

Інститут проблем моделювання в енергетиці ім. Г.Є. Пухова
Національна академія наук України

Інститут проблем моделювання в енергетиці ім. Г.Є. Пухова
Національна академія наук України

Кваліфікаційна наукова
праця на правах рукопису

Сіроткін Олексій Вікторович

УДК 004.942

ДИСЕРТАЦІЯ

**Метод побудови паралельних чисельних моделей динамічних систем на базі
протоколу реактивних потоків**

122 – Комп’ютерні науки

12 – Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів
і текстів інших авторів мають посилання на відповідне джерело

Олексій СІРОТКІН

Науковий керівник: Мохор Володимир Володимирович, член-
кореспондент НАН України, доктор технічних наук,
професор

Київ –2025

АНОТАЦІЯ

Сіроткін О. В. *Метод побудови паралельних чисельних моделей динамічних систем на базі протоколу реактивних потоків.* – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії (PhD) за спеціальністю 122 "Комп'ютерні науки". – Інститут проблем моделювання в енергетиці ім. Г.Є. Пухова Національної академії наук України, Київ, 2025.

Дисертація присвячена розробці методу паралельного моделювання складних систем з використанням парадигми реактивних потоків як протоколу синхронізації загального призначення. Запропонований метод впроваджує структуроване представлення станів динамічних об'єктів через підстани, перехідні функції та графове моделювання, що забезпечує модульність, повторне використання та масштабованість. Особлива увага приділяється побудові узгоджених графів переходів, їх реалізації як обчислювальних графів на логічних процесорах та оптимізації моделювання за допомогою методів згортання та інтерактивних шаблонів push-pull. Практична перевірка продемонстрована на класичній задачі змішування сольових розчинів, що підкреслює доцільність підходу порівняно з традиційними методами паралельного моделювання.

Основними завданнями роботи є проведення комплексного аналітичного огляду існуючих методів та підходів до паралельного моделювання, включаючи Time Warp, акторно-орієнтовані фреймворки та реактивні адаптації HLA, з метою виявлення їхніх обмежень та визначення перспективних напрямків удосконалення; розробка формальної структури для представлення складних динамічних систем через підстани, перехідні функції та узгоджені графи переходів; розробка методів відображення цих графів на обчислювальні графи, реалізовані на реактивних

потоках; вивчення впливу топології графів, стратегій оптимізації та потенційних невідповідностей на точність, масштабованість та продуктивність моделювання; та перевірка запропонованого підходу за допомогою теоретичного аналізу та експериментальних тематичних досліджень, таких як модель змішування сольових розчинів.

Вирішення цих завдань дозволить забезпечити надійне та масштабоване моделювання складних систем в інженерній, енергетичній та інформаційній сферах, особливо в умовах, що вимагають як ефективності, так і стійкості обчислювальної інфраструктури. Використання синхронізації на основі реактивних потоків у сучасних платформах моделювання підвищить модульність, можливість повторного використання та інтерактивність моделей, прискорить швидкість моделювання в розподілених системах та мінімізує ризики невідповідності або збою. Це фактор для розвитку сучасного стану паралельних обчислень та забезпечення надійних інструментів для стабільної роботи критичної інфраструктури та наукових досліджень.

Дисертація складається зі вступу, 5 розділів, загальних висновків, списку літератури та додатків.

У вступі обґрунтовується актуальність теми дослідження, окреслюється науково-технічна проблема паралелізації моделювання складних систем, обґрунтовується доцільність застосування парадигми реактивних потоків як протоколу синхронізації загального призначення та відображається зв'язок роботи з ширшими дослідженнями в галузі розподілених обчислень та технологій моделювання. Також формулюється мета та завдання дослідження, підкреслюється наукова новизна та практична значущість отриманих результатів, вказується особистий внесок заявника та надається інформація про апробацію результатів та їх публікацію в наукових виданнях.

У першому розділі наведено аналітичний огляд існуючих підходів до паралельного моделювання. В огляді розглядаються алгоритми Time Warp, реактивні розширення стандарту HLA, акторно-орієнтовані фреймворки HPC та архітектури CQRS+ES, аналізуючи їхні переваги та обмеження. Показано, що хоча ці підходи забезпечують рішення для синхронізації та розподілу, вони часто є або занадто складними на рівні протоколу, або не мають теоретичної спільності. Реактивні потоки, навпаки, пропонують баланс між виразним математичним моделюванням та ефективною реалізацією. У розділі робиться висновок, що використання реактивних потоків як протоколу синхронізації є перспективним напрямком, оскільки цей підхід забезпечує модульність, масштабованість та адаптивність моделювання, одночасно зменшуючи ризики невідповідності та підвищуючи ефективність у розподілених середовищах.

У другому розділі представлені структури та формалізація підстанів як основа підходу до моделювання. Він вводить метод декомпозиції станів об'єкта на підстани з унікальними ключами, що забезпечує узгодженість, дозволяючи водночас гнучке представлення залежностей між змінними. Методологія розширена для визначення відображень (даних, отриманих в результаті спостережень або вимірювань) та моделей (прогнозів або аналітичних конструкцій) як альтернативних, але сумісних представлень.

Ключовим фокусом третього розділу є розробка узгоджених графів переходів, де функції систематично об'єднуються в орієнтовані ациклічні графи, що відображають причинно-наслідкові зв'язки між змінними. Формулюються правила та принципи для забезпечення локальної та глобальної узгодженості цих графів, включаючи лінійне впорядкування ключів, монотонність переходів та уникнення суперечностей у змінних областях. Для подальшого підвищення точності та надійності вводяться механізми для усунення невідповідностей, які можуть виникнути через неповні або суперечливі підстани. У розділі також представлено

концепцію графів моделювання, отриманих з графів переходів, де підстани та переходи явно представлені, що дозволяє безпосереднє виконання моделі. Описано метод обчислення початкових підстанів, побудови графових моделей та представлення залежностей у вигляді кортежів графів та наборів параметрів. Ця структура забезпечує інваріантність до неповних даних та стійкість до розріджених або шумних вхідних даних, тим самим підвищуючи точність та стабільність моделювання. Нарешті, формалізовано функції перетворення запропонованого графового підходу, що показує, як вони дозволяють масштабоване представлення динамічних систем з покращеною модульністю, повторним використанням та адаптивністю порівняно з традиційними методами моделювання.

У четвертому розділі дисертації демонструється, як парадигма реактивних потоків може бути використана для побудови обчислювального графа з формальної графічної моделі, представленої раніше. Окреслено правила перетворення, які показують, як підстани, функції та їхні відношення можуть бути систематично відображені на логічні процесори та канали зв'язку. Це забезпечує конкретний механізм для ініціювання та керування обчисленнями в межах можливих мереж відношень. Особлива увага приділяється методам оптимізації, оскільки безпосереднє відображення всіх функцій на процесори може призвести до неефективних або навіть нескінченних обчислювальних структур. Такі підходи, як згортання циклічних послідовностей та консолідація кількох функцій в одному процесорі, ілюструють, як можна спростити структуру графа, тим самим зменшуючи кількість вузлів обробки та покращуючи використання ресурсів. Ці методи підкреслюють важливість балансування формальної коректності з обчислювальною ефективністю.

У п'ятому розділі представлено детальну практичну демонстрацію запропонованого підходу з використанням класичного прикладу змішування сольового розчину у двох з'єднаних резервуарах. Модельований об'єкт був описаний

з точки зору його керівних диференціальних рівнянь, які потім були представлені як в аналітичній формі, так і як система підстанів та перехідних функцій. Це подвійне представлення дозволило побудувати узгоджені графи переходів та отримати відповідні графи моделювання. У цьому розділі ілюструється процедура побудови обчислювального графа з використанням реактивних потоків, де логічні процесори відповідають перехідним функціям, а підстани передаються як повідомлення через канали. Була реалізована початкова неоптимізована версія графа, після чого були проведені оптимізації, такі як згортання рекурентних структур у компактні цикли та зменшення надлишкових обчислень. Експериментальні результати моделювання показали, що обчислювальний граф на основі реактивних потоків дав результати, що узгоджуються з аналітичними рішеннями та традиційними методами числового інтегрування. Вихідні графіки концентрації солі в обох резервуарах показали майже ідентичну поведінку між аналітичною моделлю та моделюванням на основі графів, що підтверджує точність та достовірність підходу. Крім того, у розділі висвітлюються переваги інтерактивного моделювання. Були застосовані шаблони синхронізації типу «push–pull», що дозволяє моделюванню динамічно реагувати на зовнішні вхідні дані або зміни параметрів системи під час виконання. Ця гнучкість не лише підтверджує теоретичні принципи, викладені раніше, але й демонструє потенціал методу для реальних застосувань, що вимагають швидкої реакції та масштабованості.

Результати цього тематичного дослідження підтверджують практичну доцільність запропонованої методології, перевіряють правильність розробленої структури та демонструють її переваги в модульності, точності та адаптивності порівняно з традиційними методами паралельного моделювання. Вирішення цієї проблеми допоможе забезпечити надійне та ефективне виконання масштабних моделей складних систем, особливо в галузях, де стабільність, стійкість та адаптивність є критично важливими. В умовах швидко зростаючих обчислювальних

потреб, будь то в інженерії, енергетиці чи інформаційних системах, стає важливим не лише точно моделювати динамічну поведінку об'єктів, але й гарантувати масштабованість та стійкість інфраструктури моделювання.

Використання реактивних потоків як протоколу синхронізації в сучасних моделювальних фреймворках значно підвищить модульність та можливість повторного використання моделей, прискорить виконання розподілених обчислень та знизить ризики неузгодженості або збоїв під час паралельного виконання. Такий підхід є ключовим фактором для стабільного функціонування передових обчислювальних систем, забезпечуючи як теоретичну точність, так і практичну застосовність у дослідженнях та промисловості.

Ключові слова: паралельне моделювання, реактивні потоки, граф переходів, обчислювальний граф, підстани, логічні процесори, масштабованість, синхронізація, моделювання, оптимізація.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

1. O. Sirotkin, A. Prymushko, I. Puchko, H. Kravtsov, M. Yaroshynskyi, V. Artemchuk, Parallel simulation using reactive streams: A graph-based approach for dynamic modeling and optimization, *Computation* 2025, 13(5), 103; <https://doi.org/10.3390/computation13050103>. **Index in SCOPUS Q2**. (Особистий внесок – приймав участь у проведенні аналітичного огляду, та концептуалізації для побудови методу побудови паралельних чисельних моделей динамічних систем на базі протоколу реактивних потоків, узагальненні результатів та підготовці всіх розділів).
2. M. Yaroshynskyi, A. Prymushko, I. Puchko, O. Sirotkin, D. Sinko, Akka as a tool for modelling and managing a smart grid system, *Journal of Edge Computing* 2025, 4(1), pp.105–115. <https://doi.org/10.55056/jec.822>. **Index in SCOPUS**. (Особистий внесок – приймав участь у дослідженні розумних мереж, та редагуванні всіх розділів, також приймав участь розробці програмного забезпечення).
3. О.В. Сіроткін, М.С. М.С. Ярошинський, Д.П. Сінько, С.Б. Гунько, Д.О. Манолук, Моделювання у фазовому просторі під-станів, *Електронне моделювання* 2025, 47(2):28-45, <https://doi.org/10.15407/emodel.47.03.028>. Фахове видання категорії Б. (Особистий внесок – приймав участь у проведенні аналітичного огляду, та розробці програмного забезпечення).
4. М.С. Ярошинський, О.В. Сіроткін, Д.П. Сінько, С.Б. Гунько, Д.О. Манолук, Коректність пласкої класифікації, *Електронне моделювання* 2023, 45(2):34-43, <https://doi.org/10.15407/emodel.45.02.034>. Фахове видання категорії Б. (Особистий внесок – приймав участь у дослідженні пласкої класифікації, та редагуванні всіх розділів).

5. О.В. Сіроткін. *Паралельне моделювання з використанням реактивних потоків*: Матеріали XLI науково-технічна конференція молодих вчених та спеціалістів інституту проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України. Київ, 17 травня 2023 р. С. 167-175.
6. О.В. Сіроткін. *Циклічно спревалена імовірнісна графічна модель: пропозиція, основана на структурованих результатах*: Матеріали Круглого столу «Meaningful Artificial Intelligence – 2024» Київ, 26 січня 2024 р. С. 15-21.

ANNOTATION

Sirotkin O. *A Method for Constructing Parallel Numerical Models of Dynamic Systems Based on The Reactive Streams Protocol*. – Qualification scientific work with the manuscript copyright.

The thesis on receipt of Doctor of Philosophy (PhD) scientific degree of the specialty 122 "Computer Science". – G.E. Pukhov Institute for Modelling in Energy Engineering of the National Academy of Sciences of Ukraine, Kyiv, 2025.

The dissertation is devoted to the development of methods for parallel simulation of complex systems using the reactive streams paradigm as a general-purpose synchronization protocol. The proposed approach introduces a structured representation of object states through substates, transition functions, and graph-based modeling, enabling modularity, reusability, and scalability. Special attention is given to constructing consistent transition graphs, implementing them as computational graphs on logical processors, and optimizing simulation through folding techniques and interactive push–pull patterns. Practical validation is demonstrated on the classical saline mixing problem, highlighting the feasibility of the approach compared with traditional parallel simulation methods.

The main tasks of the work are to conduct a comprehensive analytical review of existing methods and approaches to parallel simulation, including Time Warp, actor-based frameworks, and reactive adaptations of HLA, in order to identify their limitations and determine promising directions for improvement; to develop a formal framework for representing complex dynamic systems through substates, transition functions, and consistent transition graphs; to design methods for mapping these graphs into computational graphs implemented with reactive streams; to study the influence of graph topology, optimization strategies, and potential inconsistencies on the accuracy,

scalability, and performance of simulations; and to validate the proposed approach through theoretical analysis and experimental case studies such as the saline mixing model.

Solving these tasks will make it possible to ensure reliable and scalable modeling of complex systems in engineering, energy, and information domains, especially under conditions that require both efficiency and resilience of computational infrastructure. The use of reactive-streams-based synchronization in modern simulation platforms will significantly enhance modularity, reusability, and interactivity of models, accelerate simulation speed on distributed systems, and minimize risks of inconsistency or failure. This is a key factor for advancing the state of the art in parallel computation and providing robust tools for the stable operation of critical infrastructures and scientific research.

The dissertation consists of an introduction, 5 chapters, general conclusions, a list of references, and appendices.

The introduction substantiates the relevance of the research topic, outlines the scientific and technical problem of parallelizing simulations of complex systems, justifies the feasibility of applying the reactive streams paradigm as a general-purpose synchronization protocol, and reflects the connection of the work with broader research in distributed computing and simulation technologies. It also formulates the purpose and objectives of the research, highlights the scientific novelty and practical significance of the results obtained, indicates the personal contribution of the applicant, and provides information about the approbation of the results and their publication in scientific outlets.

The first chapter provides an analytical review of existing approaches to parallel simulation. The review examines Time Warp algorithms, reactive extensions of the HLA standard, actor-based HPC frameworks, and CQRS+ES architectures, analyzing their advantages and limitations. It is shown that while these approaches provide solutions to synchronization and distribution, they are often either too complex at the protocol level or lack theoretical generality. Reactive streams, by contrast, offer a balance between

expressive mathematical modeling and efficient implementation. The chapter concludes that using reactive streams as a synchronization protocol is a promising direction, since this approach ensures modularity, scalability, and adaptability of simulation, while reducing the risks of inconsistency and improving efficiency in distributed environments.

The second chapter presents the proposed structures and formalization of substates as the foundation of the modeling approach. It introduces a method for decomposing object states into substates with unique keys, ensuring consistency while allowing flexible representation of dependencies between variables. The methodology is extended to define reflections (data obtained from observation or measurement) and models (predictions or analytical constructs) as alternative but compatible representations.

A key focus of the third chapter is the development of consistent transition graphs, where functions are systematically joined into directed acyclic graphs that capture the causal relationships among variables. Rules and principles are formulated to ensure local and global consistency of these graphs, including linear ordering of keys, monotonicity of transitions, and avoidance of contradictions in variable domains. To further improve accuracy and reliability, mechanisms are introduced to eliminate inconsistencies that might arise from incomplete or contradictory substates. The chapter also presents the concept of simulation graphs derived from transition graphs, where substates and transitions are explicitly represented, enabling direct execution of the model. A method for computing initial substates, constructing graph models, and representing dependencies as tuples of graphs and parameter sets is described. This structure ensures invariance to incomplete data and resilience to sparse or noisy inputs, thus enhancing modeling accuracy and stability. Finally, the transformation functions of the proposed graph-based approach are formalized, showing how they allow scalable representation of dynamic systems with improved modularity, reusability, and adaptability compared to traditional simulation methods.

The fourth chapter of the dissertation demonstrates how the reactive stream paradigm can be used to construct a computational graph from the formal graphical model presented earlier. Transformation rules are outlined that show how substates, functions, and their relations can be systematically mapped to logical processors and communication channels. This provides a concrete mechanism for initiating and controlling computations within probabilistic networks of relations. Special attention is paid to optimization techniques, since directly mapping all functions to processors can lead to inefficient or even infinite computational structures. Approaches such as the collapse of cyclic sequences and the consolidation of multiple functions within a single processor illustrate how the graph structure can be simplified, thereby reducing the number of processing nodes and improving resource utilization. These techniques emphasize the importance of balancing formal correctness with computational efficiency.

The fifth chapter presents a detailed practical demonstration of the proposed approach using a classical case study of saline mixing in two connected tanks. The modeled object was described in terms of its governing differential equations, which were then represented in both analytical form and as a system of substates and transition functions. This dual representation made it possible to construct consistent transition graphs and derive corresponding simulation graphs. The chapter illustrates the procedure for building a computational graph using reactive streams, where logical processors correspond to transition functions and substates are transmitted as messages through channels. An initial unoptimized version of the graph was implemented, followed by optimizations such as folding recurrent structures into compact cycles and reducing redundant computations. Experimental results of the simulation demonstrated that the reactive-streams-based computational graph produced results consistent with analytical solutions and traditional numerical integration methods. Output plots of salt concentration in both tanks showed near-identical behavior between the analytical model and the graph-based simulation, confirming the accuracy and validity of the approach. Additionally, the

chapter highlights the advantages of interactive simulation. Push–pull synchronization patterns were applied, enabling the simulation to react dynamically to external inputs or changes in system parameters during runtime. This flexibility not only validates the theoretical principles outlined earlier but also demonstrates the potential of the method for real-world applications requiring responsiveness and scalability. The results of this case study confirm the practical feasibility of the proposed methodology, verify the correctness of the developed framework, and showcase its advantages in modularity, accuracy, and adaptability over traditional parallel simulation techniques.

Solving this problem will help ensure reliable and efficient execution of large-scale simulations of complex systems, particularly in domains where stability, resilience, and adaptability are critically important. In conditions of rapidly growing computational demands, whether in engineering, energy, or information systems, it becomes essential not only to accurately model the dynamic behavior of objects but also to guarantee scalability and robustness of the simulation infrastructure.

The use of reactive streams as a synchronization protocol in modern simulation frameworks will significantly enhance modularity and reusability of models, accelerate the execution of distributed computations, and minimize the risks of inconsistency or failure during parallel execution. This approach is a key factor for the stable functioning of advanced computational systems, ensuring both theoretical rigor and practical applicability in research and industry.

Keywords: parallel simulation, reactive streams, transition graph, computational graph, substates, logical processors, scalability, synchronization, modeling, optimization.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	17
ВСТУП.....	19
РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ ДО ПОБУДОВИ ПАРАЛЕЛЬНИХ ЧИСЕЛЬНИХ МОДЕЛЕЙ ДИНАМІЧНИХ СИСТЕМ	27
1.1 Паралельне моделювання: концепції, методи та застосування	28
1.1.1 Основні концепції	28
1.1.2 Як використовується паралельне моделювання	30
1.1.3 Сучасні тенденції	31
1.2 Огляд існуючих методів до паралельного моделювання	32
1.2.1 Алгоритм Time Warp	32
1.2.2 Фреймворк RxHLA.....	37
1.2.3 Архітектура CQRS + ES.....	41
1.2.4 Фреймворк акторів HPC (високопродуктивні обчислення)	46
ВИСНОВКИ ДО РОЗДІЛУ 1	51
РОЗДІЛ 2 ВИЗНАЧЕННЯ БАЗОВИХ ЕЛЕМЕНТІВ МЕТОДІВ ПАРАЛЕЛЬНОГО МОДЕЛЮВАННЯ	53
2.1. Моделювання об'єктів та фізичного часу	54
2.2. Набір змінних V як представлення модельованого об'єкта	55
2.3. Стани \mathfrak{B} , \mathfrak{X} , \mathfrak{Y} , та \mathfrak{G} як значення змінних V , X , Y , та G	58
2.4. Представлення залежності Y від X як набору функцій: $Y = FX \mathfrak{G}$	61
2.5. Підстані $\mathfrak{S}\mathfrak{A}q$ як розклад стану \mathfrak{B}	63
2.6. Представлення залежності Y від X як набору підстанів: $Y = \mathfrak{S}X \mathfrak{G}$	67
2.7. Відображення $YX\mathfrak{G}$ як запис змін значень змінних.....	70
2.8. Модель $Y(X \mathfrak{G})$ як імітація змін змінних V	72
2.9. Моделювання моделі $Y(X \mathfrak{G})$ як обчислення підмножини $\mathfrak{Y} \subseteq \mathbb{Y}n$	74
2.10. Зведення всього разом: Відображення $YX\mathfrak{G}$, модель $YX\mathfrak{G}$	76
ВИСНОВКИ ДО РОЗДІЛУ 2	79

РОЗДІЛ 3 ФОРМАЛЬНОЇ СТРУКТУРИ ОПИСУ ДИНАМІКИ ПІДСТАНІВ ЗА ДОПОМОГОЮ ПЕРЕХІДНИХ ФУНКЦІЙ	80
3.1. Перехідні функції $\Theta \mathfrak{D}$	81
3.2. Граф переходу $\Gamma \mathfrak{G}$ та граф моделювання $\gamma \mathfrak{G}$	85
3.3. Побудова узгодженого графа переходів $\Gamma \mathfrak{G}$	90
3.4. Обчислюваність графа моделювання $\gamma \mathfrak{G}$ та початкової множини підстанів \mathfrak{S}'	94
3.5. Представлення залежності Y від X вигляді графової моделі $YX\mathfrak{G}\Gamma$	95
3.6. Моделювання графа моделі $YX\mathfrak{G}\Gamma$	98
ВИСНОВКИ ДО РОЗДІЛУ 3	102
РОЗІЛ 4 ПОБУДОВА ТА ВИКОРИСТАННЯ ОБЧИСЛЮВАЛЬНОГО ГРАФА НА ОСНОВІ ПАРАДИГМИ РЕАКТИВНИХ ПОТОКІВ	103
4.1. Реактивні потоки та графова модель $YX\mathfrak{G}\Gamma$	103
4.2. Оптимізація графа C	106
4.3. Моделювання моделі $YX\mathfrak{G}\Gamma$ за допомогою графа C	108
4.4. Проблеми і перспективні шляхи подальшого розвитку запропонованого методу	109
ВИСНОВКИ ДО РОЗДІЛУ 4	113
РОЗІЛ 5 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ МЕТОДУ ПОБУДОВИ ПАРАЛЕЛЬНИХ ЧИСЕЛЬНИХ МОДЕЛЕЙ ДИНАМІЧНИХ СИСТЕМ НА БАЗІ ПРОТОКОЛУ РЕАКТИВНИХ ПОТОКІВ	114
5.1 Опис змодельованого об'єкта та побудова моделі $Y(X \mathfrak{G})$	115
5.2 Побудова та моделювання графової моделі $YX\mathfrak{G}\Gamma$	125
5.3 Побудова та обчислення графа C з використанням моделі $YX\mathfrak{G}\Gamma$	137
ВИСНОВКИ ДО РОЗДІЛУ 5	148
ЗАГАЛЬНІ ВИСНОВКИ	149
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	152
ДОДАТОК А СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ ТА ВІДОМОСТІ ПРО АПРОБАЦІЮ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ	160
ДОДАТОК Б АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ	162

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Щоб зробити формальний опис більш однорідним та зрозумілим, визначимо деякі правила позначення:

- поряд зі знаком рівності ($=$), використовуватимемо знак присвоєння ($:=$) коли присвоюємо ім'я оператору;
- змінні позначаються малими літерами, наприклад, x , y , v ;
- набір змінних позначається великими літерами, наприклад, X , Y , V . Оператор $X := \begin{bmatrix} y \\ v \end{bmatrix}$ буде читатися як " X це набір змінних y та v ";
- позначаємо значення, які можна пов'язати зі змінними, малими літерами франкфуртської літери, наприклад, \mathfrak{x} , \mathfrak{y} , \mathfrak{v} . Оператор $x = \mathfrak{x}$ буде читатися як "значення \mathfrak{x} пов'язане зі змінною x ";
- множину всіх значень можна пов'язати зі змінною (тобто доменом змінної), яку позначаємо двозначними символами у малому регістрі, наприклад, \mathfrak{x} , \mathfrak{y} , \mathfrak{v} . Оператор $x: \mathfrak{x}$ читатиметься як "змінна x , яка може бути пов'язана з будь-яким значенням з \mathfrak{x} " а оператор $\mathfrak{x} \in \mathfrak{x}$ читатиметься як "значення \mathfrak{x} з домену \mathfrak{x} ";
- набори значень, які можна пов'язати з набором змінних, позначаємо великими франкфуртськими символами, наприклад, \mathfrak{X} , \mathfrak{Y} , \mathfrak{V} . Оператор $\mathfrak{X} := \begin{bmatrix} \mathfrak{y} \\ \mathfrak{v} \end{bmatrix}$ читатиметься як " \mathfrak{X} це набір значень \mathfrak{y} та \mathfrak{v} ," а оператор $X = \mathfrak{X}$ читатиметься як "набір значень \mathfrak{X} пов'язаний з набором змінних X ";
- набір усіх наборів значень, які можна пов'язати з набором змінних (простором значень), позначаємо двозначними символами у великому регістрі з розмірним індексом, наприклад \mathfrak{X}^n , \mathfrak{Y}^n , \mathfrak{V}^n , де $n \in \mathbb{N}$. Оператор $X: \mathfrak{X}^n$ буде читатися як " множина змінних X , які можуть бути пов'язані з будь-яким

- значенням з n -вимірного простору значень \mathbb{X}^n , а оператор $\mathfrak{X} \in \mathbb{X}^n$ буде читатися як "множина значень \mathfrak{X} (точка) з n -вимірного простору значень \mathbb{X}^n ;
- довільну множину значень з області визначення змінних позначаємо жирними малими франкфуртськими символами, наприклад, $\mathfrak{x}, \mathfrak{y}, \mathfrak{v}$. Оператор $\mathfrak{x} := \begin{Bmatrix} \mathfrak{x}_1 \\ \mathfrak{x}_2 \\ \mathfrak{x}_3 \end{Bmatrix}$ буде читатися як " \mathfrak{x} це множина, що включає $\mathfrak{x}_1, \mathfrak{x}_2$ та \mathfrak{x}_3 ";
 - довільні множини значень позначаємо жирними великими франкфуртськими символами, наприклад, $\mathfrak{X}, \mathfrak{Y}, \mathfrak{Z}$. Оператор $\mathfrak{X} := \begin{Bmatrix} \mathfrak{x}_1 \\ \mathfrak{x}_2 \end{Bmatrix}$ буде читатися як " \mathfrak{X} це множина, що включає \mathfrak{x}_1 та \mathfrak{x}_2 ";
 - множину підмножин елементів множини позначаємо синтаксисом $<$ підмножина $>(<$ множина $>)$, наприклад $\mathfrak{y} = \mathfrak{y}(\mathfrak{X})$, де $\mathfrak{X} := \begin{Bmatrix} \mathfrak{x}_1 = [\mathfrak{y}_1, \mathfrak{v}_1] \\ \mathfrak{x}_2 = [\mathfrak{y}_2, \mathfrak{v}_2] \end{Bmatrix}$, $\mathfrak{y} = \begin{Bmatrix} \mathfrak{y}_1 \\ \mathfrak{y}_2 \end{Bmatrix}$. Таке позначення можна інтерпретувати як функцію $\mathfrak{y}: \mathfrak{X} \rightarrow \mathfrak{y}$;
 - не рівні, але еквівалентні об'єкти будуть позначені символом \Leftrightarrow , наприклад $f\left(\begin{Bmatrix} 1 \\ 2 \end{Bmatrix}\right) \neq \begin{Bmatrix} 3 \\ 4 \end{Bmatrix}$, але $f\left(\begin{Bmatrix} 1 \\ 2 \end{Bmatrix}\right) \Leftrightarrow \begin{Bmatrix} 3 \\ 4 \end{Bmatrix}$, де $f(\mathfrak{x}) = \mathfrak{x} + 2$.

ВСТУП

Обґрунтування вибору теми дослідження. Сучасні складні динамічні системи, у галузях енергетики, біології, фізичного моделювання та соціальних процесів, потребують усе більш точних і масштабованих методів моделювання, які неможливі без розпаралелювання обчислень, тобто без використання паралельних чисельних моделей.

Чисельна модель є програмою (програмним засобом) що слугує віртуальним аналогом фізичної сутності, що дозволяє візуалізацію, маніпулювання та аналіз у цифровому середовищі. Паралельні чисельні моделі це підклас чисельних моделей, які використовують паралельні обчислення для прискорення моделювання.

Традиційні підходи до паралельного моделювання, такі як алгоритм Time Warp, RxHLA або actor-based HPC-платформи, хоча й довели свою ефективність, залишаються складними у реалізації та вимагають низькорівневих і спеціалізованих протоколів синхронізації, а також не дуже добре адаптовані до сучасної архітектури обчислювальних систем оскільки були розроблені з урахуванням старих архітектур. Це ускладнює створення нових моделей та уповільнює їх інтеграцію в практичні застосування.

Одним із головних викликів побудови паралельних моделей є узгоджене й ефективне виконання чисельного моделювання у багатопроцесорних і розподілених середовищах. Використання класичних методів часто призводить до значних обчислювальних витрат, труднощів із балансуванням навантаження та обмежень у масштабуванні. Водночас зростає потреба у гнучких інструментах, які дозволяють швидко створювати, модифікувати й перевіряти моделі без втрати точності чи надійності.

У цьому контексті особливої актуальності набуває застосування протоколів паралельних обчислень загального призначення. Зокрема протоколу реактивних потоків як універсального засобу паралелізації та синхронізації обчислень. Його використання дає змогу суттєво спростити процес розробки паралельних моделей, представивши їх у вигляді графів переходів і обчислювальних графів. Такий підхід поєднує математичну строгість із практичною зручністю, забезпечуючи модульність, повторне використання функціональних блоків і можливість автоматичної оптимізації.

Додатковим аргументом є здатність запропонованого методу працювати в інтерактивному режимі (push- та pull-шаблони), що дозволяє моделі реагувати на зовнішні події в реальному часі. Це відкриває перспективи для його застосування у сферах, де необхідна оперативна адаптація до змінних умов, наприклад у цифрових енергетичних системах, керуванні складними технологічними процесами або дослідженні великих соціально-економічних моделей.

Таким чином, актуальне наукове завдання полягає у розробленні методу побудови паралельних чисельних моделей динамічних систем на базі протоколу реактивних потоків. Його вирішення сприятиме підвищенню надійності, масштабованості та зручності розробки моделей, а також дозволить врахувати сучасну архітектуру обчислювальних систем, що є ключовим фактором для розвитку сучасних обчислювальних технологій.

Зв'язок роботи з науковими програмами, планами, темами, грантами.

Зв'язок роботи з науковими програмами, планами, темами. Тематика дисертаційної роботи відповідає пріоритетним напрямам розвитку науки і техніки в Україні. Робота виконувалась відповідно до плану наукових досліджень Інституту проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України, зокрема в рамках НДР «Розвиток наукових засад алгебраїчної теорії сильного штучного інтелекту стосовно кібернетичної безпеки об'єктів критичної інфраструктури в галузі енергетики» (№

ДР 0123U100913, 2023-2027 рр.) та «Розвиток розподіленої енергетики в умовах ринку електричної енергії України з використанням технологій та систем цифровізації. Розділ 1. Організаційні та математичні моделі взаємодії учасників децентралізованого ринку електроенергії» (№ ДР 0125U000237, 2025-2026 рр.).

Мета і завдання дослідження. Метою роботи є забезпечення ефективного виконання чисельного моделювання у багатопроцесорних і розподілених середовищах, шляхом розроблення методу побудови паралельних чисельних моделей динамічних систем на базі протоколу реактивних потоків.

Акцент зроблено на спрощенні процесу розробки моделей завдяки використанню високорівневого, універсального протоколу реактивних потоків, а також завдяки модульному поданню станів та автоматизованій побудові графів переходів, що зменшує трудомісткість і ризик помилок при реалізації моделі. Це дозволяє створювати масштабовані та узгоджені обчислювальні схеми без необхідності розробки спеціалізованих протоколів паралелізації та синхронізації обчислень.

Для досягнення цієї мети були поставлені такі **основні завдання**:

1. Провести аналітичний огляд існуючих підходів до паралельного моделювання (алгоритм *Time Warp*, RxHLA, CQRS+ES, actor-based НРС-платформи) та визначити їхні переваги й обмеження.
2. Розробити метод побудови паралельних чисельних моделей складних динамічних систем на базі протоколу реактивних потоків.
3. Запропонувати підхід до подання станів динамічних систем у вигляді множини підстанів із використанням унікальних ключів.
4. Запропонувати подання динамічних моделей у вигляді графа переходів і розроблено підхід його перетворення у структуровану графову модель.

5. Запропонувати використання push- та pull-шаблонів для паралельного інтерактивного моделювання складних динамічних систем.
6. Розробити метод перетворення графів переходів на обчислювальні графи, що реалізуються у парадигмі реактивних потоків (зокрема, з використанням бібліотеки AKKA Streams).
7. Реалізувати демонстраційний приклад обчислювального графа на базі бібліотеки AKKA Streams для задачі змішування сольових розчинів.
8. Розробити практичні рекомендації щодо перспектив використання паралельних моделей складних динамічних систем, в різних галузях (енергетика, фізичне моделювання, біологічні та соціальні системи).

Об'єктом дослідження є розроблення засобів паралельного моделювання складних динамічних систем у багатопроцесорних та розподілених середовищах.

Предметом дослідження є методи і засоби побудови паралельних чисельних моделей динамічних систем на базі протоколу реактивних потоків.

Методи дослідження. У дисертаційній роботі при розв'язанні поставлених наукових задач комплексно використовувалися методи системного і функціонального аналізу, математичного та графового моделювання, теорії алгоритмів, методи дослідження операцій, об'єктно-орієнтованого програмування, планування наукового експерименту та обробки його результатів тощо. Основою дослідження є побудова модельних графів і їх реалізація у вигляді обчислювальних графів із застосуванням протоколу реактивних потоків.

У роботі застосовано:

- методи формальної логіки та математичного аналізу для побудови визначень підстанів, моделей і відображень;
- апарат теорії графів для конструювання узгоджених графів переходів і перевірки їхньої коректності;

- методи об'єктно-орієнтованого програмування для реалізації структур підстанів і графів у програмному коді;
- інструменти з бібліотеки AKKA Streams для побудови обчислювальних графів та організації паралельних обчислень;
- чисельні методи (зокрема метод Ейлера) для апроксимації диференціальних рівнянь у процесі моделювання;
- методи порівняльного аналізу для верифікації результатів моделювання на прикладі задачі змішування сольових розчинів та співставлення з аналітичним розв'язком.

Теоретична частина дослідження ґрунтується на аналітичних та чисельних методах, тоді як практична реалізація виконувалася у вигляді паралельного моделювання з використанням реактивних потоків, що дозволило перевірити масштабованість, інтерактивність та оптимізаційні можливості підходу.

Наукова новизна отриманих результатів:

1. Розроблено метод побудови паралельних чисельних моделей складних динамічних систем на базі протоколу реактивних потоків, який, на відміну від існуючих, враховує поточну архітектуру обчислювальних систем і сучасні методи розроблення програмного забезпечення, що дозволяє підвищити ефективність виконання чисельного моделювання у багатопроцесорних і розподілених середовищах.
2. Запропоновано підхід до подання станів динамічних систем у вигляді множини підстанів із використанням унікальних ключів, що, на відміну від монолітних станів в традиційних паралельних моделях, забезпечує модульність, повторне використання та узгодженість при побудові графів переходів і обчислювальних графів в паралельних моделях.

3. Запропоновано подання динамічних моделей у вигляді графа переходів і розроблено підхід його перетворення у структуровану графову модель, що на відміну від використання імперативних описів (наборів інструкцій для виконання машинами) істотно спрощує процес створення та масштабування моделей у багатопроцесорних і розподілених середовищах.
4. Запропоновано використання push- та pull-шаблонів для паралельного інтерактивного моделювання складних динамічних систем, що на відміну від класичних шаблонів, забезпечує ефективне використання обчислювальних ресурсів та менший час відгуку моделі, під час синхронізації отриманих моделей з реальною системою по параметрах моделі.
5. Розроблено метод перетворення графів переходів на обчислювальні графи, що реалізуються у парадигмі реактивних потоків (зокрема, з використанням бібліотеки АККА Streams). Такий метод, на відміну від традиційних (побудованих на основі потоків інструкцій), забезпечує більш природне поєднання абстрактної специфікації з виконуваними обчисленнями, а також спрощує процес масштабованої реалізації моделей у багатопроцесорних та розподілених середовищах.

Практичне значення отриманих результатів полягає у наступному:

1. Реалізовано демонстраційний приклад обчислювального графа на базі бібліотеки АККА Streams для задачі змішування сольових розчинів, результати якого співпадають з аналітичними розрахунками, що підтверджує коректність підходу. На цьому прикладі апробовано методи оптимізації обчислювальних графів та інтерактивного моделювання.
2. Розроблено практичні рекомендації щодо перспектив використання паралельних моделей складних динамічних систем, в різних галузях (енергетика, фізичне моделювання, біологічні та соціальні системи).

Особистий внесок здобувача. Безпосередньо автором здійснено:

- проведено аналітичний огляд існуючих підходів до паралельного моделювання (алгоритм Time Warp, RxHLA, CQRS+ES, actor-based HPC-платформи) та визначено їхні переваги й обмеження;
- розроблено систему формальних визначень (підстани, відображення, моделі, графи переходів), що лягли в основу побудови узгоджених модельних графів;
- створено методику перетворення графів переходів у обчислювальні графи із використанням протоколу реактивних потоків та бібліотеки АККА Streams;
- розроблено алгоритми оптимізації обчислювальних графів (згортання циклів, усунення надлишкових обчислень), які дозволяють підвищити ефективність моделювання;
- запропоновано та реалізовано push- і pull- шаблони для організації інтерактивного моделювання та синхронізації з зовнішніми подіями в реальному часі;
- здійснено практичну апробацію на задачі змішування сольових розчинів у двох резервуарах, побудовано обчислювальний граф та порівняно результати з аналітичним розв’язком.

Наукові роботи опубліковані у співавторстві з А. Примушком, І. Пучком, М. Ярошинським, Г. Кравцовим, В. Артемчуком та іншими. Співавторами є наукові керівники та колеги, з якими проведено дослідження. У наукових працях, опублікованих у співавторстві, здобувачу належить основний фактичний матеріал і творчий доробок, пов’язаний із формальними визначеннями, побудовою та оптимізацією графових моделей, а також реалізацією прикладного експерименту.

Постановка мети та завдань, а також обговорення результатів виконувалися у тісній співпраці з науковим керівниками.

Апробація результатів дисертації. Основні положення та результати дослідження, пов'язані з розробкою методу паралельного моделювання на основі протоколу реактивних потоків, були представлені та обговорені на науково-практичних конференціях різного рівня: XLI Науково-технічна конференція молодих вчених та спеціалістів (Київ, 2023, 17 травня 2023 р., форма участі – виступ і публікація тез). Та круглий стіл «Meaningful Artificial Intelligence – 2024» (Київ, 2025, 26 січня 2024 р., форма участі – виступ і публікація тез).

Результати роботи також були представлені на наукових семінарах відділу математичного та комп'ютерного моделювання Інституту проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України.

Структура та обсяг дисертації.

Дисертаційна робота викладена на 164 сторінках машинописного тексту, складається зі вступу, 5 розділів, загальних висновків, списку використаних джерел та 2 додатки. Обсяг основного тексту дисертації складає 124 сторінок друкованого тексту. Робота ілюстрована 32 рисунком. Список використаних джерел містить 73 найменування, з них 4 кирилицею та 69 латиницею.

