Київський національний університет імені Тараса Шевченка
Міністерство освіти і науки України
Інститут кібернетики імені В.М. Глушкова
Національна академія наук України

Кваліфікаційна наукова
праця на правах рукопису

Терлецький Дмитро Олександрович

УДК 004.82+004.89+004.43

ДИСЕРТАЦІЯ

ОБ'ЄКТНО-ОРІЄНТОВАНА ДИНАМІЧНА МОДЕЛЬ ПОДАННЯ ЗНАНЬ В ІНТЕЛЕКТУАЛЬНИХ ПРОГРАМНИХ СИСТЕМАХ

01.05.03 — Математичне і програмне забезпечення обчислювальних машин та систем
12 Інформаційні технології

Подається на здобуття наукового ступеня кандидата фізико-математичних наук

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

_____ Д.О. Терлецький

Науковий керівник Провотар Олександр Іванович, доктор фізико-математичних наук, професор

Київ — 2018

АНОТАЦІЯ

Терлецький Д.О. Об'єктно-орієнтована динамічна модель подання знань в інтелектуальних програмних системах. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата фізико-математичних наук (доктора філософії) за спеціальністю 01.05.03 – математичне та програмне забезпечення обчислювальних машин і систем (12 – Інформаційні технології). – Київський національний університет імені Тараса Шевченка, Інститут кібернетики імені В.М. Глушкова НАН України, Київ, 2018.

Дисертація присвячена розробці об'єктно-орієнтованої динамічної моделі подання знань з розширеними можливостями подання предметних областей для проектування та розробки інтелектуальних програмних систем на основі знань.

У роботі розглянуто та проаналізовано означення поняття *знання*, формулювання задачі *подання знань*, а також такі об'єктно-орієнтовані моделі подання знань, як системи фреймів, скрипти та динамічна пам'ять, об'єктно-орієнтоване програмування на основі класів та на основі прототипів. Окрім цього, було розглянуто та проаналізовано відомі застосування моделей подання знань та основні класи інтелектуальних програмних систем на основі знань.

Введено поняття *одноядерних* та *багатоядерних* неоднорідних класів об'єктів. Експериментально підтверджено, що використання одноядерних та багатоядерних неоднорідних класів об'єктів для збереження інформації у об'єктно-орієнтованих реляційних базах даних зменшує розміри таких БД і їх експортованих *.sql-файлів та пришвидшує виконання SQL-запитів, у порівнянні з використанням однорідних класів об'єктів та однорідних класів об'єктів, зв'язаних одиничним успадкуванням. Визначено відношення еквівалентності, агрегації, узагальнення та спеціалізації для однорідних, одноядерних та багатоядерних неоднорідних класів об'єктів.

Проаналізовано відомі механізми успадкування та проблеми надлишковості,

винятків, неоднозначності та несумісності, що виникають при проектуванні та побудові концептуальних (класових) ієрархій з використанням успадкування. Розроблено механізми P, SgFSt, SgFW, SgPSt, SgPW, MFSt, MFW, MPSt, MPW успадкування та розширену і узагальнену класифікацію механізмів успадкування. Описано підходи до вирішення проблем надлишковості, винятків, неоднозначності та несумісності, що базуються на використанні концепції багатоядерних неоднорідних класів об'єктів та SgPSt, SgPW, MPSt, MPW механізмів успадкування.

Введено поняття експлуататорів та модифікаторів об'єктів та класів об'єктів. Визначено універсальні експлуататори об'єднання, однорідного перетину, неоднорідного перетину, мультиоб'єднання, різниці, симетричної різниці та клонування, що дозволяють будувати нові об'єкти, класи, множини та мультимножини об'єктів. Визначені поняття повних, часткових, породжуючих, знищуючих, замінних і комбінованих модифікаторів, що дозволяють певним чином модифікувати структуру об'єктів та класів об'єктів. Визначені відношення модифікації об'єктів та класів об'єктів.

Також введено поняття множини та мультимножини об'єктів, класів множин та мультимножин об'єктів, а також генераторів (конструкторів) множин та мультимножин об'єктів. Визначені 3 універсальних конструктори множин та 7 універсальних конструкторів мультимножин об'єктів, а також CP, RCL, PS, D2 конструктори мультимножин об'єктів. Сформульовано та доведено теореми про потужність мультимножин об'єктів, побудованих за допомогою CP, RCL, PS, D2 конструкторів, які дозволяють завжди обчислити точну кратність кожного елементу мультимножини об'єктів та її потужність ще до генерації мультимножини.

Розроблено нову об'єктно-орієнтовану динамічну модель подання зна-нь "Об'єктно-орієнтовані динамічні мережі" (ООДМ) для проектування та розробки інтелектуальних програмних систем на основі знань.

Для запропонованої моделі подання знань досліджено та доведено ряд корисних властивостей, зокрема: розширення підмножини однорідних класів множини базових класів ООДМ за допомогою експлуататорів однорідного перетину і об'єднання є скінченними; у результаті розширення підмножини однорідних класів

множини базових класів ООДМ за допомогою експлуататора об'єднання (однорідного перетину) утворюється верхня (нижня) обмежена класова напівґратка; у результаті розширення підмножини однорідних класів множини базових класів ООДМ за допомогою експлуататорів об'єднання та однорідного перетину утворюється повна обмежена дистрибутивна модулярна класова ґратка з умовами обриву строго зростаючих та строго спадаючих ланцюгів.

Побудова обмежених напівґраток шляхом розширення підмножини однорідних класів множини базових класів ООДМ за допомогою універсальних експлуататорів об'єднання та однорідного перетину дозволяє видобувати нові знання з множини базових класів ООДМ у вигляді нових класів об'єктів та відношень часткового порядку між ними. Побудова верхньої обмеженої напівґратки шляхом розширення підмножини однорідних класів множини базових класів ООДМ за допомогою експлуататора об'єднання дає змогу проводити ефективну реструктуризацію баз даних та знань за рахунок збереження одиниці цієї напівґратки, яка є неоднорідним класом, замість множини базових однорідних класів об'єктів (атомів напівґратки).

З метою демонстрації практичного застосування усіх можливостей запропонованої моделі подання знань, в термінах моделі був описаний такий розділ геометрії, як опуклі чотирикутники.

