{m-1}\},$$

$$U_3 = \{(\tau_v)_4, (\tau_v)_8, \dots, (\tau_v)_m\}.$$

Тоді еволюція маркування під дією послідовностей кроків σ_1 і σ_2 відбувається таким чином:

$$M|\sigma_1\rangle M' = M|U_1\rangle M_1|U_2\rangle M_2|U_3\rangle M',$$

$$M|\sigma_2\rangle M' = M|U_1\rangle M'_1|U'_2\rangle M',$$

де M — початкове маркування; M' — кінцеве маркування;
 M_1, M'_1, M_2 — відповідні кроково досяжні маркування.

Поточне маркування M_v узагальненого нейрона Ne_v визначається множиною: $M_v = M_e \cup (M_p \cap M'_p) \cup M_x$,
де $M_e = \left\{ \left((e_v)_1, \{\mu_k\}_{k=1}^{(\text{Cur}-e_v)_1} \right), \dots, \left((e_v)_l, \{\mu_k\}_{k=1}^{(\text{Cur}-e_v)_l} \right) \right\}$,
 $M_p = \left\{ \left((p_v)_2, \{\mu_k\}_{k=1}^{(\text{Cur}-p_v)_2} \right), \left((p_v)_3, \{\mu_k\}_{k=1}^{(\text{Cur}-p_v)_3} \right), \left((p_v)_5, \{\mu_k\}_{k=1}^{(\text{Cur}-p_v)_5} \right), \right.$
 $\left. \left((p_v)_6, \{\mu_k\}_{k=1}^{(\text{Cur}-p_v)_6} \right), \dots, \left((p_v)_{z-1}, \{\mu_k\}_{k=1}^{(\text{Cur}-p_v)_{z-1}} \right), \left((p_v)_z, \{\mu_k\}_{k=1}^{(\text{Cur}-p_v)_z} \right) \right\}$,
 $M'_p = \left\{ \left((p_v)_1, \{\mu_k\}_{k=1}^{(\text{Cur}-p_v)_1} \right), \left((p_v)_4, \{\mu_k\}_{k=1}^{(\text{Cur}-p_v)_4} \right), \right.$
 $\left. \dots, \left((p_v)_{z-2}, \{\mu_k\}_{k=1}^{(\text{Cur}-p_v)_{z-2}} \right) \right\}$
 $M_x = \left\{ \left((x_v)_1, \{\mu_k\}_{k=1}^{(\text{Cur}-x_v)_1} \right), \dots, \left((x_v)_n, \{\mu_k\}_{k=1}^{(\text{Cur}-x_v)_n} \right) \right\}$.

Множина M_v включає маркування входів, маркування внутрішніх позицій та маркування виходів.

Початкове маркування: $M = M_e$ при

$$\left\{ M\left(\left(p_v\right)_i\right) = 0 \mid i = 1, \dots, z \right\}, \left\{ M\left(\left(x_v\right)_i\right) = 0 \mid i = 1, \dots, n \right\}.$$

Маркування першого кроку: $M = M_1$ при

$$\left\{ M\left(\left(e_v\right)_i\right) = 0 \mid i = 1, \dots, l \right\}, \left\{ M\left(\left(x_v\right)_i\right) = 0 \mid i = 1, \dots, n \right\},$$

$$\left\{ M\left(\left(p_v\right)_i\right) > 0 \mid i \in \{2, 5, \dots, z-1\} \right\},$$

$$\left\{ M\left(\left(p_v\right)_i\right) = 0 \mid i \in \{3, 6, \dots, z\} \right\}.$$

$$M = M'_1 \text{ при}$$

$$\{M((e_v)_i) = 0 \mid i = 1, \dots, l\}, \{M((x_v)_i) = 0 \mid i = 1, \dots, n\},$$

$$\{M((p_v)_i) > 0 \mid i \in \{1, 4, \dots, z - 2\}\}.$$

Маркування другого кроку:

$$M_2 = M_p \text{ при}$$

$$\{M((e_v)_i) = 0 \mid i = 1, \dots, l\}, \{M((x_v)_i) = 0 \mid i = 1, \dots, n\},$$

$$\{M((p_v)_i) = 0 \mid i \in \{2, 5, \dots, z - 1\}\},$$

$$\{M((p_v)_i) > 0 \mid i \in \{3, 6, \dots, z\}\}.$$

Кінцеве маркування:

$$M' = M_x \text{ при}$$

$$\{M((p_v)_i) = 0 \mid i = 1, \dots, z\}, \{M((e_v)_i) = 0 \mid i = 1, \dots, l\},$$

$$\{M((x_v)_i) > 0 \mid i = 1, \dots, n\}.$$

Еволюція маркування є результатом послідовних спрацювань переходів АPRO-мережі. Такі спрацювання деякого переходу $(\tau_v)_j$ відбуваються завдяки виконанню відповідних процедур $(\rho_v)_j, (\pi_v)_j, (\gamma_v)_j$ і $(\omega_v)_j$. Алгоритми їх роботи, що описані в розділі 3, задають роботу переходу у загальному вигляді. Для того, щоб мережа набула конкретних функціональних властивостей, необхідно додатково визначити процедури процесів pr_1, \dots, pr_k в процедурі

$$(\pi_v)_j = (\text{Analyze}, pr_1, \dots, pr_k), \text{ де } 1 \leq k \leq (\text{Max} _ \tau_v)_j.$$

5.1.2. Принципи функціонування програмної системи моделювання на основі тривимірної кліткової нейронної мережі

Для розв'язування тривимірних крайових задач математичної фізики будемо використовувати тривимірні кліткові мережі, структура яких ізоморфна тривимірним кільцевим циркулянтним графам. Найпростіша з таких структур містить 64 нейрони і представлена графом $G(64;1,4,16)$, у якому кроки по координатах (x, y, z) співпадають з кроками вздовж кільця кільцевого циркулянтного графа. Таким чином, кроки $(x \pm 1) \bmod 16$ відповідають крокам вздовж кільця найменшого діаметру, кільце середнього діаметру утворює сукупність вершин, які можуть бути досягнуті шляхом виконання кроків $(y \pm 4) \bmod 16$, а кроки $(z \pm 16) \bmod 64$ співпадають з кроками вздовж кільця з найбільшим діаметром. Тривимірна структура, показана на рис.5.4, є найпростішою з ряду можливих структур, оскільки вона характеризується довжиною ребра $n = 2^2$. В той же час, така структура дає уявлення про характер міжнейронних зв'язків та взаємне просторове розміщення нейронів.

Існують теореми [75], що доводять можливість існування подібних структур довільного виміру за умови, що кількість нейронів, розміщених вздовж одного ребра, дорівнює числу, еквівалентному 2^n , де $n = 2, 3, 4, \dots$. Тому такий підхід дає можливість вибору оптимальної структури для забезпечення точності моделі складних фізичних процесів з одного боку та наявних обчислювальних ресурсів з іншого боку.

Вузли структури, показані на рис.5.4, представлені фрагментами ієрархічної АPRO-мережі, які включають

переходи Ne_i , $i = 0, 1, 2, \dots$, та множини позицій

$$(d_i, u_i, r_i, l_i, a_i, b_i).$$

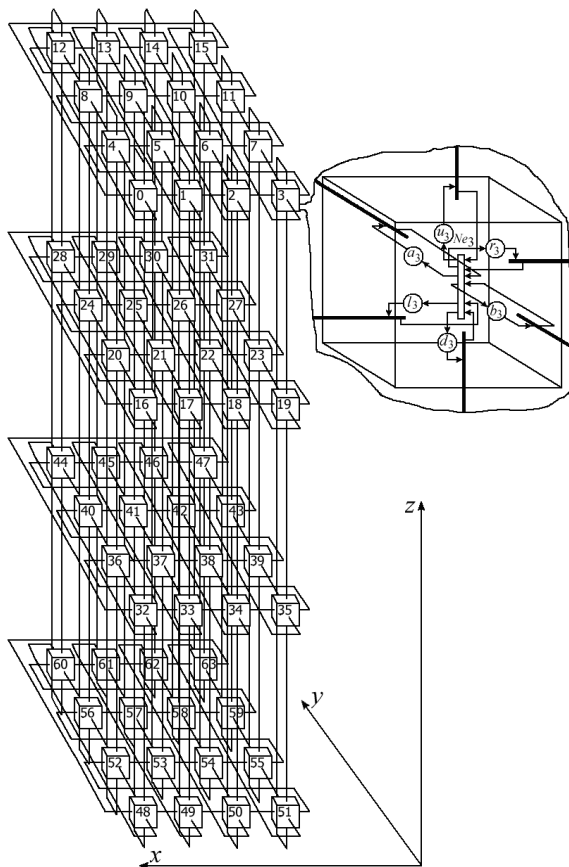


Рис.5.4. APRO-мережна модель тривимірної кліткової нейронної мережі

Структура APRO-мережі, яка включає згадані елементи, утворює ієрархічний рівень 0 формального опису моделі кліткової нейронної мережі. Рівень 1 моделі розкриває APRO-мережну структуру кліткових нейронів, представлених операторними переходами на рівні 0.

Представимо АPRO-мережу (рис.5.4) у вигляді кортежу:

$$\Phi = (P, T, F, M, V),$$

де $P = \left\{ (u_i, r_i, d_i, a_i, b_i) \mid i = \overline{0, 63} \right\}$ — множина позицій,

$T = \left\{ t_i \mid i = \overline{0, 63} \right\}$ — множина переходів,

$$F = \left\{ \left(u_i, t_{(i-16) \bmod 64} \right), \left(d_i, t_{(i+16) \bmod 64} \right), \left(r_i, t_{(i+1) \bmod 16} \right), \right. \\ \left. \left(l_i, t_{(i-1) \bmod 16} \right), \left(a_i, t_{(i+4) \bmod 16} \right), \left(b_i, t_{(i-4) \bmod 16} \right) \right\}_{i=0}^{63} \cup \\ \left\{ \left(t_i, d_{(i-16) \bmod 64} \right), \left(t_i, u_{(i+16) \bmod 64} \right), \left(t_i, l_{(i+1) \bmod 16} \right), \right. \\ \left. \left(t_i, r_{(i-1) \bmod 16} \right), \left(t_i, r_{(i-1) \bmod 16} \right), \left(t_i, a_{(i-4) \bmod 16} \right) \right\}_{i=0}^{63}$$

— множина ребер між переходами та позиціями,

$$M = \left\{ \left(u_i, \left\{ \mu_j^U \right\}_{j=1}^{\text{Max} - u_i} \right), \left(d_i, \left\{ \mu_j^D \right\}_{j=1}^{\text{Max} - d_i} \right), \left(r_i, \left\{ \mu_j^R \right\}_{j=1}^{\text{Max} - r_i} \right), \right. \\ \left. \left(l_i, \left\{ \mu_j^L \right\}_{j=1}^{\text{Max} - l_i} \right), \left(a_i, \left\{ \mu_j^A \right\}_{j=1}^{\text{Max} - a_i} \right), \left(b_i, \left\{ \mu_j^B \right\}_{j=1}^{\text{Max} - b_i} \right) \right\}_{i=0}^{63}$$

— скінченна множина маркувань,

$V = (\Delta, \Psi, \Lambda)$ — множина глобальних змінних.

$\Delta = \{ \delta_1, \delta_2, \delta_3 \}$ — показники продуктивності.

δ_1 — пропускна здатність; δ_2 — швидкість виконання операцій;

δ_3 — швидкість обробки даних.

$\Psi = \{ \psi_1, \psi_2 \}$ — показники реактивності.

ψ_1 — час проходження; ψ_2 — час реакції.

$\Lambda = \{ \lambda_1, \lambda_2 \}$ — показники використання.

λ_1 — коефіцієнт використання апаратних ресурсів; λ_2 — коефіцієнт використання програмних ресурсів.

Довільний нейрон Ne_v , заданий на рівні 0 переходом t_v , де $v = 0, \dots, 63$, представимо APRO-мережею рівня 1:

$$t_v = (P_v, T_v, E_v, X_v, F_v),$$

де $P_v = \{ (p_v)_1, \dots, (p_v)_{28} \}$ — множина позицій;

$T_v = \{ (\tau_v)_1, \dots, (\tau_v)_{17} \}$ — множина переходів нейрона;

$E_v = \{ (e_v)_1, \dots, (e_v)_6 \}$ — множина входів;

$X_v = \{ (x_v)_1, \dots, (x_v)_6 \}$ — множина виходів;

$F_v = (P_v \times T_v) \cup (T_v \times P_v) \cup (E_v \times T_v) \cup (T_v \times X_v)$ — множина ребер.

Структура APRO-мережної моделі тривимірної кліткової мережі показана на рис.5.5:

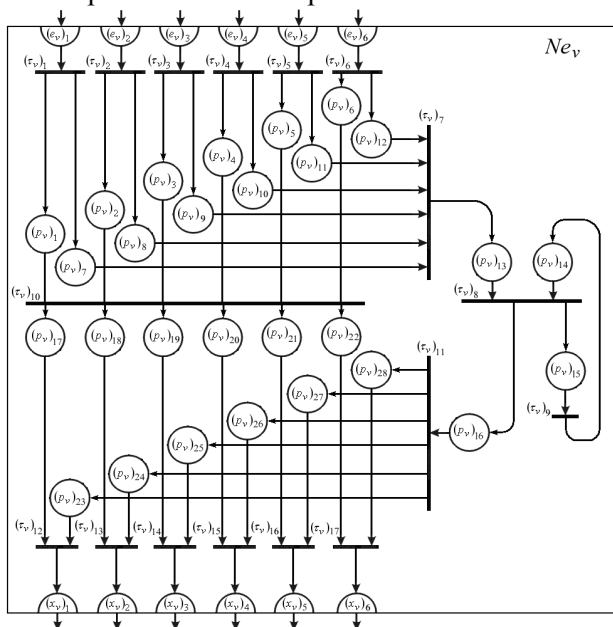


Рис.5.5. Мережна структура моделі нейрона тривимірної кліткової мережі

	Переходи нейрона $Ne_v = \{(\tau_v)_1, \dots, (\tau_v)_{17}\}$																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$(p_v)_{10}$	0	0	0	1	0	0	-1	0	0	0	0	0	0	0	0	0	0
$(p_v)_{11}$	0	0	0	0	1	0	-1	0	0	0	0	0	0	0	0	0	0
$(p_v)_{12}$	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0
$(p_v)_{13}$	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0
$(p_v)_{14}$	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0
$(p_v)_{15}$	0	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0
$(p_v)_{16}$	0	0	0	0	0	0	0	1	0	0	-1	0	0	0	0	0	0
$(p_v)_{17}$	0	0	0	0	0	0	0	0	0	1	0	-1	0	0	0	0	0
$(p_v)_{18}$	0	0	0	0	0	0	0	0	0	1	0	0	-1	0	0	0	0
$(p_v)_{19}$	0	0	0	0	0	0	0	0	0	1	0	0	0	-1	0	0	0
$(p_v)_{20}$	0	0	0	0	0	0	0	0	0	1	0	0	0	0	-1	0	0
$(p_v)_{21}$	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	-1	0
$(p_v)_{22}$	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	-1
$(p_v)_{23}$	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	0
$(p_v)_{24}$	0	0	0	0	0	0	0	0	0	0	1	0	-1	0	0	0	0
$(p_v)_{25}$	0	0	0	0	0	0	0	0	0	0	1	0	0	-1	0	0	0
$(p_v)_{26}$	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-1	0	0
$(p_v)_{27}$	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	-1	0
$(p_v)_{28}$	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	-1

	Переходи нейрона $Ne_v = \{(\tau_v)_1, \dots, (\tau_v)_{17}\}$																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$(x_v)_1$	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
$(x_v)_2$	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
$(x_v)_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
$(x_v)_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
$(x_v)_5$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
$(x_v)_6$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Згідно з описом операторного переходу, представленим у попередніх параграфах, позиція $(p_v)_i$ визначається як

$$(p_v)_i = \{ (d_v)_i, (q_v)_i \},$$

де

$(d_v)_i = \{ (\text{Id}_{-p_v})_i, (\varphi_{-p_v})_i, (\text{Cur}_{-p_v})_i, (\text{Max}_{-p_v})_i, (\delta_{-p_v})_i \}$ — параметри позиції; $(q_v)_i$ — множина міток, розміщених на даній позиції. Параметри внутрішніх позицій тривимірного нейрона на рівні 1 зведемо в таблицю 5.2.