РОЗДІЛ 1

ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ ДО ПОБУДОВИ ПАРАЛЕЛЬНИХ ЧИСЕЛЬНИХ МОДЕЛЕЙ ДИНАМІЧНИХ СИСТЕМ

Швидке зростання обчислювальних ресурсів та зростаюча складність динамічних систем зробили паралельне моделювання одним із центральних напрямків сучасної обчислювальної науки. На відміну від послідовних підходів, паралельне моделювання розподіляє обчислювальне навантаження між кількома процесорами або вузлами, що дозволяє дослідникам та інженерам досліджувати великомасштабні системи з вищою деталізацією та реалізмом. Ця парадигма стала незамінною в багатьох галузях, починаючи від кліматичного моделювання та астрофізики і закінчуючи телекомунікаціями, транспортом, виробництвом та оборонними застосуваннями.

Однак ефективність паралельного моделювання залежить не лише від чистої обчислювальної потужності, але й від методів декомпозиції, синхронізації та обміну інформацією між підсистемами. Протягом багатьох років було запропоновано низку стратегій та фреймворків для вирішення цих проблем. Класичні підходи, такі як консервативна синхронізація, гарантують правильність, але часто недовикористовують ресурси через простоту. Оптимістичні протоколи, найвідоміший з яких алгоритм Time Warp[1, 2], дозволяють спекулятивне виконання та виправлення через відкат, пропонуючи покращену продуктивність, але ціною більшої складності реалізації.

Зовсім недавно розробка моделей розподіленого програмування, таких як системи на основі акторів, фреймворки реактивного програмування та хмарні архітектури, відкрила нові можливості для проектування масштабованих та адаптивних середовищ моделювання. Такі рішення, як RxHLA, CQRS з Event Sourcing та акторні фреймворки HPC, демонструють, як сучасні програмні

парадигми можуть бути узгоджені з паралельними завданнями моделювання, кожна з яких пропонує свої переваги та обмеження.

У цьому розділі наведено аналітичний огляд існуючих підходів до побудови паралельних числових моделей динамічних систем. Розглядаючи як класичні методи, так і нові фреймворки, він закладає основу для розуміння сильних сторін, компромісів та застосовності різних стратегій синхронізації та виконання. Особлива увага приділяється порівнянням із запропонованим у цій роботі підходом на основі реактивних потоків, підкреслюючи його потенціал як універсального та масштабованого протоколу синхронізації для паралельного моделювання.

1.1 Паралельне моделювання: концепції, методи та застосування

Паралельне моделювання, це розділ обчислювальної науки, що займається моделюванням динамічних систем шляхом розподілу завдань моделювання між кількома процесорами або обчислювальними вузлами. Основна ідея в наступному: замість покрокового виконання моделювання на одній машині, робоче навантаження розділяється на менші частини, які можна обробляти одночасно. Такий підхід не тільки пришвидшує виконання, але й дозволяє моделювати великі та дуже деталізовані системи, які в іншому випадку були б важкорозв'язними.

1.1.1 Основні концепції

В основі паралельного моделювання лежить концепція декомпозиції. Складну систему, таку як транспортна мережа, енергомережа або модель гідродинаміки, можна розбити на менші компоненти або підсистеми. Кожен з цих компонентів призначається процесору (або логічному процесору у випадку програмних абстракцій). Процесори виконують свою частину моделювання паралельно, обмінюючись інформацією, коли існують залежності.

Однак, паралелізація створює проблему синхронізації. Моделювання керовані часом, а це означає, що події в одній підсистемі можуть залежати від стану іншої. Якщо процесори просувають свої локальні часові шкали без координації, можуть виникнути невідповідності. Для управління цим використовуються дві основні стратегії синхронізації:

1. Консервативна синхронізація: Консервативна синхронізація ґрунтується на принципі суворого дотримання часової послідовності подій. Її головна ідея полягає в тому, що жоден процесор не може виконати подію передчасно. Для цього кожен процесор повинен постійно перевіряти, чи є гарантії відсутності подій із меншими часовими мітками, які можуть надійти пізніше. Лише після отримання такої гарантії дозволяється продовження обчислень. Подібний підхід забезпечує високу надійність виконання, оскільки повністю виключає можливість помилок, спричинених некоректною обробкою подій у неправильному порядку. Водночас консервативна синхронізація має і свої недоліки: необхідність очікування безпечних умов для обробки часто призводить до простоїв процесорів та до неповного використання доступних обчислювальних ресурсів.
2. Оптимістична синхронізація: Оптимістична синхронізація базується на принципі спекулятивного виконання подій. У цьому підході процесори не очікують підтвердження безпечності, а продовжують рухатися вперед, обробляючи події так, ніби вони гарантовано відбуваються у правильному порядку. У випадку, коли надходить подія з меншою часовою міткою, яка мала бути виконана раніше, система виявляє порушення причинно-наслідкового зв'язку. Для усунення такої невідповідності застосовуються спеціальні механізми відкату, що дозволяють повернутися до попередньої узгодженої точки та повторно виконати обчислення у правильному порядку. Подібна стратегія часто забезпечує значне зростання продуктивності, оскільки

мінімізує прості процесорів. Проте вона потребує надійних засобів створення контрольних точок, відновлення та повторного виконання, що ускладнює реалізацію та збільшує витрати на підтримку системи.

Ще однією важливою концепцією є модель, керована подіями, де поведінка системи виражається через події, що змінюють стан об'єктів. Ця абстракція природно вписується в паралельні фреймворки, оскільки події часто можна обробляти незалежно або групувати за локальністю.

1.1.2 Як використовується паралельне моделювання

Паралельне моделювання є ключовим фактором у багатьох наукових та інженерних галузях:

- високопродуктивні наукові обчислення: кліматичні моделі, астрофізичного моделювання та молекулярна динаміка спираються на поділ величезних просторових сіток на піддомени, кожна з яких обробляється паралельно;
- телекомунікації та мережі: паралельне дискретне моделювання допомагає оцінити великомасштабні мережі, де мільйони пакетів, вузлів та взаємодій повинні моделюватися в режимі реального часу;
- транспорт та логістика: моделювання потоків транспорту, міської мобільності та ланцюгів поставок використовує паралельні моделі для дослідження заторів, стратегій оптимізації та планування інфраструктури;
- виробництво та промисловість: складні виробничі лінії та цифрові двійники промислових систем моделюються паралельно для аналізу продуктивності, прогнозування збоїв та оптимізації операцій;

- оборона та навчання: військове та тренувальне моделювання реагування на надзвичайні ситуації часто використовують розподілені паралельні архітектури для підтримки великої кількості учасників та дуже деталізованих сценаріїв.

1.1.3 Сучасні тенденції

Сьогодні розвиток паралельного моделювання значною мірою зумовлений низкою технологічних досягнень, які суттєво розширюють можливості дослідників та інженерів.

Насамперед, високопродуктивні обчислювальні кластери та хмарні інфраструктури надають масштабовані апаратні платформи, здатні ефективно обробляти великі обсяги даних та підтримувати симуляції з тисячами чи навіть мільйонами взаємодіючих компонентів. Це дозволяє переносити складні фізичні, інженерні та соціально-економічні моделі у практичну площину, забезпечуючи високу точність і швидкість розрахунків.

Важливим внеском у розвиток методів паралельного моделювання стали також програмні парадигми, що базуються на акторній моделі та реактивних потоках. Вони дозволяють розглядати комунікацію й синхронізацію як першокласні концепції, що спрощує проектування розподілених систем і робить їх більш стійкими та гнучкими до змін навантаження.

Ще одним напрямом прогресу є інтеграція методів машинного навчання. Завдяки їм системи моделювання отримують можливість адаптивно розподіляти ресурси, прогнозувати залежності подій та оптимізувати механізми відкатів або планування. Це сприяє підвищенню ефективності симуляцій і розширює спектр задач, які можна вирішувати у паралельному середовищі.

У сукупності ці досягнення створюють основу для переходу від традиційних підходів до більш інтелектуальних і масштабованих систем моделювання, здатних відповідати викликам сучасної науки та індустрії.

Таким чином, паралельне моделювання поєднує стратегії декомпозиції, розподілених обчислень та синхронізації, що дозволяє вивчати складні системи в масштабах, неможливих для послідовних методів. Його застосування охоплює практично всі наукові та інженерні галузі, що робить його одним з фундаментальних інструментів сучасної обчислювальної науки.

1.2 Огляд існуючих методів до паралельного моделювання

Проаналізуємо кілька існуючих методів до паралельного моделювання та порівняємо їх із підходом запропонованим у цій роботі (опис якого наведено в розділах 2,3 і 4).

1.2.1 Алгоритм Time Warp

Алгоритм Time Warp [1, 2] являє собою прорив у галузі паралельного дискретно-подійного моделювання (PDES). Він впроваджує оптимістичний метод синхронізації, в якому процеси виконують події, не очікуючи глобальних гарантій правильності. Якщо виникає помилка причинно-наслідкового зв'язку, система відновлює узгодженість шляхом відкату та скасування події. Цей метод усуває простоту в очікуванні, дозволяє максимально використовувати паралелізм та забезпечує універсальний підхід до високопродуктивного паралельного моделювання.

У дискретно-подійному моделюванні система моделюється як сукупність логічних процесів (ЛП) [3, 4, 5], які взаємодіють, надсилаючи повідомлення з

часовими мітками, кожне з яких представляє собою певну подію. Фундаментальна проблема синхронізації полягає в забезпеченні обробки цих подій у незменшуваному порядку часових міток по всій системі.

Консервативні методи синхронізації (наприклад, Чанді-Місра-Брайант) досягають коректності, запобігаючи порушенням, вимагаючи від LP блокуватися, доки не стане безпечно продовжувати. Ця стратегія дозволяє уникнути помилок, але може призвести до простою та недовикористання ресурсів.

Алгоритм Time Warp використовує альтернативний підхід: замість того, щоб уникати помилок причинно-наслідкового зв'язку, він оптимістично виконує події та виправляє помилки пізніше. Ця філософія проектування зміщує синхронізацію з запобігання на відновлення після помилок, відкриваючи нові можливості для паралельного прискорення.

Оптимістичне виконання і збереження стану

Кожен LP обробляє вхідні події одразу після їх надходження, у порядку локальних позначок часу, не чекаючи на підтвердження глобальної коректності. Це часто дозволяє LP просуватися швидше, ніж консервативним схемам.

Перед обробкою події LP зберігає свій локальний стан. Збереження стану може виконуватися повністю або поступово, щоб зменшити вимоги до пам'яті. Збережені стани дозволяють відкат, що є центральним для Time Warp.

Відкат, відстаючі повідомлення та анти повідомлення

Помилка причинно-наслідкового зв'язку виникає, коли надходить повідомлення про відставання, повідомлення з позначкою часу, що передує поточному локальному віртуальному часу (LVT) LP. Після виявлення відставання: LP повертається до останнього збереженого стану перед позначкою часу відставання. Усі обчислення, виконані після цієї точки, скасовуються.

Якщо LP вже надіслав повідомлення на основі неправильних обчислень, ці повідомлення необхідно скасувати. Це досягається шляхом випуску антиповідомлень, які відповідають ідентифікаторам неправильних подій, але позначають їх як скасування. Якщо LP-отримувач ще не обробив початкову подію, антиповідомлення просто скасовує її. Якщо це сталося, одержувач також відкатує, що потенційно може спричинити каскад відкатів між LP.

Локальний та глобальний віртуальний час

Кожен LP підтримує LVT, який відповідає часовій позначці його останньої правильно виконаної події. Відкати можуть зменшити LVT LP до попереднього значення.

Для координації прогресу в масштабах всієї системи алгоритм обчислює Глобальний віртуальний час (GVT), який визначається як мінімум серед: глобальних віртуальних часів усіх LP, та позначок часу всіх повідомлень, що передаються.

GVT виконує дві критично важливі функції:

1. Гарантія прогресу: жодне відкати не може вплинути на стани до GVT, що забезпечує рух вперед.
2. Колекція даних: пам'ять, що використовується для старих збережених станів та застарілих повідомлень до того, як GVT можна буде безпечно відновити.

Міркування щодо продуктивності

Оптимістичне виконання Time Warp забезпечує високий ступінь паралелізму, особливо в моделях зі слабозв'язаними потоками подій. Однак його продуктивність залежить від кількох факторів:

- частота відкатів, часті порушення причинно-наслідкових зв'язків знижують ефективність;

- стратегія збереження станів, повна чи інкрементальна контрольна точка впливає як на час відкату, так і на витрати пам'яті;
- каскадні ефекти, анти-повідомлення можуть спричиняти поширені відкати, що призводить до додаткових витрат.

Незважаючи на ці труднощі, емпіричні результати показують, що Time Warp часто перевершує консервативні підходи на великомасштабних багатопроцесорних системах, де уникнення простою є критично важливим.

Алгоритм Time Warp фундаментально змінює концепцію синхронізації в паралельному моделюванні. Дозволяючи процесам виконуватися без глобальних гарантій безпеки та використовуючи відкат з антиповідомленнями для виправлення помилок, він досягає коректності без шкоди для паралельності. Локальні механізми керування гарантують, що кожен LP може керувати своїм власним спекулятивним виконанням, тоді як глобальні механізми, такі як GVT, забезпечують загальний прогрес та узгодженість.

Таким чином, Time Warp забезпечує загальний, масштабований та ефективний протокол синхронізації для паралельного моделювання дискретних подій, закладаючи основу для передового паралельного моделювання в науці та техніці.

Порівняння із запропонованим методом

Паралельне та розподілене моделювання вже давно вимагає ефективних механізмів синхронізації для забезпечення узгодженості між паралельними логічними процесами. Серед класичних рішень алгоритм Time Warp був найвпливовішим, запроваджуючи оптимістичне виконання з відкатом. Натомість, нещодавня робота досліджує використання реактивних потоків, універсального протоколу синхронізації, який широко використовується в сучасних розподілених системах.

Time Warp спирається на оптимістичну синхронізацію: кожен логічний процес протікає незалежно, спекулятивно виконуючи події. Коли виявляється помилка причинно-наслідкового зв'язку, система повертається до попереднього узгодженого стану. Це вимагає захисту від повідомлень, збереження стану та розширеного управління відкатами [6]. Реактивні потоки, навпаки, використовують синхронізацію, керовану зворотним тиском, де дані передаються лише тоді, коли процеси вище та нижче за течією готові.

У Time Warp процеси працюють на повних станах, а відкати вимагають частих контрольних точок станів. Це часто призводить до високого споживання пам'яті. Підхід реактивних потоків розкладає стани системи на підстани з унікальними ключами, забезпечуючи модульність та узгодженість. Суперечливі підстани фільтруються під час побудови графа, що усуває необхідність коригувальних відкатів.

Технологія Time Warp забезпечує правильність, скасовуючи несумісні події, але каскадні відкати «ефект доміно» можуть значно погіршити продуктивність. Реактивні потоки уникають цього за своєю природою: як тільки граф переходів побудований послідовно, безпомилкові обчислення відбуваються детерміновано. Помилки запобігаються, а не виправляються, хоча відмовостійкість до збоїв на системному рівні залишається напрямком майбутніх досліджень.

Обидва методи спрямовані на використання паралельних архітектур, але їхні профілі масштабованості відрізняються. Time Warp, в принципі, може широко масштабуватися, але страждає від накладних витрат на відкат зі збільшенням кількості процесорів. Реактивні потоки масштабуються більш природно з багатоядерними та розподіленими системами, оскільки такі фреймворки, як АККА, забезпечують вбудоване балансування та розподіл навантаження. Крім того, реактивні потоки підтримують інтерактивне моделювання, що дозволяє

синхронізацію в реальному часі із зовнішніми вхідними даними те, що не реалізовано в Time Warp.

Впровадження Time Warp вимагає створення спеціалізованого рівня синхронізації, що робить його складним та схильним до помилок. Реактивні потоки використовують існуючі фреймворки промислового рівня, дозволяючи дослідникам та розробникам зосередитися на моделюванні, а не на синхронізації. Це робить підхід реактивних потоків більш доступним та практичним для сучасних застосувань.

Технологія Time Warp залишається віхою в дослідженнях паралельного моделювання, пропонуючи основу для оптимістичної синхронізації. Однак її залежність від відкатів створює накладні витрати та складність, що обмежує її практичну масштабованість. Реактивні потоки представляють альтернативну парадигму: модульну, з урахуванням помилок та за своєю суттю масштабовану, з синхронізацією, інтегрованою в широко використовувані програмні рамки. Замінюючи спекулятивне виконання на контрольований потік повідомлень, реактивні потоки узгоджують дослідження моделювання із сучасними практиками розподілених обчислень, пропонуючи переконливий шлях уперед.

1.2.2 Фреймворк RxHLA

Зростаюча складність та неоднорідність сучасних систем роблять розподілене моделювання критично важливим інструментом для проектування, верифікації та валідації систем. Стандарт архітектури високого рівня (HLA) вже давно використовується для таких цілей, забезпечуючи сумісність та повторне використання компонентів моделювання. Однак, створення моделювань на основі HLA залишається складним завданням через необхідність значної експертизи програмування та труднощі з керуванням асинхронними потоками зв'язку.

Розробники часто стикаються з так званою проблемою «пекла зворотних викликів», коли глибоко вкладені зворотні виклики ускладнюють підтримку коду.

Щоб вирішити ці проблеми, дослідники поєднали дві взаємодоповнюючі технології: метод MONADS та фреймворк RxHLA.

MONADS

MONADS (Архітектура MOdel-driveN для розподіленого моделювання) застосовує принципи системної інженерії, керованої моделями (MDSE), для поєднання високорівневих системних моделей та виконуваного моделювання. Починаючи з моделей SysML, MONADS автоматично генерує великі частини коду HLA- моделювання через серію перетворень модель-модель та модель-текст. Це значно зменшує зусилля, необхідні системним інженерам, які можуть зосередитися на проектуванні системи, а не на деталях низькорівневого моделювання.

RxHLA

RxHLA впроваджує реактивне програмування в область HLA [7]. Замість того, щоб покладатися на зворотні виклики, він керує подіями моделювання як спостережуваними потоками даних (Observable<DataPacket>). Розробники можуть фільтрувати, трансформувати та об'єднувати ці потоки за допомогою простих операторів, що призводить до чистішого та модульнішого коду моделювання. Фреймворк надає послуги для обробки часу моделювання, моделей об'єднаних об'єктів та трансформації даних, забезпечуючи повну сумісність зі стандартом IEEE 1516-2010 HLA.

Інтеграція MONADS та RxHLA

Інтеграція RxHLA в метод MONADS покращує генерацію коду за допомогою реактивних функцій, пропонуючи три різні стратегії:

1. Без RxHLA – традиційний HLA-код, що залишає управління подіями розробникам.

2. Автоматична конфігурація RxHLA – автоматично вставляє спостережувані RxHLA у згенерований код, спрощуючи обробку подій.
3. Налаштовувана конфігурація RxHLA – розширює автоматичний підхід, дозволяючи визначати поведінку та потоки зв'язку за допомогою діаграм послідовностей UML, які потім перетворюються на реактивний код.

Тематичне дослідження системи управління повітряним рухом продемонструвало, як ці стратегії призводять до поступово більш гнучкого та зручного для розробників коду моделювання. Реактивні оператори, такі як `filter` та `map`, можна безпосередньо застосовувати до подій моделювання, демонструючи практичні переваги цього підходу.

Поєднуючи MONADS з RxHLA, запропонований метод зменшує бар'єри для побудови розподілених моделей на основі HLA. Він дозволяє системним інженерам переходити від високорівневих специфікацій SysML до потоково-керованого коду моделі з мінімальним ручним втручанням. Такий підхід не тільки підвищує продуктивність, але й робить асинхронну обробку подій більш ефективною та зручною в обслуговуванні.

Порівняння із запропонованим методом

Паралельне та розподілене моделювання часто вимагає надійних механізмів синхронізації. У той час як традиційні алгоритми, такі як Time Warp, покладаються на оптимістичний відкат, новіші парадигми інтегрують реактивні принципи в інфраструктури моделювання. Двома помітними підходами є RxHLA, реактивне розширення стандарту архітектури високого рівня (HLA) IEEE 1516, та універсальний підхід реактивних потоків, досліджений у нещодавніх дослідженнях. Обидва прагнуть використати переваги реактивного програмування, але вони суттєво відрізняються за обсягом, складністю та гнучкістю.

RxHLA адаптує стандарт HLA для розподіленого моделювання, включаючи реактивні конструкції. Він зберігає структуру федерацій HLA, де моделювання (федерації) взаємодіють через інфраструктуру середовища виконання, але вводить реактивні розширення для поширення подій та синхронізації.

Підхід реактивних потоків, навпаки, не прив'язується до жодного специфічного для моделювання стандарту. Натомість він використовує протокол реактивних потоків загального призначення, реалізований у бібліотеках, таких як АККА Streams, як універсальний механізм синхронізації. Це відокремлює паралельне моделювання від обмежень HLA, роблячи його більш широко застосовним.

У RxHLA синхронізація залишається вбудованою в інфраструктуру середовища виконання HLA, що може зробити систему жорсткою та складною. Реактивні потоки забезпечують механізми зворотного тиску, пропонуючи гнучкий та детальний контроль над потоком повідомлень без спеціалізованих протоколів моделювання. Крім того, розкладання станів на підстави з унікальними ключами забезпечує модульну побудову графів переходів, що не є властивою архітектурі RxHLA.

RxHLA успадковує як сильні, так і слабкі сторони HLA. Він пропонує сильну сумісність між федеративними моделями, але за рахунок низького рівня складності та великих накладних витрат часу виконання. Підхід реактивних потоків, спираючись на протокол загального призначення, зменшує складність та використовує існуючі промислові фреймворки для розподілених систем. Таким чином, розробники зосереджуються на моделюванні, а не на управлінні протоколами, що спрощує впровадження та масштабування.

Обидва підходи охоплюють реактивність, але їхня гнучкість відрізняється. RxHLA впроваджує реактивність в межах фіксованих обмежень федерацій HLA, що

робить її придатною для жорстко регульованих середовищ моделювання. Реактивні потоки дозволяють інтерактивне моделювання за допомогою шаблонів push/pull, безперешкодно інтегруючи зовнішні джерела даних (наприклад, датчики, дані користувача) та підтримуючи реагування в реальному часі в ширшому діапазоні застосувань.

RxHLA являє собою важливу спробу модернізувати HLA за допомогою реактивних принципів, покращуючи подієво-кероване моделювання, зберігаючи при цьому відповідність стандарту IEEE 1516. Однак це пов'язано зі складністю та обмеженою адаптивністю за межами доменів на основі HLA. Підхід реактивних потоків, шляхом відокремлення синхронізації від стандартів моделювання, забезпечує легшу, гнучкішу та масштабованішу структуру, об'єднуючи класичне моделювання із сучасними методами розподілених обчислень. Хоча RxHLA найкраще підходить для контекстів моделювання на основі федерації, реактивні потоки пропонують узагальнюючу основу для паралельного моделювання в різних областях.

1.2.3 Архітектура CQRS + ES

Хмарні обчислення змінили спосіб проектування та експлуатації програмних систем сучасними підприємствами. Маючи практично необмежені обчислювальні ресурси, доступні на вимогу, організації тепер стикаються із завданням створення додатків, які можуть повністю використовувати ці можливості, залишаючись стійкими, ефективними та адаптивними. Традиційні багаторівневі клієнт-серверні архітектури часто не справляються з цими умовами, намагаючись забезпечити еластичність та відмовостійкість, очікувані у великомасштабних середовищах. Щоб вирішити ці проблеми, дослідники з Університету науки і технологій AGH у співпраці з Lufthansa Systems дослідили, чи можна ефективно поєднати два архітектурні шаблони розділення відповідальності за команди та запити (CQRS) та

джерело подій (ES) відповідно до принципів реактивного маніфесту для створення справді масштабованого хмарного додатку [8].

Тематичне дослідження планування польотів

Дослідження було зосереджено на розробці прототипу системи планування польотів, що є сферою, що представляє значну складність через величезну кількість взаємопов'язаних даних, які вона повинна обробляти. Розклади польотів включають сотні літаків, тисячі маршрутів та численні обмеження, такі як час наземного обслуговування, безперервність ротації та унікальність ідентифікатора польоту. Така система повинна обробляти як інтерактивні оновлення від людей-планувальників, так і обчислювально важкі фонові перевірки, зберігаючи при цьому низьку затримку. Ці характеристики зробили її ідеальним кандидатом для тестування потенціалу масштабованості CQRS та Event Sourcing.

В основі принципу CQRS лежить розділення системних обов'язків: команди, що змінюють стан, обробляються незалежно від запитів, що отримують дані. Такий поділ дозволяє оптимізувати операції читання та запису по-різному, часто навіть дозволяючи використовувати окремі бази даних, адаптовані до кожної мети. Джерела подій доповнюють цю модель, записуючи кожну зміну в системі як незмінний журнал подій, а не зберігаючи лише кінцевий стан. Такий підхід має численні переваги: повну відстежуваність усіх операцій, можливість перебудови стану в будь-який час та гнучкість у впровадженні нових моделей читання ще довго після розгортання. Разом ці шаблони тісно узгоджуються з Реактивним маніфестом, який наголошує на системах, що керуються повідомленнями, є еластичними, стійкими та адаптивними.

Прототип з використанням бібліотеки Akka

Прототип було створено з використанням сучасних реактивних технологій. Інструментарій Akka, заснований на акторній моделі, забезпечив основу для

одночасної та розподіленої обробки команд. Сторона запису використовувала шардінг для ефективного розподілу робочих навантажень між вузлами, а кешування в пам'яті покращило час відгуку. Сторона читання спиралася на Neo4j, графову базу даних, яка добре підходить для складних запитів перевірки, необхідних для домену планування. Для реалізації сховища подій команда використовувала Apache Cassandra для збереження даних та Apache Kafka як надійну чергу повідомлень, що забезпечувало надійне поширення подій між компонентами. REST API та балансування навантаження були інтегровані через Akka Clustering та Nginx, створюючи повністю реактивну, готову до хмарних технологій систему.

Тести масштабованості проводилися в хмарному середовищі, що складалося з 15 віртуальних машин. Результати були вражаючими. На стороні читання система продемонструвала майже ідеальну лінійну масштабованість: із додаванням нових вузлів пропускна здатність пропорційно зростала без погіршення часу відгуку. На стороні запису додавання вузлів зменшило час відгуку та усунуло тайм-аути під час великих навантажень. Ці результати підтверджують, що поєднання CQRS та Event Sourcing не лише підтримує горизонтальну масштабованість, але й підвищує загальну стійкість та швидкість реагування системи.

Недоліки та висновки

Звичайно, цей підхід не позбавлений своїх недоліків. Забезпечення остаточної узгодженості вимагає від розробників переосмислення звичних уявлень про потік керування та прийняття короточасних розбіжностей між моделями читання та запису. Додаткові проблеми включають накладні витрати на керування знімками подій, обробку версій подій з часом та роботу з відносною незрілістю інструментів та передових практик у цій галузі. Тим не менш, дослідники стверджують, що переваги еластичності, можливості аудиту, відмовостійкості та гнучкості значно переважають ці обмеження, особливо для хмарних застосунків, де масштабованість є першорядною.

Наслідки цієї роботи виходять за межі авіаційної галузі. Багато корпоративних сфер, від фінансів до логістики, стикаються з аналогічними проблемами масштабу та складності та можуть отримати вигоду від архітектур, заснованих на CQRS та Event Sourcing. Lufthansa Systems, переконана результатами дослідження, вже зобов'язалася включити ці шаблони в розробку своїх хмарних додатків наступного покоління. Заглядаючи в майбутнє, автори бачать особливий потенціал в інтеграції технологій обробки потоків, таких як Kafka Streams або Apache Samza, в системи CQRS+ES. Така інтеграція може забезпечити сильніші гарантії узгодженості та дозволити створювати надійні, самомасштабовані додатки, які є ключовою віхою в еволюції хмарного програмного забезпечення.

На завершення, це тематичне дослідження демонструє, що CQRS та Event Sourcing, за умови впровадження відповідно до реактивних принципів, формують потужну основу для створення масштабованих та стійких хмарних додатків. Хоча розробникам доводиться орієнтуватися в нових парадигмах проектування та знаходити компроміси, потенційні покращення продуктивності, еластичності та зручності обслуговування сигналізують про багатообіцяюче майбутнє цих архітектурних шаблонів як у дослідженнях, так і в галузевій практиці.

Порівняння із запропонованим методом

З розвитком паралельних та розподіленого моделювання дослідники звертаються за межі традиційних протоколів синхронізації до архітектур, що підтримують масштабованість та модульність. Двома перспективними напрямками є шаблон CQRS + Event Sourcing (CQRS + ES) та підхід реактивних потоків. Обидва підтримують ідеї розв'язання та модульності, проте вони виникають у різних контекстах: CQRS + ES з проектування корпоративних систем, а реактивні потоки з розподіленого реактивного програмування. Їхнє порівняння підкреслює компроміси між специфічними для предметної області архітектурними шаблонами та протоколами синхронізації загального призначення.

CQRS + ES розділяє поведінку системи на два окремі потоки:

- розділення відповідальності за запити команд (CQRS): розділяє операції на команди (зміни стану) та запити (зчитування стану);
- джерело подій (ES): записує всі зміни як незмінні події, з яких можна реконструювати стан системи.

Підхід реактивних потоків, навпаки, моделює еволюцію системи як граф переходів підстанів. Синхронізація здійснюється за допомогою передачі повідомлень, керованої зворотним тиском, що підтримується такими фреймворками, як AKKA Streams, без спеціалізованого архітектурного розділення команд і запитів.

CQRS + ES досягає узгодженості, гарантуючи, що кожна зміна стану є подією в незмінному журналі. Хоча це надійно, це вимагає постійного відтворення або проектування подій для відновлення стану, що може стати ресурсомістким.

Реактивні потоки використовують підстани з унікальними ключами для представлення еволюції системи. Замість відтворення подій, моделювання заздалегідь буде узгоджені графіки переходів, усуваючи суперечності та уникаючи накладних витрат на відновлення повних станів.

CQRS + ES сяє в галузях, де аудит, відстежуваність та повторюваність є критично важливими (наприклад, фінанси, корпоративні додатки). Однак його безпосереднє застосування до наукових або фізичних моделей часто створює зайву складність.

З іншого боку, реактивні потоки є легкими та не залежать від предметної області. Вони повторно використовують усталені розподілені фреймворки для реалізації синхронізації, звільняючи розробників від навантаження, пов'язаного з протоколом, та зосереджуючись на самій проблемі моделювання.

Системи CQRS + ES зазвичай обробляють події асинхронно, з кінцевою узгодженістю між представленнями даних. Це добре працює для корпоративних робочих навантажень, але створює затримку та обмежує пряму реакцію в реальному часі.

Реактивні потоки власно підтримують інтерактивне моделювання за допомогою шаблонів push та pull. Це робить їх добре придатними для сценаріїв реального часу або керованих користувачем сценаріїв, де моделювання повинні постійно адаптуватися до зовнішніх джерел даних.

Як CQRS + ES, так і реактивні потоки використовують модульність та подієвий підхід, але їхні цілі проектування розходяться. CQRS + ES наголошує на відстежуваності подій та довгостроковій узгодженості, що робить його потужним у транзакційних або бізнес-системах, але громіздким для високопродуктивного моделювання. Реактивні потоки, навпаки, забезпечують універсальний, масштабований протокол синхронізації, ідеальний для наукових та інженерних моделей, що вимагають швидкості реагування, ефективності та інтеграції із сучасними розподіленими платформами.

1.2.4 Фреймворк акторів HPC (високопродуктивні обчислення)

Зростаючий попит на реалістичні та детальні моделі фізичних та кіберфізичних систем вимагає обчислювальних підходів, які можуть масштабуватися далеко за межі традиційної паралельної обробки. Високопродуктивні обчислення (HPC) вже давно є основою таких моделей, але ефективність великомасштабних моделей часто обмежується бар'єрами синхронізації, які змушують усі процеси просуватися в синхронно. Перспективним напрямком подолання цього обмеження є використання десинхронізованого поширення інформації.

У десинхронізованій структурі високопродуктивних обчислень (HPC) компоненти моделювання не чекають глобальної синхронізації на кожному обчислювальному кроці [9]. Натомість процеси просуваються у власному темпі, асинхронно поширюючи оновлення між розподіленими вузлами. Така конструкція зменшує час простою та запобігає затримці швидших вузлів найповільнішими учасниками. Для підтримки узгодженості інтегровані механізми вирішення конфліктів або відкату, що дозволяє моделюванню виправляти неправильно впорядковану або затриману інформацію, не зупиняючи глобальний прогрес.

Головною перевагою десинхронізованого поширення є масштабованість. Усуваючи жорсткі бар'єри, фреймворк дозволяє моделюванням використовувати десятки тисяч ядер, графічних процесорів або гібридних прискорювачів з мінімальними накладними витратами на зв'язок. Це призводить до майже лінійного прискорення в багатьох сценаріях. Такий підхід також підвищує стійкість: нерівномірні обчислювальні навантаження на вузлах не зупиняють всю систему, що робить її добре придатною для гетерогенних середовищ.

Масштабне паралельне моделювання

Парадигма акторів, спочатку розроблена в контексті паралельного програмування, знайшла міцне підґрунтя у високопродуктивних обчисленнях (HPC). У фреймворках акторів обчислення виражаються як сукупність акторів незалежних сутностей, які інкапсулюють стан і поведінку, спілкуючись виключно через асинхронну передачу повідомлень. Ця модель пропонує природне сумісність з масштабними моделями та робочими навантаженнями, що потребують інтенсивного використання даних.

Принципи актор-орієнтованого високопродуктивного обчислення (HPC):

- інкапсуляція: Кожен актор володіє своїм станом і виконує завдання незалежно;

- асинхронний зв'язок: Інформація передається через неблокуючу передачу повідомлень, мінімізуючи глобальну синхронізацію;
- динамічне планування: Актори можуть мігрувати між потоками, вузлами або прискорювачами, що дозволяє ефективно використовувати різноманітні ресурси;
- еластична масштабованість: Нові актори можуть створюватися або знищуватися на вимогу для адаптації до змін робочого навантаження;
- ізоляція несправностей: Несправності впливають лише на відповідного актора, підтримуючи стійкість у розподілених середовищах.

Переваги, застосування та майбутні виклики

Фреймворки акторів зменшують вузькі місця синхронізації та дозволяють дрібнозернистий паралелізм, іноді масштабуючи його до мільйонів акторів в одній програмі. Динамічне балансування навантаження через міграцію акторів допомагає підтримувати високий рівень використання у великих кластерах, тоді як абстракція через акторів зміщує фокус розробника з низькорівневих викликів MPI на моделі поведінки та комунікації вищого рівня.

Фреймворки акторів HPC застосовуються в таких галузях, як молекулярна динаміка, аналітика графів, агентне моделювання та гібридна координація CPU/GPU. Їхня гнучкість робить їх особливо корисними в моделюванні складних, адаптивних або нерегулярних систем.

Незважаючи на свої переваги, акторні фреймворки стикаються з труднощами у мінімізації накладних витрат на комунікацію для дрібнозернистих акторів, інтеграції зі застарілими кодами на основі MPI та забезпеченні відтворюваності в асинхронних середовищах. Тим не менш, їхня здатність абстрагувати низькорівневі деталі, забезпечуючи при цьому масштабованість, позиціонує їх як потужну парадигму для майбутніх HPC-застосунків.

Порівняння із запропонованим методом

Паралельне моделювання вимагає архітектур, які ефективно розподіляють завдання між багатоядерними та розподіленими середовищами. Акторні фреймворки, що широко використовуються у високопродуктивних обчисленнях (HPC), та підхід реактивних потоків [10-12] використовують парадигми передачі повідомлень, але з різними акцентами. У той час як акторні системи HPC (наприклад, актори АККА [13-16] без потоків, Charm++ або користувацькі середовища виконання HPC) зосереджені на явному управлінні акторами, реактивні потоки абстрагують синхронізацію через потоки, керовані зворотним тиском. Порівняння цих двох підходів підкреслює компроміс між керуванням та абстракцією.

Фреймворки акторів HPC побудовані навколо моделі акторів, де актори є незалежними сутностями, які взаємодіють виключно за допомогою асинхронної передачі повідомлень. Ця модель добре підходить для масового паралельного виконання, ізоляції помилок та розподілу. Розробники явно керують ієрархіями акторів, протоколами повідомлень та політиками планування.

Підхід реактивних потоків базується на принципах акторів, але підвищує рівень абстракції: він визначає потоки повідомлень із вбудованим керуванням потоком. Логічні процесори (подібно до акторів) об'єднанні в обчислювальні графи, де синхронізація та зворотний тиск обробляються середовищем виконання, а не протоколами, керованими розробником.

Фреймворки акторів HPC є потужними, але ресурсоемними для розробників, вимагаючи явного обґрунтування паралельності, розподілу та планування. Вони чудово працюють, коли важливий детальний контроль, але можуть вносити значні шаблонні зміни.

Реактивні потоки повторно використовують основу акторів, але забезпечують декларативну абстракцію вищого рівня. Розробники можуть складати графи моделювання як потоки даних, покладаючись на середовище виконання для оптимізації передачі повідомлень, розподілу навантаження та зворотного тиску. Це зменшує складність та пришвидшує створення прототипів.

Хоча акторні фреймворки підтримують інтерактивність, це не є їхньою основною метою проектування; зовнішні вхідні дані мають бути явно змодельовані як повідомлення акторів.

Реактивні потоки, навпаки, вбудовують інтерактивність безпосередньо в семантику push/pull, забезпечуючи оперативність реагування в реальному часі. Зовнішні джерела даних можуть бути безперешкодно інтегровані в графи моделювання, що робить реактивні потоки більш придатними для інтерактивних або адаптивних моделей.

Як акторні фреймворки НРС, так і реактивні потоки покладаються на акторну модель у своїй основі. Акторні фреймворки пропонують максимальну гнучкість і контроль за рахунок складності, що робить їх ідеальними для ретельно оптимізованих НРС-застосунків. Реактивні потоки абстрагують значну частину цієї складності, вбудовуючи синхронізацію, зворотний тиск та інтерактивність у середовище виконання. Це позиціонує їх як масштабоване, зручне для розробників рішення для сучасного паралельного моделювання, що поєднує ретельне моделювання з промисловими розподіленими обчислювальними практиками.

ВИСНОВКИ ДО РОЗДІЛУ 1

Огляд, представлений у цьому розділі, демонструє широту та різноманітність підходів, розроблених для підтримки паралельного моделювання динамічних систем. Паралельне моделювання стало фундаментальним інструментом обчислювальної науки, поєднуючи теоретичні методи з практичними застосуваннями. Його важливість визначається не лише здатністю прискорювати обчислення, але й здатністю робити можливим вивчення дуже складних, взаємопов'язаних та великомасштабних систем, які стають все більш актуальними в сучасній науці, техніці та промисловості.

Розвиток сучасних парадигм програмування відкрив нові перспективи щодо архітектури моделювання. Фреймворк RxHLA, завдяки своїй інтеграції з MONADS, показує, як реактивне програмування може модернізувати давно встановлений стандарт HLA, спрощуючи обробку подій та покращуючи модульність. CQRS у поєднанні з Event Sourcing демонструє, як архітектурні принципи корпоративних систем можна адаптувати до великомасштабних хмарних додатків, забезпечуючи як аудитабельність, так і стійкість. Фреймворки НРС на основі акторів ще більше розширюють масштабованість, усуваючи глобальні бар'єри синхронізації та забезпечуючи дрібнозернисте, десинхронізоване поширення інформації по величезних обчислювальних ресурсах. Кожен із цих підходів розглядає конкретні обмеження попередніх методів і надає розуміння того, як моделювання може розвиватися, щоб відповідати вимогам гетерогенних та розподілених середовищ.

Однак аналіз також показує, що ці методи не позбавлені проблем. RxHLA успадковує жорсткість стандарту HLA і може стати громіздким поза спеціалізованими областями. CQRS + ES, хоча й потужні для транзакційних або корпоративних контекстів, вносять непотрібну складність при застосуванні до наукових або інженерних моделювань. Фреймворки на основі акторів чудово

масштабуються, але часто перекладають тягар синхронізації та управління комунікацією на розробників, ускладнюючи реалізацію. Таким чином, хоча кожен підхід пропонує цінні ідеї, жоден окремо не вирішує постійну суперечність між коректністю, масштабованістю та простотою використання.