Ключові слова: об'єктно-орієнтоване подання (представлення) знань, неоднорідні класи об'єктів, часткове успадкування, універсальні експлуататори об'єктів та класів, модифікатори об'єктів та класів, конструктори множин та мультимножин об'єктів, об'єктно-орієнтовані динамічні мережі, інтелектуальні програмні системи на основі знань.

ABSTRACT

Dmytro O. Terletskyi. Object-Oriented Dynamic Knowledge Representation Model within Intelligent Software Systems. – Qualification Scientific Work on the Rights of the Manuscript.

Thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science, Speciality 01.05.03 – Mathematical and Programming Software of Computing Machines and Systems (12 – Information Technologies). – Taras Shevchenko National University of Kyiv, V.M. Glushkov Institute of Cybernetics of NAS of Ukraine, Kyiv, 2018.

The main aim of the thesis is to develop object-oriented dynamic model for knowledge representation with extended possibilities of domains representation for knowledge-based systems design and development.

The thesis contains consideration and analysis of *knowledge* concept, *know-ledge representation* task, as well as object-oriented knowledge representation models, such as frame systems, scripts, dynamic memory, object-oriented class-based and prototype-based programming. In addition, well-known applications of knowledge representation models and main classes of knowledge-based systems are considered and analysed.

The concepts of *single-core* and *multi-core* inhomogeneous classes of objects are proposed. It is experimentally proved that usage of single-core and multi-core inhomogeneous classes of objects for storing the information within object-oriented relational databases reduces the sizes of such DBs and their exported *.sql-files and speeds up the execution of SQL-queries, in contrast to the use of homogeneous classes of objects and homogeneous classes of objects linked by single inheritance. The equivalence, aggregation, generalization and specialization relations for homogeneous as well as for single-core and multi-core inhomogeneous classes of objects are defined.

Well-known inheritance mechanisms and problems of redundancy, exceptions, ambiguity and incompatibility that appear during designing and constructing conceptual hierarchies using inheritance are analysed. The P, SgFSt, SgFW, SgPSt, SgPW, MFSt, MFW, MPSt, MPW inheritance mechanisms and extended and generalised classification of inheritance mechanisms are proposed. The approaches to solving the problems of redundancy, exceptions, ambiguity and incompatibility, using the proposed classification, which are based on the concept of multi-core in-

homogeneous classes of objects and SgPSt, SgPW, MPSt, and MPW inheritance mechanisms, are described.

The concepts of exploiters and modifiers of objects and classes of objects are proposed. Universal exploiters of union, homogeneous intersection, inhomogeneous intersection, multiunion, difference, symmetric difference and cloning, which allow constructing new objects, classes, sets and multisets of objects, as well as concepts of full, partial, generative, destructive, replaceable and combined modifiers, which allow modifying structure of objects and classes of objects, are defined. The relations of objects and classes of objects modification are determined.

In addition, the concepts of a set and a multiset of objects, classes of sets and multisets of objects, as well as generators (constructors) of sets and multiplicands of objects are proposed. Three universal constructors of sets and seven universal constructors of multisets of objects, as well as CP, RCL, PS, D2 constructors of multisets of objects are determined. Theorems about cardinality of multisets of objects, constructed using CP, RCL, PS, D2 constructors, which always allow calculating exact multiplicity of each element of the multiset of objects and its cardinality before constructing the multiset, are formulated and proved.

The new object-oriented dynamic knowledge representation model called "Object-oriented dynamic networks" (OODN) is proposed for knowledge-based systems design and development.

The following useful properties for the proposed knowledge representation model have been investigated and proved: extensions of the subset of homogeneous classes of the set of OODN basic classes using exploiters of union and homogeneous intersection are finite; the result of the extension of the subset of homogeneous classes of the set of OODN basic classes using exploiter of union (homogeneous intersection) is the formation of upper (lower) bounded semilattice; the result of the extension of the subset of homogeneous classes of the set of OODN basic classes using both exploiters of union and homogeneous intersection is the formation of complete bounded distribution modular lattice with the conditions of the termination of severely ascending and descending chains.

The construction of finite bounded semilattices by extending the subset of homogeneous classes of the set of OODN basic classes using universal exploiters allows new knowledge extraction from the set of OODN basic classes in the form of new classes of objects and relations of partial order between them. The construction of finite join-semilattice by extending the set of OODN basic classes using exploiter of union allows an efficient databases and knowledge bases restructuring by storing the greatest upper bound of this semilattice, which is an inhomogeneous class, instead of the set of basic homogeneous classes of objects (atoms of the semilattice).

Such section of geometry as convex quadrilaterals is described in the terms of the proposed knowledge representation model to demonstrate practical usage of all model's possibilities.

Keywords: object-oriented knowledge representation, object-oriented dynamic networks, inhomogeneous classes of objects, partial inheritance, universal exploiters of objects and classes, modifiers of objects and classes, constructors of sets and multisets of objects, intelligent systems, knowledge-based systems.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

[1] Д.О. Терлецький, "Множини як сукупність сутностей-об'єктів", *Збірник наукових праць "Комп'ютерна математика"*, № 2, с. 64–71, 2013.

[2] Д.О. Терлецький, "Конструктори множин та мультимножин об'єктів", *Проблеми програмування*, том 16, № 1, с. 18–30, 2014.

[3] D. O. Terletskyi and O. I. Provotar, "Mathematical Foundations for Designing and Development of Intelligent Systems of Information Analysis", *Проблеми програмування*, том 16, № 2-3, с. 233–241, 2014.

[4] D. Terletskyi and A. Provotar, "Object-Oriented Dynamic Networks", in *Computational Models for Business and Engineering Domains,* 1st ed., ser. International Book Series Information Science & Computing, G. Setlak and K. Markov, Eds. ITHEA, 2014, vol. 30, pp. 123–136.

[5] D. Terletskyi, “Universal and Determined Constructors of Multisets of Objects”, *Information Theories & Applications*, vol. 21, no. 4, pp. 339–361, 2014.

[6] Д. А. Терлецкий, А. И. Провотар, “Нечеткие объектно-ориентированные динамические сети. I”, *Кибернетика и системный анализ*, том 51, № 1, с. 34–40, 2015.