Таблиця 5.2

Таблиця позицій моделі нейрона Ne_v
тривимірної кліткової мережі

	Множина позицій P_v					
	Параметри позицій d_v					Мітки q_v
	Id_{-p_v}	φ_{-p_v}	Cur_{-p_v}	Max_{-p_v}	δ_{-p_v}	
$(p_v)_1$	Intran_U	μ_{tran}	$0 \div 6$	6	$\min_{0 < i \leq q_v } (\delta_{-p_v})_i$	$\{\mu_i\}_{i=1}^{\text{Cur}_{-p_v}}$

		Множина позицій P_v					Мітки q_v
		Параметри позицій d_v					
		Id_{-p_v}	φ_{-p_v}	Cur_{-p_v}	Max_{-p_v}	δ_{-p_v}	
$(p_v)_2$	Intran_R						
$(p_v)_3$	Intran_D						
$(p_v)_4$	Intran_L						
$(p_v)_5$	Intran_A						
$(p_v)_6$	Intran_B						
$(p_v)_7$	Incalc_U	μ_{-calc}	$0 \div 1$	1	$\delta_{-\mu_1}$	μ_1	
$(p_v)_8$	Incalc_R						
$(p_v)_9$	Incalc_D						
$(p_v)_{10}$	Incalc_L						
$(p_v)_{11}$	Incalc_A						
$(p_v)_{12}$	Incalc_B						
$(p_v)_{13}$	In_proc						
$(p_v)_{14}$	State_proc		1				
$(p_v)_{15}$	Mem_proc						
$(p_v)_{16}$	Out_proc	μ_{-tran}	$0 \div 12$	12	$\min_{0 < i \leq d_v } (\delta_{-\mu_i})$	$\{\mu_i\}_{i=1}^{Cur_{-p_v}}$	
$(p_v)_{17}$	Outtran_U						
$(p_v)_{18}$	Outtran_R						
$(p_v)_{19}$	Outtran_D						

	Множина позицій P_v					
	Параметри позицій d_v					Мітки
	Id_{-p_v}	φ_{-p_v}	Cur_{-p_v}	Max_{-p_v}	δ_{-p_v}	q_v
$(p_v)_{20}$	Outtran_L					
$(p_v)_{21}$	Outtran_A					
$(p_v)_{22}$	Outtran_B					
$(p_v)_{23}$	Outcalc_U	μ_{-tran}	$0 \div 6$	6		
$(p_v)_{24}$	Outcalc_R					
$(p_v)_{25}$	Outcalc_D					
$(p_v)_{26}$	Outcalc_L					
$(p_v)_{27}$	Outcalc_A					
$(p_v)_{28}$	Outcalc_B					

Колонка Id_{-p_v} містить назви позицій, що відображають їх функціональне призначення. Наприклад, назва «Intran_U» позначає позицію, яка призначена для розміщення транзитних міток, що надходять по каналу зв'язку даного нейрона з сусіднім верхнім нейроном. «Incalc_R» — позиція для розміщення міток, які надходять по каналу зв'язку від сусіднього правого нейрона та призначені для обробки даним переходом. «Outcalc_D» — позиція для розміщення вихідних міток, які містять інформацію, що є результатом обчислень у даному переході та призначені для передачі по каналу зв'язку у напрямку сусіднього нижнього нейрона. Позиція «Outtran_L» містить транзитні мітки, які даний нейрон має передати нейрону, що знаходиться зліва від нього у клітковій нейронній мережі.

Типи міток складають множину:

$$\varphi_{p_v} = \{ \mu_{\text{tran}}, \mu_{\text{calc}}, \text{Data_proc} \},$$

де μ_{tran} – транзитна мітка, μ_{calc} — мітка, що призначена для обробки в даному переході, Data_proc — мітки для обробки даних в нейроні.

Параметр Cur_{p_v} визначає поточну кількість міток на позиції в межах, вказаних в таблиці конкретно для кожної позиції. Він може змінюватись у діапазоні від 0 до Max_{p_v} .

Винятками є позиції $(p_v)_{10}$ та $(p_v)_{11}$, що містять фіксовану кількість міток, які дорівнює кількості паралельних обчислювальних процесів. З метою збереження часової цілісності моделі нейрона кожна позиція містить параметр локального часу δ_{p_v} . Принцип його формування залежить від методики просування модельного часу і відповідає мінімальному часу мітки з тих, що розміщені на даній позиції. Множина міток q_v завжди утворює певну інформаційну структуру, що дозволяє оптимізувати доступ до них в ході процесу моделювання. Типовими інформаційними структурами можуть бути масиви міток, списки та бінарні дерева. Множини позицій

$$\{ (p_v)_5, (p_v)_6, (p_v)_{11}, (p_v)_{12}, (p_v)_{21}, (p_v)_{22}, (p_v)_{27}, (p_v)_{28} \}$$

забезпечують обслуговування інформаційних потоків, пов'язаних з третім виміром кліткової мережі.

Опис переходу $(\tau_v)_j = \{ (\chi_v)_j, (N_v)_j \}$ включає:

параметри переходу

$$(\chi_v)_j = \{ (\text{Id}_{\tau_v})_j, (\delta_{\tau_v})_j, (\Delta_{\tau_v})_j, (\text{Level}_{\tau_v})_j \}$$

та функціональне ядро

$$(N_v)_j = \left\{ (\rho_v)_j, (\pi_v)_j, (\gamma_v)_j, (\omega_v)_j, (EvList_v)_j, (O_v)_j \right\} .$$

Значення параметрів переходів нейрона зводимо в таблиці 5.3.

Таблиця 5.3

Таблиця переходів моделі нейрона Ne_v
тривимірної кліткової мережі

		Переходи T_v					
		Параметри переходів			Ядро N_v		
		$\chi_v,$ $Level_ \tau_v = 1$					
	$Id_ \tau_v$	$\delta_ \tau_v$	$\Delta_ \tau_v$	ρ_v	π_v	γ_v	ω_v
$(\tau_v)_1$	Inanalyze_U			$(\rho_v (\mu_{in}^U))_1$	$(\pi_v (\mu_{in}^U))_1$	$(\gamma_v (\mu_{tr}^U, \mu_{cl}^U))_1$	$(\omega_v)_1$
$(\tau_v)_2$	Inanalyze_R			$(\rho_v (\mu_{in}^R))_2$	$(\pi_v (\mu_{in}^R))_2$	$(\gamma_v (\mu_{tr}^R, \mu_{cl}^R))_2$	$(\omega_v)_2$
$(\tau_v)_3$	Inanalyze_D			$(\rho_v (\mu_{in}^D))_3$	$(\pi_v (\mu_{in}^D))_3$	$(\gamma_v (\mu_{tr}^D, \mu_{cl}^D))_3$	$(\omega_v)_3$
$(\tau_v)_4$	Inanalyze_L			$(\rho_v (\mu_{in}^L))_4$	$(\pi_v (\mu_{in}^L))_4$	$(\gamma_v (\mu_{tr}^L, \mu_{cl}^L))_4$	$(\omega_v)_4$
$(\tau_v)_5$	Inanalyze_A			$(\rho_v (\mu_{in}^A))_5$	$(\pi_v (\mu_{in}^A))_5$	$(\gamma_v (\mu_{tr}^A, \mu_{cl}^A))_5$	$(\omega_v)_5$
$(\tau_v)_6$	Inanalyze_B			$(\rho_v (\mu_{in}^B))_6$	$(\pi_v (\mu_{in}^B))_6$	$(\gamma_v (\mu_{tr}^B, \mu_{cl}^B))_6$	$(\omega_v)_6$
$(\tau_v)_7$	InCbuffer			$\rho_v \left(\begin{matrix} \mu_{cl}^U, \mu_{cl}^R \\ \mu_{cl}^D, \mu_{cl}^L \\ \mu_{cl}^A, \mu_{cl}^B \end{matrix} \right)_7$	$\pi_v \left(\begin{matrix} \mu_{cl}^U, \mu_{cl}^R \\ \mu_{cl}^D, \mu_{cl}^L \\ \mu_{cl}^A, \mu_{cl}^B \end{matrix} \right)_7$	$(\gamma_v (\mu_{cl}^P))_7$	$(\omega_v)_7$
$(\tau_v)_8$	Neuroproc			$(\rho_v (\mu_{cl}^P, \mu_{cl}^M))_8$	$(\pi_v (\mu_{cl}^P, \mu_{cl}^M))_8$	$(\gamma_v (\mu_{out}^P, \mu_{out}^M))_8$	$(\omega_v)_8$
$(\tau_v)_9$				$(\rho_v (\mu_{out}^M))_9$	$(\pi_v (\mu_{out}^M))_9$	$(\gamma_v (\mu_{cl}^M))_9$	$(\omega_v)_9$

Переходи T_v						
Параметри переходів		Ядро N_v				
χ_v , Level $_{\tau_v} = 1$						
Id_{τ_v}	δ_{τ_v}	Δ_{τ_v}	ρ_v	π_v	γ_v	ω_v
Neuromem						
$(\tau_v)_{10}$	Router	$\left(\rho_v \begin{pmatrix} \mu_{tr}^U, \mu_{tr}^R \\ \mu_{tr}^D, \mu_{tr}^L \\ \mu_{tr}^A, \mu_{tr}^B \end{pmatrix} \right)_{10}$	$\left(\pi_v \begin{pmatrix} \mu_{tr}^U, \mu_{tr}^R \\ \mu_{tr}^D, \mu_{tr}^L \\ \mu_{tr}^A, \mu_{tr}^B \end{pmatrix} \right)_{10}$	$\left(\gamma_v \begin{pmatrix} \mu_{tr}^U, \mu_{tr}^R \\ \mu_{tr}^D, \mu_{tr}^L \\ \mu_{tr}^A, \mu_{tr}^B \end{pmatrix} \right)_{10}$	$(\omega_v)_{10}$	
$(\tau_v)_{11}$	OutCbuffer	$(\rho_v(\mu_{out}^P))_{11}$	$(\pi_v(\mu_{out}^P))_{11}$	$\left(\gamma_v \begin{pmatrix} \mu_{cl}^U, \mu_{cl}^R \\ \mu_{cl}^D, \mu_{cl}^L \\ \mu_{cl}^A, \mu_{cl}^B \end{pmatrix} \right)_{11}$	$(\omega_v)_{11}$	
$(\tau_v)_{12}$	OutAnalyze_U	$(\rho_v(\mu_{tr}^U, \mu_{cl}^U))_{12}$	$(\pi_v(\mu_{tr}^U, \mu_{cl}^U))_{12}$	$(\gamma_v(\mu_{out}^U))_{12}$	$(\omega_v)_{12}$	
$(\tau_v)_{13}$	OutAnalyze_R	$(\rho_v(\mu_{tr}^R, \mu_{cl}^R))_{13}$	$(\pi_v(\mu_{tr}^R, \mu_{cl}^R))_{13}$	$(\gamma_v(\mu_{out}^R))_{13}$	$(\omega_v)_{13}$	
$(\tau_v)_{14}$	OutAnalyze_D	$(\rho_v(\mu_{tr}^D, \mu_{cl}^D))_{14}$	$(\pi_v(\mu_{tr}^D, \mu_{cl}^D))_{14}$	$(\gamma_v(\mu_{out}^D))_{14}$	$(\omega_v)_{14}$	
$(\tau_v)_{15}$	OutAnalyze_L	$(\rho_v(\mu_{tr}^L, \mu_{cl}^L))_{15}$	$(\pi_v(\mu_{tr}^L, \mu_{cl}^L))_{15}$	$(\gamma_v(\mu_{out}^L))_{15}$	$(\omega_v)_{15}$	
$(\tau_v)_{16}$	OutAnalyze_A	$(\rho_v(\mu_{tr}^A, \mu_{cl}^A))_{16}$	$(\pi_v(\mu_{tr}^A, \mu_{cl}^A))_{16}$	$(\gamma_v(\mu_{out}^A))_{16}$	$(\omega_v)_{16}$	
$(\tau_v)_{17}$	OutAnalyze_B	$(\rho_v(\mu_{tr}^B, \mu_{cl}^B))_{17}$	$(\pi_v(\mu_{tr}^B, \mu_{cl}^B))_{17}$	$(\gamma_v(\mu_{out}^B))_{17}$	$(\omega_v)_{17}$	

Як позиції, так і переходи також мають параметр назви Id_{τ_v} , який дозволяє визначити їх основне функціональне призначення. Переходи $\{(\tau_v)_i\}_{i=1}^6$ з відповідною множиною параметрів назви $\{‘Inanalyze_U’, ‘Inanalyze_R’, ‘Inanalyze_D’, ‘Inanalyze_L’, ‘Inanalyze_A’, ‘Inanalyze_B’\}$ забезпечують попередній аналіз типів міток, що надходять від 6 сусідніх нейронів тривимірної кліткової нейронної мережі. Основною метою цього аналізу є поділ міток на транзитні мітки та обчислювальні мітки, призначені для обробки у даному переході. Функціональне призначення переходів $\{(\tau_v)_7, (\tau_v)_8, (\tau_v)_9, (\tau_v)_{11}\}$ з множиною параметрів назви $\{‘InCbuffer’, ‘Neuroproc’, ‘OutCbuffer’, ‘Neuromem’\}$ полягає у виконанні операцій над структурами даних, які є атрибутами обчислювальних міток. За результатами цих операцій у переважній більшості випадків створюються вихідні мітки, формування яких виконують вихідні переходи $\{(\tau_v)_i\}_{i=12}^{17}$ з параметрами назви $\{‘OutAnalyze_U’, ‘OutAnalyze_R’, ‘OutAnalyze_D’, ‘OutAnalyze_L’, ‘OutAnalyze_A’, ‘OutAnalyze_B’\}$. Функція маршрутизації транзитних міток покладена на перехід $(\tau_v)_{10}$ з параметром $(Id_{\tau_v})_{10} = ‘Router’$.

Всі переходи нейрона Ne_v мають однакове значення параметра $Level_{\tau_v} = 1$, який визначає ієрархічний рівень APRO-мережі, що описує модель даного нейрона. Локальні лічильники часу $\{\delta_{\tau_v}\}_j$ та періоди простою $(\Delta_{\tau_v})_j$ є змінними параметрами і формуються процедурами Timing, які входять до складу процедур активації переходів $\{\rho_v\}_j$, де j — довільний номер переходу APRO-мережної моделі нейрона (рис. 5.3).

Основними елементами ядра є процедури активації переходу ρ_v , обслуговування переходу π_v , деактивації переходу γ_v та очікування ω_v . Параметрами цих процедур є мітки, що підлягають обробці в тому чи іншому переході. В таблиці 5.3 наведено назви процедур з відповідними мітками в якості параметрів. Мітки $\{\mu_{in}^U, \mu_{in}^R, \mu_{in}^D, \mu_{in}^L, \mu_{in}^A, \mu_{in}^B\}$ утворюють множину вхідних нерозпізнаних міток, які надійшли на відповідні переходи нейрона від сусідніх нейронів. Процедури $\{(\rho_v(\mu_{in}^U))_1, (\rho_v(\mu_{in}^R))_2, (\rho_v(\mu_{in}^D))_3, (\rho_v(\mu_{in}^L))_4, (\rho_v(\mu_{in}^A))_5, (\rho_v(\mu_{in}^B))_6\}$, $\{(\pi_v(\mu_{in}^U))_1, (\pi_v(\mu_{in}^R))_2, (\pi_v(\mu_{in}^D))_3, (\pi_v(\mu_{in}^L))_4, (\pi_v(\mu_{in}^A))_5, (\pi_v(\mu_{in}^D))_6\}$ виконують попередній аналіз та обробку цих міток. Результатом проведеного аналізу є поділ міток на транзитні $\{\mu_{tr}^U, \mu_{tr}^R, \mu_{tr}^D, \mu_{tr}^L, \mu_{tr}^A, \mu_{tr}^B\}$ та обчислювальні

$$\{\mu_{cl}^U, \mu_{cl}^R, \mu_{cl}^D, \mu_{cl}^L, \mu_{cl}^A, \mu_{cl}^B\}.$$

Подальші дії переходу, зумовлені результатами поділу міток, виконують процедури

$$\left\{ \left(\gamma_v \left(\mu_{tr}^U, \mu_{cl}^U \right) \right)_1, \left(\gamma_v \left(\mu_{tr}^R, \mu_{cl}^R \right) \right)_2, \left(\gamma_v \left(\mu_{tr}^D, \mu_{cl}^D \right) \right)_3, \right. \\ \left. \left(\gamma_v \left(\mu_{tr}^L, \mu_{cl}^L \right) \right)_4, \left(\gamma_v \left(\mu_{tr}^A, \mu_{cl}^A \right) \right)_5, \left(\gamma_v \left(\mu_{tr}^B, \mu_{cl}^B \right) \right)_6 \right\}.$$

Ці дії полягають у тому, що результатом деактивації переходів $\{(\tau_v)_i\}_{i=1}^6$ є передача транзитних міток

$$\{\mu_{tr}^U, \mu_{tr}^R, \mu_{tr}^D, \mu_{tr}^L, \mu_{tr}^A, \mu_{tr}^B\}$$

на позиції $\{(p_v)_i\}_{i=1}^6$ з метою їх маршрутизації або передача

міток $\{\mu_{cl}^U, \mu_{cl}^R, \mu_{cl}^D, \mu_{cl}^L, \mu_{cl}^A, \mu_{cl}^B\}$ на позиції $\{(p_v)_i\}_{i=7}^{12}$ з метою