Це відкриває шлях до дослідження альтернативних парадигм. Підхід, заснований на реактивних потоках, як запропоновано в цій роботі, знаходиться в межах цього ландшафту, що розвивається. На відміну від традиційних протоколів синхронізації, які спираються або на спекулятивні відкати, або на жорсткі правила федерації, реактивні потоки вбудовують синхронізацію безпосередньо в потік даних через механізми зворотного тиску та контрольованого поширення повідомлень. Це усуває необхідність у великій обробці відкатів, зменшує накладні витрати на комунікацію та забезпечує безперешкодну інтеграцію з широко використовуваними розподіленими фреймворками, такими як Akka Streams.

Тому розробка методу побудови паралельних чисельних моделей динамічних систем на базі протоколу реактивних потоків є актуальним науковим завданням.

РОЗДІЛ 2

ВИЗНАЧЕННЯ БАЗОВИХ ЕЛЕМЕНТІВ МЕТОДІВ ПАРАЛЕЛЬНОГО МОДЕЛЮВАННЯ

Для розробки формальної основи моделювання важливо почати з ретельного опису фундаментальних елементів, що складають модель. У розділі 3 встановлюється теоретична основа для представлення змодельованого об'єкта в математичних термінах, зосереджуючись на тому, як його властивості, поведінку та перетворення можна фіксувати та аналізувати.

Спочатку розглядаємо роль часу в моделюванні, виділяючи, коли він є обов'язковою властивістю (як у фізичних системах), а коли його можна абстрагувати. Потім у розділі вводиться концепція представлення змодельованого об'єкта як набору змінних, кожна з яких пов'язана з областю можливих значень, та класифікація їх на незалежні змінні, залежні змінні та параметри. Ці змінні визначають простір станів, в межах якого поведінку змодельованого об'єкта можна описати як траєкторії переходів станів.

Спираючись на цю основу, у розділі досліджуються два взаємодоповнюючі підходи до вираження залежності змінних: як набір функцій (наприклад, аналітичні рівняння, задачі Коші) або як набір підстанів, які розкладають складні стани на узгоджені, несуперечливі частини. Поняття відображення вводиться як запис реальних вимірювань змодельованого об'єкта, тоді як модель формалізується як імітація цих залежностей, побудована незалежно від безпосереднього вимірювання. Нарешті, моделювання визначається як процес використання моделі для обчислення невідомих залежних змінних з відомих незалежних змінних та параметрів.

Синтезуючи ці елементи, відображення, модель та моделювання, цей розділ пропонує єдину методологію для представлення та аналізу поведінки об'єктів. Ця

структура не лише забезпечує внутрішню узгодженість між концептуальними та математичними моделями, але й закладає основу для подальших обговорень паралельного моделювання та структурованих результатів.

Також, у розділі 5 представляємо простий приклад змодельованого об'єкта, побудову моделі та моделювання об'єкта.

2.1. Моделювання об'єктів та фізичного часу

Моделювання це, в найширшому сенсі, процес побудови моделі об'єкта. [17, 18] Існує багато різновидів моделювання, але в цій статті обговоримо лише математичне моделювання. Фактично, «модель» це спеціальний об'єкт (у нашому випадку математичний об'єкт), який відображає інший об'єкт, який називатимемо змодельованим об'єктом. Модель відображає набір властивостей змодельованого об'єкта, їх тип, структуру та взаємозв'язки. [18, 19, 20, 21]

Будь-які зміни, що відбуваються в межах будь-якого фізичного об'єкта, вимагають певного часу у фізичному світі. Тим не менш, у багатьох випадках час можна ігнорувати під час моделювання. Це особливо стосується випадків, коли модельований об'єкт не є фізичним. Наприклад, якщо моделюємо певну логічну схему і нас цікавить лише набір вихідних сигналів відносно цього набору вхідних сигналів, то часом можна легко ігнорувати. Далі в цій статті розглядатимемо час як одну з властивостей модельованого об'єкта. Час є обов'язковим для фізичних об'єктів, але в загальному випадку він може бути включений або не включений до моделі.

Побудовану модель можна використовувати в моделюванні, яка по суті є імітацією поведінки модельованого об'єкта (тобто імітацією зміни його властивостей відносно інших об'єктів) [22-26]. У цій статті обговорюватимемо лише числове (комп'ютерне) моделювання.

Наприклад, можемо моделювати рух фізичного об'єкта шляхом імітації змін його координат відносно часу. Результатом є сукупність усіх змін, що відбуваються з об'єктом під час моделювання. Цей набір змін можна використовувати для аналізу та прогнозування поведінки об'єкта. Такий результат є цінним, оскільки його можна отримати без безпосереднього використання самого змодельованого об'єкта.

Моделювання вирішує певну практичну або теоретичну проблему; надалі називатимемо це завдання проблемою моделювання.

Важливою властивістю моделі є її узгодженість з модельованим об'єктом, швидкість різниці між станом і поведінкою, передбаченими моделлю, і фактичним станом і поведінкою модельованого об'єкта.

На практиці бажано, щоб наша модель якомога точніше відображала змодельований об'єкт, щоб отримати найточніший результат моделювання; проте маємо обмежені ресурси, такі як час розробки моделі та обчислювальна потужність. Це означає, що завжди буде компроміс між точністю та вартістю побудованої моделі. Таким чином, більшість моделей будуть несумісними із змодельованим об'єктом. Рівень невідповідності залежатиме, перш за все, від вимог, визначених проблемою, яку необхідно вирішити за допомогою моделювання.

2.2. Набір змінних V як представлення модельованого об'єкта

Змінна, це математичний об'єкт, що позначається деяким символом і може бути пов'язана з одним значенням з набору значень. Набір значень, які можна пов'язати зі змінною, називатиметься її доменом визначення.

Не накладаємо жодних обмежень на типи значень, що входять до області визначення змінної. Наприклад, набір цілих чисел можна вибрати як область визначення змінної, яка моделює вік людини.

$$\mathbb{N} := \left\{ \begin{array}{c} 0 \\ 1 \\ 2 \\ \dots \\ \infty \end{array} \right\}, \quad (2.1)$$

або це може бути набір значень, таких як

$$\left\{ \begin{array}{c} \text{"young"} \\ \text{"middle"} \\ \text{"old"} \end{array} \right\},$$

або будь-який інший набір значень, зручний з точки зору задачі моделювання.

Певні змодельовані об'єкти можна математично представити як набір змінних, що відображають властивості (атрибути) цього об'єкта. [27] Такий набір називатимемо змінними моделі.

Визначення: Давайте визначимо змінні моделі

$$V := \left[\begin{array}{c} v_1 : \mathbb{V}_1 \\ v_2 : \mathbb{V}_2 \\ \dots \\ v_n : \mathbb{V}_n \end{array} \right], \quad (2.2)$$

де $n \in \mathbb{N}$, як набір з n змінних v_i , яким можна пов'язати значення з множини (домену) можливих значень \mathbb{V}_i . Більше того, кожна v_i відображає властивість модельованого об'єкта.

У більшості практичних випадків V буде абстрактним. Це означає, що з усіх можливих властивостей модельованого об'єкта вибираємо лише ті, які мають значення в контексті задачі моделювання. Процес вибору значущих властивостей називається побудовою концептуальної моделі. [28-31]

Набір змінних та їхніх доменів повністю визначається задачею моделювання та відображеними властивостями. Таким чином, має існувати спосіб перетворення кількості властивості у значення з області визначення змінної. Називатимемо таке перетворення вимірюванням.

Як приклад, розглянемо резервуар, що наповнюється водою (рис. 2.1). У найпростішому випадку його можна абстрактно представити чотирма числовими змінними:

- v/t – швидкість наповнення в літрах за секунду;
- v_{tank} – об'єм резервуара в літрах;
- v_{water} – об'єм води в резервуарі також у літрах;
- t – час наповнення в секундах.

Резервуар повинен бути оснащений такими приладами для отримання числових значень вимірюваних властивостей:

- витратомір для вимірювання v/t ;
- рівнемір для вимірювання v_{water} ;
- секундомір для вимірювання t .

Об'єм v_{tank} – це відома константа.

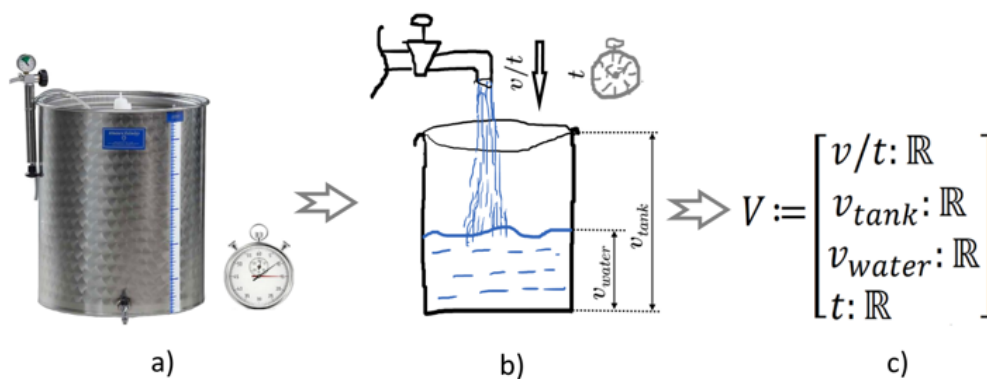


Рис. 2.1 Резервуар, що наповнюється водою: а) Змодельований (фізичний) об'єкт; б) Концептуальна модель; в) Математична модель

Змінні V можна розділити на три типи:

1. Незалежні змінні: Відображають відомі властивості або вхідні дані. Типовим прикладом є змінна, яка відображає властивість часу.

2. Залежні змінні: Відображають невідомі властивості або вихідні дані. Їхні значення змінюються залежно від зміни незалежних змінних. Наприклад, відстань, пройдена фізичним об'єктом, залежить від часу та його швидкості.
3. Параметри: Значення параметрів не залежать від незалежних та залежних змінних; це означає, що параметри не змінюються протягом усього моделювання. Як приклад можна розглянути гравітацію.

Визначення: Визначимо підмножину незалежних змінних

$$X := \begin{bmatrix} x_1 : \mathbb{X}_1 \\ \dots \\ x_n : \mathbb{X}_n \end{bmatrix}, \quad (2.3)$$

де $x_i : \mathbb{X}_i \in V$ та $0 < n < |V|$. Підмножина залежних змінних задається як

$$Y := \begin{bmatrix} y_1 : \mathbb{Y}_1 \\ \dots \\ y_n : \mathbb{Y}_n \end{bmatrix}, \quad (2.4)$$

де $y_i : \mathbb{Y}_i \in V$ та $0 < n < |V|$. Підмножина параметрів задається як

$$G := \begin{bmatrix} g_1 : \mathbb{G}_1 \\ \dots \\ g_n : \mathbb{G}_n \end{bmatrix}, \quad (2.5)$$

де $g_i : \mathbb{G}_i \in V$ та $0 \leq n < |V|$, такі що

$$\begin{aligned} X, Y, G &\subset V, \\ X \cup Y \cup G &= V; \\ (X \cap Y) \cup (X \cap G) \cup (G \cap Y) &= \emptyset. \end{aligned} \quad (2.6)$$

Тип кожної змінної залежить від конкретної задачі моделювання. Тому в цій статті вважатимемо, що будь-яка змінна v_i може мати будь-який із цих трьох типів.

2.3. Стани \mathfrak{B} , \mathfrak{X} , \mathfrak{Y} , та \mathfrak{G} як значення змінних V , X , Y , та G

Припускаємо, що змодельований об'єкт завжди має певний стан, тобто кожна властивість об'єкта має певну величину. Це можна представити як набір певних

значень v_i з доменів \mathbb{V}_i , пов'язаних з набором змінних v_i . Такий набір пов'язаних значень називатимемо станом моделі.

Визначення: Давайте визначимо стан моделі

$$V = \mathfrak{V} := \begin{bmatrix} v_1 = v_1 \in \mathbb{V}_1 \\ v_2 = v_2 \in \mathbb{V}_2 \\ \dots \\ v_n = v_n \in \mathbb{V}_n \end{bmatrix}, \quad (2.7)$$

де $v_i \in V$, $v_i \in \mathbb{V}_i$, $n \in \mathbb{N}$, та $n = |V|$, як набір значень v_i з домену \mathbb{V}_i , пов'язаних зі змінними v_i та що представляють конкретні величини властивостей модельованого об'єкта.

Можемо графічно показати стан \mathfrak{V} як точку у фазовому просторі [32-34], визначену набором змінних V . Поведінка модельованого об'єкта - це траєкторія, утворена переходами від поточного \mathfrak{V}_i до наступного \mathfrak{V}_{i+1} .

Наприклад, набір змінних

$$V := \begin{bmatrix} \frac{v}{t} : \mathbb{R} \\ v_{tank} : \mathbb{R} \\ v_{water} : \mathbb{R} \\ t : \mathbb{R} \end{bmatrix}, \quad (2.8)$$

зображення резервуара, що наповнюється водою (рис. 2.1), сформує чотиривимірний фазовий простір. Припустимо, що змінюються лише дві з чотирьох змінних, v_{water} та t ; тоді можемо згорнути простір у два виміри та представити поведінку об'єкта як лінію, що лежить на площині (рис. 2.2).

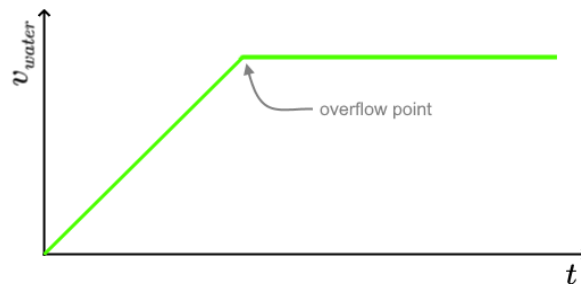


Рис. 2.2 Графік залежності v_{water} від t

Якщо переберемо всі можливі набори значень змінних V (тобто всі можливі стани \mathfrak{B}), отримаємо набір станів, які фактично утворюють фазовий простір. Такий набір станів називатимемо простором станів і позначатимемо його символом \mathbb{V}^n , де n – розмірність простору.

Наприклад, якщо V має дві булеві змінні

$$V := \begin{bmatrix} a: \text{Bool} \\ b: \text{Bool} \end{bmatrix}, \quad (2.9)$$

тоді простір станів матиме 4 можливі стани:

$$\mathbb{V}^2 := \left\{ \begin{bmatrix} a = F, b = F \\ a = F, b = T \\ a = T, b = F \\ a = T, b = T \end{bmatrix} \right\}, \quad (2.10)$$

Потужність простору \mathbb{V}^n дорівнює добутку потужностей змінних з множини V :

$$|\mathbb{V}^n| = \prod_{i=1}^n |v_i|, \quad (2.11)$$

де $n = |V|$ – це кількість змінних у V , а $v_i \in V$ – змінна з множини змінних V .

Аналогічно, можемо пов'язати певні значення з множинами змінних X , Y та G , підмножинами множини V .

Визначення: Визначимо набір значень, пов'язаних з незалежними змінними

$$X = \mathfrak{X} := \begin{bmatrix} x_1 = \mathfrak{x}_1 \in \mathbb{X}_1 \\ \dots \\ x_n = \mathfrak{x}_n \in \mathbb{X}_n \end{bmatrix}, \quad (2.12)$$

де $x_i \in X, V$, $\mathfrak{x}_i \in \mathfrak{B}$, \mathbb{X}_i та $n = |X|$. Набір значень, пов'язаних із залежними змінними, є

$$Y = \mathfrak{Y} := \begin{bmatrix} y_1 = \mathfrak{y}_1 \in \mathbb{Y}_1 \\ \dots \\ y_n = \mathfrak{y}_n \in \mathbb{Y}_n \end{bmatrix}, \quad (2.13)$$

де $y_i \in Y, V$, $\mathfrak{y}_i \in \mathfrak{B}$, \mathbb{Y}_i та $n = |Y|$. А також набір значень параметрів

$$G = \mathfrak{G} := \begin{bmatrix} g_1 = \mathfrak{g}_1 \in \mathbb{G}_1 \\ \dots \\ g_n = \mathfrak{g}_n \in \mathbb{G}_n \end{bmatrix}, \quad (2.14)$$

де $g_i \in G, V$, $\mathfrak{g}_i \in \mathfrak{B}, \mathbb{G}_i$ та $n = |G|$.

Як і у випадку з V , якщо переберемо всі можливі набори значень змінних X , Y та G , отримаємо набори \mathbb{X}^n , де $n = |X|$, \mathbb{Y}^n , де $n = |Y|$, та \mathbb{G}^n , де $n = |G|$. Кожен з них включає всі можливі \mathfrak{X} , \mathfrak{Y} та \mathfrak{G} відповідно.

Графічно можемо представити \mathbb{X}^n , \mathbb{Y}^n та \mathbb{G}^n як підпростори фазового простору \mathbb{V}^n з меншою розмірністю.

Наприклад, якщо

$$V := \begin{bmatrix} a: \mathbb{R} \\ b: \mathbb{R} \\ c: \mathbb{R} \end{bmatrix}, \quad (2.15)$$

тоді простір \mathbb{V}^3 буде тривимірним. Якщо

$$X := \begin{bmatrix} a: \mathbb{R} \\ b: \mathbb{R} \end{bmatrix}, \quad (2.16)$$

обрано для незалежних змінних, тоді підпростір \mathbb{X}^2 буде двовимірним, тобто це буде певна площа всередині \mathbb{V}^3 .

У багатьох випадках розглядатимемо лише підмножину наборів значень з просторів \mathbb{V}^n , \mathbb{X}^n , \mathbb{Y}^n та \mathbb{G}^n . Позначатимемо такі підмножини жирними символами $\mathfrak{B} \subseteq \mathbb{V}^n$, $\mathfrak{X} \subseteq \mathbb{X}^n$, $\mathfrak{Y} \subseteq \mathbb{Y}^n$ та $\mathfrak{G} \subseteq \mathbb{G}^n$ відповідно.

2.4. Представлення залежності Y від X як набору функцій: $Y = F(X|\mathfrak{G})$

Залежність змінних Y (обраних як залежні) від змінних X (обраних як незалежні) можна математично представити як набір функцій, параметризованих значеннями \mathfrak{G} змінних G , обраних як параметри.

Визначення: Визначимо залежність

$$Y = F(X|\mathfrak{G}) := \begin{bmatrix} y_1 = f_1(x_1, \dots, x_n | g_1, \dots, g_l) \\ y_2 = f_2(x_1, \dots, x_n | g_1, \dots, g_l) \\ \dots \\ y_m = f_m(x_1, \dots, x_n | g_1, \dots, g_l) \end{bmatrix}, \quad (2.17)$$

де $x_1, \dots, x_n \in X$ – множина незалежних змінних, $y_1, \dots, y_m \in Y$ – множина залежних змінних, $g_1, \dots, g_l \in \mathfrak{G}$ – множина значень параметрів, а $f_1, \dots, f_m \in \mathcal{F}$ – сукупність довільних функцій. Крім того, $n = |X|$, $m = |Y|$ та $l = |\mathfrak{G}|$.

Незалежні змінні є аргументами функції та зазвичай відповідають горизонтальним осям на графіку, тоді як залежні, це результати функції, що відповідають вертикальній осі графіка [35].

Набори функцій $F(X|\mathfrak{G})$ можна реалізувати різними способами. Наприклад, функції можна побудувати аналітично в системі рівнянь, їх можна навчити на даних за допомогою будь-яких алгоритмів машинного навчання тощо. У цій статті в основному розглянемо набір функцій, визначених в аналітичній формі. Включимо функції, сформульовані як задача Коші.

$$Y = F(X|\mathfrak{G}) := \begin{bmatrix} y'_1(X|\mathfrak{G}) = f_1(X, \hat{y}_1(X|\mathfrak{G})); \hat{y}_1(\mathfrak{X}) = \eta_1 \\ y'_2(X|\mathfrak{G}) = f_2(X, \hat{y}_2(X|\mathfrak{G})); \hat{y}_2(\mathfrak{X}) = \eta_2 \\ \dots \\ y'_m(X|\mathfrak{G}) = f_m(X, \hat{y}_m(X|\mathfrak{G})); \hat{y}_m(\mathfrak{X}) = \eta_m \end{bmatrix}, \quad (2.18)$$

де X – незалежні змінні, y_1, \dots, y_m – залежні змінні, f_1, \dots, f_m – довільні функції, \mathfrak{X} – початкові значення незалежних змінних, η_1, \dots, η_m – початкові значення залежних змінних, а $n, m \in \mathbb{N}$.

Залежність $Y = F(X|\mathfrak{G})$ можна графічно представити як геометричну фігуру у просторі станів \mathbb{V}^n , яка визначає зв'язок між X , Y та \mathfrak{G} . Ця фігура складається з точок

$$\mathfrak{Z} = \mathfrak{X} \cup \mathfrak{Y} \cup \mathfrak{G}, \quad (2.19)$$

де $\mathfrak{Z} \in \mathbb{V}^n$ – точка фазового простору, $\mathfrak{X} \in \mathbb{X}^n$ – множина значень незалежних змінних, $\mathfrak{Y} \in \mathbb{Y}^m$ – множина значень залежних змінних, а $\mathfrak{G} \in \mathbb{G}^l$ – значення параметрів.

Наприклад, уявіть, що змодельований об'єкт є тілом, яке рухається по поверхні (рис. 2.3), а його змінні моделі:

$$V := \begin{bmatrix} x: \mathbb{R} \\ y: \mathbb{R} \\ t: \mathbb{R} \end{bmatrix}, \quad (2.20)$$

де t – час, а $[x, y]$ – координати тіла. Якщо t – незалежна змінна, а $[x, y]$ – залежні, то функції

$$Y = F(X|\mathfrak{G}) := \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}, \quad (2.21)$$

буде кривою у тривимірному просторі \mathbb{V}^3 , проєкція якої на площину $[x, y]$ відповідає траєкторії руху тіла. Якщо $[x, y]$ незалежні, а t – залежна змінна, то функція

$$Y = F(X|\mathfrak{G}) := [t(x, y)], \quad (2.22)$$

буде поверхнею у \mathbb{V}^3 , яка визначає відповідність довільних координат тіла та часу, коли тіло знаходилося в цих координатах.

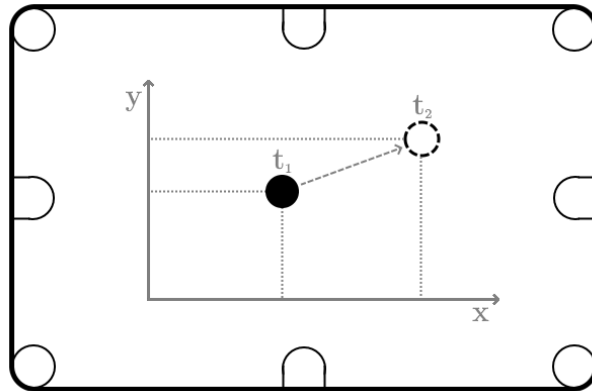


Рис. 2.3 Приклад тіла, що рухається по поверхні

2.5. Підстани $\mathfrak{S}_q^{\mathfrak{A}}$ як розклад стану \mathfrak{V}

Кожен стан \mathfrak{V} можна представити як набір підстанів, кожен з яких містить лише частину значень $v_i \in \mathfrak{V}$. Повинен існувати спосіб визначити, який із підстанів

належить до певного \mathfrak{V} . Одним із варіантів досягнення цього є використання унікального ключа для позначення всіх підстанів, що належать до певного \mathfrak{V} .

Визначення: Визначимо підстан $\mathfrak{S}^{\mathfrak{K}}_q$, де $\mathfrak{S} \subseteq \mathfrak{V}$ це частина або весь набір значень $v_i \in \mathfrak{V}$, \mathfrak{K} це деякий ключ, унікальний для стану $\mathfrak{V} \in \mathbb{V}^n$, а $q \in \mathbb{N}$ це індекс підстану з тим самим ключем, \mathfrak{K} .

Одне або декілька значень $v_i \in \mathfrak{V}$ можуть бути використані як ключ \mathfrak{K} . У цьому випадку немає сенсу включати їх до будь-якого з підстанів, оскільки вони будуть представлені в ключі.

Розподіл стану на підстани можна зобразити графічно, як на рис. 2.4.

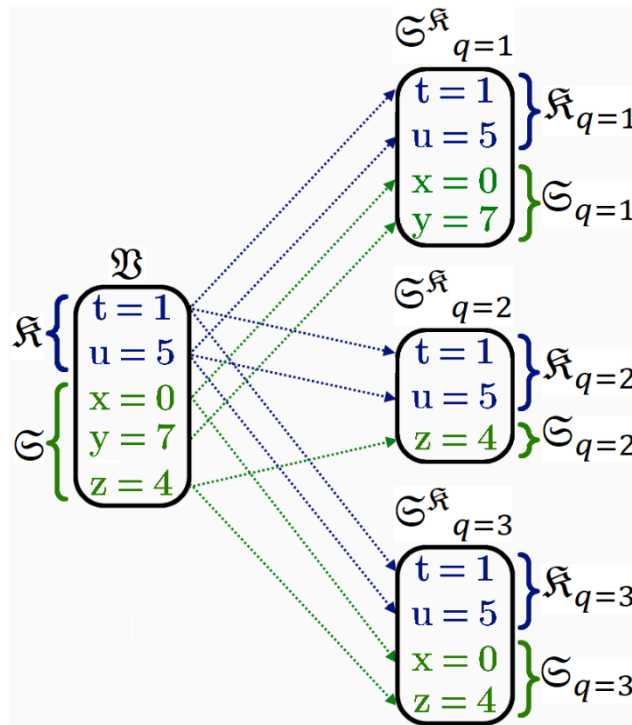


Рис. 2.4 Графічне представлення розщеплення стану на підстани

Повний стан

$$\mathfrak{V} := \begin{bmatrix} t = 1 \\ m = 5 \\ x = 0 \\ y = 7 \\ z = 4 \end{bmatrix}, \quad (2.23)$$

поділяється на підстани

$$\mathfrak{S}^{\mathfrak{K}}_{q=1} := \begin{bmatrix} x = 0 \\ y = 7 \end{bmatrix}, \quad (2.24)$$

$$\mathfrak{S}^{\mathfrak{K}}_{q=2} := [z = 4], \quad (2.25)$$

$$\mathfrak{S}^{\mathfrak{K}}_{q=3} := \begin{bmatrix} x = 0 \\ z = 4 \end{bmatrix}, \quad (2.26)$$

де ключ для кожного стану

$$\mathfrak{K} := \begin{bmatrix} t = 1 \\ m = 5 \end{bmatrix}, \quad (2.27)$$

Для деякого стану \mathfrak{V} маємо множину підстанів $\{\mathfrak{S}^{\mathfrak{K}}_q\}$ з однаковим ключем \mathfrak{K} . Позначимо цю множину жирним шрифтом $\mathfrak{S}^{\mathfrak{K}}$. При такому представленні стану \mathfrak{V} необхідно переконатися, що всі $\mathfrak{S}^{\mathfrak{K}}_q$, позначені однаковим \mathfrak{K} , не суперечать одна одній. Пара $\mathfrak{S}^{\mathfrak{K}}_q$ з однаковим ключем \mathfrak{K} може бути суперечливою, якщо одне або декілька значень $v_i \in \mathfrak{S}, \mathfrak{V}$ відрізняються під одним і тим самим індексом.

Наприклад, для стану \mathfrak{V} (формула 2.23), підстани

$$\mathfrak{S}^{\mathfrak{K}}_{q=1} := \begin{bmatrix} x = 0 \\ y = 7 \end{bmatrix}, \quad (2.28)$$

i

$$\mathfrak{S}^{\mathfrak{K}}_{q=2} := \begin{bmatrix} x = 0 \\ y = 6 \end{bmatrix}, \quad (2.29)$$

суперечливі, тоді як підстани

$$\mathfrak{S}^{\mathfrak{K}}_{q=1} := \begin{bmatrix} x = 0 \\ y = 7 \end{bmatrix}, \quad (2.30)$$

i

$$\mathfrak{S}^{\mathfrak{K}}_{q=2} := \begin{bmatrix} x = 0 \\ z = 6 \end{bmatrix}, \quad (2.31)$$

не є.

Визначення: Визначимо множину підстанів

$$\mathfrak{S}^{\mathfrak{K}} := \{\mathfrak{S}^{\mathfrak{K}}_q, \mathfrak{S}^{\mathfrak{K}'}_{q'} \mid \mathfrak{K} = \mathfrak{K}' \wedge q \neq q'''\}, \quad (2.32)$$

де для всіх $\mathfrak{S}^{\mathfrak{K}}_q \in \mathfrak{S}^{\mathfrak{K}}$, критерій узгодженості

$$\nexists \mathfrak{S}^{\mathfrak{K}}_i, \mathfrak{S}^{\mathfrak{K}}_j \in \mathfrak{S}^{\mathfrak{K}} (\exists v_l \in \mathfrak{S}^{\mathfrak{K}}_i \neq v_l \in \mathfrak{S}^{\mathfrak{K}}_j \forall l), \quad (2.33)$$

є правдою.

У цій статті поговоримо про довільні множини підстанів $\mathfrak{S}^{\mathfrak{K}}_q$, єдиною вимогою для яких є відповідність критерію узгодженості.

Визначення: Визначимо довільний набір підстанів

$$\mathfrak{S}^K := \{\mathfrak{S}^{\mathfrak{K}} \cup \mathfrak{S}^{\mathfrak{K}'} \mid \mathfrak{K} \neq \mathfrak{K}'\}, \quad (2.34)$$

як об'єднання множин $\mathfrak{S}^{\mathfrak{K}}$ з різними \mathfrak{K} .

Зауважте, що ці визначення не вимагають наявності в множині $\mathfrak{S}^{\mathfrak{K}}$ (і, отже, в \mathfrak{S}^K) достатньої кількості підстанів $\mathfrak{S}^{\mathfrak{K}}_q$, для покриття всіх значень $v_i \in \mathfrak{V}$.

Наприклад, незважаючи на те, що множина підстанів

$$\mathfrak{S}^{\mathfrak{K}} := \left\{ \begin{array}{l} \mathfrak{S}^{\mathfrak{K}}_{q=1} = [x = 0] \\ \mathfrak{S}^{\mathfrak{K}}_{q=2} = [z = 4] \end{array} \right\}, \quad (2.35)$$

побудований на стані \mathfrak{V} (формула 2.23) є правильним з точки зору заданого визначення, не можемо відновити \mathfrak{V} , оскільки значення змінної u не визначено в жодному з підстанів $\mathfrak{S}^{\mathfrak{K}}$.

Також зазначимо, що за визначенням множина \mathfrak{S}^K може містити більше одного підстану $\mathfrak{S}^{\mathfrak{K}}_q$ з тим самим ключем \mathfrak{K} . Однак, на довільній множині \mathfrak{S}^K , яка містить дублікати ключів \mathfrak{K} , можемо побудувати множину \mathfrak{S}^K , яка їх не містить.

Для цього нам потрібно об'єднати всі підстани з однаковим ключем в один підстан:

$$\mathfrak{S}^K_d \xrightarrow{\forall \mathfrak{S}^{\mathfrak{K}} \subseteq \mathfrak{S}^K_d (\mathfrak{S}^{\mathfrak{K}}_1 \in \mathfrak{S}^K_u = \bigcup_{\mathfrak{K}} \mathfrak{S}^{\mathfrak{K}})} \mathfrak{S}^K_u, \quad (2.36)$$

де \mathfrak{S}_d^K – множина з дублікатами ключів, а \mathfrak{S}_u^K – множина з унікальним \mathfrak{K} . Таким чином, можна сказати, що довільну множину підстанів \mathfrak{S}^K можна розглядати як структуру ключ-значення або як сюр'єктивну функцію.

$$f: \{\mathfrak{K}\} \rightarrow \{\mathfrak{S}\}, \quad (2.37)$$

Як випливає з визначення, множину \mathfrak{S}^K можна побудувати лише з множини станів $\mathfrak{V} \subseteq \mathbb{V}^n$, у яких кожному стану $\mathfrak{V} \in \mathfrak{V}$ можна пов'язати унікальний ключ \mathfrak{K} . В іншому випадку це призведе до появи конфлікту між підстанами $\mathfrak{S}_{\mathfrak{K}}^{\mathfrak{K}}$.

Наприклад, на правильному наборі станів

$$\mathfrak{V} := \left\{ \begin{array}{l} \mathfrak{V}_1 = [t = 0, x = 1] \\ \mathfrak{V}_2 = [t = 1, x = 1] \end{array} \right\}, \quad (2.38)$$

у випадку, коли

$$\mathfrak{K} := [x = 1], \quad (2.39)$$

вибрано як ключ \mathfrak{K} , неможливо побудувати набір підстанів $\mathfrak{S}^{\mathfrak{K}}$, оскільки значення x буде неконсистентним.

Зворотне перетворення, тобто побудова $\mathfrak{V} \subseteq \mathbb{V}^n$ з довільного \mathfrak{S}^K

$$\mathfrak{S}^K \xrightarrow{\forall \mathfrak{S}_{\mathfrak{K}}^{\mathfrak{K}} \subseteq \mathfrak{S}^K (\mathfrak{V} \in \mathfrak{V} = \bigcup_{\mathfrak{K}} \mathfrak{S}_{\mathfrak{K}}^{\mathfrak{K}})} \mathfrak{V}, \quad (2.40)$$

можливо лише тоді, коли \mathfrak{S}^K містить достатньо підстанів $\mathfrak{S}_{\mathfrak{K}}^{\mathfrak{K}}$ для повної побудови кожного стану $\mathfrak{V} \in \mathfrak{V}$.

Множина підстанів \mathfrak{S}^K може бути еквівалентною простору станів \mathbb{V}^n , якщо цей простір станів містить достатньо підстанів для побудови кожного стану $\mathfrak{V} \in \mathbb{V}^n$. Позначимо таку множину через \mathbb{S}^K .

2.6. Представлення залежності Y від X як набору підстанів: $Y = \mathfrak{S}^{X|\mathfrak{G}}$

Залежність змінних Y від X можна представити як множину станів \mathfrak{S}^K . Це представлення є альтернативою множині функцій $F(X|\mathfrak{G})$. У цьому випадку зручно

вибрати значення $\mathfrak{X} \in \mathbb{X}^n$ як ключ \mathfrak{K} , а підмножину значень $\mathfrak{Y} \in \mathbb{Y}^n$ як значення \mathfrak{S} (включаючи $\mathfrak{S} = \emptyset$).

Визначення: Визначимо підстан $\mathfrak{S}^{\mathfrak{X}}_q$, де ключ $\mathfrak{K} = \mathfrak{X}$, $\mathfrak{X} \in \mathbb{X}^n$, значення $\mathfrak{S} \subseteq \mathfrak{Y}$, $\mathfrak{Y} \in \mathbb{Y}^n$, а індекс $q \in \mathbb{N}$ такий, що

$$\forall \mathfrak{S}^{\mathfrak{X}}_q, \mathfrak{S}^{\mathfrak{X}}_{q'} (\mathfrak{X} = \mathfrak{X}', q \neq q'). \quad (2.41)$$

Також, $\mathfrak{X} \subseteq \mathfrak{S}^{\mathfrak{X}}_q$ та $(\mathfrak{S}^{\mathfrak{X}}_q \setminus \mathfrak{X}) \subseteq \mathfrak{Y}$.

Оскільки значення параметрів $\mathfrak{G} \in \mathbb{G}^n$ є постійними для будь-яких можливих значень \mathfrak{X} та \mathfrak{Y} , вони також є постійними для будь-яких можливих підстанів $\mathfrak{S}^{\mathfrak{X}}_q$, складених зі значень \mathfrak{X} та \mathfrak{Y} . Таким чином, визначення $\mathfrak{S}^{\mathfrak{X}}_q$ не включає значення \mathfrak{G} . Будучи об'єднаними в множину, підстани $\mathfrak{S}^{\mathfrak{X}}_q$ матимуть однакові параметри \mathfrak{G} , але можуть мати різні значення ключа \mathfrak{X} .

Визначення: Визначимо залежність

$$Y = \mathfrak{S}^{X|\mathfrak{G}} := \{\mathfrak{S}^{\mathfrak{X}}_q\}|\mathfrak{G}, \quad (2.42)$$

що представляє залежність змінних Y від X для заданих параметрів \mathfrak{G} , що є однаковими для всіх підстанів, що входять до множини $\mathfrak{S}^{X|\mathfrak{G}}$. Водночас, підстани не повинні бути суперечливими:

$$\nexists \mathfrak{S}^{\mathfrak{X}}_i, \mathfrak{S}^{\mathfrak{X}}_j \in \mathfrak{S}^{X|\mathfrak{G}} (\exists v_l \in \mathfrak{S}^{\mathfrak{X}}_i \neq v_l \in \mathfrak{S}^{\mathfrak{X}}_j \forall l). \quad (2.43)$$

Множину $\mathfrak{S}^{X|\mathfrak{G}}$, усі підстани якої мають однаковий ключ \mathfrak{X} , позначатимемо як $\mathfrak{S}^{\mathfrak{X}}|\mathfrak{G}$.

Наприклад, повернемося до наповнення резервуара водою (рис. 2.1). Коли

$$\mathbb{S}^{X|\mathfrak{G}} = \bigcup_{\mathfrak{X} \in \mathbb{X}^n} \mathfrak{S}^{\mathfrak{X}}|\mathfrak{G}, \quad (2.44)$$

можемо графічно представити залежність $Y = \mathfrak{S}^{X|\mathfrak{G}}$ у вигляді зрізів графіка залежності $Y = F(X|\mathfrak{G})$ (рис. 2.5).

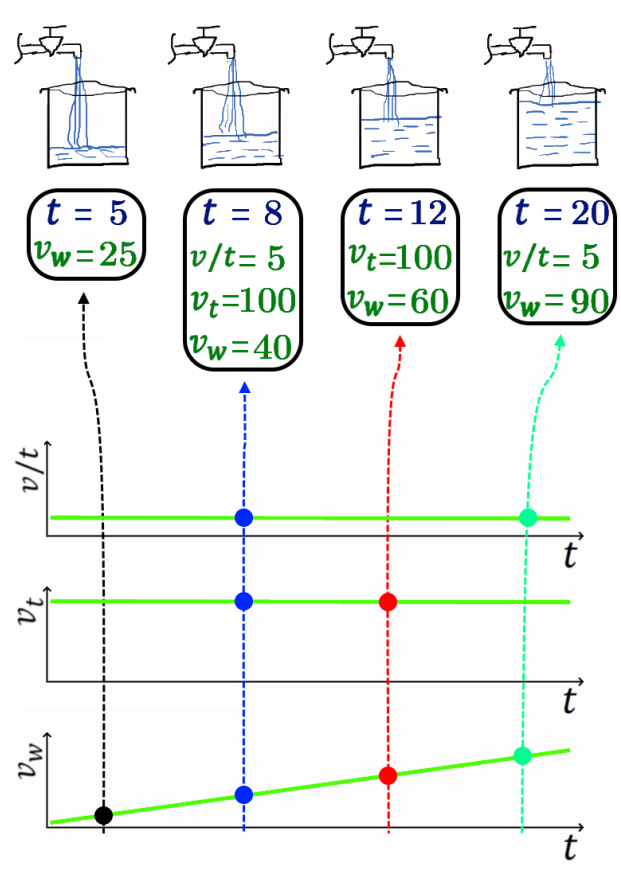


Рис. 2.5 Графічне представлення залежності $Y = \mathfrak{S}^{X|\mathfrak{G}}$ у вигляді зрізів графіка залежності $Y = F(X|\mathfrak{G})$.

Зауважимо, що не накладаємо обмеження на повноту на множину $\mathfrak{S}^{X|\mathfrak{G}}$, тобто $\mathfrak{S}^{X|\mathfrak{G}}$ може не містити всіх ключів $\mathfrak{X} \in \mathbb{X}^n$ або навіть бути порожнім: $\mathfrak{S}^{X|\mathfrak{G}} = \emptyset$. $\mathfrak{S}^{X|\mathfrak{G}}$ також може не містити всіх $\mathfrak{Y} \in \mathbb{Y}^n$ та/або може не містити достатньо $\mathfrak{S}^{\mathfrak{X}}_q$ для побудови одного або кількох повних \mathfrak{Y} .