[7] Д. А. Терлецкий, А. И. Провотар, “Нечеткие объектно-ориентированные динамические сети. II”, *Кибернетика и системный анализ*, том 52, № 1, с. 38–45, 2016.

[8] D. Terletskyi, “Inheritance in Object-Oriented Knowledge Representation”, in *Information and Software Technologies: Proceedings of 21st International Conference, ICIST 2015, Druskininkai, Lithuania, October 15-16, 2015*, ser. Communication in Computer and Information Science, G. Dregvaite and R. Domaševičius, Eds. Springer, 2015, vol. 538, pp. 293–305.

[9] D. Terletskyi, “Object-Oriented Knowledge Representation and Data Storage Using Inhomogeneous Classes”, in *Information and Software Technologies: Proceedings of 23rd International Conference, ICIST 2017, Druskininkai, Lithuania, October 12-14, 2017*, ser. Communication in Computer and Information Science, R. Domaševičius and V. Mikašytė, Eds. Springer, 2017, vol. 756, pp. 48–61.

[10] D. Terletskyi, “Exploiters-Based Knowledge Extraction in Object-Oriented Knowledge Representation”, in *Proceedings of the 24th International Workshop “Concurrency, Specification & Programming” Rzeszow, Poland, September 28-30*, Z. Suraj and L. Czaja, Eds. Rzeszow University, 2015, vol. 2, pp. 211–221.

[11] D.O. Terletskyi, “Object-Oriented Knowledge Extraction using Universal Exploiters”, in *Proceedings of 12th International Scientific and Technical Conference “Computer Science and Information Technologies” (CSIT’2017), Ukraine, Lviv, September 5-8*, 2017, pp. 257–266.

[12] D. Terletskyi, “Process of the Homogeneous Multisets Creation”, in *Proceedings of the 11th International Scientific Conference of Students and Young Scientists “Shevchenkivska vesna 2013”, Section Cybernetics, Kyiv, March 18-22*, 2013, pp. 37–39.

[13] Д.О. Терлецький, "Дослідження процесу утворення однорідних множин", *Матеріали IV Всеукраїнської науково-практичної конференції "Інформатика та системні науки", м. Полтава, 21-23 березня*, 2013, с. 278–281.

[14] Д.О. Терлецький, О.І. Провотар, "Дослідження процесу утворення неоднорідних мультимножин та їх класифікація", *Матеріали XI Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених "Теоретичні і прикладні проблеми фізики, математики та інформатики", м. Київ, 18-19 квітня*, 2013, с. 76–77.

[15] Д.О. Терлецький, "Дослідження процесу утворення та класифікації неоднорідних множин", *Матеріали IX Міжнародної наукової конференції "Інтелектуальні системи прийняття рішень і проблеми обчислювального інтелекту", м. Євпаторія, 20-24 травня*, 2013, с. 503–505.

[16] Д.О. Терлецький, "Операції над сутностями в інтелектуальному об'єктному середовищі", *Праці X Міжнародної науково-практичної конференції "Теоретичні і прикладні аспекти побудови програмних систем", м. Ялта, 26 травня - 2 червня*, 2013, с. 181–186.

[17] D.O. Terletskyi, "The System of Set Theory for Operating with Essences in the Objects Intellectual Environment", in *Proceedings of the 6th International Conference "Advanced Computer Systems and Networks: Design and Application", Lviv, September 16-18*, 2013, pp. 226–229.

[18] Д.О. Терлецький, "Декомпозиційна генерація мультимножин об'єктів", *Тези доповідей XI Міжнародної науково-практичної конференції "Математичне і програмне забезпечення інтелектуальних систем", м. Дніпропетровськ 20-22 листопада*, 2013, с. 238–239.

[19] D.O. Terletskyi, "Finitely-Generated Algebras and Constructors of Multisets of Objects", in *Proceedings of the 3rd International scientific conference of students and young scientists "Theoretical and Applied Aspects of Cybernetics", Kyiv, November 25-29*, 2013, pp. 48–63.

[20] О.І. Провотар, Д.О. Терлецький, "Конструктивна модифікація семантичних мереж", *Збірник тез доповідей VI Наукової конференції магістрантів та аспі-*

рантів "Прикладна математика та комп'ютинг", м. Київ, 16-18 квітня, 2014, с. 171–176.

[21] Д.О. Терлецький, О.І. Провотар, "Класові та об'єктні модифікатори", *Матеріали XII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених "Теоретичні і прикладні проблеми фізики, математики та інформатики", м. Київ, 24-25 квітня*, 2014, с. 85–87.

[22] Д.О. Терлецький, "Представлення знань за допомогою об'єктно-орієнтованих динамічних мереж", *Матеріали XIV Міжнародної наукової конференції ім. проф. Т.А. Таран "Інтелектуальний аналіз інформації", м. Київ, 14-16 травня*, 2014, с. 207–213.

[23] Д.О. Терлецький, "Комбіновані модифікатори класів та об'єктів", *Матеріали X Міжнародної наукової конференції "Інтелектуальні системи прийняття рішень і проблеми обчислювального інтелекту", м. Херсон, 28-31 травня*, 2014, с. 337–339.

[24] Д.О. Терлецький, "Модифікатори нечітких об'єктів та класів", *Матеріали III Міжнародної науково-практичної конференції "Обчислювальний інтелект (результати, проблеми, перспективи)", м. Черкаси, 12-15 травня*, 2015, с. 111–112.

[25] Д.О. Терлецький, "Комбіновані модифікатори нечітких об'єктів та класів", *Матеріали XIII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених "Теоретичні і прикладні проблеми фізики, математики та інформатики", м. Київ, 21-23 травня*, 2015, с. 89–92.

[26] Д.О. Терлецький, "Узагальнення концепції неоднорідних класів об'єктів", *Матеріали IV Міжнародної науково-практичної конференції "Обчислювальний інтелект (результати, проблеми, перспективи)", м. Київ, 16-18 травня*, 2017, с. 148–149.