обробки. Ядро переходу $(\tau_v)_7$ включає процедури :

$$\left(\rho_v \begin{pmatrix} \mu_{cl}^U, \mu_{cl}^R, \\ \mu_{cl}^D, \mu_{cl}^L, \\ \mu_{cl}^A, \mu_{cl}^B \end{pmatrix} \right)_7, \left(\pi_v \begin{pmatrix} \mu_{cl}^U, \mu_{cl}^R, \\ \mu_{cl}^D, \mu_{cl}^L, \\ \mu_{cl}^A, \mu_{cl}^B \end{pmatrix} \right)_7, \left(\gamma_v \left(\mu_{cl}^P \right) \right)_7,$$

які виконують функції формування єдиного потоку даних, що призначений для обробки в АPRO-мережній моделі нейрона. Послідовність міток, сформована на позиції $(p_v)_{13}$ у відповідності до часу створення, підлягає обробці в переході $(\tau_v)_8$ за допомогою процедур його ядра $(\rho_v(\mu_{cl}^P, \mu_{cl}^M))_8$, $(\pi_v(\mu_{cl}^P, \mu_{cl}^M))_8$, $(\gamma_v(\mu_{out}^P, \mu_{out}^M))_8$. Оскільки алгоритм роботи переходу $(\tau_v)_8$ для формування вихідних міток μ_{out}^P використовує не тільки дані з атрибутів вхідних міток μ_{cl}^P , але й результати попередніх циклів активації, його процедури активації та обробки включають як параметр також мітку μ_{cl}^M . Ця мітка продукується процедурою деактивації $(\gamma_v(\mu_{cl}^M))_9$ переходу $(\tau_v)_9$ за запитом переходу $(\tau_v)_8$, представленим атрибутами його вихідної мітки μ_{out}^M . Основним переходом, що забезпечує реалізацію алгоритму маршрутизації, є перехід $(\tau_v)_{10}$. Процедури ядра цього переходу

$$\left(\rho_v \begin{pmatrix} \mu_{tr}^U, \mu_{tr}^R, \\ \mu_{tr}^D, \mu_{tr}^L, \\ \mu_{tr}^A, \mu_{tr}^B \end{pmatrix} \right)_{10}, \left(\pi_v \begin{pmatrix} \mu_{tr}^U, \mu_{tr}^R, \\ \mu_{tr}^D, \mu_{tr}^L, \\ \mu_{tr}^A, \mu_{tr}^B \end{pmatrix} \right)_{10}, \left(\gamma_v \begin{pmatrix} \mu_{tr}^U, \mu_{tr}^R, \\ \mu_{tr}^D, \mu_{tr}^L, \\ \mu_{tr}^A, \mu_{tr}^B \end{pmatrix} \right)_{10}$$

реалізують алгоритми вибору вихідного каналу в залежності від пункту призначення мітки. Процедури ядра переходів

$$\left\{ (\tau_v)_i \right\}_{i=12}^{17}$$

$$\left\{ \left(\rho_v(\mu_{tr}, \mu_{cl}) \right)_i, \left(\pi_v(\mu_{tr}, \mu_{cl}) \right)_i, \left(\gamma_v(\mu_{out}) \right)_i \mid i = \overline{12, 17} \right\}$$

формують вихідні потоки міток типу μ_{out} шляхом об'єднання потоку транзитних міток типу μ_{tr} з мітками типу μ_{cl} , які були створені як результат обробки даних у нейроні.

Режим обміну даними базується на тому факті, що граф моделі кліткової мережі ізоморфний циркулянтному графу. А отже, для забезпечення транзитних пересилок на моделі можуть бути застосовані алгоритми оптимальних пересилок, запропоновані в розділі 2 даної роботи.

Застосування згаданих алгоритмів будемо розглядати на прикладі тривимірної моделі кліткової мережі, показаної на рис.4.2. Суть алгоритму, в даному випадку, зводиться до пошуку маршруту, який задовольняв би умову:

$$K = \min(|x| + |y| + |z|)$$

$$\text{при } 0 \leq m = \text{Id}_{out} - \text{Id}_{t_j} = xs_1 + ys_2 + zs_3 < N,$$

де (x, y, z) — координати циркулянтного графа, у вершинах

якого розміщені переходи АPRO-мережі; $s_1 = \frac{\sqrt[3]{N}}{2} - 1$,

$s_2 = \sqrt[3]{N}$, $\sqrt[3]{N^2}$ — кроки нумерації переходів вздовж координат (x, y, z) .

Розглянемо принципи побудови алгоритму визначення найкоротшого шляху між цільовим нейроном Ne_{out} і довільним нейроном Ne_j . Для цього позначимо через $(x_{\min}, y_{\min}, z_{\min})$ кількості кроків вздовж відповідних координат при прокладанні оптимального маршруту від нейрона Ne_j до Ne_{out} . Як показано в розділі 2, для тривимірної циркулянтної мережі існує три можливих алгоритми прокладання траси між двома вершинами. Кожний з цих алгоритмів має модифікації в залежності від

послідовності проходження траси вздовж певних координат. Як і в двовимірному випадку, внутрішні дії довільного проміжного нейрона по передачі транзитної мітки складаються з послідовності кроків σ , що спричиняють зміну маркувань:

$$M|\sigma\rangle M_3 = M|U_1\rangle M_1|U_2\rangle M_2|U_3\rangle M_3.$$

Під час виконання кроку

$$U_1 \subset \{(\tau_v)_1, (\tau_v)_2, (\tau_v)_3, (\tau_v)_4, (\tau_v)_5, (\tau_v)_6\}$$

спрацьовують переходи з процедурами обробки

$$\left\{ \left(\rho_v \left(\mu_{in}^U \right) \right)_1, \left(\rho_v \left(\mu_{in}^R \right) \right)_2, \left(\rho_v \left(\mu_{in}^D \right) \right)_3, \right. \\ \left. \left(\rho_v \left(\mu_{in}^L \right) \right)_4, \left(\rho_v \left(\mu_{in}^A \right) \right)_5, \left(\rho_v \left(\mu_{in}^B \right) \right)_6 \right\},$$

які виконують дії розпізнавання транзитних міток, що надходять на даний нейрон через відповідні канали зв'язку. Якщо мітка кваліфікована як транзитна, то вона завдяки спрацюванню відповідної процедури деактивації з множини

$$\left\{ \left(\gamma_v \left(\mu_{tr}^U, \mu_{cl}^U \right) \right)_1, \left(\gamma_v \left(\mu_{tr}^R, \mu_{cl}^R \right) \right)_2, \left(\gamma_v \left(\mu_{tr}^D, \mu_{cl}^D \right) \right)_3, \right. \\ \left. \left(\gamma_v \left(\mu_{tr}^L, \mu_{cl}^L \right) \right)_4, \left(\gamma_v \left(\mu_{tr}^A, \mu_{cl}^A \right) \right)_5, \left(\gamma_v \left(\mu_{tr}^B, \mu_{cl}^B \right) \right)_6 \right\}$$

стає елементом маркування M_1 на одній з позицій

$$\left\{ (p_v)_1, (p_v)_2, (p_v)_3, (p_v)_4, (p_v)_5, (p_v)_6 \right\}.$$

Процедури переходів кроку $U_2 \subset \{(\tau_v)_{10}, (\tau_v)_{11}\}$

містять внутрішні процедури реалізації алгоритмів транзитних пересилок, які визначають прямий та альтернативний маршрути просування міток. Переходи кроку

$$U_3 = \{(\tau_v)_{12}, (\tau_v)_{13}, (\tau_v)_{14}, (\tau_v)_{15}, (\tau_v)_{16}, (\tau_v)_{17}\}$$

разом з виходами

$$\left\{ (x_v)_1, (x_v)_2, (x_v)_3, (x_v)_4, (x_v)_5, (x_v)_6 \right\}$$

формують черги вихідних міток

$$\left(p, p^a \mu_{out} \right), \text{ де } \left\{ p, p^a \right\} \subset Ne_j^\bullet, 0 \leq j \leq N.$$

$Ne_j^\bullet = \{ l_j, r_j, u_j, d_j, a_j, b_j \}$ — множина вихідних позицій нейрона Ne_j . Розглянемо алгоритм процедури **Transmit_1**(μ_{out}), що є внутрішньою процедурою процедури обробки ρ_v та реалізує алгоритм прокладання траси по початковому перерізу у тривимірній клітковій мережі нейрона Ne'_{out} , який має координати (x'_{out}, y'_{out}) , ідентичні координатам (x_{out}, y_{out}) цільового нейрона N_{out} , і знаходиться на одному (x, y) -перерізі з нейроном Ne_j .

Procedure Transmit_1 (μ_{out});

begin

$x := 0$; $Ring_x := 0$; $Ring_y := N$;

$z_j := Id_t_j \operatorname{div} \sqrt[3]{N^2}$; $z_{out} := Id_t_{out} \operatorname{div} \sqrt[3]{N^2}$;

$Id_t'_{out} := (Id_t_{out} + (z_j - z_{out}) \cdot s) \operatorname{mod} N$;

While $Ring_x \neq Ring_y$ **do**

begin

$x := x + 1$;

$Ring_x := (Id_t_j + s_1 \cdot x) \operatorname{mod} \sqrt[3]{N^2}$;

$y = 0$;

While $Ring_x \neq Ring_y$ **do**

begin

$y := y + 1$;

$Ring_y := (Id_t'_j - s_2 \cdot y) \operatorname{mod} \sqrt[3]{N^2}$;

end;

end;

$x_{\min} := x$; $y_{\min} := y$;

```

For  $i := 1$  to 3 do
  begin
    Case  $i$  do
      begin
        1:begin  $x' := x; y' := y - s_2$ ; end;
        2:begin  $x' := x - s_2; y' := y - (s_2 - s_1)$ ; end;
        3:begin  $x' := x - s_2; y' := y + (s_2 - s_1)$ ; end;
      end;
      If  $(x' + y') < (x_{\min} + y_{\min})$  then
        begin  $x_{\min} := x'; y_{\min} := y'$ ; end;
      end;
    If  $|z_{out} - z_j| \leq \frac{\sqrt[3]{N}}{2}$  then
       $z_{\min} := z_{out} - z_j$  else  $z_{\min} := (z_{out} - z_j) - \sqrt[3]{N}$ ;
    If  $x_{\min} \neq 0$  then
      begin
        If  $x_{\min} < 0$  then  $p := l_j$  else  $p := r_j$ ;
      end else
      begin
        If  $y_{\min} \neq 0$  then
          begin
            If  $y_{\min} < 0$  then  $p := b_j$  else  $p := a_j$ ;
          end else
          begin
            If  $z_{\min} \neq 0$  then
              begin
                If  $z_{\min} < 0$  then  $p := u_j$  else  $p := d_j$ ;
              end
            end
          end
        end
      end
    end
  end

```

```
    end;
  end;
end;
If  $y_{\min} \neq 0$  then
begin
  If  $y_{\min} < 0$  then  $p^a := b_j$  else  $p^a := a_j$ ;
end else
begin
  If  $x_{\min} \neq 0$  then
begin
  If  $x_{\min} < 0$  then  $p^a := l_j$  else  $p^a := r_j$ ;
end else
begin
  If  $z_{\min} \neq 0$  then
begin
  If  $z_{\min} < 0$  then  $p^a := u_j$  else  $p^a := d_j$ ;
end;
end;
end;
end.
```

Наступний крок алгоритму включає пошук оптимального маршруту між нейроном Ne_j та нейроном Ne'_{out} . Прокладання траси відбувається спочатку вздовж координати x , а потім вздовж координати y . Зміна порядку проходження координат дає можливість прокладання альтернативного маршруту. Особливість даного підходу полягає у тому, що вибір між прямим та альтернативним маршрутом в рамках даного алгоритму може відбуватись на кожному транзитному нейроні. Це дає можливість забезпечити високу надійність передачі даних за умови рівномірного розподілу навантаження на вузли кліткової мережі.

Алгоритм прокладання траси по кінцевому перерізу реалізує процедура **Transmit_2**(μ_{out}). Як і в попередньому випадку, в рамках даного алгоритму розв'язується задача оптимального вибору напрямку передачі транзитної мітки. Прямий напрямок задає змінна p , а альтернативний напрямок представлений далі.

Procedure Transmit_2 (μ_{out});

begin

$x := 0$; $Ring_x := 0$; $Ring_y := N$;

$z_j := Id_t_j \text{ div } \sqrt[3]{N^2}$; $z_{out} := Id_t_{out} \text{ div } \sqrt[3]{N^2}$;

$Id_t'_{out} := (Id_t_j + (z_{out} - z_j) \cdot s_3) \text{ mod } N$;

If $|z_{out} - z_j| \leq \frac{\sqrt[3]{N}}{2}$ **then**

$z_{min} := z_{out} - z_j$ **else** $z_{min} := (z_{out} - z_j) - \sqrt[3]{N}$;

While $Ring_x \neq Ring_y$ **do**

begin

$x := x + 1$;

$Ring_x := (Id_t'_{out} + s_1 \cdot x) \text{ mod } \sqrt[3]{N^2}$;

$y = 0$;

While $Ring_x \neq Ring_y$ **do**

begin

$y := y + 1$;

$Ring_y := (Id_t_{out} - s_2 \cdot y) \text{ mod } \sqrt[3]{N^2}$;

end;

end;

$x_{min} := x$; $y_{min} := y$;

For $i := 1$ **to** 3 **do**

begin

Case i **do**

```

begin
  1:begin  $x' := x; y' := y - s_2$ ; end;
  2:begin  $x' := x - s_2; y' := y - (s_2 - s_1)$ ; end;
  3:begin  $x' := x - s_2; y' := y + (s_2 - s_1)$ ; end;
end;
If  $(x' + y') < (x_{\min} + y_{\min})$  then
  begin  $x_{\min} := x'; y_{\min} := y'$ ; end;
end;
If  $z_{\min} \neq 0$  then
  begin
    If  $z_{\min} < 0$  then
      begin  $p := u_j; p^a := u_j$ ; end else
        begin  $p := d_j; p^a := d_j$  end;
    end else
      begin
        If  $x_{\min} \neq 0$  then
          begin
            If  $x_{\min} < 0$  then  $p := l_j$  else  $p := r_j$ ;
          end else
            begin
              If  $y_{\min} \neq 0$  then
                begin
                  If  $y_{\min} < 0$  then  $p := b_j$  else  $p := a_j$ ;
                end;
              end;
            end;
          end else
            begin
              If  $y_{\min} \neq 0$  then
                begin
                  If  $y_{\min} < 0$  then  $p^a := b_j$  else  $p^a := a_j$ ;
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;

```

```

end else
begin
    If  $x_{\min} < 0$  then  $p^a := l_j$  else  $p^a := r_j$ ;
end;
end;
end.

```

Головна відмінність даного алгоритму від попереднього полягає у тому, що проміжний нейрон Ne'_{out} з координатами (x'_{out}, y'_{out}) , ідентичними координатам (x_j, y_j) початкового нейрона N_j , знаходиться на одному (x, y) -перерізі з нейроном Ne_{out} . Перший етап прокладання траси включає пересилку транзитних міток між початковим нейроном Ne_j та проміжним нейроном Ne'_{out} через проміжні нейрони з номерами $(Ne_j \pm \sqrt[3]{N^2}) \bmod N$.

На другому етапі алгоритм реалізує пошук оптимального маршруту між нейроном Ne'_{out} та нейроном Ne_{out} . Як і в попередньому випадку, прокладання траси відбувається спочатку вздовж координати x , а потім вздовж координати y . Але в даному випадку траса пролягає по (x, y) -перерізу цільового нейрона Ne_{out} . Змінивши порядок проходження координат, одержуємо альтернативний варіант прокладання маршруту. Як і в попередньому випадку, алгоритм має розподілений характер і дозволяє модифікувати маршрут на кожному з проміжних нейронів із врахуванням завантаження того чи іншого напрямку пересилки даних. Поєднавши дві попередні версії алгоритмів, одержуємо алгоритм прокладання траси по початковому та кінцевому перерізах, який реалізує процедура **Transmit_3** (μ_{out}) .