Представлення $Y = \mathfrak{S}^{X|\mathfrak{G}}$ еквівалентне представленню $Y = F(X|\mathfrak{G})$ тоді і тільки тоді, коли для кожного $\mathfrak{X} \in \mathbb{X}^n$ у заданій точці $Y = F(X|\mathfrak{G})$ представлення рівні:

$$Y = \mathfrak{S}^{X|\mathfrak{G}} \Leftrightarrow Y = F(X|\mathfrak{G}) \Rightarrow \forall \mathfrak{X} \in \mathbb{X}^n (F(\mathfrak{X}|\mathfrak{G}) = \mathfrak{S}^{\mathfrak{X}|\mathfrak{G}}), \quad (2.45)$$

Для кожного ключа \mathfrak{X} існує множина підстанів $\mathfrak{S}_{\mathfrak{q}}^{\mathfrak{X}}$, які охоплюють усі можливі значення $\mathfrak{Y} \in \mathbb{Y}^n$. Позначимо цю множину через $\mathbb{S}^{\mathfrak{X}|\mathfrak{G}}$. Ця множина може не задовольняти критерію узгодженості (формула 2.43) і матиме потужність

$$|\mathbb{S}^{\mathfrak{X}|\mathfrak{G}}| = \prod_{i=1}^n |y_i|, \quad (2.46)$$

Якщо об'єднаємо множини $\mathbb{S}^{\mathfrak{X}|\mathfrak{G}}$ для всіх можливих ключів $\mathfrak{X} \in \mathbb{X}^n$, отримаємо множину всіх можливих підстанів. Позначимо її через

$$\mathbb{S}^{\mathbb{X}|\mathfrak{G}} = \bigcup_{\mathfrak{X} \in \mathbb{X}^n} \mathbb{S}^{\mathfrak{X}|\mathfrak{G}}, \quad (2.47)$$

Кардинальність цієї множини, коли $\mathfrak{S} = \mathfrak{Y}$, буде

$$|\mathbb{S}^{\mathbb{X}|\mathfrak{G}}| = \sum_{\mathfrak{X} \in \mathbb{X}^n} |\mathbb{S}^{\mathfrak{X}|\mathfrak{G}}|, \quad (2.48)$$

Більше того, $|\mathbb{S}^{\mathbb{X}|\mathfrak{G}}| \leq |\mathbb{V}^n|$, оскільки з множини \mathbb{G}^n використовується лише один набір значень \mathfrak{G} (зауважте, що потужності будуть однаковими у випадку $|\mathbb{G}^n| = 1$).

На практиці частіше зустрічатимемо розріджене $\mathfrak{S}^{\mathbb{X}|\mathfrak{G}}$, де неможливо повністю побудувати \mathfrak{Y} для кожного $\mathfrak{X} \in \mathbb{X}^n$. Використання розрідженого $\mathfrak{S}^{\mathbb{X}|\mathfrak{G}}$ знизить точність моделювання. Загалом, це не є проблемою з інженерної точки зору, оскільки збільшення або зменшення потужності $\mathfrak{S}^{\mathbb{X}|\mathfrak{G}}$ дозволяє нам вибрати прийнятний рівень точності для вирішення конкретної задачі моделювання.

2.7. Відображення $\check{\mathfrak{Y}}(\bar{X}|\mathfrak{G})$ як запис змін значень змінних

Можемо відобразити поведінку змодельованого об'єкта, вимірюючи його властивості та записуючи відповідні значення змінних X , Y , та G . Абстрагуючись від конкретної реалізації, називатимемо такий запис відображенням змодельованого об'єкта.

Визначення: Визначимо відображення $\check{\mathfrak{Y}}(\bar{X}|\mathfrak{G})$ як довільне представлення залежності залежних змінних $\check{\mathfrak{Y}}$ від незалежних змінних \bar{X} та значень параметрів \mathfrak{G} .

Більше того, ця залежність будується шляхом вивчення та вимірювання властивостей модельованого об'єкта.

Можемо графічно зобразити побудову відбиття $\check{Y}(\bar{X}|\mathfrak{G})$, додавши точки

$$\mathfrak{B} = \bar{\mathfrak{X}} \cup \check{\mathfrak{Y}} \cup \mathfrak{G}, \quad (2.49)$$

у простір станів \mathbb{V}^n у координатах $\bar{X}, \check{Y}, \mathfrak{G}$ де $\bar{\mathfrak{X}} \in \overline{\mathbb{X}^n}$, $\check{\mathfrak{Y}} \in \check{\mathbb{Y}^n}$ та $\mathfrak{G} \in \mathbb{G}^n$. Додані точки утворюватимуть геометричну фігуру, яка відображає поведінку змодельованого об'єкта.

Відображення можна представити як набір функцій

$$\check{Y}(\bar{X}|\mathfrak{G})^F = F(\bar{X}|\mathfrak{G}) = \check{Y}, \quad (2.50)$$

або як набір станів

$$\check{Y}(\bar{X}|\mathfrak{G})^S = \mathfrak{S}^{\bar{X}|\mathfrak{G}} = \check{Y}, \quad (2.51)$$

У першому випадку набір функцій можна побудувати, записавши отримані або виміряні значення змінних X , Y , та G . [3] У другому випадку, зі значень $\check{\mathfrak{Y}}$, отриманих або виміряних відносно $\bar{\mathfrak{X}}$ та \mathfrak{G} , можна безпосередньо побудувати підстан $\mathfrak{S}_{q=1}^{\bar{\mathfrak{X}}}$ та додати його до набору підстанів $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$.

Наприклад, для резервуара, який наповнюється водою (рис. 2.1, формула 2.8), можемо розподілити набір змінних як

$$\bar{X} := [\bar{t}: \mathbb{R}], \quad (2.52)$$

$$\check{Y} := [\widetilde{v_{water}}: \mathbb{R}], \quad (2.53)$$

$$G := \left[\begin{array}{c} \frac{v}{t}: \mathbb{R} \\ v_{tank}: \mathbb{R} \end{array} \right], \quad (2.54)$$

У цьому випадку, записуючи значення секундоміра (який відображає змінну t) та рівнеміра (який відображає v_{water}), отримуємо функцію $v_{water}(t)$, яка відображає залежність v_{water} від t . На практиці ця функція буде визначена лише на певному інтервалі або кількох інтервалах часу $t_{measuring}$, протягом яких проводилося вимірювання. Це можна графічно зобразити на рис. 2.6.

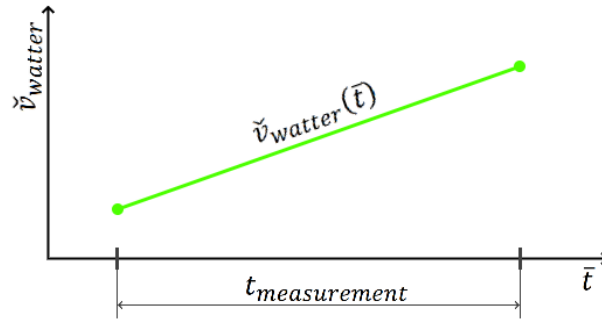


Рис. 2.6 Приклад часткової функції $\check{v}_{water}(t)$

2.8. Модель $\hat{Y}(\bar{X}|\mathfrak{G})$ як імітація змін змінних V

Одним із кількох способів можемо визначити залежності змінних Y від X та G без безпосереднього вимірювання властивостей модельованого об'єкта [28, 36, 37]. Називатимемо залежність, визначену таким чином, моделлю модельованого об'єкта.

Визначення: Модель модельованого об'єкта $\hat{Y}(\bar{X}|\mathfrak{G})$ – це довільне представлення або реалізація залежності залежних змінних \hat{Y} від незалежних змінних \bar{X} та значень параметрів \mathfrak{G} .

Більше того, ця залежність будується без безпосередньої участі модельованого об'єкта.

Можемо графічно представити модель $\hat{Y}(\bar{X}|\mathfrak{G})$ як геометричну фігуру у просторі станів V^n , що складається з точок

$$\mathfrak{V} = \bar{\mathfrak{X}} \cup \hat{\mathfrak{Y}} \cup \mathfrak{G}, \quad (2.55)$$

що визначають зв'язок між змінними \hat{Y} та \bar{X} і параметрами \mathfrak{G} , де $\bar{\mathfrak{X}} \in \bar{X}^n$, $\hat{\mathfrak{Y}} \in \hat{Y}^n$ та $\mathfrak{G} \in G^n$.

Модель може бути реалізована як набір, можливо, часткових функцій

$$\hat{Y}(\bar{X}|\mathfrak{G})^F = F(\bar{X}|\mathfrak{G}) = \hat{Y}, \quad (2.56)$$

або як набір станів

$$\hat{Y}(\bar{X}|\mathfrak{G})^S = \mathfrak{S}^{\bar{X}|\mathfrak{G}} = \hat{Y}. \quad (2.57)$$

У першому випадку набір функцій можна визначити аналітично або іншим способом. У другому випадку багато підставів необхідно заздалегідь побудувати тим чи іншим способом.

Наприклад, можемо імітувати резервуар для заповнення (рис. 2.1, формули 2.8) за допомогою простої параметричної моделі

$$v_{water} = v_{water}(t) = t * v/t, \quad (2.58)$$

або у формі задачі Коші

$$\frac{\partial v_{water}}{\partial t} = v/t; v_{water(0)}=0, \quad (2.59)$$

Графічно це можна зобразити, як на рис. 2.7.

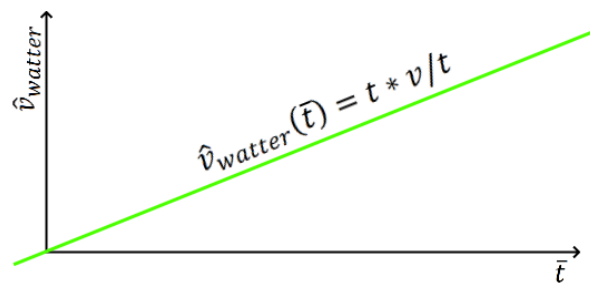


Рис. 2.7 Приклад імітаційної фракції $\hat{v}_{water}(t)$

Можемо визначити модель $\hat{Y}(\bar{X}|\mathfrak{G})$ так, щоб вона повністю співпадала з певним відображенням $\check{Y}(\bar{X}|\mathfrak{G})$; проте набагато розумніше та корисніше побудувати $\hat{Y}(\bar{X}|\mathfrak{G})$ для прогнозування змін у модельованому об'єкті.

З практичної точки зору, нас цікавить, наскільки точно побудована модель $\hat{Y}(\bar{X}|\mathfrak{G})$ відповідає змодельованому об'єкту. Один із способів визначення відповідності це порівняти модель та відображення $\check{Y}(\bar{X}|\mathfrak{G})$ (тобто обчислити величину їхньої невідповідності тим чи іншим чином). Позначимо значення невідповідності через ε .

Наприклад, для випадку, коли всі змінні V мають домен \mathbb{R} , можемо визначити $\varepsilon \in \mathbb{R}$ як інтегральну суму різниці значень \check{Y} та \hat{Y} для кожного $\bar{x} \in \bar{\mathbb{X}}^n$:

$$\varepsilon = \sum_{\bar{x} \in \bar{\mathbb{X}}^n} \sum_{i=1}^{|Y|} (\check{Y}(\bar{x}|\mathfrak{G})_i - \hat{Y}(\bar{x}|\mathfrak{G})_i), \quad (2.60)$$

де $\mathfrak{G} \in \mathbb{G}^n$.

2.9. Моделювання моделі $\hat{Y}(\bar{X}|\mathfrak{G})$ як обчислення підмножини $\hat{\mathfrak{Y}} \subseteq \hat{\mathbb{Y}}^n$

Завдання моделювання можна звести до отримання або обчислення підмножини невідомих значень залежних змінних \hat{Y} з підмножини відомих значень незалежних змінних \bar{X} та значень параметрів \mathfrak{G} за допомогою певної моделі $\hat{Y}(\bar{X}|\mathfrak{G})$.

Визначення: Визначимо моделювання як оператор

$$\bar{x} \xrightarrow{\hat{Y}(\bar{X}|\mathfrak{G})} \hat{\mathfrak{y}}, \quad (2.61)$$

де $\bar{x} \in \bar{\mathbb{X}}^n$ – можливо впорядкована множина унікальних відомих значень незалежних змінних, $\hat{\mathfrak{y}} \in \hat{\mathbb{Y}}^n$ – бажана множина можливо не унікальних значень залежних змінних, а $\hat{Y}(\bar{X}|\mathfrak{G})$ – певна модель, що використовується для отримання бажаного $\hat{\mathfrak{y}} \in \hat{\mathbb{Y}}^n$ для заданого $\bar{x} \in \bar{\mathbb{X}}^n$.

Для випадку, коли модель реалізована як набір функцій (формула 2.56), моделювання

$$\bar{x} \xrightarrow{\hat{Y}(\bar{X}|\mathfrak{G})^F} \hat{\mathfrak{y}}, \quad (2.62)$$

це просто обчислення результату $\hat{\mathfrak{y}} \in \hat{\mathfrak{Y}}$ для кожного аргументу $\bar{x} \in \bar{\mathfrak{X}}$:

$$\bar{x} \xrightarrow{\forall \bar{x} \in \bar{\mathfrak{X}} (\hat{\mathfrak{y}} \in \hat{\mathfrak{Y}} = F(\bar{X}|\mathfrak{G})(\bar{x}))} \hat{\mathfrak{y}}, \quad (2.63)$$

де

$$\hat{\mathfrak{y}} = F(\bar{X}|\mathfrak{G})(\bar{x}), \quad (2.64)$$

що є операцією для обчислення $\hat{\mathfrak{y}} \in \hat{\mathbb{Y}}^n$ для заданого $\bar{x} \in \bar{\mathbb{X}}^n$. Для моделі, реалізованої як набір підстанів (формула 2.57), моделювання полягає у знаходженні всіх підстанів для кожного ключа $\bar{x} \in \bar{\mathfrak{X}}$, а потім побудові значень $\hat{\mathfrak{y}} \in \hat{\mathfrak{Y}}$ зі знайдених підстанів

$$\bar{\mathfrak{X}} \xrightarrow{\forall \bar{\mathfrak{x}} \in \bar{X} (\mathfrak{Y} \in \hat{Y} = \cup \mathfrak{S}^{\bar{X}|\mathfrak{G}}(\bar{\mathfrak{x}}))} \hat{\mathfrak{Y}}, \quad (2.65)$$

де

$$\mathfrak{S}^{\bar{X}|\mathfrak{G}}(\bar{\mathfrak{x}}) = \mathfrak{S}^{\bar{\mathfrak{x}}|\mathfrak{G}}, \quad (2.66)$$

– це операція вибору підмножини $\mathfrak{S}^{\bar{\mathfrak{x}}|\mathfrak{G}} \subseteq \mathfrak{S}^{\bar{X}|\mathfrak{G}}$ підстанів $\mathfrak{S}^{\bar{\mathfrak{x}}|\mathfrak{G}}$ з тим самим ключем $\bar{\mathfrak{x}}$.

Наприклад, для тіла, що сповільнюється прямолінійно (рис. 2.8), змінні моделі такі:

$$\begin{aligned} \bar{X} &:= [\bar{t}: \mathbb{R}] \\ \hat{Y} &:= [\hat{x}: \mathbb{R}], \\ \mathfrak{G} &:= [k: \mathbb{R} = .5] \end{aligned} \quad (2.67)$$

де t – час, x – положення тіла, а k – коефіцієнт уповільнення. Модель:

$$\hat{Y}(\bar{X}|\mathfrak{G})^F = F(\bar{X}|\mathfrak{G}) = \hat{Y} := [\hat{x} = \bar{t} * k]. \quad (2.68)$$

Нас може цікавити множина положень тіла $\hat{\mathfrak{Y}} \subset \hat{\mathbb{Y}}^n$ протягом певного інтервалу часу

$$\bar{\mathfrak{x}} := \begin{bmatrix} \bar{\mathfrak{x}}_1 = [t = t_{begin}] \\ \dots \\ \bar{\mathfrak{x}}_n = [t = t_{end}] \end{bmatrix}. \quad (2.69)$$

У випадку моделі $\hat{Y}(\bar{X}|\mathfrak{G})^F$, множину $\hat{\mathfrak{Y}}$ можна отримати простим обчисленням кожного $\hat{\mathfrak{Y}}_i$ з відповідного $\bar{\mathfrak{x}}_i$:

$$\hat{\mathfrak{Y}} := \begin{bmatrix} \hat{\mathfrak{Y}}_1 = F(\bar{X}|\mathfrak{G})(\bar{\mathfrak{x}}_1) = [t_1 * k] \\ \dots \\ \hat{\mathfrak{Y}}_n = F(\bar{X}|\mathfrak{G})(\bar{\mathfrak{x}}_n) = [t_n * k] \end{bmatrix}. \quad (2.70)$$

У випадку моделі $\hat{Y}(\bar{X}|\mathfrak{G})^S$, кожен $\hat{\mathfrak{Y}}_i$ має бути побудований з підстанів $\mathfrak{S}^{\bar{\mathfrak{x}}}_q$, знайдених у

$$\mathfrak{S}^{\bar{X}|\mathfrak{G}} := \begin{bmatrix} \mathfrak{S}^{\bar{\mathfrak{x}}}_{q=1, begin} = [x_{begin}]^{[t_{begin}]}_1 \\ \dots \\ \mathfrak{S}^{\bar{\mathfrak{x}}}_{q=1, end} = [x_{end}]^{[t_{end}]}_1 \end{bmatrix}, \quad (2.71)$$

згідно з відповідним $\bar{\mathfrak{X}}_i$:

$$\hat{\mathfrak{Y}} := \begin{bmatrix} \hat{\mathfrak{Y}}_1 = \cup \mathfrak{S}^{\bar{X}|\mathfrak{G}}(\bar{\mathfrak{X}}_1) \\ \vdots \\ \hat{\mathfrak{Y}}_n = \cup \mathfrak{S}^{\bar{X}|\mathfrak{G}}(\bar{\mathfrak{X}}_n) \end{bmatrix}, \quad (2.72)$$

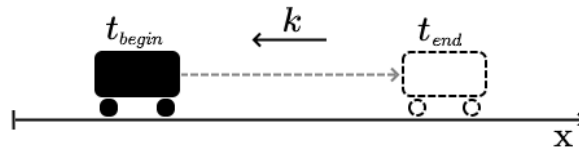


Рис. 2.8 Тіло, що рухається прямолінійно з уповільненням

Моделювання може бути інтерактивним, тобто воно може реагувати на зовнішні події та видавати результати безпосередньо під час розрахунку. У найпростішому випадку інтерактивне моделювання можна представити як серію моделювань.

$$\{\bar{\mathfrak{X}}_i \xrightarrow{\hat{Y}(\bar{X}|\mathfrak{G})_i} \hat{\mathfrak{Y}}_i\}, \quad (2.73)$$

набору моделей $\{\hat{Y}(\bar{X}|\mathfrak{G})_i\}$ для відповідних наборів підмножин значень незалежних змінних $\{\bar{\mathfrak{X}}_i\}$, послідовно отриманих під час взаємодії, та наборів залежних $\{\hat{\mathfrak{Y}}_i\}$, послідовно повернутих як результати моделювання.

2.10. Зведення всього разом: Відображення $\check{Y}(\bar{X}|\mathfrak{G})$, модель $\hat{Y}(\bar{X}|\mathfrak{G})$

Тепер можемо об'єднати всі раніше визначені частини та встановити спільний підхід до моделювання. Образно цей підхід показана на рис. 2.9. Це не єдиний можливий підхід, але він буде використовуватися в цій статті.

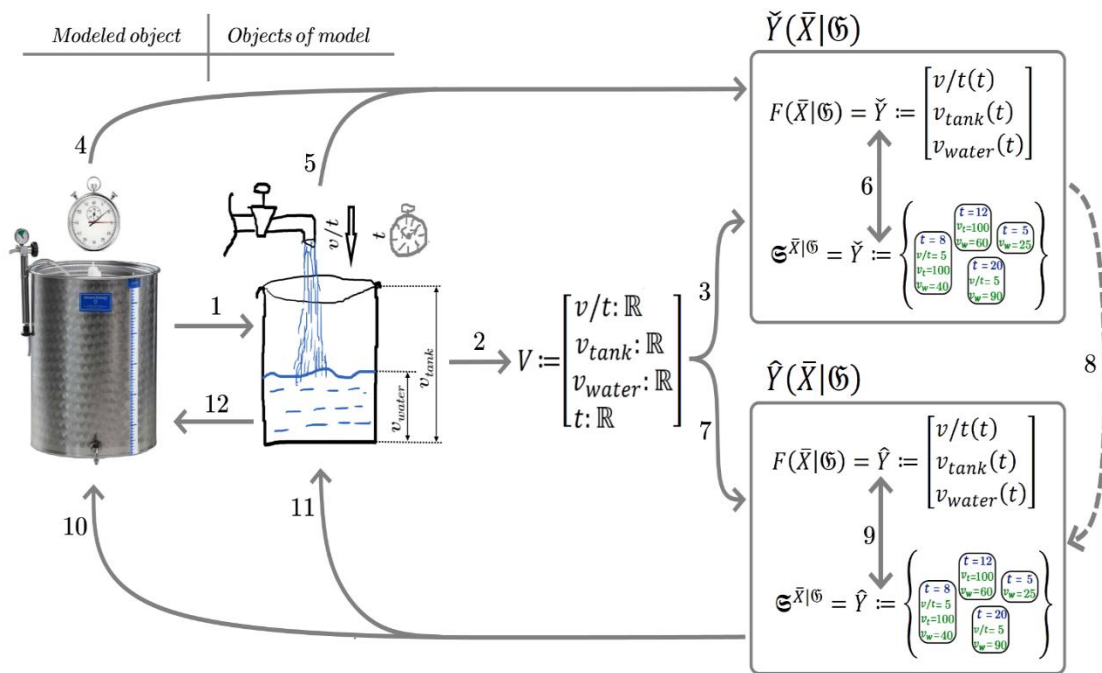


Рис. 2.9 Один з можливих підходів до моделювання

Можемо розділити об'єкти, що беруть участь у моделюванні, на фактично змодельований об'єкт (фізичний чи ні) та використані математичні об'єкти.

Коли вирішуємо будь-яку задачу моделювання, перше, що потрібно зробити, це побудувати концептуальну модель (абстрактне представлення модельованого об'єкта, рис. 2.9 (1)). Іншими словами, необхідно вибрати суттєві властивості модельованого об'єкта та їх суттєві взаємозалежності, щоб достатньо добре відобразити об'єкт для вирішення задачі моделювання.

Набір змінних V (рис. 2.9 (2)) можна побудувати на концептуальній моделі, представивши вибрані властивості модельованого об'єкта як математичні змінні v_i та визначивши набір значень v_i для кожної з них.

Відображення $\check{Y}(\bar{X}|\mathfrak{G})$, яке представляє функціональну залежність змінних V (рис. 2.9 (3)), може бути побудоване шляхом безпосереднього вимірювання властивостей (рис. 2.9 (4)) або отримане аналітично з концептуальної моделі (рис. 2.9 (5)). Відображення може бути реалізовано як набір підстанов $\check{Y} = \mathfrak{S}^{\bar{X}|\mathfrak{G}}$ або як

набір функцій $\check{Y} = \mathfrak{S}^{\bar{X}|\mathfrak{G}}$. Більше того, набір підстанів може бути перетворений на набір функцій і навпаки (рис. 2.9 (6)).

Маючи достатньо інформації про модельований об'єкт та множину змінних V (рис. 2.9 (7)), можемо побудувати його модель $\hat{Y}(\bar{X}|\mathfrak{G})$ (рис. 2.9 (8)) тим чи іншим способом, наприклад, аналітично у вигляді набору функцій $\hat{Y} = F(\bar{X}|\mathfrak{G})$ або чисельно як набір підстанів $\hat{Y} = \mathfrak{S}^{\bar{X}|\mathfrak{G}}$. Як і у випадку з відображенням, набір підстанів можна перетворити на набір функцій і навпаки (рис. 2.9 (9)).

Якщо побудована модель $\hat{Y}(\bar{X}|\mathfrak{G})$ достатньо добре імітує змодельований об'єкт, можемо використовувати її для прогнозування змін об'єкта (рис. 2.9 (10)) або для корекції його концептуальної моделі (рис. 2.9 (11)).

Далі, скориговану концептуальну модель можна використовувати для вивчення змодельованого об'єкта та впливу на його властивості (рис. 2.9 (12)).

ВИСНОВКИ ДО РОЗДІЛУ 2

У цьому розділі сформовано теоретичну основу для моделювання, визначивши ключові конструкції для математичного представлення об'єктів. Спершу уточнено роль часу: він критичний для опису еволюції фізичних систем, але може абстрагуватися, якщо не впливає на задачу.

Об'єкт подано як набір змінних із відповідними областями значень, розділеними на незалежні, залежні та параметри. Це дозволило визначити стан (конкретне присвоєння значень) і простір станів (усі можливі стани). Поведінку описано як траєкторії в просторі станів, що відображають зміни властивостей.

Запропоновано підхід до подання станів динамічних систем у вигляді множини підстанів із використанням унікальних ключів, що, на відміну від монолітних станів в традиційних паралельних моделях, забезпечує модульність, повторне використання та узгодженість при побудові графів переходів і обчислювальних графів в паралельних моделях.

Розрізнено відображення (фіксація спостережень) і модель (імітація залежностей). Моделювання визначено як процес обчислення залежних змінних із незалежних та параметрів, як у функціональному, так і у підстанному підходах.

Таким чином, змінні, стани, підстани, відображення й моделі утворюють цілісну методологію, що пов'язує реальні явища з їх обчислювальними представленнями. Вона забезпечує мову та процедури для побудови моделей, поєднуючи точність і обчислювальну практичність. Представлені поняття, зокрема взаємодія підстанів, слугуватимуть основою для подальших методів моделювання у наступних розділах.

РОЗДІЛ 3

ФОРМАЛЬНОЇ СТРУКТУРИ ОПИСУ ДИНАМІКИ ПІДСТАНІВ ЗА ДОПОМОГОЮ ПЕРЕХІДНИХ ФУНКЦІЙ

У цьому розділі розробляється формальна структура, необхідна для опису динаміки підстанів за допомогою перехідних функцій та їхньої організації в графові структурі. Вона починається з введення поняття перехідної функції, яка визначає, як невідомі підстани можуть бути виведені з відомих за певними правилами та параметрами. Ці функції потім виражаються в графічній формі, що дозволяє візуалізувати та систематично поєднувати їхні зв'язки.

Спираючись на цю основу, у розділі графи переходів визначено як спрямовані ациклічні графи (DAG), що складаються з взаємопов'язаних перехідних функцій, а графи моделювання – як їх втілення з конкретними підстанами та переходами. Особливий акцент робиться на вимогах до підтримки узгодженості в цих графах, забезпечуючи їх логічну когерентність та відсутність суперечностей.

Потім обговорення переходить до обчислюваності, зосереджуючись на тому, як графи моделювання можуть бути виведені з початкових підстанів та як відсутність невідповідностей гарантує правильну роботу. Вводиться представлення залежностей змінних у графовій формі разом з ілюстративними прикладами, такими як модель пружини Слінкі, які демонструють практичне застосування цієї структури.

Розділ завершується викладом підходів до моделювання на основі графових моделей, включаючи стратегії оптимізації та інтерактивного керування. Розглядаючи шаблони push та pull, у ньому висвітлюються методи синхронізації моделей із зовнішніми процесами або вхідними даними користувача.

Завдяки цим розробкам, розділ 3 забезпечує методологічну основу для побудови узгоджених, обчислюваних та адаптивних структур моделі, які слугують

вирішальним кроком до ширшої структури циклічного спрямованого ймовірнісного графічного моделювання.

У розділі 5 наводимо простий приклад побудови та моделювання графа переходів.

3.1. Перехідні функції $\theta^{|\mathfrak{D}}$

Можемо побудувати систему переходів на множині всіх можливих підстанів $\mathbb{S}^{X|\mathfrak{G}}$ (формула 2.47), додавши її до множини переходів [38-40]. Кожен елемент цієї множини визначає перехід від одного або кількох відомих підстанів $\mathfrak{S}^x_{q,k=1}, \dots, \mathfrak{S}^x_{q,k=n}$ до деякого невідомого підстану \mathfrak{S}^x_q та може бути параметризований деякою підмножиною значень параметрів з множини \mathfrak{G} .

Визначення: На множині підстанів $\mathbb{S}^{X|\mathfrak{G}}$, визначимо множину переходів:

$$\theta^{|\mathfrak{G}} := \left\{ \mathfrak{S}^x_{q,k=1}, \dots, \mathfrak{S}^x_{q,k=n} \xrightarrow{\theta^{|\mathfrak{D}}} \mathfrak{S}^x_q \right\}, \quad (3.1)$$

де $k, n \in N$, $\mathfrak{S}^x_{q,k=1}, \dots, \mathfrak{S}^x_{q,k=n}, \mathfrak{S}^x_q \in \mathbb{S}^{X|\mathfrak{G}}$ такі що

$$\mathfrak{S}^x_{q,k} \neq \mathfrak{S}^x_q, \quad (3.2)$$

$$\{\mathfrak{S}^x_{q,k=1}, \dots, \mathfrak{S}^x_{q,k=n}\} \cup \{\mathfrak{S}^x_q\} \subseteq \mathbb{S}^{X|\mathfrak{G}}, \quad (3.3)$$

$$\mathbb{S}^{X|\mathfrak{G}} \subseteq \mathbb{S}^{X|\mathfrak{G}}. \quad (3.4)$$

Тут $\theta^{|\mathfrak{D}}$ це довільне перехідне відношення від $\mathfrak{S}^x_{q,k=1}, \dots, \mathfrak{S}^x_{q,k=n}$ до \mathfrak{S}^x_q , параметризоване $\mathfrak{D} \subseteq \mathfrak{G}$.

На певній множині переходів $\theta^{|\mathfrak{G}}$ можемо визначити множину функцій переходу, кожна з яких описує підмножину $\theta^{|\mathfrak{G}}$. Іншими словами, функція переходу дозволяє нам знайти деякий перехід $\theta^{|\mathfrak{D}}$ шляхом отримання або обчислення невідомого підстану.

$$\mathfrak{S}^x_q \xleftarrow{\theta^{|\mathfrak{D}}}, \quad (3.5)$$

на основі набору відомих підстанів

$$\mathfrak{S}_{q,k=1}^{\mathfrak{x}}, \dots, \mathfrak{S}_{q,k=n}^{\mathfrak{x}} \xrightarrow{\theta^{|\mathfrak{D}}}, \quad (3.6)$$

використовуючи певне правило побудови, можливо параметризоване деяким $\mathfrak{D} \subseteq \mathfrak{G}$.

Визначення: На множині переходів $\theta^{|\mathfrak{G}}$ визначаємо множину функцій переходу

$$\Theta^{|\mathfrak{G}} := \left\{ S: \mathbb{S}_{k=1}, \dots, S: \mathbb{S}_{k=n} \xrightarrow{\theta^{|\mathfrak{D}}} \hat{S}: \hat{\mathbb{S}} \right\}, \quad (3.7)$$

де $k, n \in N$, $S_{k=1}, \dots, S_{k=n}$ – функціональні аргументи з доменом $\mathbb{S}_{k=1}, \dots, \mathbb{S}_{k=n} \subseteq \mathbb{S}^{X|\mathfrak{G}}$ є результатом функції домену $\hat{\mathbb{S}} \subseteq \mathbb{S}^{X|\mathfrak{G}}$, а $\theta^{|\mathfrak{D}}$ – довільне правило, яке можна параметризувати за допомогою підмножини параметрів $\mathfrak{D} \subseteq \mathfrak{G} \in \mathbb{G}^n$ і яке дозволяє отримати або обчислити невідомий підстан $\hat{\mathfrak{S}}_q^{\mathfrak{x}} \in \hat{\mathbb{S}}$ певного переходу $\theta^{|\mathfrak{D}} \in \Theta^{|\mathfrak{G}}$, використовуючи набір відомих підстанів $\mathfrak{S}_{q,k=1}^{\mathfrak{x}} \in \mathbb{S}_{k=1}, \dots, \mathfrak{S}_{q,k=n}^{\mathfrak{x}} \in \mathbb{S}_{k=n}$.

Графічно функцію переходу можна представити як перехід від точки \mathfrak{V}_n до наступної точки \mathfrak{V}_{n+1} у просторі станів V^n :

$$\mathfrak{V}_{n+1} = \theta^{|\mathfrak{D}}(\mathfrak{V}_n), \quad (3.8)$$

тут,

$$\mathfrak{V}_n = \bar{\mathfrak{X}}_n \cup \hat{\mathfrak{Y}}_n \cup \mathfrak{G}, \quad (3.9)$$

і

$$\hat{\mathfrak{Y}}_n = \bigcup \mathfrak{S}^{\bar{X}|\mathfrak{G}}(\bar{\mathfrak{X}}_n), \quad (3.10)$$

(рис. 3.1 (b)). Це відрізняється від представлення у вигляді набору функцій $\hat{\mathfrak{Y}} = F(\bar{X}|\mathfrak{G})(\bar{\mathfrak{X}})$, які безпосередньо пов'язують кожен $\bar{\mathfrak{X}}$ з кожним $\hat{\mathfrak{Y}}$ (рис. 3.1 (a)).

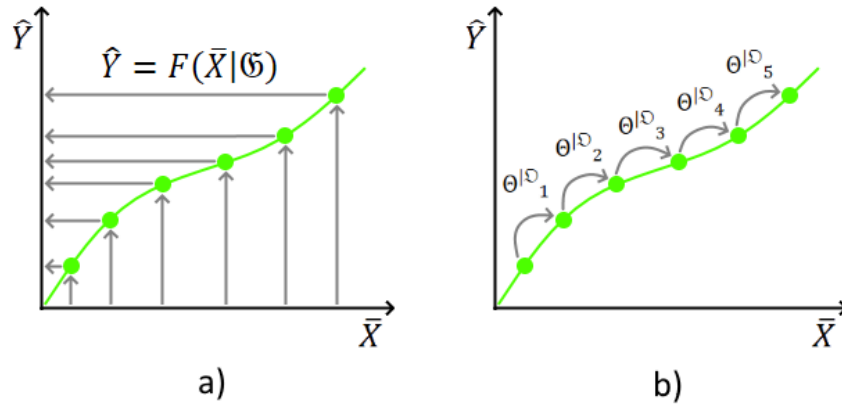


Рис. 3.1 Графічне представлення: а) набору функцій $\hat{\mathfrak{Y}} = F(\bar{X}|\mathfrak{G})(\bar{x})$; б) перехідної функції $\Theta^{|\mathfrak{D}}$

Множина $\theta^{|\mathfrak{G}}$ може бути еквівалентною деякій функції $\Theta^{|\mathfrak{D}}$, тобто для будь-якого набору її аргументів,

$$\mathfrak{S}_{q,k=1}^{\mathfrak{x}} \in \mathbb{S}_{k=1}, \dots, \mathfrak{S}_{q,k=n}^{\mathfrak{x}} \in \mathbb{S}_{k=n}, \quad (3.11)$$

і результат

$$\mathfrak{S}_q^{\mathfrak{x}} = \Theta^{|\mathfrak{D}}(\mathfrak{S}_{q,k=1}^{\mathfrak{x}}, \dots, \mathfrak{S}_{q,k=n}^{\mathfrak{x}}), \quad (3.12)$$

у множині $\theta^{|\mathfrak{G}}$ існує єдиний перехід

$$\mathfrak{S}_{q,k=1}^{\mathfrak{x}'} \dots, \mathfrak{S}_{q,k=n}^{\mathfrak{x}'} \xrightarrow{\theta^{|\mathfrak{D}}} \mathfrak{S}_q^{\mathfrak{x}'}, \quad (3.13)$$

такий, що

$$\mathfrak{S}_q^{\mathfrak{x}} = \mathfrak{S}_q^{\mathfrak{x}'} \wedge \mathfrak{S}_{q,k=1}^{\mathfrak{x}} = \mathfrak{S}_{q,k=1}^{\mathfrak{x}'} \wedge \dots \wedge \mathfrak{S}_{q,k=n}^{\mathfrak{x}} = \mathfrak{S}_{q,k=n}^{\mathfrak{x}'} \quad (3.14)$$

Позначимо таку множину через $\theta^{|\mathfrak{D}} \Leftrightarrow \Theta^{|\mathfrak{D}}$.

Зазначимо, що за визначенням група зв'язаних підстанів $\mathfrak{S}_{q,k=1}^{\mathfrak{x}}, \dots, \mathfrak{S}_{q,k=n}^{\mathfrak{x}}, \mathfrak{S}_q^{\mathfrak{x}}$ повинна належати до однієї і тільки однієї множини $\mathfrak{S}^{X|\mathfrak{G}}$ і, отже, вони не є суперечливими (формула 2.43). Водночас, незв'язані підстани можуть бути суперечливими, оскільки, в загальному випадку, змінні $\mathbb{S}_{k=1}, \dots, \mathbb{S}_{k=n}, \mathbb{S}$ можуть приймати будь-які значення з множини всіх можливих підстанів $\mathbb{S}^{X|\mathfrak{G}}$.

Кожна перехідна функція $S_{k=1}, \dots, S_{k=n} \xrightarrow{\Theta^{|\mathcal{D}}}$ можна представити як орієнтований граф (рис. 3.2) [41–44].

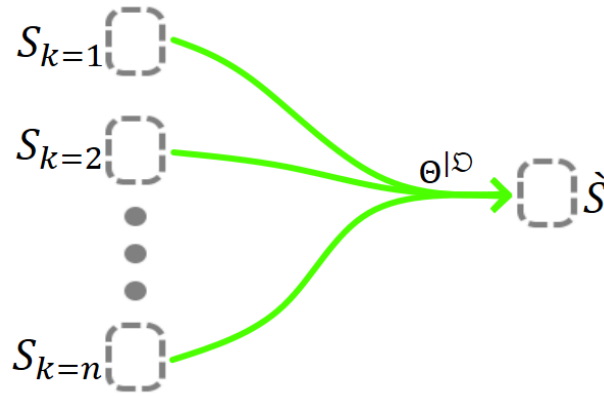


Рис. 3.2 Графове представлення перехідної функції $\Theta^{|\mathcal{D}}$

Множину переходів $\theta^{|\mathcal{D}} \Leftrightarrow \Theta^{|\mathcal{D}}$ можна представити як набір графів з такою ж структурою, як $\Theta^{|\mathcal{D}}$ (рис. 3.3). Більше того, у випадку, коли різні значення аргументів $\mathfrak{S}_{q,k=1}, \dots, \mathfrak{S}_{q,k=n}$ відповідають одному й тому ж результату \mathfrak{S}_q , покажемо такі переходи як окремі графи, по одному для кожного унікального набору аргументів.

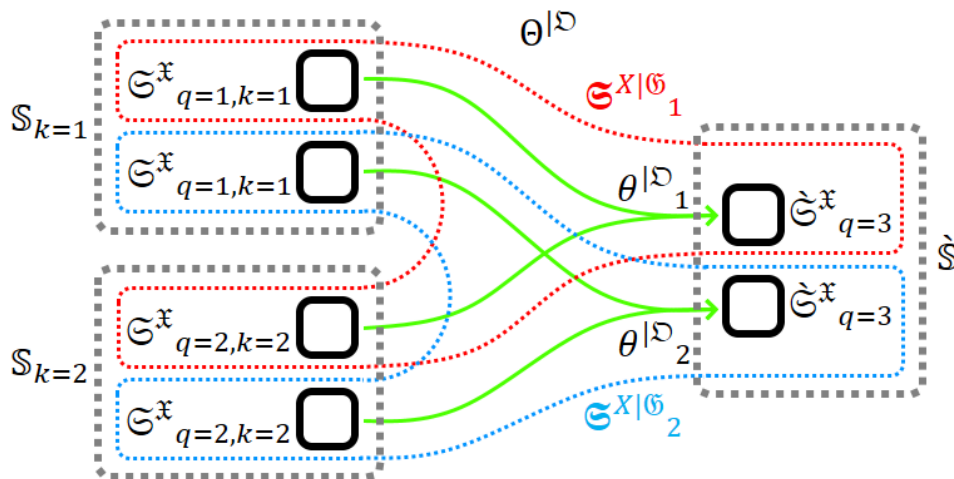


Рис. 3.3 Представлення перехідної функції $\Theta^{|\mathcal{D}}$ як набору переходів $\theta^{|\mathcal{D}}$.