[27] Д.О. Терлецький, "Узагальнення концепції неоднорідних класів нечітких об'єктів", *Матеріали XIII Міжнародної наукової конференції "Інтелектуальні системи прийняття рішень і проблеми обчислювального інтелекту", с. Залізний Порт, 22-26 травня*, 2017, с. 308–310.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

$p(A)$	–	властивість об'єкта A
$vf(A)$	–	функція верифікації якісної властивості $p(A)$
$f(A)$	–	метод об'єкта A
$P(A)$	–	специфікація об'єкта A
$F(A)$	–	сигнатура об'єкта A
$D(A)$	–	розмірність об'єкта A
$func(A)$	–	функціональність об'єкта A
$D(T)$	–	розмірність класу об'єктів T
$func(T)$	–	функціональність класу об'єктів T
$Core(T)$	–	ядро одноядерного неоднорідного класу об'єктів T
$Core_{i_m}^{m}(T)$	–	i_m-те ядро рівня m багатоядерного неоднорідного класу об'єктів T
$pr(t)$	–	проекція типу t неоднорідного класу об'єктів
$h(T)$	–	міра неоднорідності класу об'єктів T
$\cup$	–	експлуататор об'єднання
$\cap$	–	експлуататор однорідного перетину
$\Cap$	–	експлуататор неоднорідного перетину
$\setminus$	–	експлуататор різниці
$\div$	–	експлуататор симетричної різниці
$Clone_i()$	–	експлуататор клонування
$\nexists$	–	не існує
$Eq(a, b)$	–	функція еквівалентності аргументів a та b
МПЗ	–	модель(і) подання (представлення) знань
ООП	–	об'єктно-орієнтоване програмування
ООДМ	–	об'єктно-орієнтовані(а) динамічні(а) мережі(а)

ВСТУП

Актуальність теми дослідження. Дане дисертаційне дослідження присвячене розробці динамічної об'єктно-орієнтованої моделі подання знань для проектування та розробки інтелектуальних програмних систем на основі знань.

На сьогодні проектування та розробка інтелектуальних програмних систем є актуальною та важливою задачею, як в області штучного інтелекту, так і в області інженерії програмного забезпечення. Такі системи мають широке практичне застосування для вирішення різних задач інтелектуального пошуку, класифікації, діагностики, моніторингу, прогнозування, розпізнавання, планування, навчання, прийняття рішень тощо. Одним з ключових елементів багатьох інтелектуальних програмних систем є моделі подання знань, які дозволяють трансформувати знання експертів про ту чи іншу предметну область у деякий "машино-зрозумілий" формат, що дає змогу ефективно використовувати ці знання для вирішення конкретних практичних задач у цих доменах.

Створення таких систем дає змогу автоматизувати та підвищити ефективність інтелектуальних процесів у різних сферах людської діяльності. Однак розробка будь-якої інтелектуальної програмної системи на основі знань передбачає реалізацію певної моделі подання знань за допомогою деякої мови програмування. На сьогодні найбільш популярними є мови, що підтримують об'єктно-орієнтоване програмування, тому розробка та дослідження об'єктно-орієнтованих моделей подання знань є актуальною та практично обґрунтованою задачею.

Вагомий внесок у дослідження та розвиток об'єктно-орієнтованих моделей подання знань зробили такі вчені як M. Minsky, I. Graham, P. J. Ha-yes, P. L. Jones, P. D. Karp, B. J. Kuipers, R. J. Brachman, H. J. Levesque, R. C. Schank, R. P. Abelson, R. E. Cullingford, W. Lehnert, I. D. Craig, G. Booch, B. Meyer, M. Mezini, M. Abadi, L. Cardelli, B. Stroustrup, G. Blas-chek, D. S. Touretzky, R. Al-Asady, C. J. Date. Отримані ними результати стали основою багатьох інтелектуальних програмних

систем на основі знань та стимулювали активний розвиток об'єктно-орієнтованого підходу до подання знань.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконувалась відповідно до плану наукових досліджень кафедри інформаційних систем факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка в рамках держбюджетних тем "Створення теоретичних основ, методів та засобів інтелектуалізації інформаційно-комунікаційних технологій для розподілених комп'ютерних систем", номер держреєстру 0106U005860 (2010-2012 рр.) та "Теоретико-математичні методи дослідження і розробки інтелектуальних інформаційних систем в неформалізованих і слабко формалізованих предметних областях", номер держреєстру 0111U006938 (2013-2015 рр.), а також кафедральної наукової теми "Моделі та технології інтелектуальних обчислень" номер НДР 16КФ015-01 (2016-2018 рр.).

Мета і завдання дисертаційного дослідження. Метою дисертаційного дослідження є побудова об'єктно-орієнтованої динамічної моделі представлення знань з розширеними можливостями формалізації предметних областей для проектування та розробки інтелектуальних інформаційних систем на основі знань. Для досягнення поставленої мети у дисертації були розв'язані наступні задачі:

- аналіз об'єктно-орієнтованих моделей подання знань та інтелектуальних програмних систем на основі знань;
- розробка ефективної об'єктно-орієнтованої динамічної моделі подання знань;
- розробка ефективних підходів до проектування концептуальних (класових) ієрархій та мереж;
- розробка методів розширення та модифікації базових знань;
- розробка конструктивних методів побудови та класифікації абстрактних понять *множина* та *мультимножина об'єктів*;
- дослідження властивостей запропонованої моделі подання знань;
- демонстрація ефективності та практичного застосування запропонованої моделі подання знань.

Об'єкт дослідження – об'єктно-орієнтовані моделі представлення знань, інтелектуальні інформаційні системи на основі знань.

Предмет дослідження – представлення об'єктів, класів та відношень; механізми наслідування; побудова, аналіз та методи модифікації концептуальних ієрархій та мереж; побудова множин і мультимножин об'єктів та класів об'єктів; методи видобування, збереження та передачі знань; ефективність представлення знань; інтеграція з об'єктно-орієнтованими мовами програмування.

Методи дослідження. Для розв'язання поставлених задач використовуються методи об'єктно-орієнтованого програмування, об'єктно-орієнтованого аналізу та представлення знань, теорії множин та мультимножин, теорії нечітких множин, а також теорії ґраток та напівґраток.