Procedure *transmit_3* (μ_{out});

begin

$x := 0$; $Ring_x := 0$; $Ring_y := N$;

$z_j := Id_t_j \operatorname{div} \sqrt[3]{N^2}$; $z_{out} := Id_t_{out} \operatorname{div} \sqrt[3]{N^2}$;

If $|z_{out} - z_j| \leq \frac{\sqrt[3]{N}}{2}$ **then** $z_{min} := z_{out} - z_j$ **else**

$z_{min} := (z_{out} - z_j) - \sqrt[3]{N}$;

While $Ring_x \neq Ring_y$ **do**

begin

$x := x + 1$; $y := 0$;

$Ring_x := \left(\left((Id_t_{j+s_1} \cdot x) \operatorname{mod} \sqrt[3]{N^2} \right) + z_{min} \sqrt[3]{N^2} \right) \operatorname{mod} N$;

While $Ring_x \neq Ring_y$ **do**

begin

$y := y + 1$;

$Ring_y := (Id_t_{out} - s_2 \cdot y) \operatorname{mod} \sqrt[3]{N^2}$;

end;

end;

$x_{min} := x$; $y_{min} := y$;

For $i := 1$ **to** 3 **do**

begin

Case i **do**

begin

1:begin $x' := x$; $y' := y - s_2$; **end**;

2:begin $x' := x - s_2$; $y' := y - (s_2 - s_1)$; **end**;

3:begin $x' := x - s_2$; $y' := y + (s_2 - s_1)$; **end**;

end;

```

If  $(x' + y') < (x_{\min} + y_{\min})$  then
  begin  $x_{\min} := x'; y_{\min} := y';$  end;
end;
If  $x_{\min} \neq 0$  then
begin
  If  $x_{\min} < 0$  then  $p := l_j$  else  $p := r_j;$ 
end else
begin
  If  $z_{\min} \neq 0$  then
  begin
    If  $z_{\min} < 0$  then  $p := u_j$  else  $p := d_j;$ 
  end else
  begin
    If  $y_{\min} \neq 0$  then
    begin
      If  $y_{\min} < 0$  then  $p := b_j$  else  $p := a_j;$ 
    end;
    end;
  end;
end;
If  $y_{\min} \neq 0$  then
begin
  If  $y_{\min} < 0$  then  $p^a := b_j$  else  $p^a := a_j;$ 
end else
begin
  If  $z_{\min} \neq 0$  then
  begin
    If  $z_{\min} < 0$  then  $p^a := u_j$  else  $p^a := d_j;$ 
  end else

```

```
begin  
If  $x_{\min} \neq 0$  then  
  begin  
    If  $x_{\min} < 0$  then  $p^a := l_j$  else  $p^a := r_j$ ;  
  end;  
end;  
end;  
end.
```

В рамках даного алгоритму оптимальна траса спочатку прокладається від нейрона Ne_j вздовж координати x по (x, y) -перерізу, що співпадає з перерізом початкового нейрона Ne_j , до деякого проміжного нейрона Ne'_j . Другий крок полягає у переході між (x, y) перерізами вздовж координати z від проміжного нейрона Ne'_j до іншого проміжного нейрона Ne'_{out} , який знаходиться на тому ж (x, y) -перерізі, що і цільовий нейрон Ne_{out} , і його координати x та y дорівнюють відповідним координатам проміжного нейрона Ne'_j . На завершальному кроці відбувається прокладання траси від проміжного нейрона Ne'_{out} до цільового нейрона Ne_{out} вздовж координати y . Альтернативний варіант прокладання траси з використанням даного алгоритму включає прокладання траси вздовж координати y по початковому перерізу та вздовж координати x по кінцевому перерізу. Основна відмінність даного алгоритму від попередніх двох полягає у тому, що у даному випадку виникає можливість використання різних трас переходу між перерізами вздовж координати z , що підвищує надійність організації комунікацій між довільними нейронами мережі.

Розміщення мітки μ_{out} на одній із позицій $\{u, r, d, l, a, b\}$ є завершальним етапом алгоритму маршрутизації і може бути виконане за допомогою процедури **Out**, скрипт якої співпадає зі скриптом відповідної процедури двовимірного нейрона. Як і у випадку двовимірного нейрона, суть даної процедури полягає у перевірці завантаження того чи іншого напрямку передачі даних. У випадку негативних результатів такої перевірки процедура забезпечує передачу даних з урахуванням альтернативної траєкторії прокладання траси. Комунікації нейрона з навколишнім середовищем кліткової мережі виконують множини входів $\{(e_v)_i \mid i = 1, 2, \dots, 6\}$ та виходів $\{(x_v)_i \mid i = 1, 2, \dots, 6\}$.

Значення параметрів входів зведено в таблицю 5.4.

Таблиця 5.4

Таблиця входів моделі нейрона Ne_v тривимірної кліткової мережі

	Входи E_v						
	Параметри входів η_v				Ядро N_v		Мітки
	Max $_e_v = 1$, Level $_e_v = 1$						q_v
	Id $_e_v$	Cur $_e_v$	δ $_e_v$	Cur $_e_v$	ρ_v	ω_v	
$(e_v)_1$	Entran_U	0 ÷ 1	$(\rho_v(\mu_{in}^U))_1$		$(\omega_v)_1$		μ_{in}^U
$(e_v)_2$	Entran_R	0 ÷ 1	$(\rho_v(\mu_{in}^R))_2$		$(\omega_v)_1$		μ_{in}^R
$(e_v)_3$	Entran_D	0 ÷ 1	$(\rho_v(\mu_{in}^D))_3$		$(\omega_v)_3$		μ_{in}^U

Входи E_v							
Параметри входів η_v $\text{Max}_{-e_v} = 1, \text{Level}_{-e_v} = 1$					Ядро N_v		Мітки q_v
	Id_{-e_v}	Cur_{-e_v}	δ_{-e_v}	Cur_{-e_v}	ρ_v	ω_v	
$(e_v)_4$	Entran_L	$0 \div 1$	$(\rho_v(\mu_{in}^L))_4$			$(\omega_v)_4$	μ_{in}^U
$(e_v)_5$	Entran_A	$0 \div 1$	$(\rho_v(\mu_{in}^A))_5$			$(\omega_v)_5$	μ_{in}^A
$(e_v)_6$	Entran_B	$0 \div 1$	$(\rho_v(\mu_{in}^B))_6$			$(\omega_v)_6$	μ_{in}^B

В моделі використовується структура входів без черг, про що свідчить значення параметра $(\text{Max}_{-e_v})_j = 1$. Таким чином, параметр $\text{Cur}_{-e_v} = 0$ при відсутності міток на відповідному вході нейрона Ne_v , і $\text{Cur}_{-e_v} = 1$, якщо вхід зайнятий обробкою чергової мітки. Значення параметрів $(\delta_{-e_v})_j$ — лічильник часу входу $(e_v)_j$ та $(\Delta_{-e_v})_j$ — період простою входу $(e_v)_j$ змінюються в процесі моделювання і визначаються функцією активації $(\rho_v)_j$ ідентично до переходів.

Для зв'язку з вихідними позиціями операторних переходів, які представляють нейрони кліткової мережі, використовується множина елементів АPRO-мережі, які одержали назву виходів. Виходи для моделі нейрона Ne_v тривимірної кліткової мережі утворюють множину $X_v = \{(x_v)_i \mid i = 1, \dots, 6\}$.

Значення параметрів виходів зведено в таблицю 5.5.

Таблиця 5.5

Таблиця виходів моделі нейрона Ne_v тривимірної кліткової мережі

	Виходи X_v				
	Параметри виходів o_v $\text{Max } x_v = 1,$			Ядро N_v	Мітки q_v
	$\text{Id } x_v$	$\text{Cur } x_v$	φx_v	γ_v	
$(x_v)_1$	Exit_U	$0 \div 1$	μ_tran	$(\gamma_v(\mu_{out}^U))_1$	μ_{out}^U
$(x_v)_2$	Exit_R	$0 \div 1$		$(\gamma_v(\mu_{out}^R))_2$	μ_{out}^R
$(x_v)_3$	Exit_D	$0 \div 1$		$(\gamma_v(\mu_{out}^D))_3$	μ_{out}^D
$(x_v)_4$	Exit_L	$0 \div 1$		$(\gamma_v(\mu_{out}^L))_4$	μ_{out}^L
$(x_v)_5$	Exit_A	$0 \div 1$		$(\gamma_v(\mu_{out}^A))_5$	μ_{out}^A
$(x_v)_6$	Exit_B	$0 \div 1$		$(\gamma_v(\mu_{out}^B))_6$	μ_{out}^B

Як видно з таблиці 5.5, виходи $\{(x_v)_i \mid i = 1, \dots, 6\}$ також не містять черг міток. Мітки, що проходять через виходи, характеризуються однаковим типом μ_tran , оскільки містять атрибути даних, призначені для передачі сусіднім нейронам. Основою ядра виходів є функція деактивації γ_v , яка реалізує розміщення міток на вихідних позиціях.

Громіздкість задавання даної моделі в програмному середовищі може бути значно зменшеною у випадку застосування спеціальних підходів до формалізації АPRO-мережних структур за допомогою алгебри процесів. Розглянута у попередньому параграфі алгебра процесів дозволяє представити АPRO-мережну модель у вигляді паралельних обчислювальних процесів, які паралельно розвиваються та асинхронно взаємодіють між собою.

Головний процес P моделі кліткової нейронної мережі будемо розглядати як послідовність процесів вводу і обробки інформації та процесу локальної взаємодії: $P := A.B.O$. Процес вводу A є складним процесом, який відображає множину процесів вводу робочого навантаження в кожний нейрон. У випадку однотипного задавання робочого навантаження процес A може бути представлений оператором клонування $A := k \times A_1 \mid_{0 \leq k < N}$, який в залежності від значення константи k дозволяє описати процес вводу інформації в довільний вузол КНС. Якщо кожний нейрон одержує індивідуальне робоче навантаження, то процес $A = C \{A_0, \dots, A_{k-1}\}$ розглядається як паралельна композиція k процесів $A := A_0 \mid A_1 \mid \dots \mid A_i \mid \dots \mid A_{k-1}$, де A_i — процес вводу інформації в нейрон Ne_i . В загальному випадку від робочого навантаження в кліткову нейронну мережу допускає існування певних підмножин нейронів, що характеризуються ідентичними робочими навантаженнями. В цьому випадку процес A матиме вигляд:

$$A := (k_0 \times A_0) \mid \dots \mid (k_i \times A_i) \mid \dots \mid (k_{n-1} \times A_{n-1}),$$

де k_i — кількість нейронів, що входять до i -ї підмножини; A_i — процес вводу робочого навантаження, ідентичний для всіх k_i нейронів i -ї підмножини; n — кількість підмножин при

загальній кількості нейронів кліткової нейронної мережі:

$$N = \sum_{i=0}^{n-1} k_i.$$

Процес обробки інформації B складається з множини процесів $B = C \{ B_i \mid i = 0, \dots, N-1 \}$, кожен елемент якої відображає процес обробки інформації в окремому нейроні. У загальному випадку цей процес представимо виразом алгебри процесів

$$B := (B_0 \triangleleft B_1 \mid B_2 \mid \dots \mid B_{N-1}) \\ \mid (B_1 \triangleleft B_0 \mid B_2 \mid \dots \mid B_{N-1}) \mid \dots \mid (B_{N-1} \triangleleft B_0 \mid B_1 \mid \dots \mid B_{N-2}),$$

що задає правила асинхронної взаємодії кожного нейрона кліткової мережі з паралельною композицією, до якої входить решта нейронів даної мережі. Довільний процес обробки інформації B_i в нейроні Ne_i складається з двох процесів, які відображають основні функції нейрона — обчислювальну та комунікаційну. Під час функціонування нейрона згадані процеси взаємодіють між собою, утворюючи загальний процес B_i . Вхідний комунікаційний процес

$$K_i^{in} = C \{ K_i^{Uin}, K_i^{Rin}, K_i^{Din}, K_i^{Lin}, K_i^{Ain}, K_i^{Bin} \}$$

включає множину процесів, які забезпечують аналіз даних, що надходять на вхід нейрона з метою сепарації транзитної інформації. Спочатку відбувається керований вибір процесу, що забезпечує аналіз певного каналу зв'язку. Якщо результат аналізу даного каналу вказує на те, що одержані дані є транзитною інформацією, то запускається процес M для визначення подальшого маршруту шляхом вибору відповідного вихідного каналу зв'язку. Якщо дані, одержані по відповідному вхідному каналу, призначені для нейрона Ne_i , то відбувається керована передача даних обчислювальному процесу S_i . Результатом обох дій є керована передача даних після

виконання операції вибору та процеси комунікації через вихідні канали зв'язку. Таким чином, розгорнутий вираз для процесу обробки інформації має вигляд:

$$B_i := \left(\left(\left(v_i!K_i^{Uin} + v_i!K_i^{Rin} + v_i!K_i^{Din} + v_i!K_i^{Lin} + v_i!K_i^{Ain} + v_i!K_i^{Bin} \right) \cdot M \right) + \left(\left(v_i!K_i^{Uin} + v_i!K_i^{Rin} + v_i!K_i^{Din} + v_i!K_i^{Lin} + v_i!K_i^{Ain} + v_i!K_i^{Bin} \right) \triangleright \Re(X_i = S_i) \right) \right) \triangleright (K_i^{Uout} + K_i^{Rout} + K_i^{Dout} + K_i^{Lout} + K_i^{Aout} + K_i^{Bout}),$$

де $v_i = (v_i^{Uin}, v_i^{Rin}, v_i^{Din}, v_i^{Lin}, v_i^{Ain}, v_i^{Bin})$ — змінна, значення якої визначає вибір відповідного вхідного комунікаційного процесу з множини

$$K_i^{in} = \{K_i^{Uin}, K_i^{Rin}, K_i^{Din}, K_i^{Lin}, K_i^{Ain}, K_i^{Bin}\},$$

M — процес визначення маршруту транзитних даних,

S_i — обчислювальний процес в нейроні Ne_i ,

$$K_i^{out} = C \{K_i^{Uout}, K_i^{Rout}, K_i^{Dout}, K_i^{Lout}, K_i^{Aout}, K_i^{Bout}\} —$$

вихідний комунікаційний процес, який включає множину вихідних комунікаційних процесів для відповідних каналів зв'язку.

Процес міжнейронної взаємодії включає множину процесів локальної взаємодії

$$O = C \{O_j^{Out} \mid j := 0, \dots, N - 1\},$$

де кожен процес O_j^{Out} включає процеси асинхронної взаємодії нейрона Ne_i з сусідніми нейронами

$$O_i^{out} = C \{O_i^{Uout}, O_i^{Rout}, O_i^{Dout}, O_i^{Lout}, O_i^{Aout}, O_i^{Bout}\}.$$

Правила такої взаємодії формально представлені за допомогою виразу:

$$O_i^{out} := \left(\begin{array}{l} \Re \left(X_i = O_i^{Uout} \right) \triangleleft \triangleright K_{\left(i - \sqrt[3]{N^2} \right) \bmod N}^{Din} \\ \Re \left(X_i = O_i^{Dout} \right) \triangleleft \triangleright K_{\left(i + \sqrt[3]{N^2} \right) \bmod N}^{Uin} \\ \Re \left(X_i = O_i^{Rout} \right) \triangleleft \triangleright K_{\left(i + \frac{\sqrt[3]{N}}{2} - 1 \right) \bmod \sqrt[3]{N^2}}^{Lin} \\ \Re \left(X_i = O_i^{Lout} \right) \triangleleft \triangleright K_{\left(i - \frac{\sqrt[3]{N}}{2} + 1 \right) \bmod \sqrt[3]{N^2}}^{Rin} \\ \Re \left(X_i = O_i^{Aout} \right) \triangleleft \triangleright K_{\left(i + \sqrt[3]{N} \right) \bmod \sqrt[3]{N^2}}^{Bin} \\ \Re \left(X_i = O_i^{Bout} \right) \triangleleft \triangleright K_{\left(i - \sqrt[3]{N} \right) \bmod \sqrt[3]{N^2}}^{Ain} \end{array} \right)$$

Об'єднавши вказані вирази, одержимо формальний опис процесу функціонування кліткової мережі:

1-й рівень:

$$P := A.B.O$$

2-й рівень:

$$P := C \left\{ A_i \mid i := 0, \dots, k-1 \right\}. C \left\{ B_j \mid j = 0, \dots, N-1 \right\}. C \left\{ O_j^{out} \mid j = 0, \dots, N-1 \right\}$$

3-й рівень:

$$\begin{aligned} P &:= (k_0 \times A_0) \cdots (k_i \times A_i) \cdots (k_{n-1} \times A_{n-1}). \\ &(B_0 \triangleleft \triangleright B_1 \mid B_2 \mid \cdots \mid B_{N-1}) \mid (B_1 \triangleleft \triangleright B_0 \mid B_2 \mid \cdots \mid B_{N-1}) \mid \cdots \mid (B_{N-1} \triangleleft \triangleright B_0 \mid B_1 \mid \cdots \mid B_{N-2}). \\ &(O_0^{out} \mid O_1^{out} \mid \cdots \mid O_{N-1}^{out}) \end{aligned}$$