3.2. Граф переходу $\Gamma^{|\mathfrak{G}}$ та граф моделювання $\gamma^{|\mathfrak{G}}$

Можемо об'єднати функцію $\Theta^{|\mathfrak{P}}_j \in \Theta^{|\mathfrak{G}}$ та набір функцій $\Theta^{|\mathfrak{D}}_{i,k=1}, \dots, \Theta^{|\mathfrak{D}}_{i,k=n} \in \Theta^{|\mathfrak{G}}$, представлених у вигляді графів (рис. 3.2), шляхом об'єднання результируючих вузлів

$$\xrightarrow{\Theta^{|\mathfrak{D}}_{i,k=1}} \dot{S}: \dot{S}_{i,k=1}, \dots, \xrightarrow{\Theta^{|\mathfrak{D}}_{i,k=n}} \dot{S}: \dot{S}_{i,k=n}, \quad (3.15)$$

та вузли аргументів

$$S: S_{k=1,j}, \dots, S: S_{k=n,j} \xrightarrow{\Theta^{|\mathfrak{D}}_j}, \quad (3.16)$$

з вузлами проміжної змінної

$$\left(\xrightarrow{\Theta^{|\mathfrak{D}}_{i,k=1}} S_{k=1}: \dot{S}_{i,k=1}, \dots, \xrightarrow{\Theta^{|\mathfrak{D}}_{i,k=n}} S_{k=n}: \dot{S}_{i,k=n} \right) \xrightarrow{\Theta^{|\mathfrak{D}}_j}, \quad (3.17)$$

(рис. 3.4).

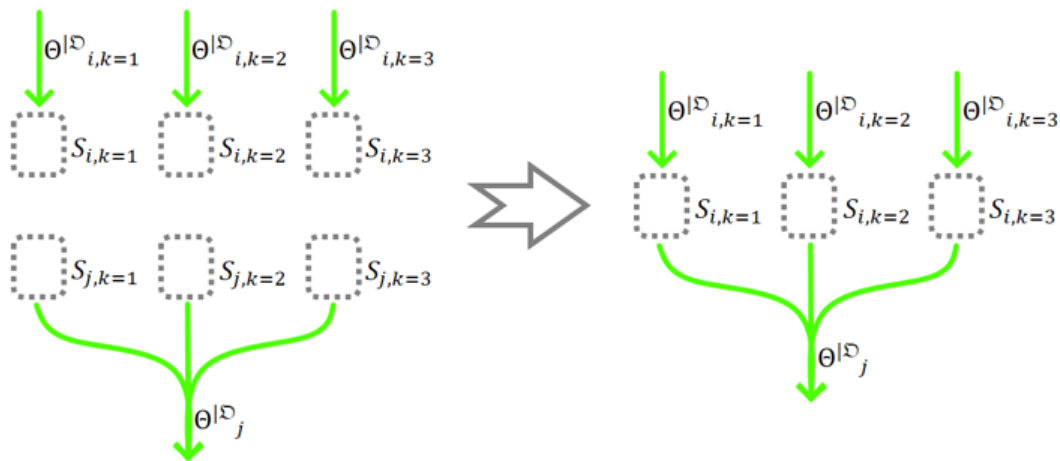


Рис. 3.4 Об'єднання функції $\Theta^{|\mathfrak{P}}_j \in \Theta^{|\mathfrak{G}}$

Продовжимо таким самим чином послідовно об'єднувати функції, що входять до одного набору $\Theta^{|\mathfrak{G}}$ (можливо, використовуючи ту саму функцію більше одного разу); отримаємо деякий DAG (рис. 3.5). Називатимемо такий DAG графом

переходів. Також, за бажанням, можемо об'єднати дві або більше кореневих змінних $S_{k,i=1}, \dots, S_{k,i=n}$, які не мають вхідних ребер, тим самим зменшуючи загальну кількість вузлів.

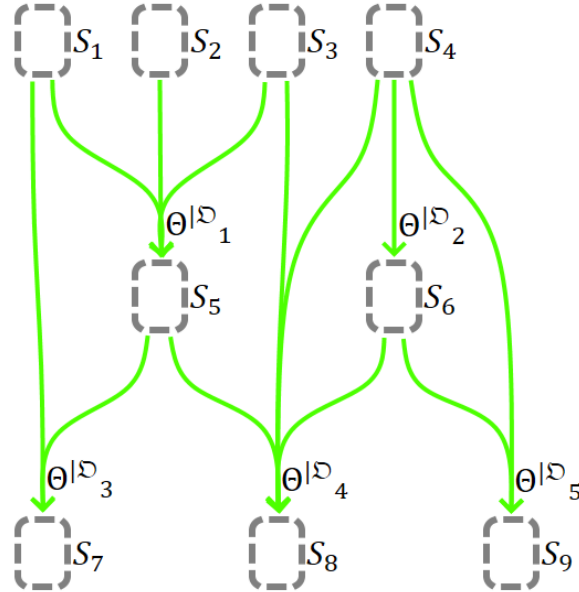


Рис. 3.5 Приклад перехідного DAG, побудованого з функцій $\theta^{|\mathcal{P}}_j \in \Theta^{|\mathcal{G}}$

Визначення: Визначаємо граф переходів $\Gamma^{|\mathcal{G}}$ як DAG, побудований на множині функцій переходу $\Theta^{|\mathcal{G}}$ шляхом послідовного об'єднання довільних підмножин функцій

$$\theta^{|\mathcal{D}}_i, \theta^{|\mathcal{D}}_{j,k=1}, \dots, \theta^{|\mathcal{D}}_{j,k=n} \in \Theta^{|\mathcal{G}}, \quad (3.18)$$

і шляхом об'єднання результуючого вузла

$$\dot{S}_i: \dot{S}_i \xleftarrow{\theta^{|\mathcal{D}}_i}, \quad (3.19)$$

та вузли аргументів

$$S_{k,j,k=1}: S_{k,j,k=1} \xrightarrow{\theta^{|\mathcal{D}}_{j,k=1}}, \dots, \quad (3.20)$$

$$\dots, S_{k,j,k=n}: S_{k,j,k=n} \xrightarrow{\theta^{|\mathcal{D}}_{j,k=n}}, \quad (3.21)$$

такий, що

$$\mathbb{S}_i \subseteq \mathbb{S}_{k,j,k=1}, \dots, \mathbb{S}_{k,j,k=n}, \quad (3.22)$$

$$\mathbb{S}_i \cap \mathbb{S}_{k,j,k=1}, \dots, \mathbb{S}_{k,j,k=n} \neq \emptyset, \quad (3.23)$$

у проміжні змінні вузли

$$\xrightarrow{\Theta^{|\mathcal{D}|}_i} S: \mathbb{S}_i \xrightarrow{\Theta^{|\mathcal{D}|}_{j,k=1}, \dots, \Theta^{|\mathcal{D}|}_{j,k=n}}, \quad (3.24)$$

Крім того, кореневі вузли

$$S_{k,i=1} \cup \dots \cup S_{k,i=n}, \quad (3.25)$$

та їхні домени

$$\mathbb{S}_{k,i=1} \cap \dots \cap \mathbb{S}_{k,i=n}, \quad (3.26)$$

також можуть бути поєднані.

Зазначимо, що це визначення не накладає жодних обмежень на структуру графа, окрім його ациклічності (результат наступного об'єднаного $\Theta^{|\mathcal{D}|}$ не може бути пов'язаний з аргументом жодного вже об'єднаного $\Theta^{|\mathcal{D}|}$) та неперервності (всі вузли графа $\Gamma^{|\mathcal{G}|}$ з'єднані принаймні одним ребром).

Коли об'єднуємо перехідні функції $\Theta^{|\mathcal{D}|}$, також об'єднуємо переходи $\theta^{|\mathcal{D}|}$ з еквівалентної множини $\theta^{|\mathcal{D}|} \Leftrightarrow \Theta^{|\mathcal{D}|}$, утворюючи набір складніших DAG з такою ж структурою, як і граф $\Gamma^{|\mathcal{G}|}$, але які складаються з підстанів \mathfrak{S}_q^x та переходів $\theta^{|\mathcal{D}|}$ (рис. 3.6) [45-47]. Називаємо такі DAG графами моделювання.

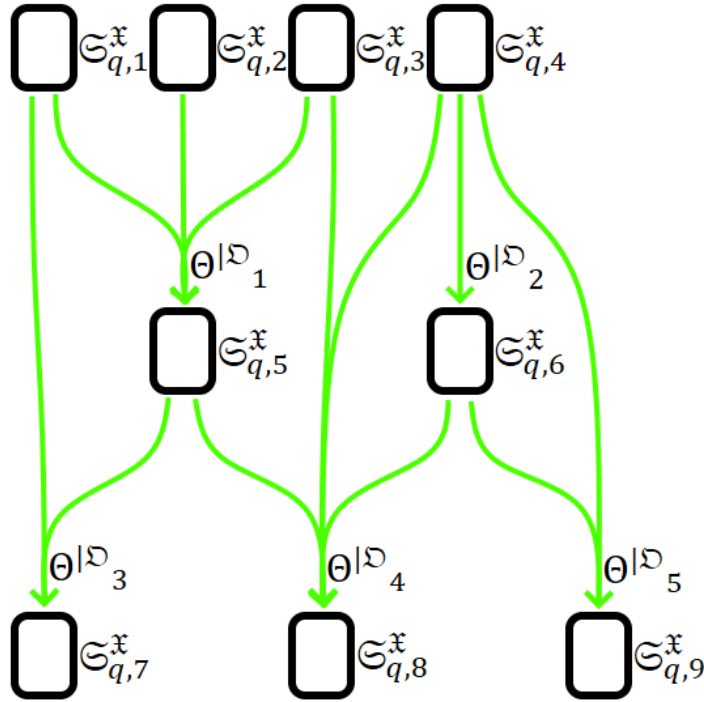


Рис. 3.6 Приклад графа моделювання, який можна отримати з графа переходу на рис. 3.5

Визначення: Граф моделювання $\gamma^{|\mathfrak{G}|}$ визначається як DAG, отриманий шляхом побудови графа переходів $\Gamma^{|\mathfrak{G}|}$; він має таку ж структуру, як $\Gamma^{|\mathfrak{G}|}$. Граф $\gamma^{|\mathfrak{G}|}$ складається з конструкцій вигляду

$$\xrightarrow{\theta^{|\mathfrak{D}|}_i} \mathfrak{S}_{q,i}^x \xrightarrow{\theta^{|\mathfrak{D}|}_{j,k=1}, \dots, \theta^{|\mathfrak{D}|}_{j,k=n}}, \quad (3.27)$$

які виникають в результаті об'єднання переходів

$$\theta^{|\mathfrak{D}|}_i \in \boldsymbol{\theta}^{|\mathfrak{D}|}_i \Leftrightarrow \Theta^{|\mathfrak{D}|}_i, \quad (3.28)$$

i

$$\begin{aligned} & \theta^{|\mathfrak{D}|}_{j,k=1} \in \boldsymbol{\theta}^{|\mathfrak{D}|}_{j,k=1} \Leftrightarrow \\ & \Leftrightarrow \theta^{|\mathfrak{D}|}_{j,k=1}, \dots, \theta^{|\mathfrak{D}|}_{j,k=n} \in \boldsymbol{\theta}^{|\mathfrak{D}|}_{j,k=n} \Leftrightarrow, \\ & \Leftrightarrow \Theta^{|\mathfrak{D}|}_{j,k=n} \end{aligned} \quad (3.29)$$

що належать до множини об'єднаних функцій

$$\Theta^{|\mathfrak{D}|}_i, \Theta^{|\mathfrak{D}|}_{j,k=1}, \dots, \Theta^{|\mathfrak{D}|}_{j,k=n} \in \boldsymbol{\Theta}^{|\mathfrak{G}|}, \quad (3.30)$$

такий, що

$$\mathfrak{S}_{q,i}^{\mathfrak{x}} \xleftarrow{\theta^{\mathfrak{D}}_i} = \mathfrak{S}_{q,j,k=1}^{\mathfrak{x}} \xrightarrow{\theta^{\mathfrak{D}}_{j,k=1}} =, \dots, = \mathfrak{S}_{q,j,k=n}^{\mathfrak{x}} \xrightarrow{\theta^{\mathfrak{D}}_{j,k=n}}. \quad (3.31)$$

Зазначимо, що всі $\gamma^{\mathfrak{G}}$ матимуть структуру, яка точно відповідає $\Gamma^{\mathfrak{G}}$. Згідно з визначенням $\gamma^{\mathfrak{G}}$, під час побудови $\Gamma^{\mathfrak{G}}$ неповні графи $\gamma^{\mathfrak{G}}$ зі структурами, що не збігаються зі структурою $\Gamma^{\mathfrak{G}}$, будуть відкинуті. Таким чином, підстани $\mathfrak{S}_q^{\mathfrak{x}}$, що входять до відкинутих графів $\gamma^{\mathfrak{G}}$, також будуть видалені з доменів \mathbb{S} змінних S , що входять до побудованої $\Gamma^{\mathfrak{G}}$.

Позначимо деякий довільний набір графів $\{\gamma^{\mathfrak{G}}\}$ через $\boldsymbol{\gamma}^{\mathfrak{G}}$. Згідно з визначеннями графів $\Gamma^{\mathfrak{G}}$ та $\gamma^{\mathfrak{G}}$, кожен з підстанів $\mathfrak{S}_q^{\mathfrak{x}}$ з доменів \mathbb{S} змінних вузлів S належатиме одному з графів моделювання $\gamma^{\mathfrak{G}}$. Усі терміни $\mathfrak{S}_q^{\mathfrak{x}}$, які не належать жодному $\gamma^{\mathfrak{G}}$, будуть відкинуті під час побудови $\Gamma^{\mathfrak{G}}$, разом з неповним $\gamma^{\mathfrak{G}}$.

Таким чином, можемо представити граф $\Gamma^{\mathfrak{G}}$ як еквівалентну множину графів $\gamma^{\mathfrak{G}}$. Позначимо таку множину як $\boldsymbol{\gamma}^{\mathfrak{G}} \Leftrightarrow \Gamma^{\mathfrak{G}}$; ця множина включатиме всі $\mathfrak{S}_q^{\mathfrak{x}}$ з усіх доменів \mathbb{S} :

$$\cup \mathbb{S}(\Gamma^{\mathfrak{G}}) = \cup_{\gamma^{\mathfrak{G}} \in \boldsymbol{\gamma}^{\mathfrak{G}} \Leftrightarrow \Gamma^{\mathfrak{G}}} \mathfrak{S}^{\mathfrak{x}}(\gamma^{\mathfrak{G}}), \quad (3.32)$$

де $\mathbb{S}(\Gamma^{\mathfrak{G}})$ – множина доменів \mathbb{S} змінних вузлів S з графа $\Gamma^{\mathfrak{G}}$, а $\mathfrak{S}^{\mathfrak{x}}(\gamma^{\mathfrak{G}})$ – множина всіх $\mathfrak{S}_q^{\mathfrak{x}}$, що належать до $\gamma^{\mathfrak{G}}$, що є узгодженим:

$$\exists \mathfrak{S}^{\mathfrak{X}^{\mathfrak{G}}} (\mathfrak{S}^{\mathfrak{x}}(\gamma^{\mathfrak{G}}) = \mathfrak{S}^{\mathfrak{X}^{\mathfrak{G}}}), \quad (3.33)$$

Більше того, всі графи моделювання $\gamma^{\mathfrak{G}}$ матимуть однаковий набір параметрів \mathfrak{G} , розділений на частини \mathfrak{D} .

Проіндексуємо кожен вузол з множини $\mathbb{S}(\Gamma^{\mathfrak{G}})$ індексом глибини

$$d = \max(\text{len}(\{\mathbf{S}_{root} \dots S_d\})), \quad (3.34)$$

де $d \in \mathbb{N}$, $\{\mathbf{S}_{root} \dots S_d\}$ – множина всіх можливих шляхів від будь-якого кореневого вузла $S_{root} \in \mathbf{S}_{root}$ (тобто вузла, який не має вхідних ребер) до індексованого вузла

S_d , а $len()$ – довжина шляху (кількість ребер у шляху). Також проіндексуємо всі функції переходу $\Theta^{|\mathcal{D}}$ тим самим індексом d , що й індекс результуючого вузла.

$$\dots \xrightarrow{\Theta^{|\mathcal{D}}_d} S_d, \quad (3.35)$$

Таким чином, кожен кореневий вузол S_{root} матиме $d = 0$, а кожен листовий вузол S_{leaf} (тобто такий, що не має вихідних ребер) матиме $d = n$, де n – мінімальна кількість ребер до найближчого кореневого вузла $S_{d=0}$.

3.3. Побудова узгодженого графа переходів $\Gamma^{|\mathcal{G}}$

З практичної точки зору, хочемо мати можливість конструювати переходи $\Gamma^{|\mathcal{G}}$ з набору попередньо визначених функцій переходів $\Theta^{|\mathcal{G}}$ це іншими словами, будувати моделі з набору готових функціональних блоків, подібно до програмного забезпечення Simulink. Для успішної реалізації цього підходу необхідно гарантувати узгодженість $\Gamma^{|\mathcal{G}}$ на локальному рівні, тобто на рівні окремих функцій $\Theta^{|\mathcal{F}}$.

Можемо представити деякий граф моделювання $\gamma^{|\mathcal{G}}$ як множину спрямованих шляхів (дишляхів), що охоплюють усі підстани $\mathcal{S}^x_q \in \mathcal{S}^x(\gamma^{|\mathcal{G}})$ та переходи $\theta^{|\mathcal{D}} \in \theta(\gamma^{|\mathcal{G}})$.

Визначення: Давайте визначимо d-шлях

$$p^{|\mathcal{G}} := \mathcal{S}^x_{q,l=0} \xrightarrow{\theta^{|\mathcal{D}}_{l=1}} \dots \xrightarrow{\theta^{|\mathcal{D}}_{l=n}} \mathcal{S}^x_{q,l=n}, \quad (3.36)$$

у графі моделювання $\gamma^{|\mathcal{G}} \in \mathcal{Y}^{|\mathcal{G}} \Leftrightarrow \Gamma^{|\mathcal{G}}$, де $\mathcal{S}^x_{q,l=0} \in \mathcal{S}_{l=0}$, $\mathcal{S}^x_{q,l=n} \in \mathcal{S}_{l=n}$, $l \in \mathbb{N}$ – індекс вузла, що знаходиться в дишляху, таким чином, що $l = 0$ відповідає деякому кореневому вузлу $\mathcal{S}^x_{q,root}$, а $l = n$ відповідає деякому листовому вузлу $\mathcal{S}^x_{q,leaf}$ у графі $\gamma^{|\mathcal{G}}$, $\mathcal{S}^x_{q,l=0}, \dots, \mathcal{S}^x_{q,l=n} \in \mathcal{S}^x(\gamma^{|\mathcal{G}})$, $\theta^{|\mathcal{D}}_{l=1}, \dots, \theta^{|\mathcal{D}}_{l=n} \in \theta(\gamma^{|\mathcal{G}})$, $\mathcal{S}_{l=0}, \dots, \mathcal{S}_{l=n} \in \mathcal{S}(\Gamma^{|\mathcal{G}})$, та $\mathcal{S}^x(p^{|\mathcal{G}}) \subseteq \mathcal{S}^x(\gamma^{|\mathcal{G}})$.

Позначимо через $\mathbf{p}^{|\mathfrak{G}}$ деяку довільну множину шляхів, яка не обов'язково пов'язана з тим самим графом $\gamma^{|\mathfrak{G}}$.

Множина шляхів $\mathbf{p}^{|\mathfrak{G}}$ може бути еквівалентною графу $\gamma^{|\mathfrak{G}}$, якщо шляхи в цій множині містять усі підстани $\mathfrak{S}^{\mathfrak{x}}_q \in \mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}})$ та переходи $\theta^{|\mathfrak{D}} \in \theta(\gamma^{|\mathfrak{G}})$:

$$\begin{aligned} \mathbf{p}^{|\mathfrak{G}} \Leftrightarrow \gamma^{|\mathfrak{G}} &\Rightarrow \bigcup_{\mathbf{p}^{|\mathfrak{G}} \ni \mathbf{p}^{|\mathfrak{G}}} \mathfrak{S}^{\mathfrak{x}}(\mathbf{p}^{|\mathfrak{G}}) = \\ &= \mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}}_j) \wedge \bigcup_{\mathbf{p}^{|\mathfrak{G}} \ni \mathbf{p}^{|\mathfrak{G}}} \theta\left(\mathbf{p}(\gamma^{|\mathfrak{G}}_j)\right) = \\ &= \theta(\gamma^{|\mathfrak{G}}_j), \end{aligned} \quad (3.37)$$

Щоб гарантувати умову узгодженості

$$\forall \gamma^{|\mathfrak{G}} \in \mathbf{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}(\mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}}) \subseteq \mathfrak{S}^{X|\mathfrak{G}}), \quad (3.38)$$

для графа $\Gamma^{|\mathfrak{G}}$ (тобто, щоб гарантувати, що кожен із графів моделювання $\gamma^{|\mathfrak{G}}$, описаних $\Gamma^{|\mathfrak{G}}$, не міститиме жодних несумісних підстанів), граф $\Gamma^{|\mathfrak{G}}$ повинен відповідати наступним двом обмеженням:

1. Для кожного графа $\gamma^{|\mathfrak{G}}_j \in \mathbf{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$, кожен підстан $\mathfrak{S}^{\mathfrak{x}}_{q,j} \in \mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}}_j), \mathfrak{S}^{\mathfrak{x}}(\mathbf{p}^{|\mathfrak{G}}_j)$ (тобто розташований на одному з усіх можливих шляхів $\mathbf{p}^{|\mathfrak{G}}_j$) повинен мати унікальний ключ $\mathfrak{X} \subseteq \mathfrak{S}^{\mathfrak{x}}_{q,j}$ щодо $\mathbf{p}^{|\mathfrak{G}}_j$.
2. Для кожного графа $\gamma^{|\mathfrak{G}} \in \mathbf{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$, значення $\mathfrak{y} \in \mathfrak{y}$ деякої змінної $\mathfrak{y}: \mathfrak{y} \in \mathfrak{Y}$ повинні належати лише до множини підстанів $\mathfrak{S}^{\mathfrak{x}}_{q,j} \in \mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}}_j)$ таких, що в $\gamma^{|\mathfrak{G}}$ існує принаймні один шлях $\mathbf{p}^{|\mathfrak{G}} \in \mathbf{\mathbf{p}^{|\mathfrak{G}}} \Leftrightarrow \gamma^{|\mathfrak{G}}$, включаючи всі ці підстани.

На локальному рівні (тобто без вивчення всього графа $\Gamma^{|\mathfrak{G}}$), вищезазначені обмеження можна виконати, застосовуючи такі принципи побудови [44]:

- I. Набір ключів \mathfrak{X}^n має бути лінійно впорядкований.
- II. Кожна перехідна функція

$$(S: \mathbb{S}_{k=1}, \dots, S: \mathbb{S}_{k=n}) \xrightarrow{\Theta^{|\mathbb{D}|}} \hat{S}: \hat{\mathbb{S}}, \quad (3.39)$$

(де $\Theta^{|\mathbb{D}|} \in \Theta(\Gamma^{|\mathbb{G}|})$) для кожного переходу

$$(\mathfrak{S}^{\mathfrak{x}}_{q,j,k=1}, \dots, \mathfrak{S}^{\mathfrak{x}}_{q,j,k=n}) \xrightarrow{\theta^{|\mathbb{D}|}_j} \mathfrak{S}^{\mathfrak{x}}_{q,j}, \quad (3.40)$$

(де $\theta^{|\mathbb{D}|}_j \in \theta(\mathfrak{p}^{|\mathbb{G}|}_j)$, $\theta^{|\mathbb{D}|} \Leftrightarrow \Theta^{|\mathbb{D}|}$) у деякому графі $\gamma^{|\mathbb{G}|}_j \in \mathbf{\gamma}^{|\mathbb{G}|} \Leftrightarrow \Gamma^{|\mathbb{G}|}$ має генерувати результуючий підстан $\mathfrak{S}^{\mathfrak{x}}_{q,j} \in \mathbb{S}, \mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathbb{G}|}_j)$ такий, що його ключ $\mathfrak{X}_{q,j} \subseteq \mathfrak{S}^{\mathfrak{x}}_{q,j}$ завжди задовольнятиме умови

$$\mathfrak{X}_{q,j} > \max(\mathfrak{X}_{q,j,k=1}, \dots, \mathfrak{X}_{q,j,k=n}), \quad (3.41)$$

або

$$\mathfrak{X}_{q,j} < \min(\mathfrak{X}_{q,j,k=1}, \dots, \mathfrak{X}_{q,j,k=n}), \quad (3.42)$$

де

$$\mathfrak{X}_{q,j,k} \subseteq \mathfrak{S}^{\mathfrak{x}}_{q,j,k} \in \mathbb{S}_k, \mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathbb{G}|}_j), \quad (3.43)$$

III. Для кожної змінної $y: y \in Y$ її значення $\eta \in y$ повинні належати не більше ніж одному кореневому вузлу $S: \mathbb{S}_{l=0}$:

$$\forall y: y \in Y (|\{S: \mathbb{S}_{l=0} \in S(\Gamma^{|\mathbb{G}|}) \mid \exists \eta \in y, \eta(\mathbb{S}_{l=0})\}| \leq 1), \quad (3.44)$$

де

$$\eta(\mathbb{S}) := \cup \mathfrak{S}(\mathbb{S}), \quad (3.45)$$

$$\mathfrak{S}(\mathbb{S}) := \{\mathfrak{S}_q \mid \mathfrak{S}_q \subset \mathfrak{S}^{\mathfrak{x}}_q \in \mathbb{S}\}, \quad (3.46)$$

(тобто, множина всіх значень η у всіх підстанах $\mathfrak{S}^{\mathfrak{x}}_q$ утворює домен визначення змінної \mathbb{S}).

IV. Якщо для деякого вузла $S: \mathbb{S}_i \in S(\Gamma^{|\mathbb{G}|})$ та деякої змінної $y: y \in Y$ умова $\exists \eta \in y, \eta(\mathbb{S}_i)$ є істинною, тоді або вузол S_i має бути коренем, або має існувати функція переходу

$$(\dots, S: \mathbb{S}_{i,k=0}, \dots, S: \mathbb{S}_{i,k=n}, \dots) \xrightarrow{\Theta^{|\mathbb{D}|}_i} \hat{S}_i, \quad (3.47)$$

з одним або кількома аргументами $S: \mathbb{S}_{i,k}$, для яких виконується умова $\exists \eta \in \mathcal{Y}, \eta(\mathbb{S}_i)$ та в графі $\Gamma^{|\mathbb{G}|}$ існує ланцюг

$$S_{i,k=0} \xrightarrow{\Theta^{|\mathbb{D}|}_{i,k=1}}, \dots, \xrightarrow{\Theta^{|\mathbb{D}|}_{i,k=n}} S_{i,k=n}, \quad (3.48)$$

що включає всі $S_{i,k}$. Більше того, для останнього аргументу $S_{i,k=n}$ у ланцюжку не повинно бути іншої функції

$$(\dots, S_{i,k=n}, \dots) \xrightarrow{\Theta^{|\mathbb{D}|}_i'} \hat{S}_i', \quad (3.49)$$

для яких виконується умова $\exists \eta \in \mathcal{Y}, \eta(\hat{S}_i')$.

На практиці принцип (I) можна легко реалізувати, оскільки лінійно впорядковані множини є поширеними. Наприклад, час, швидкість тощо можна представити за допомогою змінних з \mathbb{R} . Далі, якщо домени всіх незалежних змінних розташовані в лінійному порядку, то набір ключів \mathbb{X}^n також буде в лінійному порядку.

Принцип (II) стверджує, що парі ключ-значення постійно збільшується або зменшується під час обчислення графа моделювання $\gamma^{|\mathbb{G}|}$. Цей підхід можна застосувати, наприклад, до фізичних моделей, де незалежні змінні зазвичай є раціональними числами, які збільшуються або зменшуються під час моделювання.

Принцип (III) виконується, якщо граф $\Gamma^{|\mathbb{G}|}$ має один кореневий вузол $S: \mathbb{S}_{l=0} \in S(\Gamma^{|\mathbb{G}|})$ такий, що в кожному графі $\gamma^{|\mathbb{G}|}_j$ буде лише один підстан $\mathbb{S}^x_{q,j,l=0} \in \mathbb{S}_{l=0}$, тим самим виключаючи можливість того, що значення $\eta \in \mathcal{Y}$ однієї й тієї ж змінної $y: y \in \mathcal{Y}$ знаходяться в різних підстанах $\mathbb{S}^x_{q,j,l=0}$.

Інший підхід до реалізації (III) полягає в тому, щоб кожен кореневий вузол $S: \mathbb{S}_{l=0}$ включав значення $\eta \in \mathcal{Y}$ лише з власного унікального набору змінних $y_1, \dots, y_n \subset \mathcal{Y}$, таких що

$$\forall y_i, y_j (i \neq j, y_i \cap y_j = \emptyset). \quad (3.50)$$

Такий підхід, наприклад, зручний у графах $\Gamma^{|\mathfrak{G}|}$, що використовуються для інтерактивного моделювання, де кожен наступний вузол $S_{l=0}$ відображає наступний вхід даних ззовні моделювання.

На практиці, простий спосіб реалізації принципу (IV) полягає в перевірці, чи додавання наступної функції $\Theta^{|\mathfrak{D}|}$ для утворення \hat{S} не включає змінні, які вже є в результатах функцій, що мають спільні аргументи S з $\Theta^{|\mathfrak{D}|}$. Наприклад, якщо є вузли $S_{k=1}$ та $S_{k=2}$, для яких $y(S_{k=1}) = [a, b]$ та $y(S_{k=2}) = [x, y]$, де

$$y(S: \mathbb{S}) := \{y: y \in Y \mid \exists \eta \in y, \eta(S)\}, \quad (3.51)$$

і ці вузли є аргументами деякої функції

$$(S_{k=1}, S_{k=2}) \xrightarrow{\Theta^{|\mathfrak{D}|}_{j=1}} \hat{S}_{j=1}, \quad (3.52)$$

для якого результатом є $y(\hat{S}_{j=1}) = [a, x]$, тоді можемо додати лише функцію

$$(S_{k=1}, S_{k=2}) \xrightarrow{\Theta^{|\mathfrak{D}|}_{j=1}} \hat{S}_{j=1}, \quad (3.53)$$

для яких $y(\hat{S}_{j=2}) = [b, y]$ та або $y(\hat{S}_{j=2}) = [y]$, або $y(\hat{S}_{j=2}) = [b]$, але не $y(\hat{S}_{j=2}) = [a, b, y]$.

3.4. Обчислюваність графа моделювання $\gamma^{|\mathfrak{G}|}$ та початкової множини підстанів \mathfrak{S}'

На практиці нам потрібно буде знайти певний граф моделювання $\gamma^{|\mathfrak{G}|} \in \mathbf{\Upsilon}^{|\mathfrak{G}|} \Leftrightarrow \Gamma^{|\mathfrak{G}|}$ з деякого відомого набору узгоджених підстанів $\mathfrak{S}^{X|\mathfrak{G}} \subseteq \mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}|})$, пов'язаних з вузлами S графа $\Gamma^{|\mathfrak{G}|}$. Називатимемо $\mathfrak{S}^{X|\mathfrak{G}}$ початковим набором підстанів.

Визначення: Визначимо початковий набір підстанів

$$\mathfrak{S}' := \{S = \mathfrak{S}_q^{\mathfrak{X}}\}, \quad (3.54)$$

пов'язані з певними вузлами S графа $\Gamma^{|\mathfrak{G}|}$ такі, що

$$\exists! \gamma^{|\mathfrak{G}|} \left(\mathfrak{S}' \subseteq \mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}|}) \right), \quad (3.55)$$

де $S: \mathbb{S} \in S(\Gamma^{|\mathfrak{G}|})$, $\mathfrak{S}_q^{\mathfrak{X}} \in \mathbb{S}$, $\mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}|})$, $\gamma^{|\mathfrak{G}|} \in \mathbf{\Upsilon}^{|\mathfrak{G}|} \Leftrightarrow \Gamma^{|\mathfrak{G}|}$.

Пошук конкретного графа $\gamma^{|\mathbb{G}|}$ з деякою множиною \mathfrak{S}' можна імперативно представити як обчислення всіх функцій $\theta^{|\mathbb{D}|} \in \Theta(\Gamma^{|\mathbb{G}|})$, використовуючи \mathfrak{S}' як початкові аргументи для цих функцій.

Зауважте, що визначення вимагає, щоб \mathfrak{S}' було підмножиною однієї й тільки однієї множини $\mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathbb{G}|})$. Однак, у загальному випадку, деяка $\mathfrak{S}^{X|\mathbb{G}|}$ може бути підмножиною більш ніж однієї $\mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathbb{G}|})$. У цьому випадку, в імперативному представленні пошуку, єдиний графік $\gamma^{|\mathbb{G}|}$ не може бути обчислений з такої $\mathfrak{S}^{X|\mathbb{G}|}$, оскільки для деяких або всіх функцій $\theta^{|\mathbb{D}|} \in \Theta(\Gamma^{|\mathbb{G}|})$ не всі аргументи визначені.

Представляючи пошук конкретного $\gamma^{|\mathbb{G}|}$ у вигляді обчислення функцій $\theta^{|\mathbb{D}|} \in \Theta(\Gamma^{|\mathbb{G}|})$, помічаємо, що всі $\theta^{|\mathbb{D}|}$ будуть обчислені лише тоді, коли значення всіх кореневих вузлів $S_{root} \in S(\Gamma^{|\mathbb{G}|})$ відомі або можуть бути отримані певним чином. Таким чином, \mathfrak{S}' є підмножиною унікальної множини $\mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathbb{G}|})$ (формула 3.55) тоді і тільки тоді, коли для кожного початкового вузла $S_{root} \in S(\Gamma^{|\mathbb{G}|})$ існує шлях

$$S_{root} \xrightarrow{\theta^{|\mathbb{D}|}_1} \dots \xrightarrow{\theta^{|\mathbb{D}|}_n} S_{def}, \quad (3.56)$$

де $S_{def} \in S(\Gamma^{|\mathbb{G}|}), S(\mathfrak{S}')$ вузол, значення якого визначено в \mathfrak{S}' . І всі функції $\theta^{|\mathbb{D}|}_i$ на цьому оборотному вузлі [44].

Ще однією важливою властивістю цього підходу є відсутність глітчів, описаних у [48,49]. Оскільки потрібно знайти лише один граф $\gamma^{|\mathbb{G}|}$, ніколи не існує несумісних підстанів $\mathfrak{S}^{\mathfrak{X}}_q$.

3.5. Представлення залежності Y від X вигляді графової моделі $\hat{Y}(\bar{X}|\mathbb{G})^{\Gamma}$

Залежність залежних змінних Y від незалежних змінних X можна представити як кортеж графа переходів $\Gamma^{|\mathbb{G}|}$ та множину початкових підстанів \mathfrak{S}' із заданими значеннями параметрів \mathbb{G} .

Визначення: Давайте визначимо пару

$$Y = \langle \Gamma^{|\mathfrak{G}}, \mathfrak{S}' \rangle^X := \bigcup_{\mathfrak{x} \in \mathbb{X}^n} \mathfrak{S}^{\mathfrak{x}} \left(\gamma(\Gamma^{|\mathfrak{G}} | \mathfrak{S}') \right) (X), \quad (3.57)$$

що представляє залежність змінних Y від змінних X , параметризованих значеннями \mathfrak{G} , де

$$\gamma^{|\mathfrak{G}} = \gamma(\Gamma^{|\mathfrak{G}} | \mathfrak{S}'), \quad (3.58)$$

це графік моделювання $\gamma^{|\mathfrak{G}} \in \mathbf{Y}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$, знайдений для заданих $\Gamma^{|\mathfrak{G}}$, \mathfrak{S}' та \mathfrak{G} , та

$$\mathfrak{Y} = \bigcup_{\mathfrak{x} \in X} \mathfrak{S}^{X|\mathfrak{G}}(\mathfrak{x}), \quad (3.59)$$

це операція об'єднання підстанів $\mathfrak{S}_q^{\mathfrak{x}} \in \mathfrak{S}^{X|\mathfrak{G}}$ з тим самим ключем $\mathfrak{x} \in \mathbb{X}^n$ у множину значень $\mathfrak{Y} \in \mathbb{Y}^n$.

Представлення $Y = \langle \Gamma^{|\mathfrak{G}}, \mathfrak{S}' \rangle^X$ можна використовувати для реалізації моделі $\hat{Y}(\bar{X}|\mathfrak{G})$; називатимемо цю реалізацію графовою моделлю та позначаємо її як

$$\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma = \langle \Gamma^{|\mathfrak{G}}, \mathfrak{S}' \rangle^{\bar{X}} = \hat{Y}, \quad (3.60)$$

Ця реалізація подібна до представлення у вигляді набору підстанів (формула 2.57), за винятком того, що множину $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$ спочатку потрібно знайти як

$$\mathfrak{S}^{\bar{X}|\mathfrak{G}} = \mathfrak{S}^{\mathfrak{x}} \left(\gamma(\Gamma^{|\mathfrak{G}} | \mathfrak{S}') \right), \quad (3.61)$$

Наприклад, можемо змоделювати рух пружини Слінкі вниз по сходах зі змінною висотою (рис. 3.7), використовуючи графічну модель $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$. Вибираємо

$$\begin{aligned} \bar{X} &:= [\bar{n}] \\ \hat{Y} &:= [\hat{t}], \\ \mathfrak{G} &:= \emptyset, \end{aligned} \quad (3.62)$$

де n – кількість кроків, а t – час.

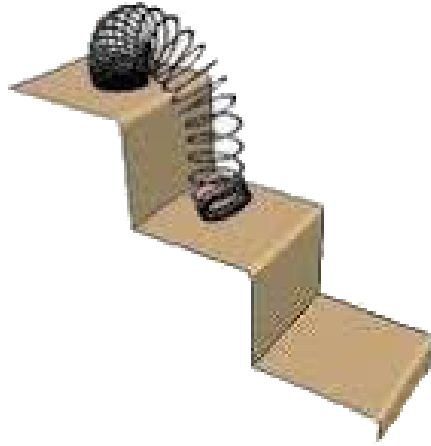


Рис. 3.7 Гра «Слінкі»

Припустимо, що у нас є 3 сходи. Перехід з 1-ї сходинки на 2-гу займає 5 секунд, а з 2-ї на 3-ю 7 секунд. Граф

$$\Gamma^{|\mathfrak{G}} := S_{d=0} \xrightarrow{\Theta^{|\mathfrak{D}}_{d=1}} S_{d=1} \xrightarrow{\Theta^{|\mathfrak{D}}_{d=2}} S_{d=2}, \quad (3.63)$$

де

$$\Theta^{|\mathfrak{D}}_{d=1} \left([\hat{t}]^{[\bar{n}]}_{q=1} \right) := [\hat{t} + 5]^{[\bar{n}+1]}_{q=1}, \quad (3.64)$$

і

$$\Theta^{|\mathfrak{D}}_{d=2} \left([\hat{t}]^{[\bar{n}]}_{q=1} \right) := [\hat{t} + 7]^{[\bar{n}+1]}_{q=1}. \quad (3.65)$$

Спочатку

$$\mathfrak{S}' := \left\{ S_{d=0} = [\hat{t} = 0]^{[\bar{n}=1]}_{q=1} \right\}, \quad (3.66)$$

у цьому випадку знайдений граф моделювання має вигляд

$$\begin{aligned} \gamma^{|\mathfrak{G}} = \gamma(\Gamma^{|\mathfrak{G}} | \mathfrak{S}') &= [\hat{t} = 0]^{[\bar{n}=1]}_{q=1, d=0} \xrightarrow{\Theta^{|\mathfrak{D}}_{d=1}} \\ &\xrightarrow{\Theta^{|\mathfrak{D}}_{d=1}} [\hat{t} = 5]^{[\bar{n}=2]}_{q=1, d=1} \xrightarrow{\Theta^{|\mathfrak{D}}_{d=2}} [\hat{t} = 12]^{[\bar{n}=3]}_{q=1, d=2}, \end{aligned} \quad (3.67)$$

набір підстанів є

$$\mathfrak{S}^{\bar{X}|\mathfrak{G}} = \mathfrak{S}^{\mathfrak{X}}(\gamma^{|\mathfrak{G}}) = \left\{ \begin{array}{l} [\hat{t} = 0]^{[\bar{n}=1]}_{q=1} \\ [\hat{t} = 5]^{[\bar{n}=2]}_{q=1} \\ [\hat{t} = 12]^{[\bar{n}=3]}_{q=1} \end{array} \right\}, \quad (3.68)$$

а множина зібраних значень $\hat{\mathfrak{Y}}$ є

$$\left[\begin{array}{l} \hat{\mathfrak{Y}}_1 = \cup \mathfrak{S}^{\bar{X}|\mathfrak{G}}(\bar{\mathfrak{X}}_1 = [\bar{n} = 1]) = [\hat{t} = 0] \\ \hat{\mathfrak{Y}}_2 = \cup \mathfrak{S}^{\bar{X}|\mathfrak{G}}(\bar{\mathfrak{X}}_2 = [\bar{n} = 2]) = [\hat{t} = 5] \\ \hat{\mathfrak{Y}}_3 = \cup \mathfrak{S}^{\bar{X}|\mathfrak{G}}(\bar{\mathfrak{X}}_3 = [\bar{n} = 3]) = [\hat{t} = 12] \end{array} \right], \quad (3.69)$$

3.6. Моделювання графа моделі $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$

Для графової моделі $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$ можемо визначити моделювання як оператор

$$\bar{\mathfrak{X}} \xrightarrow{\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma} \hat{\mathfrak{Y}}, \quad (3.70)$$

де $\bar{\mathfrak{X}} \subseteq \bar{\mathbb{X}}^n$ – підмножина відомих значень множини незалежних змінних $\bar{X} \subset V$, а

$\hat{\mathfrak{Y}} \subseteq \hat{\mathbb{Y}}^n$ – підмножина невідомих значень залежних змінних $\hat{Y} \subset V$.