Наукова новизна отриманих результатів. Основні результати дисертаційного дослідження полягають у розробці об'єктно-орієнтованої динамічної моделі подання знань для проектування та розробки інтелектуальних програмних систем на основі знань. В рамках дисертації автором отримано наступні результати:
вперше:

— введено поняття одноядерних та багатоядерних неоднорідних класів об'єктів; визначено відношення еквівалентності, агрегації, узагальнення та спеціалізації для однорідних, одноядерних та багатоядерних неоднорідних класів об'єктів;

— розроблено механізми P, SgFSt, SgFW, SgPSt, SgPW, MFSt, MFW, MPSt, MPW успадкування та розширену і узагальнену класифікацію механізмів успадкування;

— введено поняття експлуататорів об'єктів та класів об'єктів; визначено універсальні експлуататори об'єднання, однорідного перетину, неоднорідного перетину, мультиоб'єднання, різниці, симетричної різниці та клонування, що дозволяють будувати нові об'єкти, класи, множини та мультимножини об'єктів;

— введено поняття модифікаторів об'єктів та класів об'єктів; визначено концепції повних, часткових, породжуючих, знищуючих, замінних і комбіно-

ваних модифікаторів, що дозволяють модифікувати структуру об'єктів та класів об'єктів; визначено відношення модифікації об'єктів та класів об'єктів;

— побудовано універсальні конструктори множин і мультимножин об'єктів та CP, RCL, PS, D2 конструктори мультимножин об'єктів;

— розроблено точні методи обчислення кратностей елементів та потужностей мультимножин об'єктів згенерованих за допомогою CP, RCL, PS, D2 конструкторів;

— розроблено таку об'єктно-орієнтовану динамічну модель подання знань, як "Об'єктно-орієнтовані динамічні мережі" (ООДМ);

— доведено скінченність розширень підмножини однорідних класів множини базових[1] класів ООДМ за допомогою універсальних експлуататорів об'єднання та однорідного перетину;

— доведено що підмножина однорідних класів множини базових класів ООДМ, розширена за допомогою експлуататора об'єднання (однорідного перетину), разом з цим експлуататором утворює верхню (нижню) обмежену класову напівґратку;

— доведено що підмножина однорідних класів множини базових класів ООДМ, розширена за допомогою експлуататорів об'єднання та однорідного перетину, разом з цими експлуататорами утворює повну обмежену дистрибутивну модулярну класову ґратку з умовами обриву строго зростаючих та строго спадаючих ланцюгів;

— доведено що на основі найбільших верхніх граней верхніх обмежених класових напівґраток, побудованих шляхом розширення підмножини однорідних класів множини базових класів ООДМ за допомогою універсального експлуататора об'єднання, можна відновити усі елементи цих напівґраток;

удосконалено:

— механізми одиничного та множинного успадкування;

— підходи до вирішення проблем надлишковості, винятків, неоднозначності

[1]Маються на увазі класи об'єктів, які використовуються для утворення нових класів об'єктів.

та несумісності, що виникають у процесі побудови концептуальних ієрархій;

— методи побудови концептуальних ієрархій та мереж;

набули подальшого розвитку:

— поняття типу, класу, множини та мультимножини об'єктів;

— підходи до генерації класів об'єктів під час виконання програм;

— методи об'єктно-орієнтованого видобування знань;

— підходи до проектування об'єктно-орієнтованих баз даних та баз знань;

— підходи до реструктуризації об'єктно-орієнтованих баз даних та баз знань.

Теоретичне і практичне значення отриманих результатів.

— Використання *одноядерних* та *багатоядерних неоднорідних класів об'єктів* для збереження інформації у об'єктно-орієнтованих реляційних базах даних, зменшує розміри таких БД та їх експортованих *.sql-файлів, а також пришвидшує виконання SQL-запитів, у порівнянні з використанням однорідних класів об'єктів та однорідних класів об'єктів, зв'язаних одиничним успадкуванням.

— Введені поняття неоднорідних класів об'єктів та розроблена класифікація механізмів успадкування дозволяють більш ефективно і раціонально проектувати та будувати концептуальні ієрархії, уникаючи проблем надлишковості, винятків та неоднозначності.

— Визначені універсальні експлуататори об'єктів та класів об'єктів дозволяють динамічно розширювати ООДМ та є кроком до практичного розв'язання задачі генерації класів під час виконання програм (Runtime Class Generation).

— Визначені модифікатори об'єктів та класів об'єктів дозволяють динамічно моделювати зміни сутностей об'єктів з часом і тим самим оновлювати (актуалізовувати) ООДМ.

— Запропоновані конструктори множин (мультимножин) дозволяють конструктивно будувати множини (мультимножини) об'єктів та їх класи.

— Розроблені точні методи обчислення кратностей елементів та потужностей

мультимножин об'єктів згенерованих за допомогою CP, RCL, PS, D2 конструкторів, дозволяють оцінити розмір та деякі властивості мультимножин об'єктів ще до початку їх генерації.

— Побудова найбільшої верхньої грані верхньої напівґратки на основі підмножини однорідних класів множини базових класів ООДМ, розширеної за допомогою експлуататора об'єднання, дозволяє проводити ефективну реструктуризацію баз даних та баз знань за рахунок збереження одиниці цієї напівґратки, яка є неоднорідним класом об'єктів, замість збереження однорідних базових класів об'єктів (атомів напівґратки).

Особистий внесок здобувача. Усі наукові результати та положення, які складають основну частину дисертаційної роботи, отримані автором самостійно. Роботи [1–20] написані здобувачем особисто і всі наукові результати, що містяться в них, є оригінальними. Роботи [21–27] написані у співавторстві з науковим керівником. У цих роботах здобувачу належать наступні результати: [21] – запропоновані поняття експлуататорів об'єктів та класів об'єктів, визначені універсальні експлуататори об'єднання, перетину, різниці, симетричної різниці та клонування; [22] – запропонована об'єктно-орієнтована модель подання знань – об'єктно-орієнтовані динамічні мережі; [23] – визначені поняття нечітких об'єктів, однорідних та неоднорідних класів нечітких об'єктів; [24] – запропоноване узагальнення об'єктно-орієнтованих динамічних мереж на нечіткий випадок; [25] – запропоноване поняття універсального конструктора неоднорідних мультимножин об'єктів; [26] – описаний процес генерації нових класів об'єктів за допомогою універсальних експлуататорів; [27] – запропоновані поняття модифікаторів об'єктів та класів об'єктів.