4-й рівень:

$$\begin{aligned} P &:= \left\{ \left(A_i \leftrightarrow K_{i_0}^{in} \mid K_{i_1}^{in} \mid \cdots \mid K_{i_{k_i}}^{in} \right) \mid \left(A_j \leftrightarrow K_{j_0}^{in} \mid K_{j_1}^{in} \mid \cdots \mid K_{j_{k_j}}^{in} \right) \right\}_{i, j=0}^{n-1} \cdot \\ &\left\{ \left(\left(\left(v_i ! K_i^{Uin} + v_i ! K_i^{Rin} + v_i ! K_i^{Din} + v_i ! K_i^{Lin} + v_i ! K_i^{Ain} + v_i ! K_i^{Bin} \right) M \right) + \right. \right. \\ &\left. \left(\left(v_i ! K_i^{Uin} + v_i ! K_i^{Rin} + v_i ! K_i^{Din} + v_i ! K_i^{Lin} + v_i ! K_i^{Ain} + v_i ! K_i^{Bin} \right) \triangleright \Re \left(X_i = S_i \right) \right) \right\} \triangleright \triangleleft \\ &\left(K_i^{Uout} + K_i^{Rout} + K_i^{Dout} + K_i^{Lout} + K_i^{Aout} + K_i^{Bout} \right) \\ &\left(\left(\left(v_{j_y} ! K_{j_y}^{Uin} + v_{j_y} ! K_{j_y}^{Rin} + v_{j_y} ! K_{j_y}^{Din} + v_{j_y} ! K_{j_y}^{Lin} + v_{j_y} ! K_{j_y}^{Ain} + v_{j_y} ! K_{j_y}^{Bin} \right) M \right) + \right) \Bigg\}_{i, j_y=0}^{N-1} \\ &\left(\left(v_{j_y} ! K_{j_y}^{Uin} + v_{j_y} ! K_{j_y}^{Rin} + v_{j_y} ! K_{j_y}^{Din} + v_{j_y} ! K_{j_y}^{Lin} + v_{j_y} ! K_{j_y}^{Ain} + v_{j_y} ! K_{j_y}^{Bin} \right) \triangleright \right) \triangleright \\ &\left(\Re \left(X_{j_y} = S_{j_y} \right) \right) \\ &\left(K_{j_y}^{Uout} + K_{j_y}^{Rout} + K_{j_y}^{Dout} + K_{j_y}^{Lout} + K_{j_y}^{Aout} + K_{j_y}^{Bout} \right) \Bigg\}_{\substack{i, j_y=0 \\ y=0, N-2}} \end{aligned}$$

$$\left(\begin{array}{l} \Re(X_i = O_i^{Uout}) \triangleleft K_{(i-\sqrt{N^2}) \bmod N}^{Din} \\ \Re(X_i = O_i^{Dout}) \triangleleft K_{(i+\sqrt{N^2}) \bmod N}^{Uin} \\ \Re(X_i = O_i^{Rout}) \triangleleft K_{\left(\frac{i+\sqrt{N^2}}{2}-1\right) \bmod \sqrt{N^2}}^{Lin} \\ \Re(X_i = O_i^{Lout}) \triangleleft K_{\left(\frac{i-\sqrt{N^2}}{2}+1\right) \bmod \sqrt{N^2}}^{Rin} \\ \Re(X_i = O_i^{Aout}) \triangleleft K_{(i+\sqrt{N^2}) \bmod \sqrt{N^2}}^{Bin} \\ \Re(X_i = O_i^{Bout}) \triangleleft K_{(i-\sqrt{N^2}) \bmod \sqrt{N^2}}^{Am} \end{array} \right) \left| \left| \left(\begin{array}{l} \Re(X_j = O_j^{Uout}) \triangleleft K_{(j-\sqrt{N^2}) \bmod N}^{Din} \\ \Re(X_j = O_j^{Dout}) \triangleleft K_{(j+\sqrt{N^2}) \bmod N}^{Uin} \\ \Re(X_j = O_j^{Rout}) \triangleleft K_{\left(\frac{j+\sqrt{N^2}}{2}-1\right) \bmod \sqrt{N^2}}^{Lin} \\ \Re(X_j = O_j^{Lout}) \triangleleft K_{\left(\frac{j-\sqrt{N^2}}{2}+1\right) \bmod \sqrt{N^2}}^{Rin} \\ \Re(X_j = O_j^{Aout}) \triangleleft K_{(j+\sqrt{N^2}) \bmod \sqrt{N^2}}^{Bin} \\ \Re(X_j = O_j^{Bout}) \triangleleft K_{(j-\sqrt{N^2}) \bmod \sqrt{N^2}}^{Am} \end{array} \right) \right|^{N-1} \right)_{i,j=0 \atop i \neq j}$$

Перехід до нижчих рівнів формального опису моделі кліткової мережі пов’язаний з описом активностей, які утворюють процеси 4-го рівня. Структура цих активностей повністю залежить від фізичної постановки задачі та методів її розв’язання.

5.2. Розробка програмної системи моделювання

Для реалізації математичних моделей в комп’ютерних системах за останні десятиліття створено велику кількість програмних систем моделювання, які дозволяють описувати та досліджувати складні фізичні процеси [76, 77]. В основі цих програмних систем моделювання завжди лежить певний математичний апарат, який і визначає коло задач, що можуть бути розв’язані з їх використанням. Переважна більшість програмного забезпечення в сфері математичного моделювання залишається орієнтованою на однопроцесорні комп’ютерні засоби, що істотно обмежує складність проблем, доступних для розгляду. Природним вирішенням проблеми обмежених ресурсів є використання паралельних обчислювальних систем. Тому велика група програмних систем моделювання згаданого типу орієнтована саме на такий підхід. Їх загальним недоліком є призначення для реалізації на мультипроцесорних системах, доступ до яких значно утруднений як з технічних, так і з фінансових причин. Саме в останні роки, завдяки поширенню кластерних систем та технології Grid, виникли передумови бурхливого розвитку програмного забезпечення для реалізації

паралельних математичних моделей складних фізичних процесів. В результаті з'явився цілий рід успішних проектів, які базуються на застосуванні чисельних методів розв'язування крайових задач математичної фізики [78]. Їх аналіз дозволяє впевнено зробити висновок, що головна проблема, з якою зіткнулись розробники, полягає в забезпеченні ефективного використання розподілених ресурсів у ході обчислювального процесу. Вирішення цієї проблеми полягає в застосуванні формальних засобів як проміжної ланки між математичною постановкою задачі та моделлю паралельного обчислювального процесу для її розв'язування. Саме такий підхід запропоновано в даній роботі. В розділі 2 представлено моделі кліткових нейронних мереж на базі тривимірних циркулянтних графів, що забезпечують оптимальну організацію обчислювального процесу паралельного розв'язування крайових задач. В якості основи для представлення цих моделей у вигляді програм застосовано формальні засоби. В розділі 3 запропоновано формальні засоби для такого опису, а в розділі 4 наведено опис моделі кліткової мережі за допомогою згаданих формальних засобів. На основі формалізованих специфікацій кліткових нейронних мереж побудовано програмну систему моделювання для моделювання складних фізичних процесів. Для опису структури моделі та принципів її функціонування створено мову моделювання.

5.2.1. Розробка мови моделювання

Моделі в програмній системі моделювання задаються мовою, яка ґрунтується на алгебрі процесів, описаній в попередньому розділі. Крім того, мова містить ряд операторів для спрощення задавання складних моделей, що включають значну кількість однакових або подібних компонентів.

Першою і основною вимогою до розроблюваної мови є наявність в ній операторів, що дозволяють описувати моделі у виразах алгебри процесів. Мова також повинна забезпечувати

просте і зручне задавання реального робочого навантаження для створеної моделі. Важливою характеристикою мови моделювання є усунення синтаксичних недоліків алгебри процесів.

5.2.1.1. Типи даних, змінні і масиви мови моделювання

Всі змінні, що використовуються в мові, мають тип числа з плаваючою комою подвійної точності (double). У всіх операціях, які потребують цілого типу даних, значення змінних автоматично округляються. Над змінними можна здійснювати побітові логічні операції та операції зсуву. У логічних виразах ненульове значення змінної вважається логічною одиницею, нульове – логічним нулем.

Імена змінних завжди починаються зі знаку \$, що робить їх відмінними від решти ідентифікаторів, що зустрічаються в програмі. Окремих операторів опису змінних в мові не передбачено, оскільки всі змінні мають однаковий тип, що автоматично змінюється в залежності від операції, в якій бере участь змінна. Тому пам'ять під змінну виділяється інтерпретатором в той момент, коли змінна вперше зустрічається в тексті програми. Область видимості змінних глобальна в межах програми.

Змінні можна використовувати в суфіксній формі, коли за ім'ям змінної розміщуються в квадратних дужках один або більше виразів. Це дає можливість формувати зі змінних «масиви». Але це є лише синтаксичним спрощенням. Всі змінні зберігаються інтерпретатором в єдиному однорідному списку, і кожна змінна в суфіксній формі перетворюється на змінну в звичайній формі. Тобто вираз $i[2>1][2+3]$ і вираз i_1_5 означають одну і ту саму змінну і можуть використовуватися взаємозамінно.

Такий підхід до формування типів змінних і обробки масивів впливає з мети створення мови моделювання. Оскільки основною метою мови є спрощення задавання моделі у вигляді

списку виразів алгебри процесів, то таких типів даних і рівня контролю за перетворенням типів цілком достатньо. Оскільки мова не орієнтована на високопродуктивні обчислення чи на операції зі значною кількістю даних, то розглянутий підхід до обробки масивів має ряд переваг. Зокрема, при формуванні певних складених об'єктів алгебри процесів часто постає необхідність роботи з розрідженими масивами (масивами, що мають великі розміри, проте надзвичайно малу кількість ненульових елементів). В цьому випадку подання масиву у вигляді списку його значень має перевагу в використанні пам'яті та обчислювальних ресурсів.

Особливість мови моделювання полягає в тому, що вона включає принципово новий тип об'єктів — об'єкти алгебри процесів. Основний тип об'єктів алгебри процесів — процес. Активність є окремим випадком процесу. Над об'єктами алгебри процесів визначені свої оператори і операції, які описані окремо. Процеси не можуть використовуватись в звичайних операціях.

5.2.1.2. Загальні операції та оператори

В мові реалізовано наступні *арифметичні операції*: додавання (+), віднімання (-), множення (*), ділення (/), ділення по модулю (%), інкремент (++), декремент (--). Перші чотири операції використовують тип даних з плаваючою комою, останні три вважають змінні цілими. Інкремент і декремент можуть використовуватись у префіксній і суфіксній формі. Знак віднімання допускає унарне використання.

Також можна використовувати такі *порозрядні операції*: порозрядне АБО (|), порозрядне І (&), порозрядне виключне АБО (^), зсув вправо (>>), зсув вправо з заповненням старшого біта нулем (>>>), зсув вліво (<<), порозрядне інвертування (~). Виконуємо всі операції трактуючи змінні, як 32-бітні цілі.

До змінних можуть бути застосовані *операції відношення*: операція дорівнює (==), не дорівнює (!=), більше (>), менше (<),

більше-дорівнює (\geq), менше-дорівнює (\leq). При виконанні операцій також вважаємо змінні числами з плаваючою комою, а результат — в залежності від операції 1 чи 0.

Також в мові присутні *логічні операції*: логічне І (&&), логічне АБО (||), логічне НЕ (!). Змінні в цих операціях вважаємо булевими змінними, а результат також дорівнює 1 чи 0.

Старшинство операцій у мові є наступним:

1. (), []
2. ++, --, ~, !, унарний -
3. *, /, %
4. +, -
5. >>, >>>, <<
6. >, >=, <, <=
7. ==, !=
8. &
9. ^
10. |
11. &&
12. ||

Мова включає кілька стандартних операторів для управління потоком виконання програми. Оператор розгалуження *if* має стандартний синтаксис мови C, і використовується для вибору однієї з гілок алгоритму. Оператор циклу *for* має спрощений синтаксис та використовується для виконання певної частини програми фіксовану кількість разів. Оператор циклу *while* має синтаксис, аналогічний мові C, і є стандартним циклом з передумовою. Оператор циклу *do-while* має синтаксис, аналогічний мові C, і є стандартним циклом з постумовою. Оператор присвоєння (=) використовується для надання значень змінним. Правила перетворення типів при присвоєнні детально описані в попередньому пункті.

5.2.1.3. Операції і оператори алгебри процесів

Процес у мові може використовуватись у звичайній формі, суфікській формі та суфікській формі з зазначенням області змінних. Процес у звичайній формі описується ідентифікатором (на відміну від змінних, перед ім'ям процесу не ставиться знак \$). Якщо процес визначається у суфікській формі, за ім'ям процесу в квадратних дужках можуть бути записані один або більше виразів. Так само, як і з масивами змінних, такий підхід є лише синтаксичним спрощенням для опису великої кількості однотипних процесів. При виконанні програми ім'я процесу в суфікській формі перетворюється на звичайне ім'я. Наприклад, процес `Processor[2*$i+$j][$j]` для $\$i=1$, $\$j=3$ буде перетворений у `Processor_5_3`. Це дає можливість описувати аналогічними виразами схожі об'єкти і долає один з основних недоліків, який виникає через використання алгебр процесів – складність і громіздкість моделі при описуванні значної кількості подібних об'єктів.

При визначенні процесу в круглих дужках можна задати ідентифікатор області змінних. Кожна активність, що зв'язана з конкретним класом (реальне робоче навантаження) має можливість зберігати певні змінні в єдиній області змінних і розділяти їх з іншими активностями. Ідентифікатор області змінних вказує, з якою саме областю буде працювати активність. Активності з різних областей змінних не можуть обмінюватись інформацією через розділювані змінні. Приклад опису процесу в суфікській формі з зазначенням області змінних: `Processor[1][2]` (`Workspace`).

На множині процесів визначені операції алгебри процесів, описані в розділі 3 даної роботи.

Результатами всіх операцій алгебри процесів є визначення процесів. Операції використовуються для ієрархічного опису складних процесів на основі простих, керуючись синтаксисом і семантикою запропонованої алгебри процесів.

В мові визначені два оператори алгебри процесів.

Оператор присвоєння алгебри процесів ($=$) використовується для асоціації певного імені зі складеним процесом, тобто для задавання складеного процесу на основі більш простих.

Оператор задавання реального робочого навантаження має синтаксис, що складається з ключового слова `define`, імені активності, імені зовнішнього класу та необов'язкового списку фактичних параметрів.

5.2.2. Структура програмної системи моделювання

Структура програмної системи моделювання представлена множиною підсистем пакетів, що пов'язані зв'язками, показаними на рис.5.6.

Ядром програмної системи моделювання є `ModelingCore` — пакет класів, який створюється першим, і є власне “вхідною точкою” програмної системи моделювання. `ModelingCore` створює `MainGUI` — пакет класів (вікон і діалогів) для спілкування з користувачем. Головний інтерфейс `MainGUI` містить команди системного налаштування. Зокрема, за допомогою цього інтерфейсу користувач ініціює створення `APROnetBuilder` — пакет класів, головна мета якого — згенерувати структурне представлення `APRO`-мережі. `APROnetBuilder` створює `APROBuilderGUI`, через який користувач матиме змогу задавати модель. Структурна модель `APRO`-мережі може бути безпосередньо сформована шляхом використання графічних елементів інтерфейсу `APROBuilderGUI` у випадку, коли кількість її об'єктів не перевищує 128.

При побудові моделей з більшою кількістю об'єктів або моделей зі складною структурою зв'язків між переходами та позиціями використовують алгебру процесів. Для цього фінальна команда задавання моделі “`build`” запускає процес парсинга.

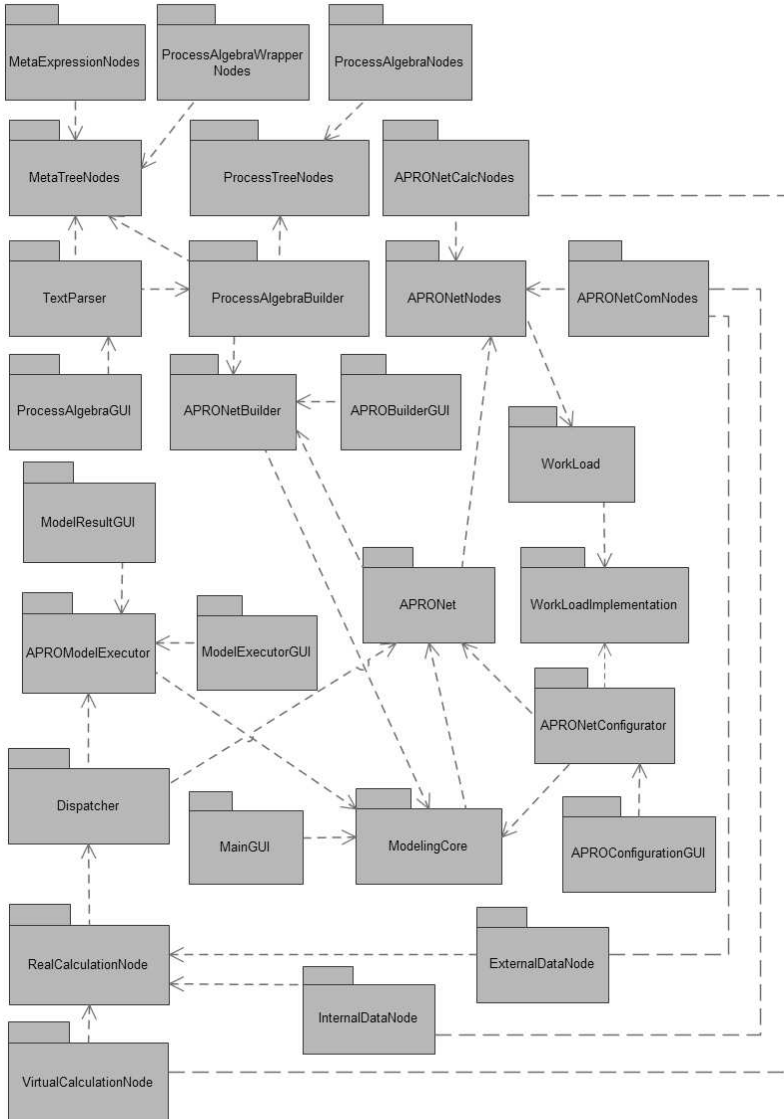


Рис.5.6. UML-діаграма пакетів програмної системи
МОДЕЛЮВАННЯ

Коли користувач ініціює генерування мережі, APRONetBuilder створює ProcessAlgebraBuilder – проміжний будівельник. ProcessAlgebraBuilder створює TextParser, який включає інтерфейс ProcessAlgebraGUI. При цьому виникає такий потік виконання: за допомогою ProcessAlgebraGUI формується скрипт моделі на мові моделювання ПМАП, який отримує TextParser і формує з нього дерево розбору. Згадане дерево розбору передається в ProcessAlgebraBuilder. Пакет ProcessAlgebraBuilder розкриває цикли, розгалуження, тобто “виконує” дерево розбору і формує на виході дерево виразів алгебри процесів, яке передає в APRONetBuilder.