Моделювання можна реалізувати як пошук графа моделювання $\gamma^{|\mathfrak{G}} \in \mathcal{Y}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$ для заданої початкової множини \mathfrak{S}' та множини \mathfrak{G} . Тоді, з множини $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$, $\hat{\mathfrak{Y}} \in \hat{\mathfrak{Y}}$ будується для кожного $\bar{\mathfrak{X}} \in \bar{\mathfrak{X}}$ як:

$$\bar{\mathfrak{X}} \xrightarrow{\forall \bar{\mathfrak{X}} \in \bar{\mathfrak{X}} (\hat{\mathfrak{Y}} \in \hat{\mathfrak{Y}} = \cup \mathfrak{S}^{\bar{X}|\mathfrak{G}}(\gamma^{|\mathfrak{G}}(\mathfrak{S}'))(\bar{\mathfrak{X}}))} \hat{\mathfrak{Y}}, \quad (3.71)$$

У задачі моделювання можемо значно оптимізувати пошук графа $\gamma^{|\mathfrak{G}}$. Оскільки множина $\bar{\mathfrak{X}}$ зазвичай значно менша за $\bar{\mathbb{X}}^n$, можемо шукати або обчислювати лише частину підстанів з $\mathfrak{S}^{\bar{X}|\mathfrak{G}}(\gamma^{|\mathfrak{G}})$, які містять усі необхідні ключі $\bar{\mathfrak{X}} \in \bar{\mathfrak{X}}$:

$$\mathfrak{S}^{\bar{\mathfrak{X}}} = \{\mathfrak{S}_q^{\bar{\mathfrak{X}}} \in \mathfrak{S}^{\bar{X}|\mathfrak{G}}(\gamma^{|\mathfrak{G}}) \mid \mathfrak{X}(\mathfrak{S}_q^{\bar{\mathfrak{X}}}) \in \bar{\mathfrak{X}}\}, \quad (3.72)$$

Також можемо оптимізувати \mathfrak{S}' , включивши підстани, які є максимально близькими (з точки зору відстані в графу $\Gamma^{|\mathfrak{G}}$) до підстанів від бажаного $\mathfrak{S}^{\bar{\mathfrak{X}}}$ або

навіть еквівалентними цим підстанам. Це зменшить кількість обчислень, не пов'язаних з пошуком $\mathfrak{S}^{\bar{x}}$ (рис. 3.8).

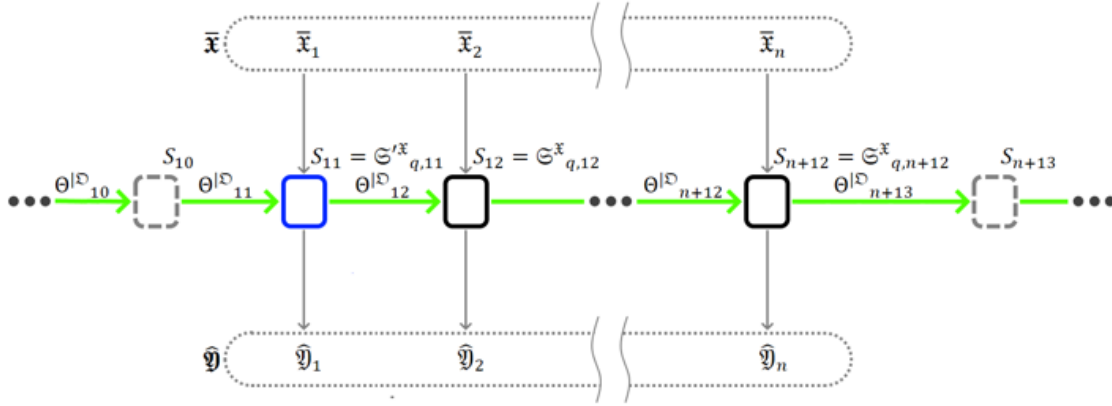


Рис. 3.8 Зменшення кількості обчислень шляхом включення підстанів, які максимально наближені до тих, що знаходяться в бажаному $\mathfrak{S}^{\bar{x}}$

Для графічної моделі $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$ також можна виконати інтерактивне моделювання. У найпростішому випадку це вимагає багатьох моделей $\{\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma_i\}$; однак, цікавішим та оптимальнішим підходом є інтерактивне маніпулювання значеннями вузлів $S \in S(\Gamma^{|\mathfrak{G}|})$ при використанні імперативних представлень (послідовне обчислення функцій $\Theta^{|\mathfrak{P}|}$) операції $\gamma(\Gamma^{|\mathfrak{G}|}|\mathfrak{S}')$. Цей підхід був коротко розглянутий у [42].

Тут можливі два шаблони:

1. Шаблон push. Цей шаблон може допомогти синхронізувати модель з деякими зовнішніми процесами (наприклад, для синхронізації з реальним часом). Суть шаблону полягає в тому, що деяку функцію $\Theta^{|\mathfrak{D}|}$ неможливо обчислити, доки не визначено всі її аргументи S ; таким чином, можемо локально призупинити моделювання, залишивши деякі кореневі вузли $S_{root} \in S(\Gamma^{|\mathfrak{G}|})$ неініціалізованими. Потім можемо продовжити її, визначивши ці вузли.

2. Шаблон pool. Цей шаблон можна використовувати для реалізації асинхронної реакції моделі на деякі зовнішні події, наприклад, для реагування на введення користувача. Як і в попередньому випадку, деякі $S_{root} \in S(\Gamma^{|\mathcal{G}|})$ залишаються неініціалізованими. Однак моделювання на цьому не зупиняється. Їхні значення будуються за потреби для обчислення наступного $\Theta^{|\mathcal{D}|}$.

Використовуючи цей підхід, образно можна уявити, що невизначений S_{root} обчислюється за допомогою деякого набору невідомих перехідних функцій, можливо, також об'єднаних у граф переходів. Іншими словами, існує деяка «тіньова» або «невідома» частина графа $\Gamma^{|\mathcal{G}|}$, і в результаті її обчислення S_{root} ініціалізується (рис. 3.9).

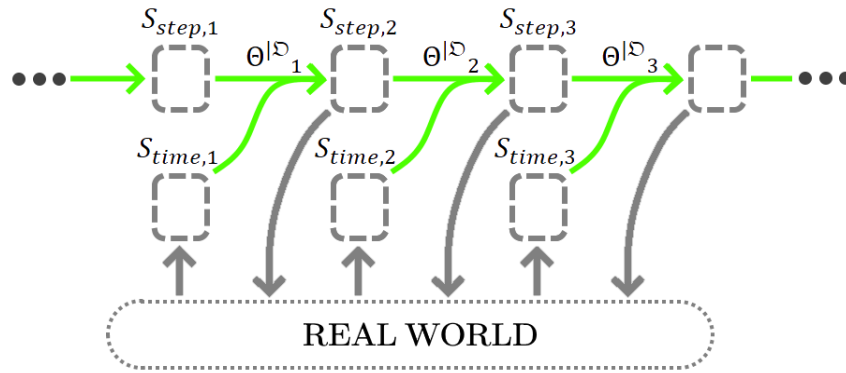


Рис. 3.9 Представлення входу/виходу як набору невідомих перехідних функцій

Наприклад, можемо зробити модель Слінкі (рис. 3.7) інтерактивною, що дозволить користувачеві змінювати висоту сходів у режимі реального часу, тобто змінювати час, протягом якого Слінкі перетинає сходи. Для цього визначимо функції як

$$\Theta^{|\mathcal{D}|}_d \left([\hat{t}]^{[\bar{n}]}_{q=1}, [\hat{h}]^{[\bar{n}]}_{q=2} \right) := [\hat{t} + \hat{h}]^{[\bar{n}+1]}_{q=1}, \quad (3.73)$$

Тоді графік переходу матиме вигляд

$$\Gamma^{|\mathbb{G}} := \left((S_{t,d=0}, S_{h,d=0}) \xrightarrow{\Theta^{|\mathbb{D}}_{d=1}} S_{t,d=1}, S_{h,d=1} \right) \xrightarrow{\Theta^{|\mathbb{D}}_{d=2}} S_{t,d=2}, \quad (3.74)$$

Використовуватимемо шаблон push таким чином, що моделювання зупинятиметься на кожній невизначеній змінній $S_{h,d}$, доки користувач не ініціалізує її значення:

$$\gamma^{|\mathbb{G}} := \left(([\hat{t} = 0]^{[\bar{n}=1]}_{q=1, d=0}, S_{h,d=0} = \emptyset) \xrightarrow{\Theta^{|\mathbb{D}}_{d=1}} S_{t,d=1} = \emptyset, S_{h,d=1} = \emptyset \right) \xrightarrow{\Theta^{|\mathbb{D}}_{d=2}} S_{t,d=2} = \emptyset \mid \bar{n} = 1; \quad (3.75)$$

$$\gamma^{|\mathbb{G}} := \left(([\hat{t} = 0]^{[\bar{n}=1]}_{q=1, d=0}, [\hat{h} = 5]^{[\bar{n}=1]}_{q=2, d=0}) \xrightarrow{\Theta^{|\mathbb{D}}_{d=1}} [\hat{t} = 5]^{[\bar{n}=2]}_{q=1, d=1}, S_{h,d=1} = \emptyset \right) \xrightarrow{\Theta^{|\mathbb{D}}_{d=2}} S_{t,d=2} = \emptyset \mid \bar{n} = 2; \quad (3.76)$$

$$\gamma^{|\mathbb{G}} := \left(([\hat{t} = 0]^{[\bar{n}=1]}_{q=1, d=0}, [\hat{h} = 5]^{[\bar{n}=1]}_{q=2, d=0}) \xrightarrow{\Theta^{|\mathbb{D}}_{d=1}} [\hat{t} = 5]^{[\bar{n}=2]}_{q=1, d=1}, [\hat{h} = 7]^{[\bar{n}=2]}_{q=2, d=1} \right) \xrightarrow{\Theta^{|\mathbb{D}}_{d=2}} [\hat{t} = 12]^{[\bar{n}=3]}_{q=1, d=2} \mid \bar{n} = 3. \quad (3.77)$$

ВИСНОВКИ ДО РОЗДІЛУ 3

Запропоновано подання динамічних моделей у вигляді графа переходів і розроблено підхід його перетворення у структуровану графову модель, що на відміну від використання імперативних описів (наборів інструкцій для виконання машинами) істотно спрощує процес створення та масштабування моделей у багатопроцесорних і розподілених середовищах.

Особливу увагу приділено узгодженості: неправильна композиція переходів може спричинити суперечності, тому визначено принципи побудови, що гарантують логічну зв'язаність, ациклічність та однозначність результатів. Це робить можливим створення складних моделей із багаторазових функціональних блоків, подібно до Simulink [50-52].

У цьому розділі також показано, що графи моделювання можна надійно згенерувати з початкового набору підстанів, що пов'язує абстрактні правила з конкретними завданнями. Представлення залежностей між змінними у формі графа забезпечує операційну реалізацію теорії, а приклад пружини Слінкі проілюстрував можливості аналізу динамічних процесів.

Запропоновано використання push- та pull-шаблонів для паралельного інтерактивного моделювання складних динамічних систем, що на відміну від класичних шаблонів, забезпечує ефективне використання обчислювальних ресурсів та менший час відгуку моделі, підчас синхронізації отриманих моделей з реальною системою по параметрах моделі.

РОЗІЛ 4

ПОБУДОВА ТА ВИКОРИСТАННЯ ОБЧИСЛЮВАЛЬНОГО ГРАФА НА ОСНОВІ ПАРАДИГМИ РЕАКТИВНИХ ПОТОКІВ

У цьому розділі представлено побудову та використання обчислювального графа на основі парадигми реактивних потоків. Окреслено перетворення графової моделі на мережу логічних процесорів і каналів, а також методи оптимізації результуючого графа для зменшення надлишковості та підвищення ефективності. В цьому розділі розглядається використання обчислювального графа для моделювання моделі, з висвітленням основних етапів виконання та міркувань, пов'язаних з обчислювальною складністю, а у розділі 5 реалізуємо простий обчислювальний граф та виконуємо моделювання з його допомогою.

4.1. Реактивні потоки та графова модель $\hat{Y}(\bar{X}|\mathfrak{G})^G$

Концепція реактивних потоків була сформульована у 2014 році в маніфесті [10-12] та розширена розробниками бібліотеки АККА інструментами для складання реактивних потоків у обчислювальні графи, [13, 48] які вже широко використовуються на практиці [9, 53-55]. Вузли графа – це логічні процесори, а ребра – канали, що представляють потік повідомлень, що передають дані; кожен із процесорів певним чином трансформує повідомлення. Загалом, реактивні потоки є реалізацією відомої парадигми програмування потоків даних [53].

У цьому розділі буде використано підхід, описаний у [45, 56-58]. (тобто складемо обчислювальну систему з невеликих блоків, які обробляють потоки даних). Однак використовуватимемо реактивні потоки для виконання всієї важкої роботи з розподілу обчислень та балансування навантаження.

Позначимо повідомлення (значення) через M , логічні процесори (реактори) через LP , канали, що з'єднують процесори, через D , а числовий граф – через C .

Можемо перетворити довільну графову модель $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$ (формула 3.60) на обчислювальний граф C :

- 1) Для представлення кожного підстану $\mathfrak{S}_q^x \in \mathfrak{S}^x(\Gamma|\mathfrak{G})$ повідомленням $M = [\mathfrak{S}_q^x]$.
- 2) Для заміни всіх $\Theta^{|\mathfrak{P}} \in \Theta(\Gamma|\mathfrak{G})$, для яких $S'_{k=1}, \dots, S'_{k=n} \in S(\mathfrak{S}')$, еквівалентними процесорами LP^{eval} :

$$S'_{k=1} \Rightarrow D_{k=1}, \dots, S'_{k=n} \Rightarrow D_{k=n} \xrightarrow{\Theta^{|\mathfrak{P}} \Rightarrow LP^{eval}} \hat{S} \Rightarrow D, \quad (4.1)$$

та всі $\Theta^{|\mathfrak{P}}$ (для яких $\hat{S} \in S(\mathfrak{S}')$) з процесорами LP^{eval} , еквівалентними оберненим функціям $\Theta^{-1|\mathfrak{P}}$:

$$\hat{S} \Rightarrow D \xrightarrow{\Theta^{|\mathfrak{P}} \Rightarrow \Theta^{-1|\mathfrak{P}} \Rightarrow LP^{eval}} S_{k=1} \Rightarrow D_{k=1}, \dots, S_{k=n} \Rightarrow D_{k=n}, \quad (4.2)$$

- 3) Для послідовної заміни всіх функцій $\Theta^{|\mathfrak{P}} \in \Theta(\Gamma|\mathfrak{G})$ та всіх аргументів $S_{k=1}, \dots, S_{k=n}$, які вже замінені каналами $D_{k=1}, \dots, D_{k=n}$, що еквівалентні LP^{eval} :

$$D_{k=1}, \dots, D_{k=n} \xrightarrow{\Theta^{|\mathfrak{P}} \Rightarrow LP^{eval}} \hat{S} \Rightarrow D, \quad (4.3)$$

І послідовно замінити всі $\Theta^{|\mathfrak{P}}$, результат \hat{S} яких вже був замінений каналом D , на LP^{eval} , що еквівалентно оберненим функціям $\Theta^{-1|\mathfrak{P}}$:

$$D \xrightarrow{\Theta^{|\mathfrak{P}} \Rightarrow \Theta^{-1|\mathfrak{P}} \Rightarrow LP^{eval}} S_{k=1} \Rightarrow D_{k=1}, \dots, S_{k=n} \Rightarrow D_{k=n}, \quad (4.4)$$

В результаті отримуємо граф C , що містить еквівалентну LP^{eval} для кожного $\Theta^{|\mathfrak{P}} \in \Theta(\Gamma|\mathfrak{G})$, але, можливо, з різними структурами порівняно з $\Gamma|\mathfrak{G}$, оскільки його побудова проводилася, починаючи з $S' \in S(\mathfrak{S}')$, а не з кореневих вузлів $S_{root} \in S(\Gamma|\mathfrak{G})$ (рис. 4.1).

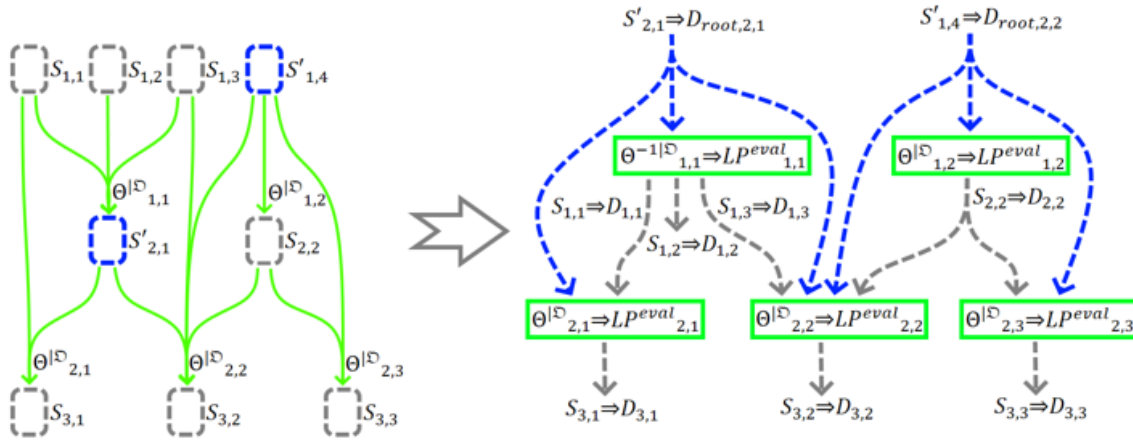


Рис. 4.1 Приклад побудови обчислювального графа C за моделлю $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$

Далі, кожен кореневий канал $D_{root} \in D(C)$ має бути підключений до логічного процесора LP^{init} , завданням якого є надсилання відповідного $M' = [\mathfrak{S}'^x_q]$ (де $\mathfrak{S}'^x_q \in \mathfrak{S}^x(\mathfrak{G}')$), що запускає обчислювальний процес (рис. 4.2).

Більше того, всі або частина каналів $D \in D(C)$ повинні бути з'єднані з одним або кількома $LP^{collect}$, які збиратимуть частину або всі обчислені підстани $\mathfrak{S}^x_q \in \mathfrak{S}^x(\gamma^{|\mathfrak{G}})$, що належать до графа $\gamma^{|\mathfrak{G}} \in \mathbf{\gamma}^{|\mathfrak{G}} \Leftrightarrow \Gamma^{|\mathfrak{G}}$, заданого множиною початкових станів \mathfrak{S}' (рис. 4.2).

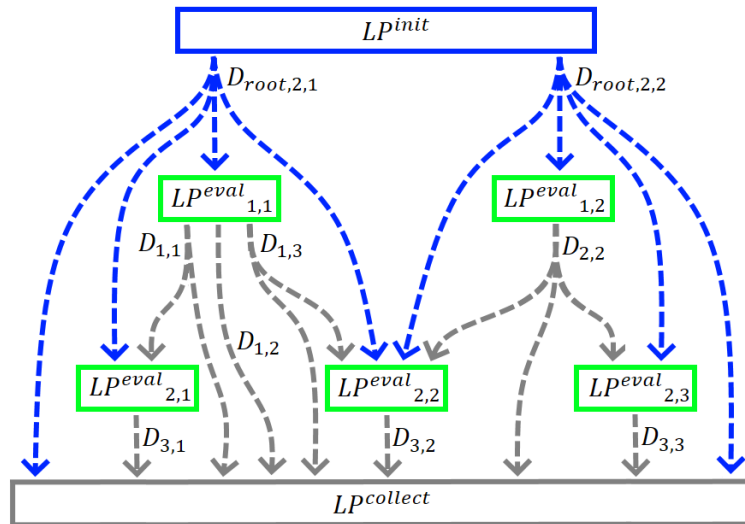


Рис. 4.2 Додавання LP^{init} та $LP^{collect}$ до обчислювального графа C

4.2. Оптимізація графа C

Проста заміна функцій $\Theta^{|\mathfrak{P}|}$ на процесори LP^{eval} призводить до надзвичайно неоптимального та потенційно нескінченного графа обробки C , що не є добре з точки зору мінімізації обчислювальних ресурсів. Щоб вирішити цю проблему, можемо оптимізувати граф C . Наприклад, розглянемо два методи оптимізації:

1. Згортання циклічних послідовностей у графі $\Gamma^{|\mathfrak{G}|}$:

Розглянемо ланцюг довільної довжини тих самих функцій $\theta^{|\mathfrak{P}|}$, як на рис. 4.3 (а). Це можна перетворити на ланцюг логічних процесорів LP^{eval} однакової довжини, як на рис. 4.3 (b). Можемо згорнути цей ланцюг в один LP^{eval} , додавши цикл повернення повідомлень, як на рис. 4.3 (c). Таким чином, через один LP^{eval} пройде більше одного повідомлення M , тому якщо $\theta^{|\mathfrak{P}|}$ має більше одного аргументу, це може призвести до колізій. Щоб вирішити колізії, а також реалізувати розрив циклу, нам потрібно визначити кількість ітерацій циклу повідомлень M . Простий спосіб зробити це - додати лічильник ітерацій для кожного циклу в C . Інший підхід полягає у використанні значень, чутливих до історії [44]. Як складніший приклад, розглянемо графік $\Gamma^{|\mathfrak{G}|}$ на рис. 4.4 (а), який можна перетворити та згорнути в компактний графік C , як на рис. 4.4 (b).

2. Згортання графа C :

Всередині кожного LP^{eval} можемо реалізувати більше однієї функції $\Theta^{|\mathfrak{P}|} \in \Theta(\Gamma^{|\mathfrak{G}|})$, таким чином зменшуючи кількість вузлів у графі C . Це згортання можна виконувати в широкому діапазоні, аж до реалізації всіх $\Gamma^{|\mathfrak{G}|}$ в одному LP^{eval} . Наприклад, граф C з рис. 4.4 (b) можна згорнути в один LP^{eval} і він виглядатиме як рис. 4.4 (c).

Загалом, задача оптимізації для графа C є досить складною та виходить за рамки цієї статті.

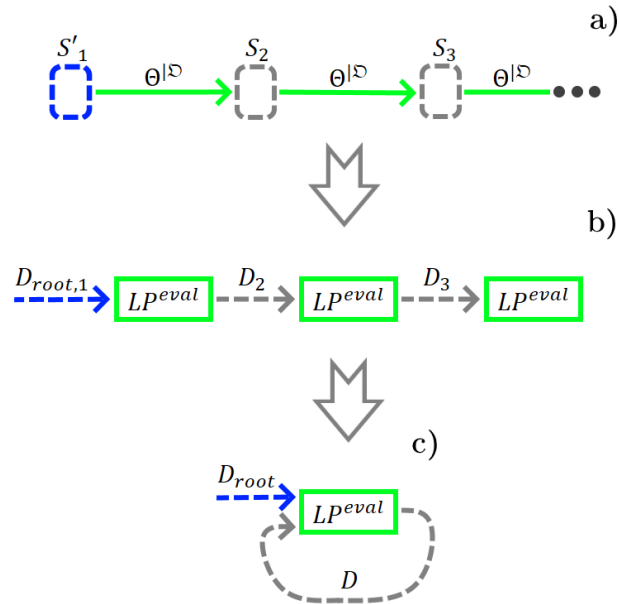


Рис. 4.3 Приклад згортання простого обчислювального графа C

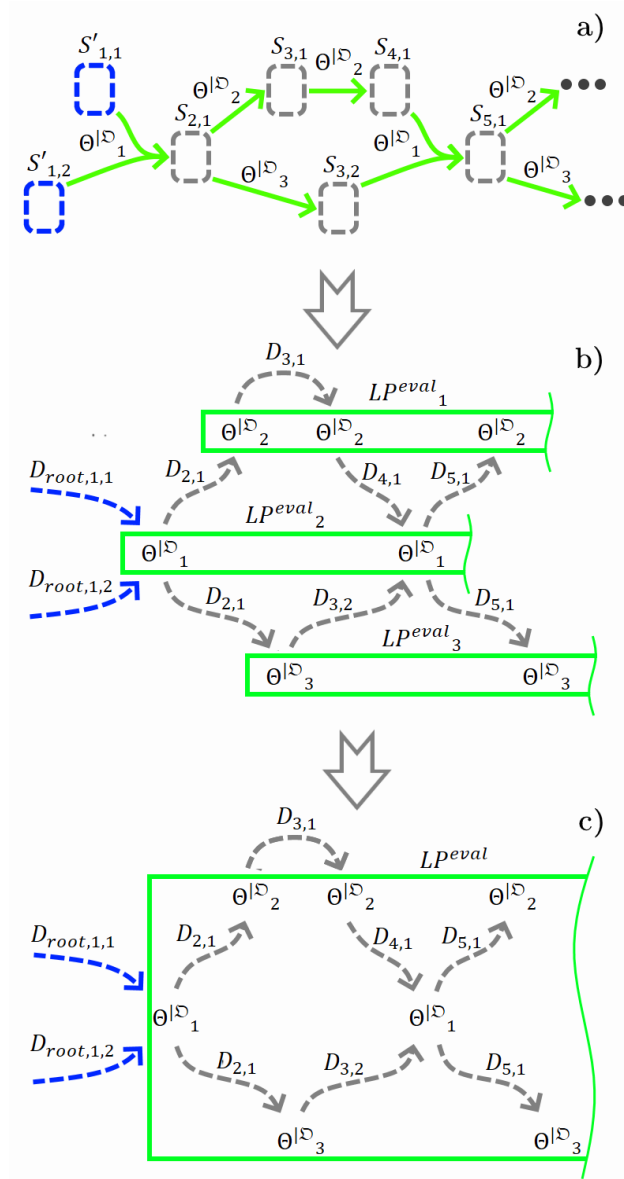


Рис. 4.4 Приклад згортання складнішого обчислювального графа C

4.3. Моделювання моделі $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$ за допомогою графа C

У найпростішому випадку можемо змодельовати модель $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$ (формула 3.70) за допомогою побудованого на ній графа C у два етапи:

1. Обчисліть набір підстанів

$$\mathfrak{S}^{\bar{X}|\mathfrak{G}} = \mathfrak{S}^{\mathfrak{X}} \left(\gamma(\Gamma^{|\mathfrak{G}}|\mathfrak{S}') \right), \quad (4.5)$$

Для цього ініціалізуємо обчислення, надсилаючи M' повідомлень за допомогою процесорів LP^{init} . Використовуючи процесори $LP^{collect}$, збираємо всі обчислені повідомлення, $M = [\mathfrak{S}_q^x]$.

2. Знайти всі підстани для кожного ключа $\bar{x} \in \bar{\mathfrak{X}}$, а потім зібрати значення $\hat{\mathfrak{y}} \in \hat{\mathfrak{Y}}$ зі знайдених підстанів (формула 2.65).

У більшості випадків цей підхід буде обчислювально ресурсоємним, оскільки на практиці зазвичай $|\mathfrak{X}(\gamma^{|\mathfrak{G}})| > |\bar{\mathfrak{X}}|$.

Загалом, оптимізація моделювання – це мінімізація кількості обчислених підстанів \mathfrak{S}_q^x таких, що $\mathfrak{X}(\mathfrak{S}_q^x) \notin \bar{\mathfrak{X}}$. Тут можливі кілька підходів, наприклад, побудова мінімалістичного $\Gamma^{|\mathfrak{G}}|$ з певною відомою колекцією $\bar{\mathfrak{X}}$. Як альтернатива, можна використовувати лінійні алгоритми, які відсікають обчислення $\Theta^{|\mathfrak{P}}|$, результат яких не обов'язково повинен охоплювати $\bar{\mathfrak{X}}$. Однак ця тема виходить за рамки цієї статті.

4.4. Проблеми і перспективні шляхи подальшого розвитку запропонованого методу

Хоча запропонований підхід пропонує багатообіцяючий напрямок, залишається кілька проблем:

1. Оптимізація графа. Подібно до проблем, що спостерігалися в попередніх системах паралельного моделювання [3, 4], початкова побудова обчислювального графа \mathcal{C} може бути неоптимальною. Методи згортання циклічних послідовностей та зменшення надлишкових обчислень є важливими, але необхідні подальші дослідження для автоматизації та оптимізації цього процесу без шкоди для точності.
2. Дискретизація неперервних моделей. Хоча наша структура ефективно обробляє дискретні події та переходи станів, багато фізичних явищ є за

своєю суттю неперервними. Ця проблема не є унікальною та була відзначена в дослідженнях, що стосуються обмежень моделювання дискретних подій у неперервних системах [8, 27]. У майбутніх роботах слід дослідити методи підвищення точності моделювання з неперервними значеннями без надмірних обчислювальних витрат.

3. Відмовостійкість та надійність. Як підкреслюється в дослідженнях відмовостійкого розподіленого реактивного програмування [42, 44], забезпечення надійності в умовах збоїв мережі або вузлів є критично важливим питанням. Хоча наша стаття торкається теми відмовостійкості, комплексні стратегії підвищення надійності обчислювального графа потребують подальшого дослідження.
4. Автоматизація побудови графів. Сучасні методи побудови графа переходів Γ^{G} є ручними або напівавтоматизованими. Розробка інструментів або предметно-орієнтованих мов для полегшення як ручної, так і автоматичної побудови цих графів, як обговорюється в роботах, таких як робота про мову EdgeC [59, 60], підвищила б зручність використання та сприяла б ширшому впровадженню.

Перспективні шляхи подальшого розвитку та вдосконалення запропонованого методу розпаралелювання моделювання з використанням реактивних потоків:

1. Ефективна оптимізація обчислювального графа C та моделювання на ньому. Цю тему розглянуто в розділах 5.2 та 5.3. Однак, через її складність та обсяг, вона не вписується в цю статтю. Загалом, це дуже важливе питання з практичної точки зору. Його вирішення значно зменшить кількість ресурсів, необхідних для виконання моделювання. Ще одне цікаве питання – автоматизація оптимізації графа C . Припустимо, що спочатку у нас є неоптимальний C , наприклад, отриманий методом, описаним у розділі 4.1. Щоб автоматично зробити

С компактним та обчислювально простим, без втрати точності та узгодженості. Для вирішення завдання оптимізації можна використовувати методи машинного навчання. Наприклад, агенти переобладнання підкріплення можуть бути навчені досліджувати різні конфігурації графа (тобто різні способи згортання або згортання обчислювального графа) та вивчати, які конфігурації дають найкращу продуктивність з точки зору затримки, пропускну здатності або споживання ресурсів [61-65]. Також такі методи, як пошук нейронної архітектури (NAS), можуть бути адаптовані для оптимізації макета та параметрів обчислювального графа. Це включає автоматичне вирішення питань згортання циклічних послідовностей, балансування навантаження між логічними процесорами та мінімізацію надлишкових обчислень [64-66].

2. Точне моделювання моделей з неперервними значеннями. Багато властивостей модельованих об'єктів можна представити у вигляді неперервних величин, наприклад, значення з множини \mathbb{R} . Однак моделювання (обчислення якого базується на пересиланні повідомлень) є за своєю суттю дискретним. Відкритим залишається питання щодо того, наскільки точно можна обчислювати неперервні величини. Питання полягає в тому, як підвищити точність обчислення таких величин без збільшення вимог до комп'ютерних ресурсів.
3. Відмовостійкість реактивних потоків. У цій роботі не торкалися відмовостійкості моделювання, але в більшості реальних/практичних застосувань відмовостійкість є дуже важливою. Це питання було частково досліджено в [42], але також пропонуємо це для майбутньої роботи.
4. Ручне та автоматичне конструювання графів С. З практичної точки зору, цікаво мати можливість використовувати певне IDE для ручного

конструювання обчислювального графа C , і робити це таким чином, щоб відповідний граф $\Gamma^*(G)$ був узгодженим та оптимальним. Наприклад, це можна зробити аналогічно пакету Simulink, інструменту SwiftVis [59, 67], [53, 68] або мові XFRP [49, 69]. Також цікаво знайти способи автоматизації конструювання C . Наприклад, модель спочатку можна визначити як певний набір правил, за якими граф C можна автоматично і навіть динамічно будувати. Спеціалізовані мови програмування також є цікавою областю для дослідження. Наприклад, мову EdgeC [59, 60] можна розглядати як інструмент для опису обчислювальних графів.

5. Також тут можна широко застосовувати методи машинного навчання. Наприклад, методи навчання графів з графових нейронних мереж (GNN) можна застосовувати для вивчення структури оптимального обчислювального графа з історичних даних. Навчена модель потім може запропонувати або автоматично побудувати більш ефективний граф на основі поточних вимог моделювання. Адаптивні алгоритми машинного навчання з плануванням можуть динамічно налаштовувати планування завдань на логічних процесорах, оптимізуючи порядок виконання та балансуєчи навантаження [70-73]. Це особливо корисно в інтерактивних або реальних симуляціях, де умови можуть часто змінюватися.
6. Тестування зі складними моделями та порівняння з іншими підходами до паралелізації. У цій роботі наведено невеликий, простий приклад паралельного моделювання, щоб показати, як описаний підхід можна реалізувати на практиці. Однак питання перевірки цього підходу з великими та складними моделями та порівняння його ефективності з іншими підходами до паралелізації залишаються відкритими.

ВИСНОВКИ ДО РОЗДІЛУ 4

У цьому розділі розроблено метод перетворення графів переходів на обчислювальні графи, що реалізуються у парадигмі реактивних потоків (зокрема, з використанням бібліотеки AKKA Streams). Такий метод, на відміну від традиційних (побудованих на основі потоків інструкцій), забезпечує більш природне поєднання абстрактної специфікації з виконуваними обчисленнями, а також спрощує процес масштабованої реалізації моделей у багатопроцесорних та розподілених середовищах.

Особливу увагу було приділено методам оптимізації, оскільки пряме відображення всіх функцій на процесори може призвести до неефективних або навіть нескінченних обчислювальних структур. Такі підходи, як згортання циклічних послідовностей та консолідація кількох функцій в межах одного процесора, ілюструють, як можна спростити структуру графа, тим самим зменшуючи кількість вузлів обробки та покращуючи використання ресурсів. Ці методи підкреслюють важливість балансування формальної коректності з обчислювальною ефективністю.

Нарешті, було розглянуто принципи моделювання моделі за допомогою побудованого обчислювального графа. У розділі було наголошено як на простій двоетапній процедурі моделювання, так і на проблемах, що виникають при роботі з великою кількістю підстанів. Хоча детальні стратегії мінімізації непотрібних обчислень залишилися поза межами цієї роботи, потенційні напрямки, такі як ліниве обчислення та вибіркове конструювання графів, були визначені як перспективні шляхи подальшого розвитку.

РОЗІЛ 5

ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ МЕТОДУ ПОБУДОВИ ПАРАЛЕЛЬНИХ ЧИСЕЛЬНИХ МОДЕЛЕЙ ДИНАМІЧНИХ СИСТЕМ НА БАЗІ ПРОТОКОЛУ РЕАКТИВНИХ ПОТОКІВ

Експериментальні дослідження, представлені в цьому розділі, підтвердили практичну можливість побудови паралельних числових моделей динамічних систем на основі протоколу реактивних потоків. Використовуючи класичну задачу змішування як приклад, експерименти продемонстрували, як теоретичні конструкції, такі як розкладання станів, графи переходів та представлення підстанів, можуть систематично перетворюватися на функціональні реалізації.

Проведені моделювання показали, що різні представлення моделі, чи то у формі прямих наборів функцій, колекцій підстанів чи графів переходів, призводять до узгоджених результатів. Ця узгодженість підкреслює стійкість базового методу та його незалежність від конкретної форми реалізації. Крім того, впровадження інтерактивних моделей підкреслило гнучкість підходу, що дозволяє динамічно змінювати параметри системи в режимі реального часу без порушення цілісності процесу моделювання.

Значним внеском експериментальних досліджень стала реалізація методу в сучасних середовищах обробки потоків, зокрема Akka Streams. Ця реалізація продемонструвала, що синхронізація на основі реактивних потоків дозволяє створювати ефективні, масштабовані та оптимізовані графи моделювання. Порівняння між наївними та оптимізованими структурами графів додатково проілюструвало потенціал для покращення продуктивності шляхом структурного вдосконалення та повторного використання логічних процесорів.

Загалом, результати цього розділу обґрунтовують застосовність запропонованого методу як в академічному, так і в практичному контекстах. Вони ілюструють, як абстрактні математичні формулювання можуть бути вбудовані в програмовані середовища та ефективно виконані в різних парадигмах моделювання. Ця експериментальна основа служить містом до подальших розробок, де на основі представленого підходу можна досліджувати складніші системи та масштабніші паралельні обчислення.

5.1 Опис змодельованого об'єкта та побудова моделі $\hat{Y}(\bar{X}|\mathfrak{G})$

Як приклад, розглянемо класичну модель змішування сольових розчинів (рис. 5.1). Тут об'єкт моделювання являє собою систему з двох з'єднаних резервуарів об'ємом $v_1 = 4L$ та $v_2 = 8L$. Протягом часу t сольовий розчин циркулює з першого резервуара до другого зі швидкістю $q_3 = 5L/m$ та у зворотному напрямку зі швидкістю $q_2 = 2L/m$. Крім того, сольовий розчин заливається в перший резервуар зі швидкістю $q_1 = 3L/m$ та зливається з другого резервуара з тією ж швидкістю $q_4 = 3L/m$, тобто об'єм сольового розчину в резервуарах не змінюється. Спочатку перший та другий резервуари повністю заповнені розчинами з початковими концентраціями солей $\omega_1 = 0g/L$ та $\omega_2 = 20g/L$ відповідно. Сольовий розчин з концентрацією $\omega_3 = 10g/L$ постійно подається в перший резервуар. Таким чином, набір змінних, що відображають властивості, що цікавлять, виглядатиме так:

$$V = \begin{bmatrix} t \\ \omega_1 \\ \omega_2 \\ \omega_3 \\ v_1 \\ v_2 \\ q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}. \quad (5.1)$$

Завдання моделювання полягає в прогнозуванні зміни концентрацій солей $\omega_1 = 0 \text{ g/L}$ та $\omega_2 = 20 \text{ g/L}$ з часом t .