Апробація результатів дослідження. Основні результати дисертаційного дослідження були представлені на наукових семінарах:

— Науковий семінар "Інтелектуальні інформаційні системи" кафедри інформаційних систем факультету кібернетики Київського національного університету імені Тараса Шевченка, 28 листопада 2017 року;

— Науковий семінар відділів методів та технологічних засобів побудови ін-

телектуальних програмних систем, теорії цифрових автоматів, інтелектуальних інформаційних технологій, автоматизації програмування, методів дискретної оптимізації, математичного моделювання та аналізу складних систем, методів індуктивного моделювання та керування Інституту кібернетики імені В.М. Глушкова НАН України, 12 грудня 2017 року;

та на міжнародних і всеукраїнських наукових конференціях:

- International Scientific Conference of Students and Young Scientists "Shevchenkivska Vesna: Cybernetics", 2013;
- Всеукраїнська науково-практична конференція "Інформатика та системні науки", 2013;
- Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених "Теоретичні і прикладні проблеми фізики, математики та інформатики", 2013/2014/2015;
- Міжнародна конференція "Інтелектуальні системи прийняття рішень і проблеми обчислювального інтелекту", 2013/2014/2017;
- Міжнародна науково-практична конференція "Теоретичні і прикладні аспекти побудови програмних систем", 2013;
- International Scientific and Technical Conference "Advanced Computer Systems and Networks: Design and Application", 2013;
- Міжнародна науково-практична конференція "Математичне і програмне забезпечення інтелектуальних систем", 2013;
- International Scientific Conference of Students and Young Scientists "Theoretical and Applied Aspects of Cybernetics", 2013;
- Наукова конференція магістрантів та аспірантів "Прикладна математика та комп'ютинг", 2014;
- Міжнародна наукова конференція імені професора Т.А. Таран "Інтелектуальний аналіз інформації", 2014;
- Міжнародна науково-практична конференція з програмування "УкрПРОГ", 2014;
- International Conference on "Intelligent Information and Engineering Systems",

Poland, 2014;

— Міжнародна науково-практична конференція “Обчислювальний інтелект (результати, проблеми, перспективи)”, 2015/2017;

— International Workshop on “Concurrency, Specification and Program-ming”, Poland, 2015;

— International Conference on “Information and Software Technologies”, Lithuania, 2015/2017;

— International Scientific and Technical Conference “Computer Science and Information Technologies”, 2017;

Публікації. Основні результати дисертаційного дослідження опубліковані у 27 наукових працях, серед яких 5 робіт у виданнях, що входять до переліку наукових фахових видань України [1, 2, 21, 23, 24]; 5 робіт у виданнях, що індексуються у міжнародній наукометричній базі Web of Science [4, 5, 20, 23, 24]; 7 робіт у виданнях, що індексуються у міжнародній наукометричній базі Scopus [4, 5, 17, 20, 21, 23, 24]; 2 роботи у виданнях, що індексуються у міжнародній наукометричній базі zbMATH [23, 24]; 4 роботи у виданнях, що індексуються у міжнародній бібліографічній базі DBLP [4, 5, 17, 20]; 1 робота у виданні, що індексується у міжнародній бібліографічній базі IEEE Xplore [20]; 2 роботи у виданнях іншої країни за напрямком дисертації [3, 22]; 16 робіт у матеріалах, працях та тезах доповідей міжнародних та всеукраїнських наукових конференцій [6–16, 18, 19, 25–27].

Структура та обсяг дисертації. Дисертація складається з анотації, переліку умовних позначень, вступу, трьох розділів, висновків, списку використаних джерел, який містить 140 найменувань та двох додатків. Загальний обсяг дисертації становить 200 сторінок, основний текст складає 168 сторінок.

РОЗДІЛ 1
ОБ'ЄКТНО-ОРІЄНТОВАНІ МОДЕЛІ ПОДАННЯ ЗНАНЬ ТА ІНТЕЛЕКТУАЛЬНІ ПРОГРАМНІ СИСТЕМИ

1.1. Подання знань

На сьогодні не існує єдиного чітко визначеного та загально прийнятого означення поняття *знання*, оскільки воно є дуже загальним та має велику кількість різних інтерпретацій. Однак, серед великої кількості існуючих означень цього поняття можна виділити декілька найбільш важливих для проектування інтелектуальних програмних систем на основі знань. Одне із таких означень було запропоноване Тревором Бенч-Капоном [28].

Означення 1.1.1. Знання – це набір синтаксичних та семантичних конвенцій, що дає можливість описувати речі.

Де під синтаксисом мається на увазі множина правил комбінування символів для утворення коректних виразів, а під семантикою – специфікація того, як повинні інтерпретуватися такі вирази.

Інше означення поняття *знання* було сформульоване Карлосом Раміресом та Бенджаміном Вальдесом [29].

Означення 1.1.2. Знання – це множина певним чином між собою асоційованих ментальних об'єктів та множин атрибутів, що утворюють стабільні з часом динамічні структури представлені за допомогою певних обчислювальних символів або шаблонів.

Ще одне важливе означення поняття *знання* було сформульоване Бернхардом Небелем [30].

Означення 1.1.3. Знання – це інформація про певну предметну область представлена за допомогою певної абстракції, яка дозволяє інтерпретувати цю інформацію.

Такими абстракціями є моделі подання знань, які дозволяють певним чином формалізувати знання про певну предметну область. Для того щоб програмні системи могли повноцінно використовувати у своїй роботі знання про ту чи іншу предметну область, необхідно щоб вони мали доступ до цих знань. У зв'язку з чим виникає потреба у поданні знань у певному "машино-зрозумілому" форматі [31]. Іншими словами, з'являється необхідність розробки ефективних моделей подання знань, які б дозволяли зберігати знання в пам'яті машини таким чином, щоб програми мали змогу їх "осмислено" обробляти та використовувати для вирішення поставлених перед ними завдань. Представлення знань у так званому "машино-зрозумілому" форматі є певною реконструкцією, перекладом або трансляцією знань експертів про ту чи іншу предметну область на мову певної моделі подання знань [30]. У зв'язку з чим означення *моделі подання (представлення) знань* можна сформулювати наступним чином.

Означення 1.1.4. Модель подання (представлення) знань – це формалізм, який дозволяє певним чином подавати знання про деяку предметну область у деякому "машино-зрозумілому" форматі.