Таким чином реалізується одна з важливих відмінностей даної програмної системи моделювання, що полягає в комбінованому підході, який поєднує процесне моделювання та моделювання списком подій. APRONetBuilder з дерева виразів алгебри процесів за заданими правилами формує APRONet, який містить у собі APRONetCalcNodes і APRONetComNodes (переходи й вершини) — реалізації загального спільного інтерфейсу APRONetNodes. Від кожного APRONetNodes є зв'язок з пакетом класів WorkLoad — звідси формуються “реальні” процедури переходу. Спочатку ці зв'язки порожні, і побудований APRONet повертається в ModelingCore, який за допомогою GUI інформує користувача про успіх побудови структури мережі (або інформує про помилки в синтаксисі, у виразах алгебри процесів тощо). Після цього ModelingCore створює APRONetConfigurator і передає йому раніше побудований APRONet. Конфігуратор створює APRONetConfigurationGUI — для прийому від користувача параметрів конфігурації (області моделювання, параметрів методу тощо.). На підставі цієї інформації конфігуратор встановлює внутрішні параметри вершин в APRONet. Відразу створюються кінцеві екземпляри класів реального робочого навантаження і приєднуються до вершин мережі.

Після цього готова мережа передається в APRONetModelExecutor — набір класів, що забезпечують

виконання моделі. Інтерфейс `ModelExecutorGUI` — інтерфейс кваліфікованого користувача для тонкого налаштування параметрів системи виконання моделі. Інтерфейс `ModelResultGUI` призначений для відображення результатів моделювання.

Отже, інтерфейс `RealCalculationNode` — окремий пакет, який виконується на фізичному вузлі кластера. Фактично, екземпляри програм, створених на основі даного пакета, передаються в чергу на кластер і розміщуються по фізичних вузлах відповідно до інструкцій локального менеджера ресурсів. Кожний `RealCalculationNode` має зв'язок за допомогою TCP-сокета, (можливо, через PVM/MPI) з `Dispatcher`. Пакет `Dispatcher` лежить в основі програми, запущеної на головному вузлі (`MasterNode`). `Dispatcher` відіграє роль посередника між обчислювальними вузлами й `APROModelExecutor`. Пакет `APROModelExecutor` системи моделювання передає в `Dispatcher` готову до виконання мережу (шляхом `java serialized classes`), коли складні структури можна перетворювати в комплект бінарних даних, придатних для передачі по мережі. `Dispatcher` розподіляє віртуальні вузли мережі по реальних вузлах `RealCalculationNode`. Пакети `RealCalculationNode` для цього створюють внутрішні класи `VirtualCalculationNode`, які інкапсулюють переходи APRO- мережі. `InternalDataNode` інкапсулюють вершини, спільні для переходів усередині реального вузла, `ExternalDataNode` інкапсулюють вершини, пов'язані з переходами в інших вузлах, і тому вимагають мережної передачі даних. `VirtualCalculationNode` виконують робоче навантаження, `RealCalculationNode` перерозподіляє дані між `InternalDataNode`, а також відповідає за передачу даних між своїми `ExternalDataNode` і `ExternalDataNode` інших вузлів (або через диспетчер, або через прямі канали, встановлені за допомогою диспетчера). Результати моделювання передаються диспетчеру, а через нього — в `APROModelExecutor`, який їх відображає графічно. Після завершення обчислень `RealCalculationNode` завершують роботу й звільняють чергу кластера.

5.2.2.1. Підсистема генерації моделі

Програмна система моделювання складається з ряду підсистем, однією з яких є підсистема генерації моделі, UML-діаграма класів якої показана на рис.5.7.

Клас `ModelingSystemCore`, що є головним класом показаного на рис.5.6 пакета `ModelingCore`, створюється першим при запуску програмної системи і знищується останнім при її завершенні. Об'єкт, породжений даним класом, створює дочірній об'єкт `MainWindow` на основі класу, що входить в пакет `MainGUI`, є нащадком стандартного Java-вікна і включає елементи головного інтерфейсу. Метод `+onNetBuildButtonClicked()` обробника подій створює об'єкт типу `APRONetDirector`, який, в свою чергу, створює дочірній об'єкт-вікно `APRONetDirectorWindow`, що є редактором тексту з меню керування та кнопками “Компілювати”, “Відкрити”, “Зберегти” тощо. Застосовано шаблон проектування “будівельник”, який проводить поділ функцій між об'єктом `Director`, відповідальним за зовнішню логіку побудови складної структури, і об'єктом `Builder`, відповідальним за її внутрішню логіку.

`APRONetDirector` може побудувати мережу з вихідного тексту по команді “build”, або відтворити раніше побудовану мережу зі збереженого XML-файлу.

У першому випадку директор створює `APRONetPABuilder` і передає в метод `+buildAPRONet` текст програми. У другому випадку `APRONetXMLBuilder` передає в метод `+buildAPRONet` текст XML-файлу. Після цього виконується процедура `+returnAPRONet()`. Оскільки обидва класи є спадкоємцями одного абстрактного класу `APRONetTextBuilder`, то для класу `APRONetDirector` їх поведінка абсолютно однакова. Головна функція цих підкласів полягає в передачі їм скрипта опису структури моделі та поверненні побудованої мережі. `APRONetPABuilder` — це головний клас, який виконує функції керування процесом побудови мережі за вхідним текстом. Його головний підклас `AbstractTextParser` є абстрактним класом перетворення тексту моделі в дерево розбору.

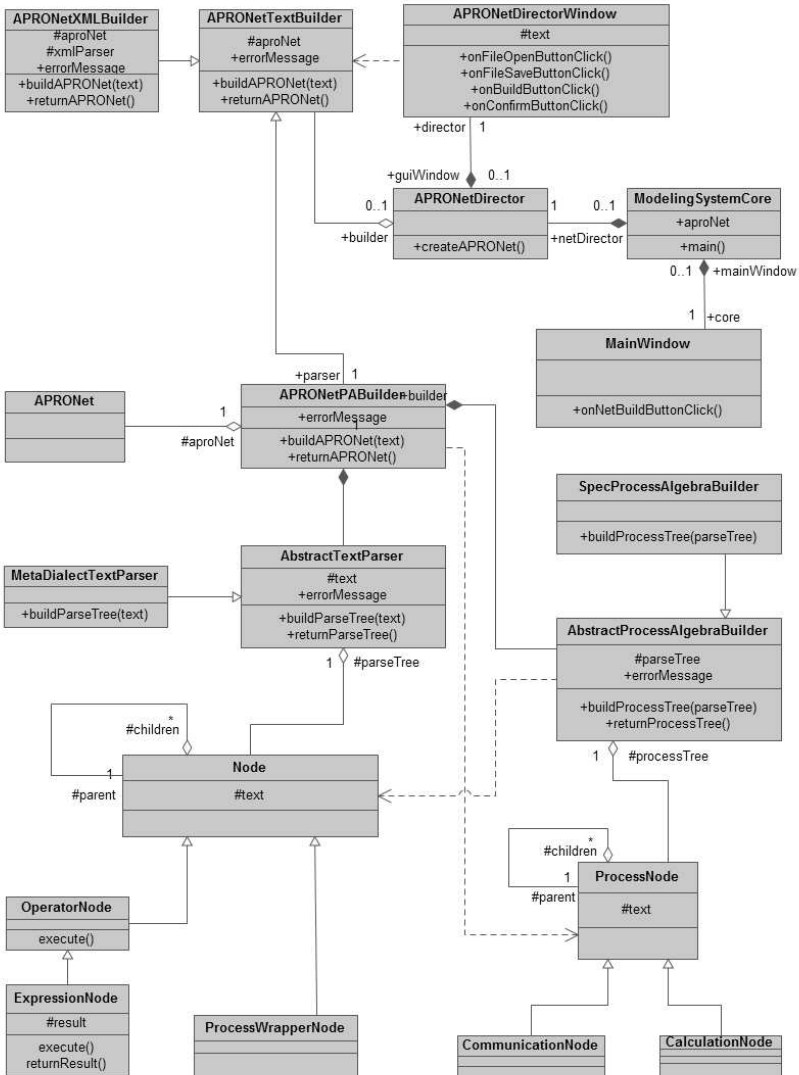


Рис.5.7. UML-діаграма класів підсистеми генерації моделі

`MetaDialectTextParser` — конкретна реалізація цього розбору. У випадку створення моделі у відповідності з виразами алгебри процесів створюється абстрактний клас `AbstractProcessAlgebraBuilder`, який перетворює дерево розбору в дерево процесів алгебри процесів. Конкретна реалізація цього класу `SpecProcessAlgebraBuilder` використовує операції алгебри процесів, описані в даній роботі. Для формування структури дерева розбору для мережі використано шаблон “контейнер”, структура якого полягає у тому, що кожен елемент може відігравати роль контейнера для подібних йому елементів. Отже, елемент дерева розбору — це абстрактний клас `Node`, конкретні реалізації якого — це вершини `OperatorNode`, представлені `ExpressionNode`. Подібним чином організована структура формування дерева розбору процесів, абстрактним класом якої виступає `ProcessNode` з конкретними реалізаціями `CommunicationNode` та `CalculationNode`. Пакет `ProcessAlgebraBuilder` містить множину класів, які забезпечують процес лексичного та синтаксичного розбору. Загальна схема роботи лексичного аналізатора така: спочатку виділяється окрема лексема, а потім ключові слова в аналізаторі розпізнаються явним виділенням безпосередньо з тексту. Якщо виділена лексема є обмежником, то вона видається як результат лексичного аналізу. Якщо виділена лексема є ключовим словом, то видається ознака відповідного ключового слова. Якщо виділена лексема є ідентифікатором — видається ознака ідентифікатора, а сам ідентифікатор зберігається окремо. Нарешті, якщо виділена лексема належить будь-якому з інших класів лексем (наприклад, лексема представляє собою число, рядок, тощо), то видається ознака відповідного класу, а значення лексеми зберігається окремо.

В процесі розбору будується дерево розбору програми з базових блоків, що є виконавчими класами програми. Кожен клас містить процедуру, які відповідає елементарній операції мови. Обхід дерева розбору з послідовним виконаннями коду кожного з класів дерева і є виконанням програми. Побудова

дерева розбору реалізована таким чином, що абстрактні вершини дерева, або вершини, що мають лише одного нащадка, не відображаються у дереві. Це дає можливість значно зменшити розмір дерева і дещо збільшити ефективність виконання програм інтерпретатором.

Якщо вхідний текст програми повністю відповідає граматиці мови, відбувається виконання програми. Виконання програми здійснюється в стандартний спосіб, в порядку слідування операторів програми. Розгалуження і цикли працюють аналогічно до відповідних конструкцій більшості мов програмування високого рівня. Арифметичні, логічні, побітові операції виконуються стандартним чином відповідно до їх старшинства. При обчисленні виразів автоматично виконуються необхідні перетворення типів змінних. Змінна ініціюється під час виконання першого оператора за її участі. В кожному виразі використовується своя копія змінної.

Ключовими операторами мови є оператор присвоєння алгебри процесів і оператор задавання робочого навантаження. Оператор задавання робочого навантаження ставить у відповідність певній активності в термінах алгебри процесів конкретну процедуру на мові Java і задає для неї вхідні аргументи. Оператор присвоєння алгебри процесів додає до списку описів процесів відповідний опис процесу з лівої частини виразу. І тому результатом роботи будь-якої програми на цій мові є список виразів розширеної алгебри процесів. Саме цей список і є досліджуваною моделлю та використовується в подальшому підсистемою імітаційного моделювання. Цикли, розгалуження, змінні і операції мови дозволяють задавати складні ієрархічні описи процесів, що включають значну кількість компонентів. Потрібно відзначити, що оператори алгебри процесів в мові є своєрідними «шаблонами» виразів алгебри процесів. У ході виконання програми шаблони наповнюються змістом — це відбувається в той момент, коли ім'я процесу в суфіксній формі перетворюється на звичайне строкове ім'я в залежності від значень змінних, що входять у

суфікс. Групові операції також виконуються і «розкриваються» на етапі виконання програми. Успішними програмами можна вважати такі, що досягають мети — тобто завершують виконання після формування одного або більше виразів алгебри процесів. Програми, що містять «вічні» цикли або не формують вирази алгебри процесів, не мають змісту.

Після того, як формування структури завершено, модель (список виразів алгебри процесів) готова до виконання. Початковою точкою моделювання є процес з назвою «SYSTEM». Процес з такою назвою обов'язково повинен бути заданий у робочому навантаженні моделі.

5.2.2.2. Підсистема вузлових обчислень

Результатом роботи підсистеми генерації моделі програмної системи моделювання є структура APRO-мережі, яка лежить в основі організації вузлових обчислень.

UML-діаграма класів підсистеми вузлових обчислень показана на рис.5.8.

Ядро програмної системи моделювання `ModelingSystemCore` включає контейнер `APRONet`, який містить масив вузлів типу `AbstractAPRONode`. Елементи `AbstractAPRONode` представлено сукупністю вершин дводольного графа APRO-мережі і деталізуються як об'єкти переходів `APROCalcNode` та об'єкти позицій `APROComNode`.

Переходи `APROCalcNode` містять два масиви: масив вхідних позицій `+inNodes` і масив вихідних позицій `+outNodes`. Методи `+pullTokens()` і `+pushTokens()` призначені для зняття міток із вхідних позицій у поточний перехід та розміщення вихідних міток на вихідних позиціях. Ці методи можуть викликатись всередині методу `+execute()`, який формує обчислювальний процес в переході, займається просуванням модельного часу та виконує реальне робоче навантаження.

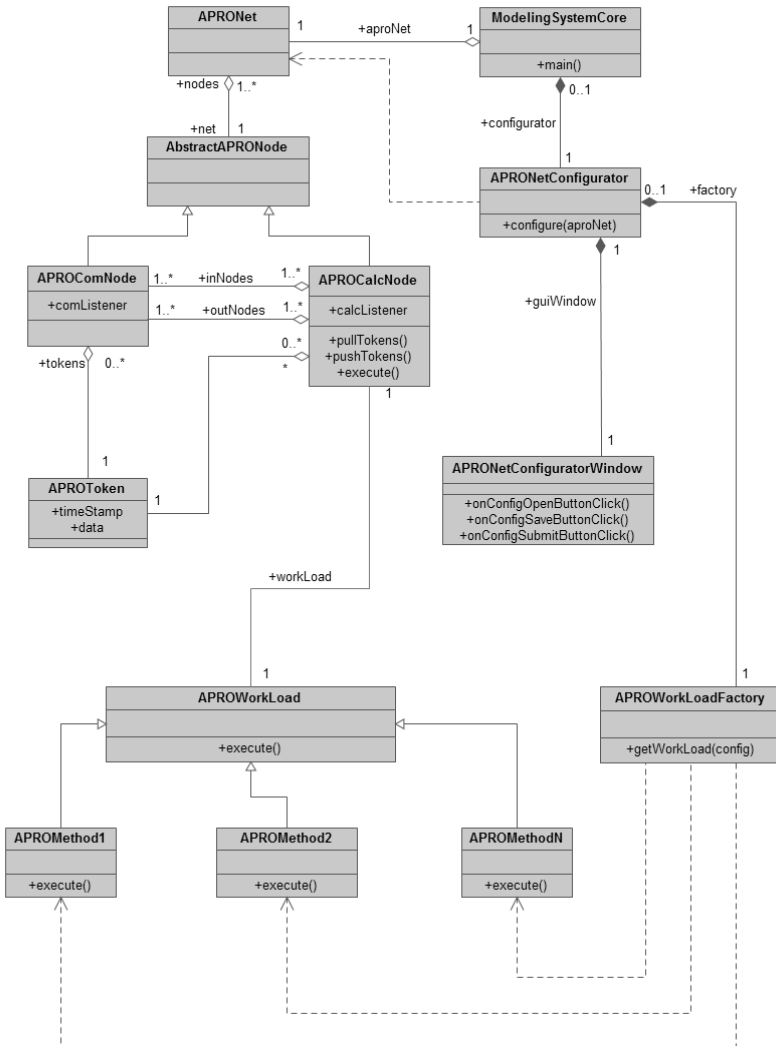


Рис.5.8. UML-діаграма класів підсистеми вузлових обчислень
 Абстрактний клас APROWorkLoad з абстрактним методом

+execute() призначений для забезпечення основних функціональних властивостей обчислювального процесу у відповідності з чисельними методами. Спадкоємці даного класу перевизначають його з метою формування конкретного обчислювального алгоритму. Структуру взаємодії представляє клас APROComNode, який асоціюється з класом APROToken, що інкапсулює методи та структури даних мітки.