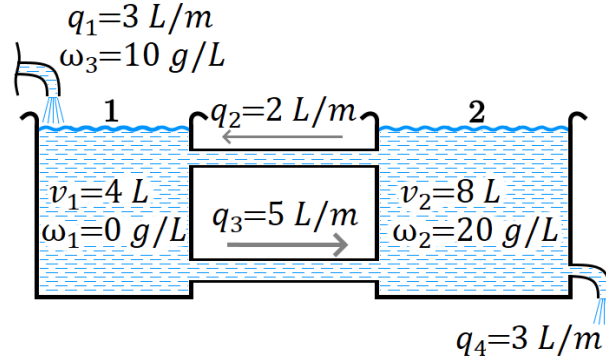


Рис. 5.1 Модель змішування сольового розчину

Як частину задачі моделювання, яку потрібно розв'язати, представляємо модельований об'єкт у вигляді моделі $\hat{Y}(\bar{X}|\mathfrak{G})^F$ (формула 2.56), розбиваючи змінні V на

$$\begin{aligned}\bar{X} &= [t]; \\ \hat{Y} &= \begin{bmatrix} \omega_1 \\ \omega_2 \end{bmatrix}; \\ \mathfrak{G} &= \begin{bmatrix} v_1 = 4 \\ v_2 = 8 \\ q_1 = 3 \\ q_2 = 2 \\ q_3 = 5 \\ q_4 = 3 \\ \omega_3 = 10 \end{bmatrix},\end{aligned}\tag{5.2}$$

та визначення їхньої залежності як набору функцій (5.3):

$$\begin{aligned}\hat{Y} &= F(\bar{X}|\mathfrak{G}) = \\ \begin{bmatrix} \hat{\omega}_1(t) = \frac{13e^{\frac{(\sqrt{105}-15)t}{16}}\sqrt{105}}{21} - \frac{13e^{\frac{(15+\sqrt{105})t}{16}}\sqrt{105}}{21} - 5e^{\frac{(\sqrt{105}-15)t}{16}} - 5e^{\frac{-(15+\sqrt{105})t}{16}} + 10 \\ \hat{\omega}_2(t) = \frac{5e^{\frac{-(15+\sqrt{105})t}{16}}\sqrt{105}}{21} - \frac{5e^{\frac{(\sqrt{105}-15)t}{16}}\sqrt{105}}{21} + 5e^{\frac{(\sqrt{105}-15)t}{16}} + 5e^{\frac{-(15+\sqrt{105})t}{16}} + 10 \end{bmatrix}.\end{aligned}\tag{5.3}$$

Які отримуються шляхом розв'язання задачі Коші

$$\begin{cases} \frac{d\hat{\omega}_1}{d\bar{t}} = \frac{3*10+2*\hat{\omega}_2-5*\hat{\omega}_1}{4} \\ \hat{\omega}_1(0) = 0 \\ \frac{d\hat{\omega}_2}{d\bar{t}} = \frac{5*\hat{\omega}_1-2*\hat{\omega}_2-3*\hat{\omega}_2}{8} \\ \hat{\omega}_2(0) = 20 \end{cases} \quad (5.4)$$

Також можемо представити змодельований об'єкт у вигляді моделі $\hat{Y}(\bar{X}|\mathfrak{G})^S$ (формула 2.57). У цьому випадку значення змінної t будуть використовуватися як ключі, а значення змінних ω_1 та ω_2 можуть бути розділені різними підстанами, таким чином отримаємо два типи підстанів $\mathfrak{S}^{\mathfrak{x}}_{q=1} = [\omega_1]^{[t]}_{q=1}$ та $\mathfrak{S}^{\mathfrak{x}}_{q=2} = [\omega_2]^{[t]}_{q=2}$. У коді можемо представити значення $\bar{\mathfrak{X}}$, $\hat{\mathfrak{Y}}$ та підстан $\mathfrak{S}^{\mathfrak{x}}_q$ як класи ООП (псевдокод 5.1 (L27)).

Один простий, але непрактичний спосіб побудови набору підстанів $\mathfrak{S}^{\bar{X}|\mathfrak{G}}$ полягає в генерації $\mathfrak{S}^{\mathfrak{x}}_{q=1}, \mathfrak{S}^{\mathfrak{x}}_{q=2} \in \mathfrak{S}^{\bar{X}|\mathfrak{G}}$ за допомогою набору функцій (формула 5.3) з деяким кроком ключа Δt (псевдокод 5.2 (L60)).

Використовуючи модель $\hat{Y}(\bar{X}|\mathfrak{G})$, можемо виконати моделювання (формула 2.61) для деякого відрізка $\bar{\mathfrak{X}} = [t_{begin}, t_{end}]$ та отримати відповідний набір значень $\hat{\mathfrak{Y}}$ (псевдокод 5.1 (L71)) є реалізацією $\hat{Y} = F(\bar{X}|\mathfrak{G})$, а псевдокод 5.2 (L70) є реалізацією $\mathfrak{S}^{\bar{X}|\mathfrak{G}} = \hat{Y}$.

Використовуючи реалізацію коду (псевдокод 5.1 та 5.2), моделюємо об'єкт. Дивлячись на вихідні графіки, бачимо, що вони схожі (рис. 5.2).

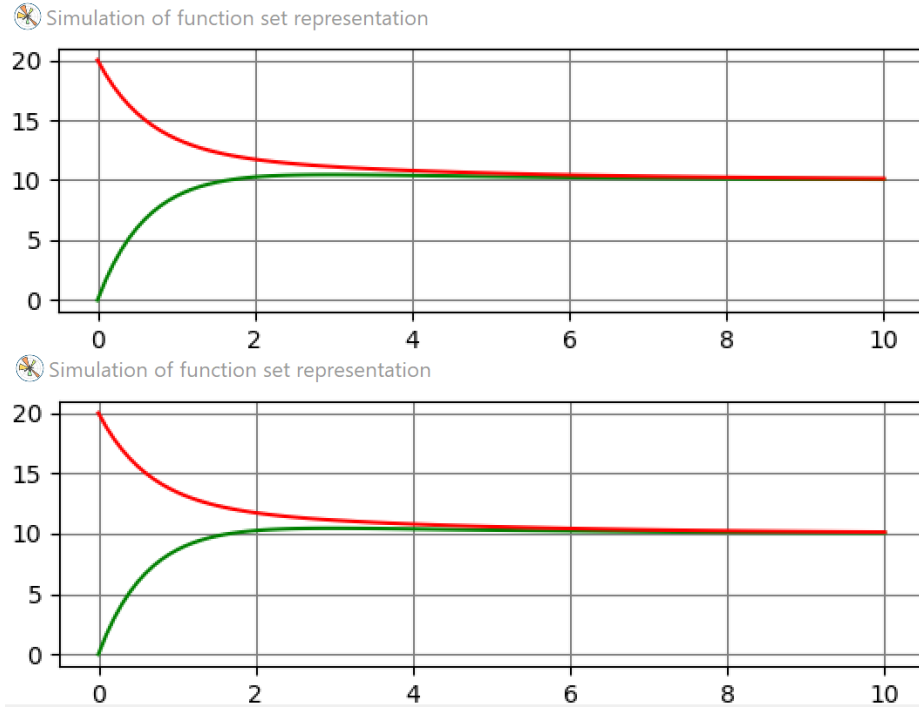


Рис. 5.2 Результати моделювання моделей $\hat{Y}(\bar{X}|\mathfrak{G})^F$ та $\hat{Y}(\bar{X}|\mathfrak{G})^S$.

Можемо порівняти результати виконання моделей $\hat{Y}(\bar{X}|\mathfrak{G})^F$ та $\hat{Y}(\bar{X}|\mathfrak{G})^S$, просто накопичуючи різні загальні вихідні значення:

$$\varepsilon = \sum_{\bar{\mathfrak{x}} \in \overline{\mathfrak{X}}^n} \sum_{i=1}^{|\mathcal{Y}|} (\hat{Y}(\bar{X}|\mathfrak{G})^F_i - \hat{Y}(\bar{X}|\mathfrak{G})^S_i). \quad (5.5)$$

Оцінюючи цей алгоритм (псевдокод 5.3 (L24)), отримуємо $\varepsilon = 1.1546319456101628e^{-14}$.

Можемо зробити моделювання інтерактивним, наприклад, дозволивши модифікацію параметра $\omega_3 \in G$ (тобто концентрації солі, насипаної в перший резервуар) у режимі реального часу.

Як описано в розділі 3, можемо представити інтерактивне моделювання як послідовність неінтерактивних моделювань (формула 2.73), де кожен набір $\bar{\mathfrak{x}}_i$ з послідовності $\bar{\mathfrak{x}}_0, \dots, \bar{\mathfrak{x}}_n \in$ частиною відрізка $[t_{begin}, t_{end}]$ та відповідає періоду, протягом якого параметри \mathfrak{G}_i не змінюються:

$$\cup_{i=0}^n t(\bar{\mathfrak{x}}_i) = [t_{begin}, t_{end}]. \quad (5.6)$$

Кожен \mathfrak{G}_i матиме свою власну модель $\hat{Y}(\bar{X}|\mathfrak{G})_i$, і моделі $\hat{Y}(\bar{X}|\mathfrak{G})_i$ та $\hat{Y}(\bar{X}|\mathfrak{G})_{i+1}$ повинні перетинатися в точці (\bar{x}, \hat{y}) , що відповідає моменту зміни \mathfrak{G}_i на \mathfrak{G}_{i+1} (рис. 5.3).

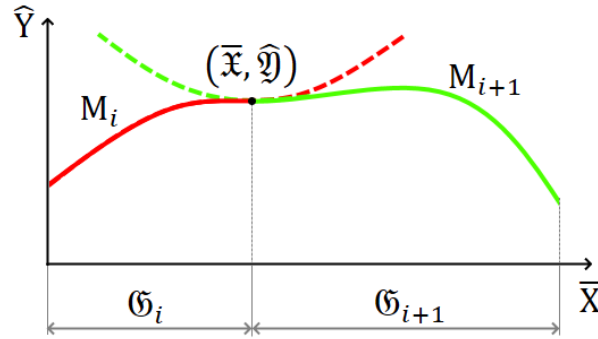


Рис. 5.3 Перехоплення моделей $M_i := \hat{Y}(\bar{X}|\mathfrak{G})_i$ та $M_{i+1} := \hat{Y}(\bar{X}|\mathfrak{G})_{i+1}$

В результаті набору моделей отримуємо набір значень залежних параметрів $\hat{Y}_0, \dots, \hat{Y}_n$, які є частинами відрізків $[\omega_{1_{begin}}, \omega_{1_{end}}]$ та $[\omega_{2_{begin}}, \omega_{2_{end}}]$:

$$\cup_{i=0}^n \omega_1(\hat{\mathfrak{Y}}_i) = [\omega_{1_{begin}}, \omega_{1_{end}}], \cup_{i=0}^n \omega_2(\hat{\mathfrak{Y}}_i) = [\omega_{2_{begin}}, \omega_{2_{end}}]. \quad (5.7)$$

У кодї інтерактивне моделювання може бути реалізовано як цикл, що послідовно ітерується через значення t_{real} у реальному часі, перевіряючи, чи змінилися значення параметрів \mathfrak{G} , а потім відображаючи результуюче $\hat{\mathfrak{Y}}_i$ моделювання в реальному часі (псевдокод 5.3).

Псевдокод 5.1 (function_set_representation.py):

```
# Definitions
```

```
class x_:
    def __init__(self, t):
        self.t = t
```

```

class  $\mathfrak{Y}$ _:
    def __init__(self,  $\omega_1$ ,  $\omega_2$ ):
        self. $\omega_1$  =  $\omega_1$ 
        self. $\omega_2$  =  $\omega_2$ 

class  $\mathfrak{E}$ _:
    def __init__(self,  $v_1$ ,  $v_2$ ,  $q_1$ ,  $q_2$ ,  $q_3$ ,  $q_4$ ,  $\omega_3$ ):
        self. $v_1$  =  $v_1$ 
        self. $v_2$  =  $v_2$ 
        self. $q_1$  =  $q_1$ 
        self. $q_2$  =  $q_2$ 
        self. $q_3$  =  $q_3$ 
        self. $q_4$  =  $q_4$ 
        self. $\omega_3$  =  $\omega_3$ 

# Parameters

 $\mathfrak{E}$  =  $\mathfrak{E}$ _(
     $v_1$  = 4, # L
     $v_2$  = 8, # L
     $q_1$  = 3, # L/m
     $q_2$  = 2, # L/m
     $q_3$  = 5, # L/m
     $q_4$  = 3, # L/m
     $\omega_3$  = 10) # g/l

```



```

# Model

def F(X,  $\mathcal{E}$ ):
    q = m.sqrt(105.0)
    em = m.exp(((q - 15.0) * X.t) / 16.0)
    ep = m.exp(-(((q + 15.0) * X.t) / 16.0))
    return  $\mathcal{Y}$ _(
         $\omega_1$  = ((13.0 * em * q) / 21.0) - ((13.0 * ep * q) /
21.0) - (5.0 * em) - (5.0 * ep) + 10.0,
         $\omega_2$  = -((5.0 * em * q) / 21.0) + ((5.0 * ep * q) /
21.0) + (5.0 * em) + (5.0 * ep) + 10.0)

# Simulations

def simulation(setX,  $\mathcal{E}$ ):
    set $\mathcal{Y}$  = np.array([])
    for x in setX:
         $\mathcal{Y}$  = F(x,  $\mathcal{E}$ )
        set $\mathcal{Y}$  = np.append(set $\mathcal{Y}$ , [ $\mathcal{Y}$ ])
    return set $\mathcal{Y}$ 

# Run simulation

setx = np.vectorize(lambda t: x_(t))(np.arange(0.0, 10.1,
0.1))
set $\mathcal{Y}$  = simulation(setx,  $\mathcal{E}$ )

```

Definitions

```
class GX_q_:
    def __init__(self, x, y, q):
        self.t = x.t
        if q == 1:
            self.w_1 = y.w_1
        elif q == 2:
            self.w_2 = y.w_2
```

Generating of set of sub-states

```
setGXG = []
for x in [x_(t) for t in np.round(np.arange(-15.0, 15.0,
0.1), 4)]:
    y = F(x)
    setGXG.append(GX_q_(x, y, q=1))
    setGXG.append(GX_q_(x, y, q=2))
```

Simulation function

```
def simulation(setx):
    sety = []
    for x in setx:
        GX_1 = None
        GX_2 = None
        for GX_q in setGXG:
            if GX_q.t == x.t and GX_q.q == 1: GX_1 = GX_q
```

```

        if  $\mathcal{G}\mathcal{X}_q.t == \mathcal{X}.t$  and  $\mathcal{G}\mathcal{X}_q.q == 2$ :  $\mathcal{G}\mathcal{X}_2 = \mathcal{G}\mathcal{X}_q$ 
        set $\mathcal{Y}$ .append( $\mathcal{Y}_-(\mathcal{G}\mathcal{X}_1.\omega_1, \mathcal{G}\mathcal{X}_2.\omega_2)$ )
    return set $\mathcal{Y}$ 

# Run simulation
set $\mathcal{X}$  = np.vectorize(lambda t:
 $\mathcal{X}_-(t)$ )(np.round(np.arange(0.0, 10.1, 0.1), 4))
set $\mathcal{Y}$  = simulation(set $\mathcal{X}$ )

```

Псевдокод 5.3 (function_set_interactive_simulation.py):

```

# Parameters

 $\Delta t = .1$ 
 $\mathcal{X}_0 = \mathcal{X}_-(t=.0)$ 
 $\mathcal{Y}_0 = \mathcal{Y}_-(\omega_1=.0, \omega_2=20.0)$ 
 $\mathcal{G}_0 = \mathcal{G}_-($ 
     $v_1 = 4, \quad \# L$ 
     $v_2 = 8, \quad \# L$ 
     $q_1 = 3, \quad \# L/m$ 
     $q_2 = 2, \quad \# L/m$ 
     $q_3 = 5, \quad \# L/m$ 
     $q_4 = 3, \quad \# L/m$ 
     $\omega_3 = 10) \# g/l$ 
up_down_step = 1

# Model (Earlier)

```

```

def F( $\Delta t$ ,  $\mathfrak{X}_0$ ,  $\mathfrak{Y}_0$ ):
    def eval(X,  $\mathfrak{E}$ ):
        t =  $\mathfrak{X}_0.t$ 
         $\omega_1$  =  $\mathfrak{Y}_0.\omega_1$ 
         $\omega_2$  =  $\mathfrak{Y}_0.\omega_2$ 
        while t <= X.t:
             $\omega_{1\_m1}$  =  $\omega_1$ 
             $\omega_{2\_m1}$  =  $\omega_2$ 
             $\omega_1$  =  $\omega_{1\_m1}$  +  $\Delta t$  * ((( $\mathfrak{E}.q_1$  *  $\mathfrak{E}.\omega_3$ ) + ( $\mathfrak{E}.q_2$ 
*  $\omega_{2\_m1}$ ) - ( $\mathfrak{E}.q_3$  *  $\omega_{1\_m1}$ )) /  $\mathfrak{E}.v_1$ )
             $\omega_2$  =  $\omega_2$  +  $\Delta t$  * ((( $\mathfrak{E}.q_3$  *  $\omega_{1\_m1}$ ) - ( $\mathfrak{E}.q_2$  *
 $\omega_{2\_m1}$ ) - ( $\mathfrak{E}.q_4$  *  $\omega_{2\_m1}$ )) /  $\mathfrak{E}.v_2$ )
            t +=  $\Delta t$ 
        return  $\mathfrak{Y}_0(\omega_1, \omega_2)$ 
    return eval

# Simulations

def simulation(M, setX,  $\mathfrak{E}$ ):
    set $\mathfrak{Y}$  = []
    for  $\mathfrak{X}$  in setX:
         $\mathfrak{Y}$  = M( $\mathfrak{X}$ ,  $\mathfrak{E}$ )
        set $\mathfrak{Y}$ .append( $\mathfrak{Y}$ )
    return set $\mathfrak{Y}$ 

# Interactive simulation

X =  $\mathfrak{X}_0$ 

```

```

Y = y_0
M = H.get_model(X, Y)
while H.not_terminated():
    t_real = H.get_next_real_time().t
    G = H.get_parameters()
    if G != G:
        M = H.get_model(X, Y)
        G = G
    X = x_(t_real)
    Y = simulation(M, [X], G)[0]
    H.show(X, Y)

```

5.2 Побудова та моделювання графової моделі $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$

Як приклад, побудуємо модель $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$ задачі змішування сольового розчину, як описано в розділі 5.1. Потім змоделюємо її за формулою 3.70. Для цього побудуємо граф переходів $\Gamma^{|\mathfrak{G}|}$ та визначити початковий набір підстанів \mathfrak{S}' .

Графік $\Gamma^{|\mathfrak{G}|}$ для цього прикладу представлятиме нескінченний ланцюг пар вузлів S , з'єднаних ребрами $\Theta^{|\mathfrak{D}|}$. Для зручності, окрім індексу глибини d , проіндексуємо вузли S індексами ширини $w \in N$, так що вузли S_d , з однаковим індексом d , матимуть різні значення w . Крім того, встановимо $w = k = q$, де k - індекс аргументу (ребер) $\Theta^{|\mathfrak{D}|}_d$, а q - індекс підстану, призначеного $S_{d,w}$. Кожна пара $S_{d,w=1}$ та $S_{d,w=2}$ відповідає певному моменту дискретного часу \bar{t} . Для простоти використовуватимемо фіксований крок за часом $\Delta t = d * \gamma$, де d - індекс глибини, а γ - коефіцієнт кроку за часом. Також обмежуємо час моделі малим інтервалом $[t_{begin}, t_{end}] \supset \bar{t}$. У цьому випадку граф $\Gamma^{|\mathfrak{G}|}$ міститиме

$$n = \frac{t_{end} - t_{begin}}{\Delta t} + 1, \quad (5.8)$$

пари вузлів $S_{d,w}$ (рис. 5.4).

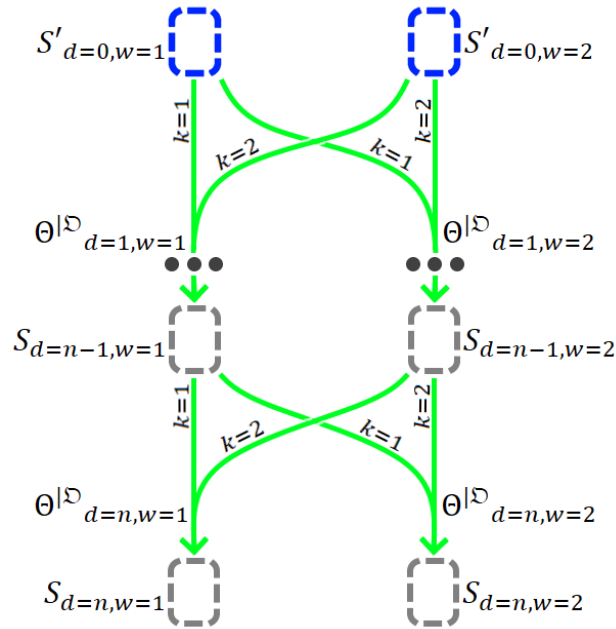


Рис.5.4 Графова модель $\hat{Y}(\bar{X}|\mathfrak{G})^G$ прикладу змішування сольового розчину.

Найпростіший спосіб реалізації перехідних функцій $\Theta^{|\Sigma}_{d,w=1}$ та $\Theta^{|\Sigma}_{d,w=2}$ полягає у використанні функцій $\hat{\omega}_1(\bar{t})$ та $\hat{\omega}_2(\bar{t})$ з множини $F(\bar{X}|\mathfrak{G})$ (формула 5.3). У цьому випадку,

$$\begin{aligned} \Theta^{|\Sigma}_{d,w=1}([\hat{\omega}_1]^{[\bar{t}]}_{k=1}, [\hat{\omega}_2]^{[\bar{t}]}_{k=2}) &= [\hat{\omega}_1(\bar{t} + \Delta t)]^{[\bar{t} + \Delta t]}_{q=1}; \\ \Theta^{|\Sigma}_{d,w=2}([\hat{\omega}_1]^{[\bar{t}]}_{k=1}, [\hat{\omega}_2]^{[\bar{t}]}_{k=2}) &= [\hat{\omega}_2(\bar{t} + \Delta t)]^{[\bar{t} + \Delta t]}_{q=2}. \end{aligned} \quad (5.9)$$

Дещо складніша реалізація полягає в переписуванні системи диференціальних рівнянь (формула 5.4), яку потрібно розв'язати методом Ейлера

$$\begin{cases} \hat{\omega}_{1,i} = \hat{\omega}_{1,i-1} + \Delta t * \frac{q_1 * \omega_3 + q_2 * \hat{\omega}_{2,i-1} - q_3 * \hat{\omega}_{1,i-1}}{v_1}, \\ \hat{\omega}_{2,i} = \hat{\omega}_{2,i-1} + \Delta t * \frac{q_3 * \hat{\omega}_{1,i-1} - q_2 * \hat{\omega}_{2,i-1} - q_4 * \hat{\omega}_{2,i-1}}{v_2}, \end{cases} \quad (5.10)$$

після ітерації Δt : $\hat{\omega}_{1,0} = 0$; $\hat{\omega}_{2,0} = 20$; $i = 1, 2, 3, \dots$

У цьому випадку

$$\begin{aligned}
& \Theta^{|\mathfrak{D}}_{d,w=1}([\hat{\omega}_1]^{[\bar{t}]}_{k=1}, [\hat{\omega}_2]^{[\bar{t}]}_{k=2}) = \\
& = \left[\hat{\omega}_1 + \Delta t * \frac{\mathfrak{D}.q_1 * \mathfrak{D}.\omega_3 + \mathfrak{D}.q_2 * \hat{\omega}_2 - \mathfrak{D}.q_3 * \hat{\omega}_1}{\mathfrak{D}.v_1} \right]^{[\bar{t}+\Delta t]}_{q=1} ; \\
& \Theta^{|\mathfrak{D}}_{d,w=2}([\hat{\omega}_1]^{[\bar{t}]}_{k=1}, [\hat{\omega}_2]^{[\bar{t}]}_{k=2}) = \\
& = \left[\hat{\omega}_2 + \Delta t * \frac{\mathfrak{D}.q_3 * \hat{\omega}_1 - \mathfrak{D}.q_2 * \hat{\omega}_2 - \mathfrak{D}.q_4 * \hat{\omega}_2}{\mathfrak{D}.v_2} \right]^{[\bar{t}+\Delta t]}_{q=2} . \tag{5.11}
\end{aligned}$$

Реалізуємо вузли S та множини ребер $\Theta^{|\mathfrak{P}}$ як класи ООП (псевдокод 5.4.(L89)). S вузлів є, по суті, змінними, які спочатку не визначені. Граф переходів $\Gamma^{|\mathfrak{G}}$ та граф моделювання $\gamma^{|\mathfrak{G}}$ можна представити як класи, що містять колекції вузлів S множин ребер $\Theta^{|\mathfrak{P}}$ (псевдокод 5.4 (L149)). Більше того, граф $\gamma^{|\mathfrak{G}}$ такий самий, як граф $\Gamma^{|\mathfrak{G}}$, але з усіма визначеними змінними S .

Завдяки простоті графа переходів $\Gamma^{|\mathfrak{G}}$, можемо реалізувати функцію $build_ \Gamma(n, \Delta t)$, яка автоматично будує $\Gamma^{|\mathfrak{G}}$ на основі заданої кількості кроків та часового кроку (псевдокод 5.4 (L190)).

Пошук графа моделювання $\gamma(\Gamma^{|\mathfrak{G}}|\mathfrak{S}')$ – це обчислення значень усіх вузлів S з початкового набору підстанів

$$\mathfrak{S}' = \{S_{d=0,w=1} = \mathfrak{S}^x_{j=1}, S_{d=0,w=1} = \mathfrak{S}^x_{j=2}\}. \tag{5.12}$$

Реалізуємо пошук як метод $\Gamma.\gamma(\mathfrak{S}')$, використовуючи індекси d та w як ключ у множині \mathfrak{S}' (псевдокод 5.4 (L156)). Метод спочатку ініціалізує вузли $S_{d=0,w=1}$ та $S_{d=0,w=2}$ початковими підстанами $\mathfrak{S}'^x_{j=1}$ та $\mathfrak{S}'^x_{j=2}$, а потім обчислює значення решти вузлів $S_{d,w}$, викликаючи кожен метод $\Theta^{|\mathfrak{D}}_{d,w}.eval()$, доки всі $S_{d,w}$ не будуть визначені. Метод $\Theta^{|\mathfrak{D}}_{d,w}.eval()$ перевіряє, чи аргументи

$$S_{d-1,w=1}, S_{d-1,w=2} \xrightarrow{\Theta^{|\mathfrak{D}}_{d,w}} ..., \tag{5.13}$$

визначені, і якщо так, оцінює результат

$$... \xrightarrow{\Theta^{|\mathfrak{D}}_{d,w}} S_{d,w}. \tag{5.14}$$

Набір підстанів $\mathfrak{S}^{\mathfrak{x}}(\gamma^{|\mathfrak{G}})$ можна отримати з графа моделювання $\gamma(\Gamma^{|\mathfrak{G}}|\mathfrak{S}')$ шляхом простого вилучення значень з вузлів S та об'єднання їх у множину $\mathfrak{S}^{\bar{\mathfrak{X}}|\mathfrak{G}}$. Реалізуємо це у вигляді методу $\gamma^{|\mathfrak{G}}.\mathfrak{S}()$ (псевдокод 5.4 (L179)); далі, $\mathfrak{S}^{\bar{\mathfrak{X}}|\mathfrak{G}}$ можна використовувати для отримання значень $\hat{\mathfrak{Y}} \in \hat{\mathfrak{Y}}$ зі значень $\bar{\mathfrak{X}} \in \bar{\mathfrak{X}}$ (розділ 5.1).

Можемо зробити моделювання інтерактивною, дозволивши змінювати параметр ω_3 (концентрацію сольового розчину, налитого в перший резервуар) у режимі реального часу. Для цього додамо додаткові вузли $S_{d,w=3}$ до графа $\Gamma^{|\mathfrak{G}}$ джерел у режимі реального часу,

$$\overline{t_{real}} \subset [0, n * \Delta t], \quad (5.15)$$

і приймати такі значення, як

$$\mathfrak{S}^{\mathfrak{x}}_{q=3} = [\hat{\omega}_3]^{[\bar{t}]}_{q=3}. \quad (5.16)$$

Також, його джерело параметра $\hat{\omega}_3$ встановлюється інтерактивно, що переміщується з множини значень параметрів \mathfrak{G} у залежні змінні Y (рис. 5.5).

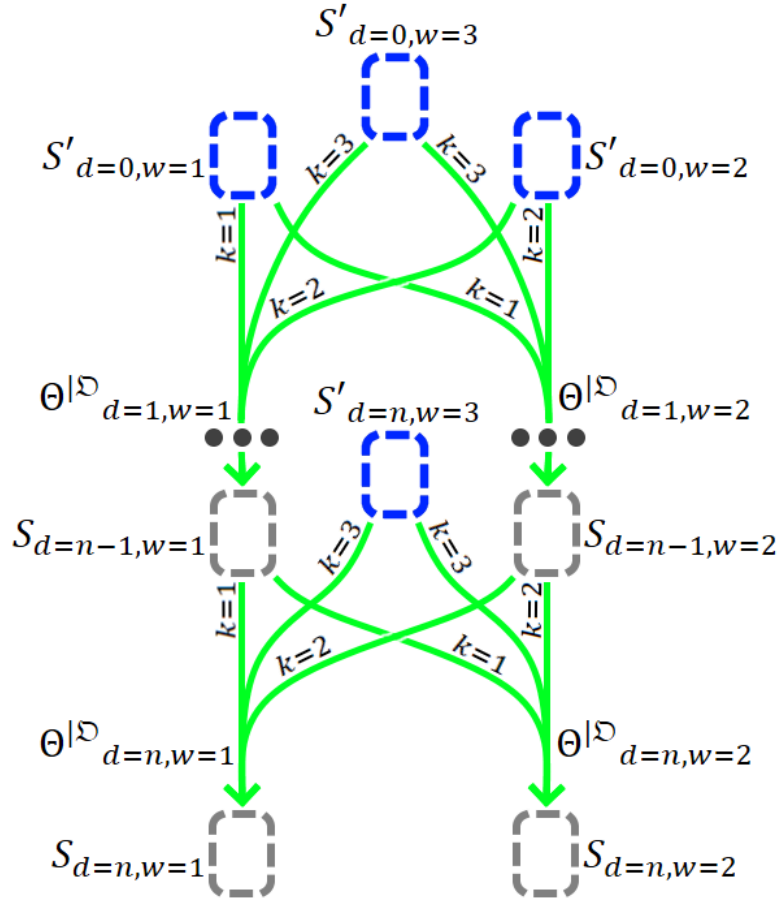


Рис. 5.5 Графова модель $\hat{Y}(\bar{X}|\mathfrak{G})^{\Gamma}$ прикладу змішування сольового розчину для інтерактивного моделювання

Значення $\mathfrak{S}_{q=3}^x$ вузла $S_{d,w=3}$, якими маніпулюємо інтерактивно, можна представити як результат обчислення деякої невизначеної або тіньової частини графа $\Gamma^{|\mathfrak{G}|}$ (розділ 3.6). Зокрема, значення параметра $\hat{\omega}_3$ можна представити як деяку невідому функцію від \bar{t} , таку як $\hat{\omega}_3 = f_{\omega_3}(\bar{t})$. У цьому випадку функція переходу для обчислення $S_{d,w=3}$ буде

$$\Theta^{|\mathfrak{P}|}_{d,w=3}(\phi^{[t_{real}]}_{w=3}) = [f_{\omega_3}(\overline{t_{real}})]^{[t_{real}]}_{w=3}. \quad (5.17)$$

Як перехідні функції використовуватимемо формули для розв'язання методом Ейлера (формула 5.2). Визначимо їх як:

$$\Theta^{|\mathcal{D}|}_{d,w=1}([\hat{\omega}_1]^{[\bar{t}]}_{k=1}, [\hat{\omega}_2]^{[\bar{t}]}_{k=2}, [\hat{\omega}_3]^{[t_{real}]}_{k=3}) =$$

$$\begin{aligned}
&= \left[\hat{\omega}_1 + (\overline{t_{real}} - \bar{t}) * \frac{\mathfrak{D}.q_1 * \hat{\omega}_3 + \mathfrak{D}.q_2 * \hat{\omega}_2 - \mathfrak{D}.q_3 * \hat{\omega}_1}{\mathfrak{P}.v_1} \right]_{q=1}^{[\overline{t_{real}}]}; \\
&\quad \Theta^{|\mathfrak{D}}_{d,w=2}([\hat{\omega}_1]_{k=1}^{[\bar{t}]}, [\hat{\omega}_2]_{k=2}^{[\bar{t}]}, [\hat{\omega}_3]_{k=3}^{[\overline{t_{real}}]}) = \\
&= \left[\hat{\omega}_2 + (\overline{t_{real}} - \bar{t}) * \frac{\mathfrak{D}.q_3 * \hat{\omega}_1 - \mathfrak{D}.q_2 * \hat{\omega}_2 - \mathfrak{D}.q_4 * \hat{\omega}_2}{\mathfrak{D}.v_2} \right]_{q=2}^{[\overline{t_{real}}]}, \quad (5.18)
\end{aligned}$$

Перепишемо функцію $build_ \Gamma(n, \Delta t)$ як функцію $build_interactive_ \Gamma \mathfrak{E}(n)$, яка буде інтерактивний граф $\Gamma^{|\mathfrak{E}}$. Окрім графа, ця функція повертає набір вузлів $\{S_{d,w=3}\}$, значеннями яких будемо маніпулювати далі (псевдокод В.2 (L192)). Також додаємо інтерактивний клас, який представляє невизначену (тіньову) частину $\Gamma^{|\mathfrak{E}}$ (псевдокод В.2 (L252)), з методом $Interaction.next_ \mathfrak{E} \mathfrak{X}_3(i)$, який буде значення для i -ї змінної $S_{d,w=3}$ (вхід), та методом $Interaction.show(X, Y)$, який відображає i -ті обчислені значення X та Y (вихід).

Псевдокод 5.4 (transition_graph_representation.py):

Parameters

```

 $\mathfrak{E}$  =  $\mathfrak{E}_$ (
    v_1 = 4, # L
    v_2 = 8, # L
    q_1 = 3, # L/m
    q_2 = 2, # L/m
    q_3 = 5, # L/m
    q_4 = 3, # L/m
     $\omega_3$  = 10) # g/l

```

```

Δt = .1

n = 10

pS = {
    (0,1): Sx_q_(t=.0, w=0, q=1),      #State for S_d=0,w=1
    (0,2): Sx_q_(t=.0, w=20, q=2)}    #State for S_d=0,w=2

# Transition implementation

def X_transition(Δt):
    def f_t(t):
        return np.round(t + Δt, 4)
    return f_t

def S_functional_transition():
    q = m.sqrt(105.0)
    def em(t):
        return m.exp(((q - 15.0) * t) / 16.0)
    def ep(t):
        return m.exp(-(((q + 15.0) * t) / 16.0))
    def f_w_1(w_1, w_2, t_prev, t, P_1):
        return ((13.0 * em(t) * q) / 21.0) - ((13.0 * ep(t)
* q) / 21.0) - (5.0 * em(t)) - (5.0 * ep(t)) + 10.0
    def f_w_2(w_1, w_2, t_prev, t, P_2):

```

```

        return -((5.0 * em(t) * q) / 21.0) + ((5.0 * ep(t) *
q) / 21.0) + (5.0 * em(t)) + (5.0 * ep(t)) + 10.0

    return f_w_1, f_w_2

```

```

# Build  $\Gamma^{\mathcal{C}}$ 

```

```

 $\Gamma^{\mathcal{C}}$  = build_ $\Gamma^{\mathcal{C}}$ (n,  $\Delta t$ ,  $\mathcal{C}$ , X_transition,
S_functional_transition)

```

```

# Eval  $\gamma^{\mathcal{C}}$  on given  $\mathcal{S}'$  and get  $\mathcal{S}^{\mathcal{X}|\mathcal{C}}$  set

```

```

 $\gamma^{\mathcal{C}}$  =  $\Gamma^{\mathcal{C}}.\gamma(p\mathcal{S})$ 

```

```

set $\mathcal{S}^{\mathcal{X}|\mathcal{C}}$  =  $\gamma^{\mathcal{C}}.\mathcal{S}()$ 

```

```

# Simulation function

```

```

def simulation(set $\mathcal{X}$ ):

```

```

    set $\mathcal{Y}$  = []

```

```

    for  $\mathcal{X}$  in set $\mathcal{X}$ :

```

```

         $\mathcal{S}_{\mathcal{X}_1}$  = None

```

```

         $\mathcal{S}_{\mathcal{X}_2}$  = None

```

```

        for  $\mathcal{S}_{\mathcal{X}_q}$  in set $\mathcal{S}^{\mathcal{X}|\mathcal{C}}$ :

```

```

            if  $\mathcal{S}_{\mathcal{X}_q}.t == \mathcal{X}.t$  and  $\mathcal{S}_{\mathcal{X}_q}.q == 1$ :

```

```

                 $\mathcal{S}_{\mathcal{X}_1} = \mathcal{S}_{\mathcal{X}_q}$ 

```

```

        if Gx_q.t == x.t and Gx_q.q == 2:
            Gx_2 = Gx_q
            setY.append(hY_(Gx_1.w_1, Gx_2.w_2))
    return setY

# Run simulation

setX = np.vectorize(lambda t:
    bX_(t))(np.round(np.arange(0.0, (n + 1) * Δt, Δt), 4))
setY = simulation(setX)

```

Псевдокод 5.5 (transition_graph_interactive_simulation.py):

```

# Parameters

G = G_(
    v_1 = 4, # L
    v_2 = 8, # L
    q_1 = 3, # L/m
    q_2 = 2, # L/m
    q_3 = 5, # L/m
    q_4 = 3) # L/m

n = 100

```

```

Δt = .1

pS = {
    (0,1): Sx_q_(t=.0, w=0, q=1),      #State for S_d=0,w=1
    (0,2): Sx_q_(t=.0, w=20, q=2)}    #State for S_d=0,w=2

init_w_3 = 10.0

up_down_step = 1

# Transition implementation

def S_transition():
    def f_w_1(w_1, w_2, w_3, t, t_next, P_1):
        return w_1 + (t_next - t) * (((P_1.q_1 * w_3) +
        (P_1.q_2 * w_2) - (P_1.q_3 * w_1)) / P_1.v_1)

    def f_w_2(w_1, w_2, w_3, t, t_next, P_2):
        return w_2 + (t_next - t) * (((P_2.q_3 * w_1) -
        (P_2.q_2 * w_2) - (P_2.q_4 * w_2)) / P_2.v_2)

    return f_w_1, f_w_2

# Build  $\Gamma^{\mathbb{C}}$ 

 $\Gamma^{\mathbb{C}}$ , setS_3 = build_interactive_ $\Gamma^{\mathbb{C}}$ (n,  $\mathbb{C}$ , S_transition)

# Helpers functions

```

```

class Interaction:
    def __init__(self,  $\Delta t$ , init_ $\omega_3$ , up_down_step, chart):
        self.input = ""
        self. $\Delta t$  =  $\Delta t$ 
        self. $\omega_3$  = init_ $\omega_3$ 
        self.up_down_step = up_down_step
        self.chart = chart
        def on_key(key):
            self.input = key
            chart.on_key_press(on_key)
    def next_ $\mathcal{G}\mathcal{X}_3$ (self, i):
        t = i * self. $\Delta t$ 
        if self.input == "up":
            self. $\omega_3$  += self.up_down_step
            self.input = ""
        if self.input == "down":
            self. $\omega_3$  -= self.up_down_step
            self.input = ""
        return  $\mathcal{G}\mathcal{X}_q$ (t, self. $\omega_3$ , q=3)
    def show(self, X, Y):

```

```
self.chart.append(x = X.t, ys = [Y.w_1, Y.w_2,
Y.w_3])
```

```
I = Interaction( $\Delta t$ , init_w_3, up_down_step, chart)
```

```
# Interactive simulation
```

```
 $\Gamma\mathcal{E}$ .init(p $\mathcal{S}$ )
```

```
i = 0
```

```
while i < n:
```

```
    S_3 = setS_3[i]
```

```
     $\mathcal{G}\mathcal{X}_3$  = I.next_ $\mathcal{G}\mathcal{X}_3$ (i)
```

```
    S_3.assign( $\mathcal{G}\mathcal{X}_3$ )
```

```
    graph_viz.update()
```

```
    set $\mathcal{G}\mathcal{X}_q$  =  $\Gamma\mathcal{E}$ .eval()
```

```
     $\mathcal{G}\mathcal{X}_1$  = None
```

```
     $\mathcal{G}\mathcal{X}_2$  = None
```

```
    for  $\mathcal{G}\mathcal{X}_q$  in set $\mathcal{G}\mathcal{X}_q$ :
```

```
        if  $\mathcal{G}\mathcal{X}_q.q$  == 1:
```

```
             $\mathcal{G}\mathcal{X}_1$  =  $\mathcal{G}\mathcal{X}_q$ 
```

```
        if  $\mathcal{G}\mathcal{X}_q.q$  == 2:
```

```
             $\mathcal{G}\mathcal{X}_2$  =  $\mathcal{G}\mathcal{X}_q$ 
```

```
    X = b $\mathcal{X}$ _ $(\mathcal{G}\mathcal{X}_3.t_{\text{real}})$ 
```


$$Y = h\eta_-(\mathfrak{S}x_{1.w_1}, \mathfrak{S}x_{2.w_2}, \mathfrak{S}x_{3.w_3})$$

$$I.show(X, Y)$$

$$i += 1$$

5.3 Побудова та обчислення графа C з використанням моделі $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$

Як приклад, побудуємо граф C , використовуючи модель $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$ для змішування розчинів солі (розділ 5.1). Для його реалізації використаємо бібліотеку АККА Streams [10-12]. Подібний підхід був запропонований у [51] для реалізації інструменту SwiftVis.