На сьогодні існує багато різних моделей подання знань, кожну з яких можна віднести до певної категорії [30, 32, 33]. Виділяють наступні категорії моделей подання знань: **логічні** (пропозиційна логіка, логіка предикатів першого порядку, дескрипційні логіки, нечіткі логіки, нечіткі дескрипційні логіки), **процедурні** (продукційні системи, нечіткі продукційні системи, скрипти), **мережеві** (семантичні мережі, концептуальні графи, нечіткі концептуальні графи), **структурні** (системи фреймів, системи нечітких фреймів, об'єктно-орієнтоване програмування на основі класів та на основі прототипів[1]), **гібридні** (онтології, нечіткі онтології).

Враховуючи специфіку процесу представлення знань про ту чи іншу предметну область, виділяють наступні характеристики моделей подання знань [28, 34]:

1. **Метафізична адекватність** – не повинно існувати жодних протиріч між

[1]Мова йде про можливості для формалізації знань про деяку предметну область в термінах об'єктно-орієнтованого програмування.

знаннями, які потрібно представити за допомогою певної моделі та їхнім представленням в термінах цієї моделі.

2. **Епістемічна адекватність** – модель повинна дозволяти представлення необхідних типів знань.
3. **Евристична адекватність** – модель повинна надавати можливості для представлення міркувань у ході вирішення проблем.
4. **Обчислювальна гнучкість** – модель повинна надавати можливості для ефективного маніпулювання представленням знань в рамках комп'ютерної системи.
5. **Однозначність** – будь-яке знання представлене в термінах моделі має бути однозначним, тобто мати лише одну інтерпретацію.
6. **Ясність** – представлені в термінах моделі знання повинні бути зрозумілими для усіх, навіть для тих хто не знайомий з цією моделлю.
7. **Однорідність** – усі знання одного типу повинні мати однаковий тип представлення в термінах моделі.
8. **Зручність нотації** – модель повинна надавати можливості зручного представлення різних типів знань.
9. **Релевантність** – подання знань в термінах моделі не повинно ускладнювати розв'язки задач, які потребують представлення цих знань в термінах певної моделі.
10. **Модулярність** – модель повинна мати механізми додавання, модифікації та видалення окремих знань.

Різні моделі подання знань орієнтовані на представлення різних типів знань та інформації, що обумовлює ефективність їхнього застосування. Виділяють наступні типи знань [34, 35]:

1. **Об'єкти** – основні структурні елементи предметної області.
2. **Класи об'єктів** – категорії об'єктів, що визначають їх специфіку.
3. **Відношення** – взаємозв'язки між різними елементами та типами знань.
4. **Сценарії** – послідовність певних подій з метою досягнення певних цілей.
5. **Уміння** – стереотипні послідовності дій для досягнення певних цілей (ал-

горитми, рецепти, традиції).

6. **Метазнання** – знання про певний тип знань.

Глобально знання можна поділити на *декларативні* та *процедурні* [29, 33–36]. Декларативні знання – це факти, які певним чином описують та визначають структуру сутностей предметної області, у той час, як процедурні знання – це методи, інструкції, рецепти, алгоритми, тощо, які можна до них застосовувати.

Практичне застосування будь-якої моделі подання знань зводиться до реалізації цієї моделі, за допомогою деякої мови програмування, в рамках певної програмної системи. Кожна модель подання знань має свою власну специфіку, яка обумовлює складність її реалізації за допомогою тої чи іншої мови програмування.

Найпопулярнішими на сьогодні є мови, що підтримують об'єктно-орієнтоване програмування, тому розробка та дослідження об'єктно-орієнтованих моделей подання знань є актуальною та практично обґрунтованою задачею. Це обумовлено тим, що об'єктно-орієнтовані моделі подання знань значно простіше реалізувати за допомогою мов що підтримують відповідну парадигму програмування, оскільки абстракція моделі та мови її реалізації є дуже схожими. У зв'язку з цим, розглянемо такі відомі об'єктно-орієнтовані моделі подання знань, як системи фреймів, скрипти та динамічну пам'ять, об'єктно-орієнтоване програмування на основі класів та на основі прототипів.

1.2. Системи фреймів

Фрейми, як структурована (об'єктно-орієнтована) статична модель для представлення стереотипних ситуацій, була запропонована М. Мінським в 1975 році [37]. Найпростішим прикладом фрейму є кадр фільму (картинка у певний момент часу). Таким чином фрейми можна розглядати як певні пакети взаємопов'язаних знань про деяку частину світу, у певний момент часу. Однак існують ситуації, що можуть повторно проявлятися в різних місцях та в різний час. Такі ситуації є стереотипними, тобто множину таких ситуацій можна описати за допомогою одного найбільш загального шаблону, для якого кожна конкретна (можлива) си-

туація буде просто прикладом (варіацією). Такий підхід дозволяє представляти певну множину можливих варіацій однієї ситуації, використовуючи єдине загальне представлення.

Фреймова модель представлення знань має два рівня абстракції, де нижній рівень відповідає за представлення конкретних об'єктів та класів [38–40], а верхній – за представлення певних концептів, ситуацій, які утворюють об'єкти та класи описані на нижньому рівні. У зв'язку з цим фреймова модель має мережево-ієрархічну структуру, що у певному розумінні є більш розвинутою версією семантичних мереж. Формальне визначення фрейму виглядає наступним чином.

Означення 1.2.1. Фрейм – це множина пар $F = \{(s_1, f_1), \ldots, (s_n, f_n)\}$, що описує певний об'єкт або клас F, де s_i – це i-й слот (термінал) фрейму, $i = \overline{1, n}$, а f_i – його заповнювач.

В якості заповнювачів слотів фреймів можуть виступати: ім'я (ідентифікатор) фрейму, відношення фрейму з іншими фреймами, атрибути фрейму та їх значення, атрибути фрейму та їх значення по замовчуванню, атрибути фрейму без значень, процедурні вкладення.

Відношення. Виділяють три основних типи відношень між фреймами: узагальнення, агрегація та асоціація. Узагальнення – це таке відношення між класом та суперкласом або між об'єктом та класом, коли клас є прикладом суперкласу або об'єкт є прикладом свого класу. Цей тип відношень може позначатися як *is-a*, *an-instance-of*, *a-kinf-of*. Агрегація – це таке відношення між класами і суперкласом, коли класи є частиною суперкласу. За зазвичай цей тип відношень може позначатися як *a-part-of*, *part-whole*. Асоціація – це такі відношення, які не належать до двох попередніх типів, наприклад *have*, *can*, *owl*, тощо.