APROComNode формує черги міток шляхом заповнення поля +data даними зі знову прибулих міток і викликає метод +execute() класу APROWorkLoad. Клас APRONetConfigurator містить інструменти налаштування обчислювального методу та середовища його виконання. Такі налаштування можуть бути здійснені за допомогою інтерфейсу APRONetConfiguratorWindow.

До елементів даного інтерфейсу входить зокрема редактор границь, який дозволяє задати параметри тривимірної області обчислень та параметри навчання кліткової нейронної мережі, формально описаної APRO-мережею. Формування конкретного робочого навантаження відбувається шляхом створення класу APROWorkLoadFactory, який є реалізацію шаблону Factory. На його вхід подаються параметри, клас на підставі цих параметрів приймає рішення про те, який саме екземпляр робочого навантаження створити, а потім повертає створений екземпляр. Шаблон зручний тим, що за його допомогою всю логіку по роботі з зовнішніми класами можна відокремити від інших елементів системи. Таким чином, на виході роботи підсистеми вузлових обчислень одержуємо об'єкт APRONet, у якому сформована раніше структура APRO-мережі поєднується з реальним робочим навантаженням.

5.2.2.3. Підсистема виконання моделі

Ще однією важливою підсистемою програмної системи моделювання є підсистема виконання моделі. UML-діаграма класів даної підсистеми показана на рис.5.9.

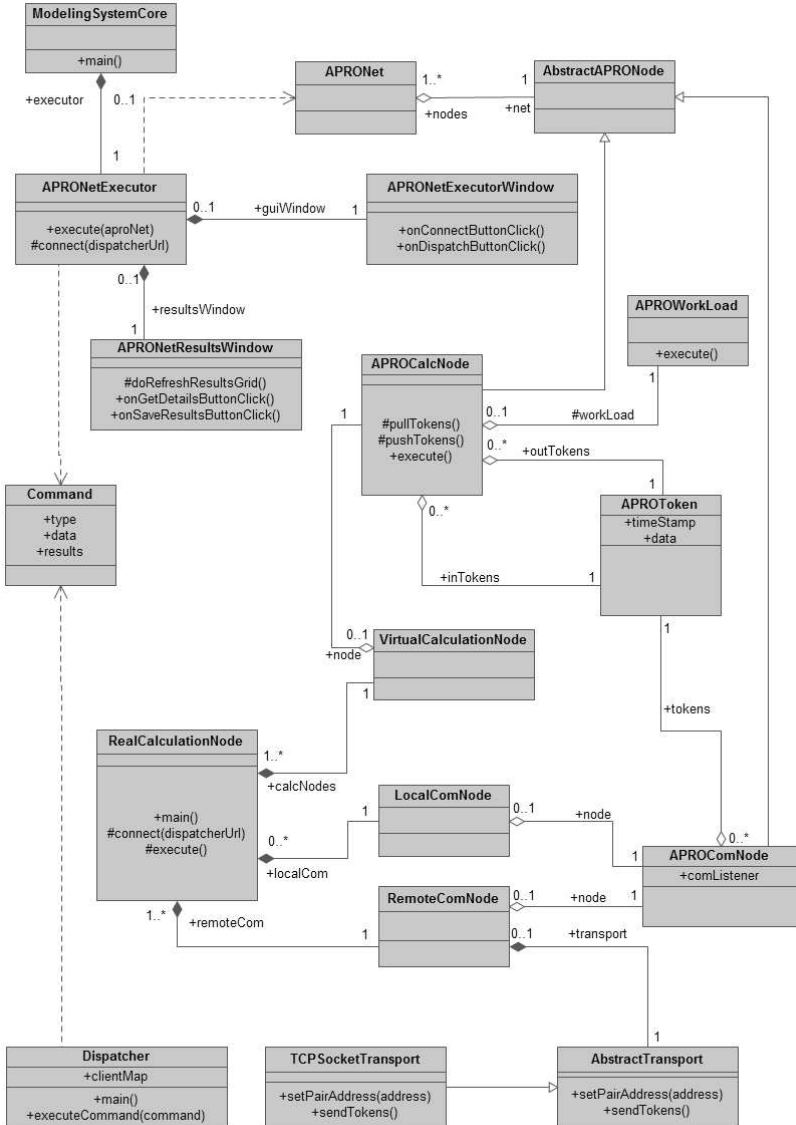


Рис.5.9. UML-діаграма класів підсистеми виконання моделі

Як і в попередньому випадку, підсистема виконання моделі створюється на основі класу `ModelingSystemCore`, що є базовим класом програмної системи моделювання. Умовою створення є успішне завершення процесу генерації моделі. Операція `+executor` породжує клас `APRONetExecutor` та передає його методу `+execute()` структуру `aproNet` як параметр. `APRONetExecutor` утворює композицію інтерфейсів `APRONetExecutorWindow` та `APRONetResultsWindow`, які забезпечують основні функції керування процесом виконання моделі та відображення результатів моделювання. За допомогою `Command` клас `APRONetExecutor` підтримує взаємодію з окремою програмою, в основі якої лежить клас `Dispatcher`.

Програма `Dispatcher` запускається на головному вузлі кластера та виконує функції посередника між головною частиною програмної системи моделювання з системою інтерфейсів та множиною локальних програм, розміщених на реальних вузлах кластера.

Головними функціями диспетчера є оптимальний розподіл фрагментів APRO-мережі по обчислювальних вузлах та збір інформації про хід обчислень. Програма, яка підтримує обчислювальний процес на реальному вузлі кластера, будується на основі контейнера `RealCalculationNode`. Базовою структурою, що містить множину віртуальних обчислювальних вузлів `VirtualCalculationNode`, є масив `+calcNodes`. Кожний обчислювальний `VirtualCalculationNode` — це контейнер для сукупності класів, що описують роботу переходу APRO-мережі. Масив `+localCom` містить множину позицій внутрішніх переходів, тобто множину позицій `LocalComNode`, в яких не потрібна мережна активність для розміщення міток.

Масив `+remoteCom` містить множину позицій `RemoteComNode`, що відповідають критерію сусідства для множини внутрішніх переходів, але фізично розміщені на інших реальних вузлах кластера. Тому для розміщення міток на таких позиціях необхідно забезпечити їх передачу через мережу. Такі відмінності між позиціями приховані від нижчих класів

обчислювальної ієрархії. Конкретні переходи, що представлені класом `APRONet`, викликають методи `+pullTokens()` і `+pushTokens()`, але джерело одержання цих методів абстраговано внутрішньою логікою `RealCalculationNode`. `AbstractTransport` — клас, що надає єдиний інтерфейс для мережної взаємодії з іншими екземплярами `RealCalculationNode` і може використовуватись при різних способах взаємодії шляхом застосування конкретних реалізацій. Клас `TCP SocketTransport` представляє конкретну реалізацію, засновану на звичайних TCP-сокетах. Всі необхідні TCP/IP адреси необхідних вузлів `TCP SocketTransport` одержує від `Dispatcher`. Клас `APROComNode` представляє конкретну реалізацію позиції APRO-мережі. Його поле `+comListener` — це клас, який представляє конкретну реалізацію шаблону `Listener`. Такий підхід дозволяє надати попередньо сформованій APRO-мережі нову функціональність. Методи `+pullTokens()` і `+pushTokens()` спочатку викликають такі ж методи з об'єкта `+comListener` і тільки потім виконують свій власний код. В той же час, в об'єкт `+comListener` встановлюються параметри реальної конфігурації на етапі розгортання `RealCalculationNode`.

5.2.3. Інтерфейс програмної системи моделювання

Адаптивний інтерфейс реалізує ряд функцій. Перший вид включає функції, які необхідні для роботи з інформаційними файлами, що містять дані про робоче навантаження та структуру нейронної мережі. Другий вид об'єднує функції редагування структури нейронної мережі та робочого навантаження. До третього виду відносять функції керування моделлю.

Безпосередньо після завантаження програмної системи моделювання доступними є тільки функції першого виду. Тому початковою дією користувача завжди є завантаження уже сформованої моделі, яка зберігається у відповідному файлі, або створення нової моделі за допомогою спеціального механізму генерації. Перший етап генерації моделі полягає у виборі

параметрів вимірності простору, в якому буде розв'язуватись крайова задача. Ці параметри вибираються, виходячи з того, що комплекс обчислень, який відповідає одній точці дискретизації у просторі, ототожнюється з одним цифровим нейроном. Створення простору відкриває можливість його редагування, яке полягає у формуванні границь розв'язування крайової задачі. Процес генерації сітки може підлягати редагуванню за допомогою редактора границь (рис.5.10).

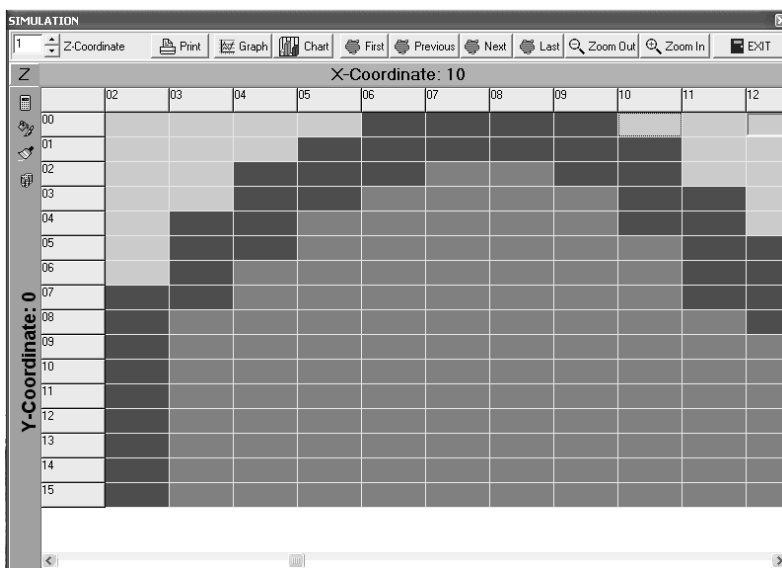


Рис.5.10. Редактор границь

Редагування границь зводиться до пошарової розмітки границь області. Кожна точка простору може бути внутрішньою, граничною або зовнішньою.

Введення цих ознак відбувається шляхом розфарбовування певних зон простору відповідним кольором.

Хід процесу моделювання відображається у тривимірному вигляді, як показано на рис.5.11.

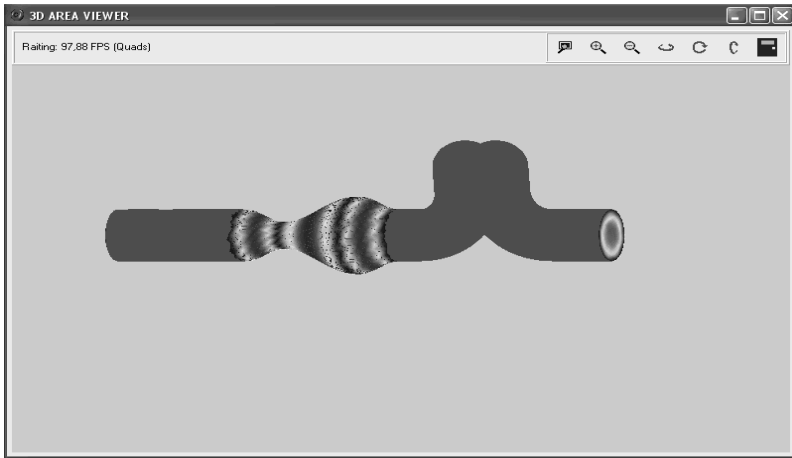


Рис.5.11. Вікно графічного відображення результатів моделювання

Головне вікно керування програмною системою моделювання (рис.5.12) містить інтерактивні можливості задавання початкових значень та констант моделі.

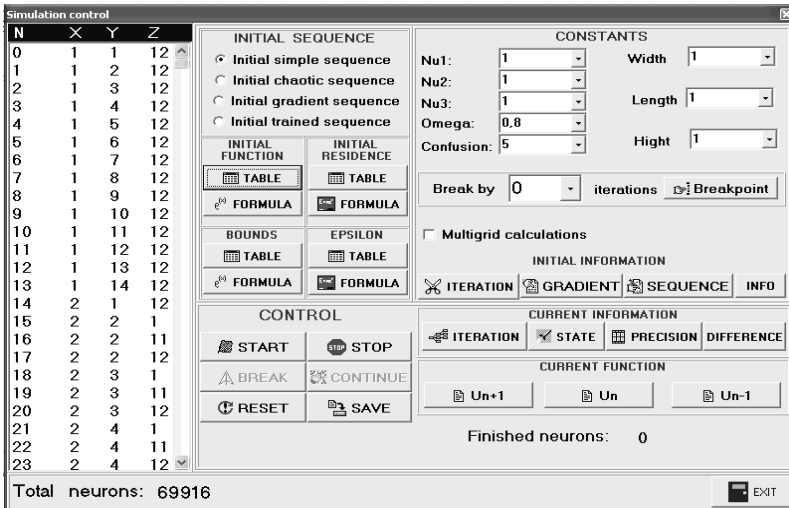


Рис.5.12. Головне вікно керування програмним комплексом

Початкові значення функцій можуть бути задані в табличному або аналітичному вигляді. Серед констант моделі найголовнішими є фізичні розміри простору та константи ітераційного процесу. Порядок запуску нейронів також може підлягати модифікації. Прийнято такі алгоритми запуску нейронів:

- проста послідовність, коли порядок запуску відповідає глобальному номеру нейрона в клітковій мережі;
- хаотична послідовність, для якої порядок запуску визначається випадковою величиною з нормальним законом розподілу;
- градієнтна послідовність, що визначає порядок запуску за вектором градієнта, колінеарного до нормалі границі;
- спеціальна послідовність, яка є результатом навчання нейронної мережі.

Значення помилки, при досягненні якої відбувається зупинка ітераційного процесу, може бути встановлене індивідуально для кожного нейрона. Як і для початкових значень функції, помилка задається таблицею або аналітично.

В ході моделювання відображається інформація про загальну кількість нейронів, які беруть участь в обчисленнях, поточна інформація про кількість пройдених часових шарів та про кількість нейронів, які досягли бажаної точності на поточному часовому шарі. Існує можливість графічного та табличного перегляду поточного стану моделі. Доступна інформація про значення невідомих змінних, які були одержані послідовно на трьох останніх ітераціях. В ході моделювання можна одержати інформацію про поточну кількість ітерацій, проведених кожним з нейронів, про стан нейрону та про досягнуту точність. При відомому точному аналітичному розв'язанні крайової задачі існує можливість обчислення різницевої поверхні між точним та наближеним розв'язком.

Моделювання може проводитись з використанням тільки локально-асинхронного методу на одній різницевій сітці,

а також з підключенням багатосіткового методу з переходами між сітками з кроками, які кратні степеням двійки.

Для режиму настройки алгоритму розв'язання крайової задачі механізм керування моделлю дозволяє зупинити процес моделювання при досягненні заданої ітерації, на заданому номері нейрона, при досягненні заданого значення функції тощо.

За допомогою даного програмного комплексу реалізовано ряд математичних моделей складних фізичних процесів, найважливішою з яких є модель перистальтичних процесів у травному тракті. Для практичного визначення ефективності створеного програмного комплексу при реалізації процесу моделювання перистальтичних процесів у травному тракті людини застосовано WEB-систему з відкритим кодом Cacti. Систему Cacti було налаштовано на збір статистичної інформації про завантаження всіх ядер процесорів, що входять до складу вузлів кластера. Результати досліджень представлено у вигляді графіків, побудованих за допомогою набору утиліт PRDTool. Вимірювання завантаження процесорних ядер проводились на кластерній системі Інституту математики НАН України simulator.imath.kiev.ua, які підтвердили середнє завантаження на рівні 80%-87%.