Можемо побудувати неоптимізовану версію графа C , просто замінивши функції $\Theta^{\mathfrak{P}}_{d,w=1}$ та $\Theta^{\mathfrak{P}}_{d,w=2}$ логічними процесорами $LP^{eval}_{d,w=1}$ та $LP^{eval}_{d,w=2}$ та додавши $LP^{collect}$, $LP^{init}_{d=0,w=1}$ та $LP^{init}_{d=0,w=2}$ (рис. 5.6).

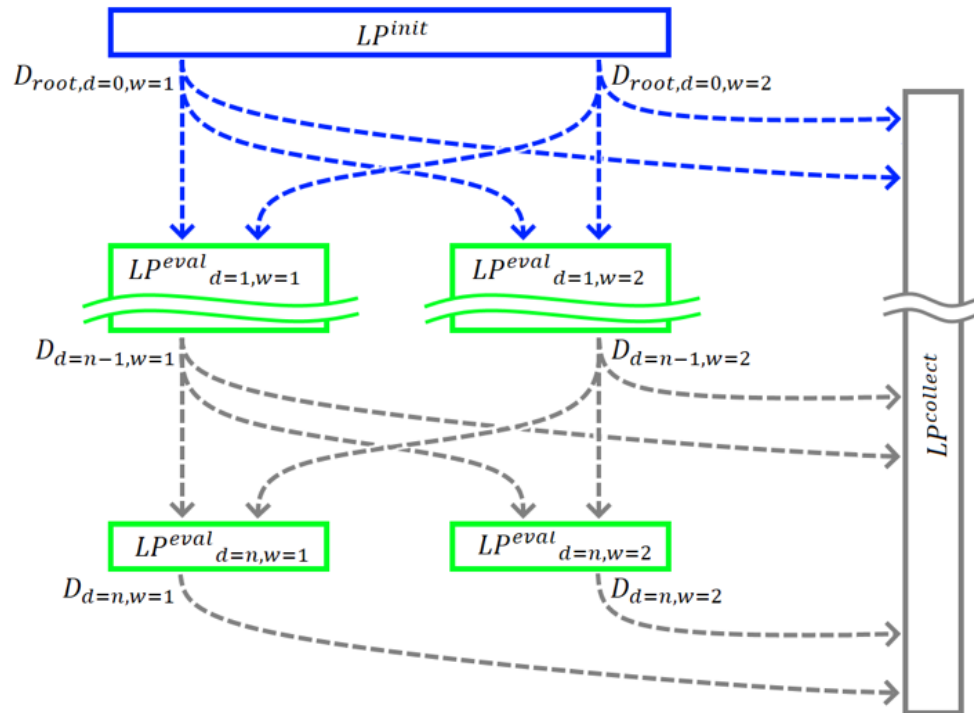


Рис. 5.6 Наївний граф C прикладу змішування фізіологічного розчину

Представимо підстани у вигляді повідомлень $M_{d,w} = [\mathfrak{S}_{q,d,w}^x]$, що генеруються відповідним $LP^{eval}_{d,w}$, де $q = w$. Зокрема, підстани з множини \mathfrak{S}' можна представити як $M_{d=0,w=1} = [S_{d=0,w=1} = \mathfrak{S}'^x_{q=1}]$ та $M_{d=0,w=2} = [S_{d=0,w=2} = \mathfrak{S}'^x_{q=2}]$.

Це працюватиме наступним чином (див. псевдокод 5.6): початкові повідомлення $M_{d=0,w=1}$ та $M_{d=0,w=2}$ надсилаються логічними процесорами $LP^{init}_{d=0,w=1}$ та $LP^{init}_{d=0,w=2}$ до процесорів $LP^{eval}_{d=1,w=1}$, $LP^{eval}_{d=1,w=2}$. Потім повідомлення розподіляться по графу, де копія кожного підстану подається до процесора $LP^{collect}$, який формує результуючий набір підстанів $\mathfrak{S}^{\bar{x}|\mathfrak{G}}$.

Оскільки для побудови графа \mathcal{C} використовувалися стандартні блоки *Zip*, *Flow.map*, *Broadcast* та *Merge* з бібліотеки АККА Streams, реалізація кожного $LP^{eval}_{d,w}$ буде вкладеним графом (рис. 5.7).

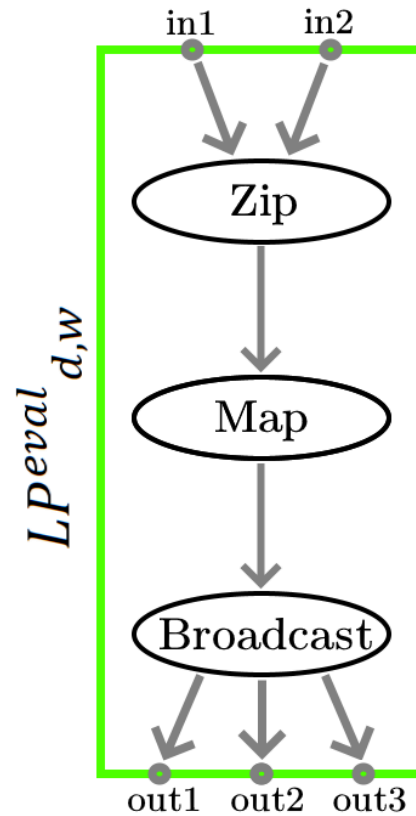


Рис. 5.7 Реалізація $LP^{eval}_{d,w}$ з використанням блоків *Zip*, *Flow.map* та *Broadcast* з бібліотеки *AKKA Streams*

Оскільки отриманий граф C складається з повторюваних пар $LP^{eval}_{d,w=1}$ та $LP^{eval}_{w=1}$, його можна оптимізувати, реалізувавши два цикли з використанням 2 логічних процесорів $LP^{eval}_{w=1}$ та $LP^{eval}_{w=2}$ (рис. 5.8).

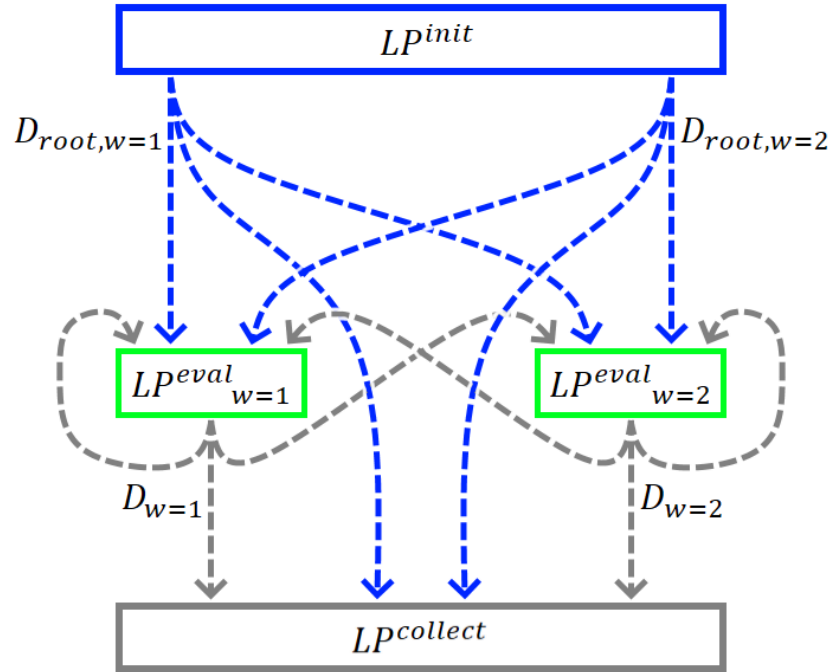


Рис. 5.8 Оптимізований графік C прикладу змішування фізіологічного розчину

Оскільки в цьому випадку необхідно визначити, які вхідні повідомлення відносяться до певних ітерацій циклу, додаємо до них лічильник ітерацій (глибини) d , $M_{d,w} = [d, \mathfrak{S}_q^x]$, та модифікуємо функцію групування Zip так, щоб вона вибирала пари вхідних $M_{d,w}$ з однаковим значенням d (псевдокод 5.7).

Коли виконаємо цей код, отримуємо результат моделювання (рис. 5.9), який збігається з нашими висновками, отриманими під час реалізації моделі $\hat{Y}(\bar{X}|\mathfrak{G})^S$ (рис. 5.2).

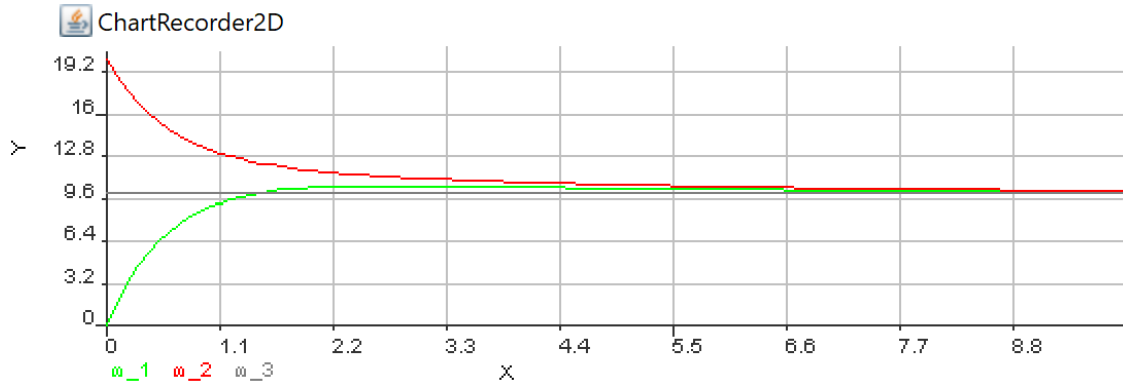


Рис. 5.9 Обчислення моделі $\hat{Y}(\bar{X}|\mathfrak{G})^\Gamma$ з використанням графа \mathcal{C}

Можемо зробити моделювання інтерактивним, додавши ще один вхід до логічних процесорів $LP^{eval}_{d,w=1}$ та $LP^{eval}_{d,w=2}$, а також процесор $LP^{init}_{d=0,w=3}$ для введення значень t у реальному часі та параметрів ω_3 (рис. 5.10).

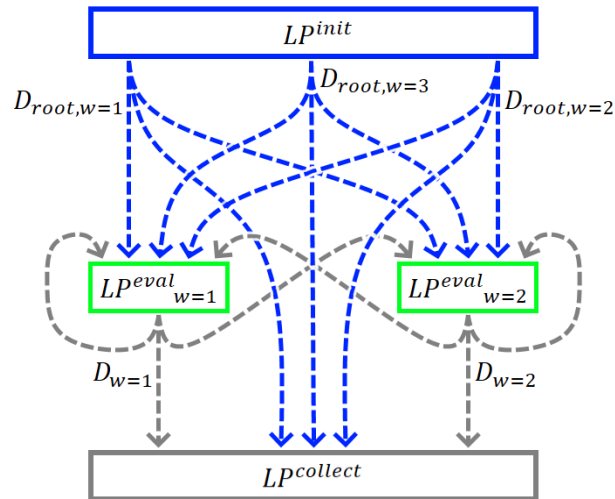


Рис. 5.10 Оптимізований граф \mathcal{C} прикладу змішування сольового розчину для інтерактивного моделювання

На відміну від попередньої версії, моделювання зупинятиметься на кожному кроці, доки процесор $LP^{init}_{d=0,w=3}$ не згенерує наступне повідомлення $M_{d,w=3} =$

$[d + 1, \mathfrak{S}_{q=3}^x]$. Повідомлення $M_{d,w=3}$ містить наступний t (шаблон проштовхування) та поточний ω_3 (шаблон pull); див. псевдокод 5.8.

Псевдокод 5.6 (ReactiveNaiveImplementation.scala):

```
object ReactiveNaiveImplementation extends ScriptBase with
Plotting with GraphVisualization{
  // Parameters
  val n = 100
  val Δt = 0.1
  val S_prime_1 = SX_1_(t = 0.0, ω_1 = 0.0)
  val S_prime_2 = SX_2_(t = 0.0, ω_2 = 20.0)
  val G = G_(
    v_1 = 4, // L
    v_2 = 8, // L
    q_1 = 3, // L/m
    q_2 = 2, // L/m
    q_3 = 5, // L/m
    q_4 = 3, // L/m
    ω_3 = 10) // g/l

  //Transition implementation
  def X_transition(Δt: D): D⇒D = t ⇒ t + Δt
  def S_functional_transition(P_1: P_1_, P_2: P_2_):
((D,D,D,D)⇒D, (D,D,D,D)⇒D) = {
    val q = math.sqrt(105.0)
    def em(t: D) = math.exp(((q - 15.0) * t) / 16.0)
    def ep(t: D) = math.exp(-(((q + 15.0) * t) / 16.0))
```

```

def f_w_1(ω_1: D, ω_2: D, t_prev: D, t: D) =
  ((13.0 * em(t) * q) / 21.0) - ((13.0 * ep(t) * q) /
21.0) - (5.0 * em(t)) - (5.0 * ep(t)) + 10.0
def f_w_2(ω_1: D, ω_2: D, t_prev: D, t: D) =
  -((5.0 * em(t) * q) / 21.0) + ((5.0 * ep(t) * q) /
21.0) + (5.0 * em(t)) + (5.0 * ep(t)) + 10.0
  (f_w_1, f_w_2)}

// Build graph
val C = buildC(n, Δt, G, X_transition,
S_functional_transition, S_prime_1, S_prime_2, sink_S_X_G,
viz_graph)

// Show graph
viz_graph.show()

// Run processing
val system = ActorSystem("ExampleSys")
val materializer = ActorMaterializer()(system)
val mat = C.run()(materializer)
}

```

Псевдокод 5.7 (ReactiveOptimizedImplementation.scala):

```

object ReactiveOptimizedImplementation extends ScriptBase
with Plotting {
  // Parameters
  val Δt = 0.1
  val S_prime_1 = SX_1_(t = 0.0, ω_1 = 0.0)

```

```

val S_prime_2 = SX_2_(t = 0.0, ω_2 = 20.0)
val G = G_(
    v_1 = 4, // L
    v_2 = 8, // L
    q_1 = 3, // L/m
    q_2 = 2, // L/m
    q_3 = 5, // L/m
    q_4 = 3, // L/m
    ω_3 = 10) // g/l

//Transition implementation
def X_transition(Δt: D): D⇒D = t ⇒ t + Δt
def S_functional_transition(P_1: P_1_, P_2: P_2_):
((D,D,D,D)⇒D, (D,D,D,D)⇒D) = {
    val q = math.sqrt(105.0)
    def em(t: D) = math.exp(((q - 15.0) * t) / 16.0)
    def ep(t: D) = math.exp(-(((q + 15.0) * t) / 16.0))
    def f_ω_1(ω_1: D, ω_2: D, t_prev: D, t: D) =
        ((13.0 * em(t) * q) / 21.0) - ((13.0 * ep(t) * q) /
21.0) - (5.0 * em(t)) - (5.0 * ep(t)) + 10.0
    def f_ω_2(ω_1: D, ω_2: D, t_prev: D, t: D) =
        -((5.0 * em(t) * q) / 21.0) + ((5.0 * ep(t) * q) /
21.0) + (5.0 * em(t)) + (5.0 * ep(t)) + 10.0
    (f_ω_1, f_ω_2)}

// Build graph
val C = buildC(Δt, G, X_transition,
S_functional_transition, S_prime_1, S_prime_2, sink_S_X_G)

```



```
// Run processing
val system = ActorSystem("ExampleSys")
val materializer = ActorMaterializer()(system)
val mat = C.run()(materializer)
}
```

Псевдокод 5.8 (ReactiveInteractiveImplementation.scala):

```
object ReactiveInteractiveImplementation extends ScriptBase
with Plotting with Input {

  // Parameters

  val n = 100

  val  $\Delta t$  = 0.1

  val S_prime_1 = SX_1_(t = 0.0,  $\omega_1$  = 0.0)
  val S_prime_2 = SX_2_(t = 0.0,  $\omega_2$  = 20.0)

  val G = G_(
    v_1 = 4, // L
    v_2 = 8, // L
    q_1 = 3, // L/m
    q_2 = 2, // L/m
    q_3 = 5, // L/m
    q_4 = 3) // L/m

  val init_ $\omega_3$  = 10.0

  val up_down_step = 1
```

```

//Transition implementation

def S_transition(P_1: P_1_, P_2: P_2_): ((D,D,D,D,D)⇒D,
(D,D,D,D,D)⇒D) = {

  def f_w_1(ω_1: D, ω_2: D, ω_3: D, t: D, t_next: D) =

    ω_1 + (t_next - t) * (((P_1.q_1 * ω_3) + (P_1.q_2 *
ω_2) - (P_1.q_3 * ω_1)) / P_1.v_1)

  def f_w_2(ω_1: D, ω_2: D, ω_3: D, t: D, t_next: D) =

    ω_2 + (t_next - t) * (((P_2.q_3 * ω_1) - (P_2.q_2 *
ω_2) - (P_2.q_4 * ω_2)) / P_2.v_2)

  (f_w_1, f_w_2)}

// Output

val sink_S_X_G = Sink.fold[(Map[D, SX_1_], Map[D, SX_2_],
Map[D, SX_3_]), M]((Map(), Map(), Map())){

  case ((s1, s2, s3), m) ⇒

  val (us1, us2, us3) = m match{

    case M(_, ss1: SX_1_) ⇒ (s1 + (ss1.t → ss1), s2, s3)

    case M(_, ss2: SX_2_) ⇒ (s1, s2 + (ss2.t → ss2), s3)

    case M(_, ss3: SX_3_) ⇒ (s1, s2, s3 + (ss3.t → ss3))}

  if(us1.contains(m.ss.t) && us2.contains(m.ss.t) &&
us3.contains(m.ss.t)){

    chart.addPoints(m.ss.t, Seq(us1(m.ss.t).ω_1,
us2(m.ss.t).ω_2, us3(m.ss.t).ω_3))}

  (us1, us2, us3)}

```

```
// Build graph

val C = buildC( $\Delta t$ ,  $\omega_3$ _input. $\omega_3$ , G, S_transition,
S_prime_1, S_prime_2, sink_S_X_G)

// Run processing

val system = ActorSystem("ExampleSys")
val materializer = ActorMaterializer()(system)
val mat = C.run()(materializer)
}
```

ВИСНОВКИ ДО РОЗДІЛУ 5

В цьому розділі ралізовано демонстраційний приклад обчислювального графа на базі бібліотеки AKKA Streams для задачі змішування сольових розчинів, результати якого співпадають з аналітичними розрахунками, що підтверджує коректність підходу. На цьому прикладі апробовано методи оптимізації обчислювальних графів та інтерактивного моделювання.

А також розроблено практичні рекомендації щодо перспектив використання паралельних моделей складних динамічних систем, в різних галузях (енергетика, фізичне моделювання, біологічні та соціальні системи).

Експерименти, проведені в цьому розділі, забезпечують всебічну перевірку запропонованого методу побудови паралельних числових моделей динамічних систем з використанням протоколу реактивних потоків. За допомогою послідовності практичних прикладів, починаючи з класичної задачі змішування сольових розчинів, робота демонструє, як теоретичні визначення та формулювання можуть бути перетворені на конкретні обчислювальні процедури.

На завершення, результати, представлені в цьому розділі, демонструють не лише доцільність, але й універсальність запропонованого методу. Метод довів свою здатність поєднувати теоретичні конструкції з практичним виконанням, пропонуючи узгоджену структуру, яку можна реалізувати як у традиційних середовищах програмування, так і в передових реактивних системах. Ці висновки створюють міцну основу для подальшого дослідження більших та складніших моделей, підтверджуючи, що підхід має потенціал для значного підвищення ефективності та гнучкості паралельного числового моделювання.

ЗАГАЛЬНІ ВИСНОВКИ

У дисертаційній роботі в результаті проведених теоретичних і практичних досліджень вирішено важливе наукове завдання з розвитку принципів побудови паралельних моделей складних динамічних систем на основі парадигми реактивних потоків як універсального протоколу синхронізації. Було сформовано формальний апарат для опису підстанів, перехідних функцій і графових моделей, а також проведено експериментальну перевірку коректності підходу на прикладі задачі змішування сольових розчинів. Основні наукові та практичні результати проведених досліджень полягають у наступному:

1. Показано еволюцію підходів до паралельного моделювання від консервативних і оптимістичних стратегій до сучасних архітектур, таких як RxHLA, CQRS+ES та HPC-фреймворки на основі акторів. Класичний компроміс між коректністю та продуктивністю залишається нерозв'язаним, адже кожен підхід має власні обмеження щодо масштабованості, простоти чи гнучкості. На цьому тлі методи на базі реактивних потоків усувають багато вад традиційних протоколів, інтегруючи синхронізацію безпосередньо у потоки даних та знижуючи комунікаційні витрати. Це робить методи на основі реактивних потоків перспективним напрямом для побудови ефективних і масштабованих моделей динамічних систем.
2. Розроблено метод побудови паралельних чисельних моделей складних динамічних систем на базі протоколу реактивних потоків, який, на відміну від існуючих, враховує поточну архітектуру обчислювальних систем і сучасні методи розроблення програмного забезпечення, що

дозволяє підвищити ефективність виконання чисельного моделювання у багатопроцесорних і розподілених середовищах.

3. Запропоновано підхід до подання станів динамічних систем у вигляді множини підстанів із використанням унікальних ключів, що, на відміну від монолітних станів в традиційних паралельних моделях, забезпечує модульність, повторне використання та узгодженість при побудові графів переходів і обчислювальних графів в паралельних моделях.
4. Запропоновано подання динамічних моделей у вигляді графа переходів і розроблено підхід його перетворення у структуровану графову модель, що на відміну від використання імперативних описів (наборів інструкцій для виконання машинами) істотно спрощує процес створення та масштабування моделей у багатопроцесорних і розподілених середовищах.
5. Запропоновано використання push- та pull-шаблонів для паралельного інтерактивного моделювання складних динамічних систем, що на відміну від класичних шаблонів, забезпечує ефективне використання обчислювальних ресурсів та менший час відгуку моделі, під час синхронізації отриманих моделей з реальною системою по параметрах моделі.
6. Розроблено метод перетворення графів переходів на обчислювальні графи, що реалізуються у парадигмі реактивних потоків (зокрема, з використанням бібліотеки AKKA Streams). Такий метод, на відміну від традиційних (побудованих на основі потоків інструкцій), забезпечує більш природне поєднання абстрактної специфікації з виконуваними обчисленнями, а також спрощує процес масштабованої реалізації моделей у багатопроцесорних та розподілених середовищах.

7. Реалізовано демонстраційний приклад обчислювального графа на базі бібліотеки AKKA Streams для задачі змішування сольових розчинів, результати якого співпадають з аналітичними розрахунками, що підтверджує коректність підходу. На цьому прикладі апробовано методи оптимізації обчислювальних графів та інтерактивного моделювання
8. Розроблено практичні рекомендації щодо перспектив використання паралельних моделей складних динамічних систем, в різних галузях (енергетика, фізичне моделювання, біологічні та соціальні системи).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- [1] D. R. Jeerson, H. A. Sowizral, Fast concurrent simulation using the time warp mechanism, Part I: Local Control, *Rand Corp Santa Monica CA*, 1982.
- [2] R. Richard, M. Fujimoto, *Parallel and Distributed Simulation Systems*, Wiley, New York, 2000.
- [3] R. Radhakrishnan, D. E. Martin, M. Chetlur, D. M. Rao, P. A. Wilsey, An Object-Oriented, Time Warp Simulation Kernel. in *D. Caromel, R. R. Oldehoeft, and M. Tholburn, editors, Proc. of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)* 1998, 1505, 13-23.
- [4] D. Jefferson, B. Beckman, F. Wieland, L. Blume, M. Diloroto, Time warp operating system, in *Proc. of the eleventh ACM Symposium on Operating systems principles* 1987, 21, 77-93.
- [5] John Aach, George M. Church, Aligning gene expression time series with time warping algorithms , *Bioinformatics*, Volume 17, Issue 6, June 2001, Pages 495–508, <https://doi.org/10.1093/bioinformatics/17.6.495>
- [6] D. M. Nicol, R. M. Fujimoto, Parallel simulation today, *Ann. Oper. Res.*, 1994, 53, 249. <https://doi.org/10.1007/BF02136831>
- [7] A. Falcone, A. Garro, Reactive HLA-based distributed simulation systems with rxhla, in *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)* 2018, 1-8.
- [8] A. Debski, B. Szczepanik, M. Malawski, S. Spahr, In Search for a scalable & reactive architecture of a cloud application: CQRS and event sourcing case study, *IEEE Software (accepted preprint)*, copyright IEEE(99), 2016.
- [9] J. Bujas, D. Dworak, W. Turek, A. Byrski, High-performance computing framework with desynchronized information propagation for large-scale simulations, *J. Comp. Sci.* 2019, 32, 70-86. <https://doi.org/10.1016/j.jocs.2018.09.004>

- [10] Reactive stream initiative, <https://www.reactive-streams.org>, accessed: July, 2025.
- [11] Davis, Adam L. Reactive Streams in Java: Concurrency with RxJava, Reactor, and Akka Streams, Apress, 2018.
- [12] Herminio Paucar Curasma and Júlio César Estrella. 2023. Reactive Software Architectures in IoT: A Literature Review. In Proceedings of the 2023 International Conference on Research in Adaptive and Convergent Systems (RACS '23). Association for Computing Machinery, New York, NY, USA, Article 25, 1–8. <https://doi.org/10.1145/3599957.3606212>
- [13] The implementation of reactive streams in AKKA: <https://doc.akka.io/docs/akka/current/stream/stream-introduction.html>, accessed: July, 2025.
- [14] Bjarno Oeyen, Joeri De Koster, and Wolfgang De Meuter. 2023. A Graph-Based Formal Semantics of Reactive Programming from First Principles. In Proceedings of the 24th ACM International Workshop on Formal Techniques for Java-like Programs (FTfJP '22). Association for Computing Machinery, New York, NY, USA, 18–25. <https://doi.org/10.1145/3611096.3601>
- [15] R. Posa, (2018). Scala Reactive Programming: Build Scalable, Functional Reactive Microservices with Akka, Play, and Lagom. Німеччина: Packt Publishing
- [16] Christian Baxter,. Mastering Akka. Germany, Packt Publishing, 2016.
- [17] A. Pietarinen, Y. Senden, S. Sharpless, A. Shepperson, T. Vehkavaara, The Commens Dictionary of Piece's Terms, "Object," 2009-03-19 (<https://web.archive.org/web/20090214004523/http://www.helsinki.fi/science/commens/terms/object.html>)
- [18] S. Sismondo, Models, Simulations, and Their Objects. Science in Context. 1999;12(2):247-260. doi:10.1017/S0269889700003409
- [19] P. Achinstein, Theoretical models, *Br. J. Philos. Sci.* 1965, 16, 102-20. <https://doi.org/10.1093/bjps/XVI.62.102>

- [20] О.В. Сіроткін, М.С. Ярошинський, Д.П. Сінько, С.Б. Гунько, Д.О. Манолук, Моделювання у фазовому просторі під-станів, *Elektron. model.* 2025, 47(2):28-45, <https://doi.org/10.15407/emodel.47.03.028>.
- [21] О.В. Сіроткін. Паралельне моделювання з використанням реактивних потоків: Матеріали ХЛІ науково-технічна конференція молодих вчених та спеціалістів інституту проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України. Київ, 17 травня 2023 р. С. 167-175.
- [22] J. Banks, J. Carson, B. Nelson, D. Nicol, *Discrete-Event System Simulation*. Prentice Hall, 2001. ISBN 0-13-088702-1.
- [23] Context in Computing: A Cross-Disciplinary Approach for Modeling the Real World. USA, Springer New York, 2014.
- [24] András Varga. "Discrete event simulation system." Proc. of the European Simulation Multiconference (ESM'2001). Vol. 17. 2001.
- [25] Choi, Byoung Kyu, and Donghun Kang. Modeling and simulation of discrete event systems. John Wiley & Sons, 2013.
- [26] David Goldsman, and Paul Goldsman. "Discrete-event simulation." Modeling and Simulation in the Systems Engineering Life Cycle: Core Concepts and Accompanying Lectures. London: Springer London, 2015. 103-109.
- [27] E. R. Babbie, *The practice of social research*, Wadsworth Publishing, 2009. ISBN 0-495-59841-0.
- [28] A. D. Myshkis, Classification of applied mathematical models - the main analytical methods of their investigation, *Elements of the Theory of Mathematical Models* 2007, 9.
- [29] Stewart Robinson. "Conceptual modeling for simulation." 2013 Winter Simulations Conference (WSC). IEEE, 2013.
- [30] Stewart Robinson. "A tutorial on conceptual modeling for simulation." 2015 Winter Simulation Conference (WSC). IEEE, 2015.

- [31] Mohammed Adel Abdelmegid, et al. "Towards a conceptual modeling framework for construction simulation." 2017 Winter Simulation Conference (WSC). IEEE, 2017.
- [32] D. D. Nolte, The tangled tale of phase space, *Phys. Today* 2010, 63, 33-38.
- [33] Thomas L. Curtright, Cosmas K. Zachos. "Quantum mechanics in phase space." *Asia Pacific Physics Newsletter* 1.01 (2012): 37-46.
- [34] Wu Baihua, Xin He, Jian Liu. "Nonadiabatic field on quantum phase space: A century after Ehrenfest." *The Journal of Physical Chemistry Letters* 15.2 (2024): 644-658.
- [35] N. B. Hastings, *Workshop Calculus: Guided Exploration with Review*, Springer Science & Business Media, 1998.
- [36] Lionel C. Briand, and Jurgen Wust. "Modeling development effort in object-oriented systems using design properties." *IEEE Transactions on Software Engineering* 27.11 (2001): 963-986.
- [37] Lionel C. Briand, John W. Daly, and Jurgen K. Wust. "A unified framework for coupling measurement in object-oriented systems." *IEEE Transactions on software Engineering* 25.1 (1999): 91-121.
- [38] R. M. Keller, Formal verification of parallel programs, *Communications of the ACM* 1976, 19, 371-384.
- [39] Martin Erwig, and Steve Kollmansberger. "Functional pearls: Probabilistic functional programming in Haskell." *Journal of functional programming* 16.1 (2006): 21-34.
- [40] Abhishek Saini, and Laurent Thiry. "Functional programming for business process modeling." *IFAC-PapersOnLine* 50.1 (2017): 10526-10531
- [41] S. Van den Vonder, J. De Koster, F. Myter, W. Meuter, Tackling the awkward squad for reactive programming: the actor-reactor model, in *Proc. of the 4th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems* 2017.

- [42] R. Mogk, L. Baumgärtner, G. Salvaneschi, B. Freisleben, M. Mezini, Fault-tolerant distributed reactive programming, in *32nd European Conference on Object-Oriented Programming (ECOOP 2018)* 2018.
- [43] K. Shibanaï, T. Watanabe, Distributed functional reactive programming on actor-based runtime, in *Proc. of the 8th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control* 2018.
- [44] O. Sirotkin¹, A. Prymushko, I. Puchko, H. Kravtsov, M. Yaroshynskyi, V. Artemchuk, Parallel simulation using reactive streams: A graph-based approach for dynamic modeling and optimization, *Computation* 2025, 13(5), 103; <https://doi.org/10.3390/computation13050103>.
- [45] М.С. Ярошинський, О.В. Сіроткін, Д.П. Сінько, С.Б. Гунько, Д.О. Манолук, Коректність пласкої класифікації, *Elektron. model.* 2023, 45(2):34-43, <https://doi.org/10.15407/emodel.45.02.034>.
- [46] O. Sirotkin, Cyclic Directed Probabilistic Graphical Model: A Proposal Based on Structured Outcomes, arXiv:2310.16525 [cs.LG] (preprint), <https://doi.org/10.48550/arXiv.2310.16525>.
- [47] О.В. Сіроткін. Циклічно спревалена імовірнісна графічна модель: пропозиція, основана на структурованих результатах: Матеріали Круглого столу «Meaningful Artificial Intelligence – 2024» Київ, 26 січня 2024 р. С. 15-21.
- [48] M. Lohstroh, I. I. Romeo, A. Goens, P. Derler, J. Castrillon, E. A. Lee, A. Sangiovanni-Vincentelli, Reactors: A deterministic model for composable reactive systems, In *Cyber Physical Systems. Model-Based Design* (pp. 59-85). Springer, Cham, 2019.
- [49] About the graphs in AKKA streams: <https://doc.akka.io/docs/akka/2.5/stream/stream-graphs.html>, accessed: August, 2025.

- [50] About Simulink package:
<https://www.mathworks.com/products/simulink.html>, accessed: July, 2019.
- [51] Steven T. Karris, Introduction to Simulink with engineering applications. Orchard Publications, 2006.
- [52] L-A. Dessaint, et al. "A power system simulation tool based on Simulink." IEEE Transactions on Industrial Electronics 46.6 (1999): 1252-1254
- [53] Z. Kurima-Blough, M. C. Lewis, L. Lacher, Modern parallelization for a dataflow programming environment, in *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pp. 101-107. *The Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp)*, 2017.
- [54] S. Kirushanth, B. Kabaso, Designing a cloud-native weigh-in-motion, in *2019 Open Innovations (OI)*, 2019.
- [55] Arsentii Prymushko, Ivan Puchko, Mykola Yaroshynskyi, Dmytro Sinko, Hryhoriy Kravtsov, Volodymyr Artemchuk. Efficient State Synchronization in Distributed Electrical Grid Systems Using Conflict-Free Replicated Data Types. IoT, 6(1), 6, 2025. <https://doi.org/10.3390/iot6010006>
- [56] Oeyen, Bjarno, Joeri De Koster, and Wolfgang De Meuter. "Reactive Programming without Functions." arXiv preprint arXiv:2403.02296 (2024).
- [57] Babaei Majid, Mojtaba Bagherzadeh, and Juergen Dingel. "Efficient reordering and replay of execution traces of distributed reactive systems in the context of model-driven development." Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. 2020.
- [58] M. Yaroshynskyi, A. Prymushko, I. Puchko, O. Sirotkin, and D. Sinko, Akka as a tool for modelling and managing a smart grid system, Journal of Edge Computing 2025, 4(1), pp.105–115. <https://doi.org/10.55056/jec.822>.

- [59] I. Kuraj, A. Solar-Lezama, Aspect-oriented language for reactive distributed applications at the edge, in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, 2020.
- [60] Jochem Broekhoff. "Programming Languages For Programs For Stateful Distributed Systems."
- [61] B. Zoph, Q. V. Le, Neural Architecture Search with Reinforcement Learning, arXiv preprint arXiv:1611.01578, 2016
- [62] T. Nakata, S. Chen, S. Saiki, et al. Enhancing Personalized Service Development with Virtual Agents and Upcycling Techniques. *Int J Netw Distrib Comput* 13, 5 (2025). <https://doi.org/10.1007/s44227-024-00043-y>
- [63] M. Liu, L. Zhang, J. Chen, et al. Large language models for building energy applications: Opportunities and challenges. *Build. Simul.* 18, 225–234 (2025). <https://doi.org/10.1007/s12273-025-1235-9>
- [64] T. N. Kipf, M. Welling, Semi-Supervised Classification with Graph Convolutional Networks. *International Conference on Learning Representations (ICLR)*, 2017
- [65] Mingshuo Nie, Dongming Chen, Huilin Chen, Dongqi Wang,; AutoMTNAS: Automated meta-reinforcement learning on graph tokenization for graph neural architecture search, *Knowledge-Based Systems*, Volume 310, 2025, 113023, <https://doi.org/10.1016/j.knosys.2025.113023>
- [66] Zeki Kuş, Musa Aydin, Berna Kiraz, Alper Kiraz, Neural Architecture Search for biomedical image classification: A comparative study across data modalities, *Artificial Intelligence in Medicine*, Volume 160, 2025, 103064, <https://doi.org/10.1016/j.artmed.2024.103064>
- [67] Devendra K. Chaturvedi, *Modeling and simulation of systems using MATLAB and Simulink*. CRC press, 2017
- [68] Mark C. Lewis, and Lisa L. Lacher. "Swiftvis2: Plotting with spark using scala." *International Conference on Data Science (ICDATA'18)*. Vol. 1. No. 1. 2018

- [69] Yoshitaka Sakurai, and Takuo Watanabe. "Towards a statically scheduled parallel execution of an FRP language for embedded systems." Proceedings of the 6th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems. 2019
- [70] Z. Wang, et al., Adaptive Resource Management for Data Stream Processing Systems Using Deep Reinforcement Learning. IEEE Transactions on Parallel and Distributed Systems, 31(5), 1100-2, 2020
- [71] Y. Chen, et al., Deep Reinforcement Learning for Task Scheduling in Cloud Computing: A Survey. IEEE Access, 8, 103904-103921, 2020
- [72] Jonathan Bassen, et al. "Reinforcement learning for the adaptive scheduling of educational activities." Proceedings of the 2020 CHI conference on human factors in computing systems. 2020.
- [73] Le Ngoc Bao Long, Sam-Sang You, Truong Ngoc Cuong, Hwan-Seong Kim, Optimizing quay crane scheduling using deep reinforcement learning with hybrid metaheuristic algorithm, Engineering Applications of Artificial Intelligence, Volume 143, 2025,110021, <https://doi.org/10.1016/j.engappai.2025.110021>.

ДОДАТОК А

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ ТА ВІДОМОСТІ ПРО АПРОБАЦІЮ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ

1. O. Sirotkin, A. Prymushko, I. Puchko, H. Kravtsov, M. Yaroshynskyi, V. Artemchuk, Parallel simulation using reactive streams: A graph-based approach for dynamic modeling and optimization, *Computation* 2025, 13(5), 103; <https://doi.org/10.3390/computation13050103>. **Index in SCOPUS Q2**. (Особистий внесок – приймав участь у проведенні аналітичного огляду, та концептуалізації для побудови методу побудови паралельних чисельних моделей динамічних систем на базі протоколу реактивних потоків, узагальненні результатів та підготовці всіх розділів).
2. M. Yaroshynskyi, A. Prymushko, I. Puchko, O. Sirotkin, D. Sinko, Akka as a tool for modelling and managing a smart grid system, *Journal of Edge Computing* 2025, 4(1), pp.105–115. <https://doi.org/10.55056/jec.822>. **Index in SCOPUS**. (Особистий внесок – приймав участь у дослідженні розумних мереж, та редагуванні всіх розділів, також приймав участь розробці програмного забезпечення).
3. О.В. Сіроткін, М.С. М.С. Ярошинський, Д.П. Сінько, С.Б. Гунько, Д.О. Манолук, Моделювання у фазовому просторі під-станів, *Електронне моделювання* 2025, 47(2):28-45, <https://doi.org/10.15407/emodel.47.03.028>. Фахове видання категорії Б. (Особистий внесок – приймав участь у проведенні аналітичного огляду, та розробці програмного забезпечення).
4. М.С. Ярошинський, О.В. Сіроткін, Д.П. Сінько, С.Б. Гунько, Д.О. Манолук, Коректність пласкої класифікації, *Електронне моделювання* 2023, 45(2):34-43, <https://doi.org/10.15407/emodel.45.02.034>. Фахове видання категорії Б. (Особистий внесок – приймав участь у дослідженні пласкої класифікації, та редагуванні всіх розділів).

5. О.В. Сіроткін. *Паралельне моделювання з використанням реактивних потоків*: Матеріали ХІІ науково-технічна конференція молодих вчених та спеціалістів інституту проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України. Київ, 17 травня 2023 р. С. 167-175.
6. О.В. Сіроткін. *Циклічно спревалена імовірнісна графічна модель: пропозиція, основана на структурованих результатах*: Матеріали Круглого столу «Meaningful Artificial Intelligence – 2024» Київ, 26 січня 2024 р. С. 15-21.

ДОДАТОК Б

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ

Основні положення та результати дослідження, пов'язані з розробкою методу паралельного моделювання на основі протоколу реактивних потоків, були представлені та обговорені на науково-практичних конференціях різного рівня:

1. XLI Науково-технічна конференція молодих вчених та спеціалістів (Київ, 2023, 17 травня 2023 р., форма участі – виступ і публікація тез).
2. Круглий стіл «Meaningful Artificial Intelligence – 2024» (Київ, 2025, 26 січня 2024 р., форма участі – виступ і публікація тез).

Результати роботи також були представлені на наукових семінарах відділу математичного та комп'ютерного моделювання Інституту проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України.