СПИСОК ЛИТЕРАТУРИ

1. Буч Г., Якобсон А., Рамбо Дж. UML. Классика CS.– СПб.: Питер, 2006.– 736 с.
2. Ильин В.В. Моделирование бизнес-процессов. Практическое использование ARIS.– М.: Издательский дом «Вильямс», 2006.– 176 с.
3. Глушков В.М., Гусев В.В., Марьянович Т.П., Сахнюк М.А. Программные средства моделирования непрерывно-дискретных систем.– К.: Наукова думка, 1975.– 152 с.
4. Бусленко Н.П., Калашников В.В., Коваленко И.Н. Лекции по теории сложных систем.– М.: Сов.радио, 1973.– 439 с.
5. Бусленко Н.П. Моделирование сложных систем.– М.: Наука, 1978.– 400 с.
6. Молчанов А.А. Моделирование и проектирование сложных систем.– К.: Вища школа, 1988.– 359 с.
7. Шеннон Р. Имитационное моделирование систем — искусство и наука.– М.: Мир, 1978.– 418 с.
8. Феррари Д. Оценка производительности вычислительных систем.– М.: Мир, 1981.– 576 с.
9. Derrick E.J. A Visual Simulation Support Environment Based on Multifaceted Conceptual Framework.– Ph.D. Dissertation, Department of Computer Science, Verginia Tech, Blacksburg: 1992.– 268 p.
10. Хоар Ч. Взаимодействующие последовательные процессы.– М.: Мир, 1989.– 264 с.

11. Milner R. Calculus of Communicating Systems // Lecture Notes in Computer Sciences.– 1980.– Vol.92.– 260 p.
12. Baeten J.C.M., Weijland W.P. Process Algebra (Cambridge Tracts in Theoretical Computer Science).– Cambridge: Cambridge University Press, 1991.– 248 p.
13. Котов В.Е. Сети Петри.– М.: Наука, 1984.– 160 с.
14. Питерсон Дж. Теория сетей Петри и моделирование систем.– М.: Мир, 1984.– 264 с.
15. David R., Alla H. Petri Nets and Grafcet. Tools for Modeling Discrete Event Systems.– New York: Prentice Hall, 1992.– 339 p.
16. Nance R.E. A History of Discrete Event Simulation Programming Languages // ACM SIGPLAN Notices.– 1993.– Vol.28, №3.– P. 149-175.
17. Гусев В.В., Марьянович Т.М., Сахнюк М.А. Система программирования НЕДИС: В 2-х ч.–К.: Ин-т кибернетики АН УССР, 1975.–ч. I.–205 с.
18. Гусев В.В., Марьянович Т.М., Сахнюк М.А. Система программирования НЕДИС: В 2-х ч.–К.: Ин-т кибернетики АН УССР, 1975.–ч. II.– 189 с.
19. Глушков В.М., Калиниченко П.А., Марьянович Т.П. и др. СЛЭНГ — система программирования для моделирования дискретных систем.– К.: Ин-т кибернетики АН УССР, 1969.– 469 с.
20. Шрайбер Т. Дж. Моделирование на GPSS.– М.: Машиностроение, 1980.– 592 с.
21. Balci O. Requirements for Model Development Environments // Computers and Operations Research.–1986.– Vol.13, №1.– P. 53-67.

22. Железнов И.Г. Сложные технические системы.– М.: Высшая школа, 1984.– 119 с.
23. Малютов М.Б., Заиграев А.Ю. Современные задачи оптимального планирования регрессионных экспериментов.– К.: Вища школа, 1989.– 64 с.
24. Ермаков С.М. Математическая теория планирования эксперимента.– М.: Наука, 1983.– С. 35-51.
25. Клейнен Дж. Статистические методы в имитационном моделировании: В 2-х т.– М.: Мир, 1978.– Т.1.– 222 с.
26. Клейнен Дж. Статистические методы в имитационном моделировании: В 2-х т.– М.: Мир, 1978.– Т.2.– 395 с.
27. Аврамчук Е.Ф., Вавилов А.А., Емельянов С.В. Технология системного моделирования.– М.: Машиностроение, 1988.– 520 с.
28. Советов Б.Я., Яковлев С.А. Моделирование систем.– М.: Высшая школа, 2001.– 343 с.
29. Баранов С.И. Синтез микропрограммных автоматов.– Л.: Энергия, 1979.– 232 с.
30. Hartmanis J., Stearns R. Algebraic structure theory of sequential machines.– New-York: Prentice-Hall, 1966.– 322 p.
31. Бусленко Н.П., Коваленко И.Н. О математическом описании элементов сложных систем // Докл. АН СССР.– 1969.– Т.187, №6.– С. 1222-1224.
32. Zeigler B.P. Theory of Modeling and Simulation.– New York: Wiley, 1976.– 435 p.
33. Cellier F.E., Wang Q., Zeigler B.P. A Five Level Hierarchy for the Management of Simulation Models // Proc. of the Winter Simulation Conf.– New Orleans: IEEEExplore.– 1990.– P. 55-64.

34. Rozenblit J. W., Zeigler B.P. Representing and Constructing System Specifications Using the System Entity Structure Concepts // Proc. of the Winter Simulation Conf.– Los Angeles: ACM.– 1993.– P. 604-611.
35. Schruben L. Simulation Graphical Modeling with Event Graphs // Communications of the ACM.– 1983.– Vol.26, №11.– P. 957-963.
36. Schruben L. SIGMA: Graphical Simulation Modeling.– San Francisco: The Scientific Press, 1992.– 309 p.
37. Yücesan E., Schruben L. Structural and Behavioral Equivalence of Simulation Models // ACM Transactions on Modeling and Computer Simulation.– 1992.– Vol.2, №1.– P. 82-103.
38. Zeigler B.P. Theory of Modeling and Simulation.– New York: John Wiley and Sons.– 1976.– 435 p.
39. Murata T. Petri Nets: Properties, Analysis and Applications // Proceedings of the IEEE.– 1989.– Vol.77, № 4.– P. 541-580.
40. Ramchandani C. Performance Evaluation of Asynchronous Concurrent Systems by Timed Petri Nets // PhD thesis.– Massachusetts Institute of Technology, Cambridge.– 1973.– 148 p.
41. Sifakis J. Use of Petri Nets for Performance Evaluation // Measuring, Modeling and Evaluating Computer Systems.– North-Holland.– 1977.– P.75-93.
42. David R., Alla H. Petri Nets and Grafcet — Tools for Modelling Discrete Event Systems.– New York: Prentice Hall, 1992.– 339 p.
43. Nutt G.J. Evaluation nets for computer system performance analysis // 1972 Fall Joint Computer Conference, AFIPS Conference Proceedings.– 1973.– Vol.41.– P. 279-286.

44. Noe J.D. Nets in modeling and simulation: Lect. Notes Comput. Sci.– 1980.– Vol.84.– P. 347-368.
45. Plotkin G.D. A Structured Approach to Operational Semantics: Technical Report DAIMI FM-19 // Computer Science Department, Aarhus University.– 1981.– 133 p.
46. Larsen K.,Skou A. Bisimulation through Probabilistic Testing // Information and Computation.– 1991.– Vol.94, №1.– P. 1-28.
47. Bochmann G.V., Vaucher J. Adding Performance Aspects to Specification Languages // Protocol Specification, Testing and Varification.– 1988.– Vol.8.– P.19-31.
48. Harvey C. Performance Engineering as an Integral Part of System Design // BT Technology Journal.– 1986.– Vol.4, №3.– P. 143-147.
49. Hilston J. A compositional Approach to Performance Modelling.– Cambridge: Cambridge University Press.– 1996.– 168 p.
50. Нестеренко Б.Б., Новотарский М.А. Имитационные модели параллельных асинхронных методов математической физики.– Киев, 1997.– 52 с.– (Препр. / НАН Украины. Ин-т математики; 97-14).
51. Meseguer J. On Semantics of Petri Nets / J.Meseguer, U.Montanari, V.Sassone // Lecture Notes in Computer Science.– 1992.– Vol. 630.– P. 286-301.
52. Fujimoto R.M. Parallel Discrete Event Simulation / R.M. Fujimoto // Communication of the ACM.– 1990.– Vol. 33, №10.– P. 30-53.

53. Fayad E.M., Schmidt D.C. Object-Oriented Application Frameworks // Communications of the ACM.– 1997.– Vol. 40, №10.– P. 32-38.
54. Chande K.M., Misra J. Distributed Simulation: A Case of Study in Design and Verification of Distributed Programs // IEEE Trans. on Software Engineering.– 1979.– Vol. 5, №5.– P. 440-452.
55. Jefferson D. Virtual time // ACM Transactions on Programming Languages and Systems.– 1985.– Vol. 7, №3.– P. 405-425.
56. Valiante G. Algorithms on Trees and Graphs.– Berlin: Springer, 2002.– 450 p.
57. Hack M.H.T. Decidability Questions for Petri Nets: Thesis Ph.D.– Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science.– 1976.– 196 p.
58. Jancar P. Decidability Questions for Bisimilarity of Petri Nets and Some Related Problems: Technical Report ECS-LFCS-93-261 // Dept. of Computer Science, University of Ostrava.– Ostrava, 1993.– 24 p.
59. Glabbeek R. J. The Linear Time – Branching Time Spectrum // Lecture Notes In Computer Science.– 1990.– Vol. 458.– P. 278-297.
60. Milner R. Communication and Concurrency.– Prentice Hall, 1995.– 272 p.
61. Bergstra J.A., Middelburg C.A. Process algebra for hybrid systems // Theoretical Computer Science.– 2005.– Vol. 335, №2/3.– P. 215-208.
62. Bergstra J.A., Middelburg C.A., Stefanescu Ch. Network algebra for asynchronous dataflow // International Journal of Computer Mathematics.– 1997.– Vol. 65.– P. 57-88.

63. Hermanns H., Herzog U., Mertsiotakis V. Stochastic Process Algebras as a Tool for Performance and Dependability Modeling // Proceedings of IEEE International Computer Performance and Dependability Symposium.– 1995.– P. 102-111.
64. Glynn P.W. A GSMP formalism for discrete event simulation // Proceedings of the IEEE.– 1989.– Vol. 77, №1.– P. 14-23.
65. Groote J.F., Vaandrager F. Structured operational semantics and bisimulation as a congruence // Information and Computation.– 1992.– Vol. 100, №2.– P. 202-260.
66. De Nicola R. Extensional Equivalence for Transition Systems // Acta Informatica.– 1987.– Vol. 24, №2.– P. 211-237.
67. Main M. Trace, failure and testing equivalences for communicating processes // International Journal of Parallel Programming.– 1987.– Vol. 16, №5.– P. 383-400.
68. Chua L.O., Yang L. Cellular Neural Networks. Theory // IEEE Transactions on Circuits and Systems.– 1988.– Vol. 35, №10.– P. 1257-1272.
69. Chua L.O., Roska T. The CNN paradigm // IEEE Transactions on Circuits and Systems.– 1993.– Vol. 40, № 3.– P. 147-156.
70. Espejo S., Carmona R. A VLSI-Oriented Continuous-Time CNN Model // International Journal of Circuit Theory and Applications.– 1996.– Vol. 24, №3, P. 341-356.
71. Keresztes P., Zaràndy Á. An emulated digital CNN implementation // J. VLSI signal processing systems for signal, image and video technology.– 1999.– Vol. 23.– P. 291-303.
72. Duncan R. A survey of parallel computer architectures / R. Duncan // Computer.– 1990.– Vol. 23.– №2.– P. 5-16.

-
73. Евреинов Э.В., Косарев Ю.Г. Однородные универсальные вычислительные системы высокой производительности.– Новосибирск.: Наука, 1978.– 320 с.
 74. Shepherd G.M. Neurobiology.– New York: Oxford University Press.– 1994.– 784 p.
 75. Новотарський М. А. Паралельні асинхронні методи та засоби моделювання перистальтичних процесів : дис. ... д-ра техн. наук.: 01.05.02 – математичне моделювання та обчислювальні методи.– К., 2010.– 403 с.
 76. Hecht F. Freefem++.–Paris: UPMC-LJLL Press.– 2005.– 304 p.
 77. Demuth H., Beale M. Neural Network Tool Box: For use with MATLAB.–Natick:The MathWork, Inc.–2005.–846 p.
 78. Henshaw W.D., Schwendeman D.W. Parallel computation of three-dimensional flows using overlapping grids with adaptive mesh refinement // J. Computational Physics.– 2008.– Vol. 227, №16.– P. 7469-7502.

ЗМІСТ

ПЕРЕДМОВА	5
Розділ 1	
Загальні питання дослідження складних систем і процесів	17
1.1. Процеси та системи як об'єкт дослідження.....	17
1.2. Моделі як інструменти дослідження.....	20
1.3. Схема класифікації моделей.....	22
1.4. Життєвий цикл моделі.....	25
1.5. Основні проблеми побудови сучасних математичних моделей.....	34
Розділ 2.	
Огляд формальних засобів опису моделей	37
2.1. Системи з дискретними подіями.....	37
2.2. Проблеми подібності моделей.....	40
2.3. Особливості життєвого циклу моделі з дискретними подіями.....	42
2.4. Формальні засоби опису моделей з дискретними подіями.....	44
2.4.1. Скінченний автомат.....	44
2.4.2. Кусково-лінійний агрегат.....	46
2.4.3. Специфікації систем з дискретними подіями.....	47
2.4.4. Графічні формальні засоби.....	50
2.5. Мережні засоби опису моделей.....	52
2.5.1. Мережі Петрі.....	53
2.5.1.1. Класи мереж Петрі.....	59
2.5.1.2. Мережі Петрі з урахуванням часу.....	61
2.5.2. Е-мережі.....	65
2.5.3. PRO-мережі.....	71
2.6. Алгебри процесів.....	75
2.6.1. Стохастичні алгебри процесів.....	76
2.6.2. Алгебри процесів з дискретними подіями.....	79

Розділ 3.

APRO-мережі	83
3.1. Структура APRO-мережі.....	83
3.2. Принципи функціонування APRO-мережі.....	94
3.2.1. Алгоритмічні аспекти функціонування APRO-мережі.....	99
3.2.1.1. Активація переходу.....	109
3.2.1.2. Робота переходу.....	114
3.2.1.3. Деактивація переходу.....	115
3.2.2. Проблема транзитних пересилок міток.....	120
3.2.3. Просування мережного часу.....	121
3.2.3.1. Методи просування мережного часу з загальним ресурсом.....	122
3.2.3.2. Консервативні методи паралельного просування мережного часу.....	128
3.2.3.3. Оптимістичні методи паралельного просування мережного часу.....	136
3.2.4. Збір статистичної інформації.....	150
3.3. Аналіз APRO-мереж.....	151
3.4. Еквівалентність APRO-мереж.....	158

Розділ 4

Алгебра процесів	163
4.1. Неформальний опис алгебри процесів.....	163
4.2. Синтаксис.....	167
4.3. Семантика алгебри процесів.....	176
4.4. Еквівалентність поведінки.....	184
4.5. Строга взаємна подібність.....	188
4.6. Слабка взаємна подібність.....	221
4.7. Алгоритми визначення еквівалентності.....	243
4.7.1. Прямий алгоритм визначення строгої взаємної подібності.....	243
4.7.2. Прискорений алгоритм визначення строгої взаємної подібності.....	246

4.7.3. Прискорений алгоритм визначення слабкої взаємної подібності.....	249
4.8. Зв'язок алгебри процесів та APRO-мереж.....	253

Розділ 5

Формальний опис програмної системи моделювання.....

5.1. APRO-мережний опис програмної системи моделювання.....	260
5.1.1. Структура моделі узагальненого нейрона та принципи його функціонування.....	260
5.1.2. Принципи функціонування програмної системи моделювання на основі тривимірної кліткової нейронної мережі.....	266
5.2. Розробка програмної системи моделювання.....	299
5.2.1. Розробка мови моделювання.....	300
5.2.1.1. Типи даних, змінні і масиви мови моделювання.....	301
5.2.1.2. Загальні операції та оператори.....	302
5.2.1.3. Операції і оператори алгебри процесів.....	304
5.2.2. Структура програмної системи моделювання.....	305
5.2.2.1. Підсистема генерації моделі.....	309
5.2.2.2. Підсистема вузлових обчислень.....	313
5.2.2.3. Підсистема виконання моделі.....	315
5.2.3. Інтерфейс програмної системи моделювання.....	318

СПИСОК ЛІТЕРАТУРИ.....

Наукове видання

НЕСТЕРЕНКО Борис Борисович
НОВОТАРСЬКИЙ Михайло Анатолійович

**ФОРМАЛЬНІ ЗАСОБИ МОДЕЛЮВАННЯ
ПАРАЛЕЛЬНИХ ПРОЦЕСІВ ТА СИСТЕМ**

Комп'ютерний набір та верстка
С.В. Новотарська

Редактор В.Е. Гонтковська

Підп. до друку 01.10.2012. Формат 60x84/16. Папір тип №1. Офс.друк.
Фіз. друк. арк. 21,08. Ум. друк. арк.. 19,57. Зам. № .

Інститут математики НАН України
01601, Київ 4, ГСП, вул. Терещенківська, 3