Атрибути. Атрибути (властивості) фреймів можуть приймати одиничні, комбіновані або діапазонні значення. Одиничні значення використовуються тоді коли потрібно описувати атрибути, які можуть приймати лише чітко визначені типи значень. Такими типами можуть бути лише символьний, чисельний та логічний. Одиничні значення атрибутів можуть використовуватися для опису властивостей

класів та об'єктів.

Комбіновані значення використовуються для опису атрибутів, які можуть приймати різні одиничні значення з певної комбінації можливих значень. Такі ситуації зазвичай виникають під час успадкування, коли тип значення або саме значення певного атрибуту чи атрибутів можуть бути різними в батьківському класі та класі, що його наслідує. Комбіновані значення атрибутів використовуються для опису класів, під час успадкування, та опису різних об'єктів в рамках одного класу.

Діапазонні значення використовуються для опису атрибутів значення яких може коливатися в певних рамках. Також вони використовуються для обмеження можливих значень атрибутів, що дозволяє реалізувати часткову верифікацію введених значень. Такий підхід дуже часто використовується в експертних системах, що побудовані на основі фреймової моделі представлення знань [40].

Процедурні вкладення. Процедурні вкладення або просто процедури – це певні набори інструкцій, що можуть бути викликані та застосовані до конкретного фрейму [39–42] в залежності від потреби або ситуації. Виділяють два типи процедур: процедури що викликаються або можуть бути викликані при певній потребі (читання, запис (модифікація) слотів, створення екземплярів (конструктор екземплярів) класу) та процедури що викликаються автоматично в залежності від зміни значень слотів чи виникнення певних ситуацій (читання та запис (модифікація) слотів).

Як було зазначено вище, одиничні фрейми, що описують певні об'єкти або класи, разом утворюють систему фреймів, яку формально можна визначити наступним чином.

Означення 1.2.2. Система фреймів – це зв'язний орієнтований граф $S = (Nod, Edg)$, де $Nod = \{F_1, \ldots, F_n\}$ – це множина фреймів, що описують об'єкти або класи $F_1, \ldots, F_n$, а $Edg = \{R_1, \ldots, R_m\}$ – це множина ребер, що представляють відношення між ними.

Успадкування. Аналогічно до семантичних мереж, при побудові фреймових систем використовується механізм успадкування [39, 40, 42–44]. Тобто фрейми,

що є складовими частинами системи фреймів, можуть успадковувати деякі властивості один одного в залежності від позиції в ієрархії.

З розвитком фреймового підходу до подання знань, було розроблено багато відповідних систем, основними сімействами яких є Unit Package, FRL, SRL та KL-ONE. Існують і окремі системи, що не відносять до якогось конкретного сімейства. В загальному було розроблено понад 50 різних систем представлення знань на основі фреймів [45]. У всіх цих системах за рахунок використання механізму одиничного успадкування виникає проблема винятків, коли деякі властивості суперкласів не є характерними для усіх їхніх підкласів [46, 47].

Множинне успадкування. Для побудови фреймових систем також використовується механізм множинного успадкування, використання якого може призводити до проблеми неоднозначності та семантичного конфлікту, в яких об'єкт або клас успадковує атрибути з однаковими іменами або взаємовиключні (несумісні) властивості одночасно від кількох суперкласів [39, 40, 42, 45–47].

Висновки. Фреймова модель, як і інші моделі подання знань, має свої переваги та недоліки.

Переваги та можливості:

— можливість представлення складних об'єктів у вигляді одного фрейму [40, 42, 43];

— можливість компактного представлення великої кількості об'єктів та класів за рахунок побудови ієрархії з використанням механізму успадкування [40, 44, 48];

— забезпечення природного способу представлення стереотипних об'єктів та класів [40, 43];

— забезпечення встановлення значень атрибутів по замовчуванню [42, 44, 49];

— можливість параметричного логічного виведення, за допомогою зміни значень за замовчуванням [44];

— поєднання декларативних та процедурних знань в одній моделі представлення знань [34, 44];

— діапазонні обмеження на значення атрибутів фреймів [44];

— виклик автоматичних процедур в залежності від зміни значень слотів чи виникнення певних ситуацій [40, 42, 44];

Недоліки та обмеження:

— проблема ідентифікації об'єктів та класів при використанні та перевизначенні значень по-замовчуванню властивостей фреймів [48, 50];

— проблема винятків при організації ієрархічної структури системи фреймів з використанням успадкування [35, 39, 40, 42, 45];

— проблема логічного (семантичного) конфлікту властивостей під час множинного успадкування [35, 39, 40, 42, 45].

Не зважаючи на свої недоліки та обмеження, системи фреймів застосовуються для побудови систем: класифікації, планування, моніторингу баз даних, розпізнавання образів та різного роду експертних систем [38, 40].

1.3. Скрипти та динамічна пам'ять

Скрипти (сценарії), як структурована модель для опису послідовностей стереотипних подій у певному контексті, була запропонована Р. К. Шанком та Р. П. Абельсоном у 1975 році [51, 52]. Прикладами стереотипних послідовностей подій можуть бути певні історії, традиції, обряди та шаблони поведінки в певних життєвих ситуаціях. Класичним прикладом є скрипт, що описує похід у ресторан. Одним з головних призначень скриптів було їх використання для організації бази знань у системах розуміння природної мови, у контексті ситуації, яку повинна зрозуміти система [43, 53]. Загальне визначення скрипта виглядає наступним чином:

Означення 1.3.1. Скрипт (сценарій) це кортеж виду (T, P, R, C, Res, S), де T – це специфічна варіація на найбільш загальний шаблон, яка представлена скриптом, P – це множина реквізитів або предметів, які є важливими складовими частинами середовища в якому виконується скрипт, R – це множина ролей або дій які виконують окремі учасники (актори), C – це множина вхідних умов, які повинні виконуватися для того щоб скрипт був викликаний, Res – це множина

результатів – фактів які мають місце після завершення виконання скрипта, S – це множина сцен, де кожна сцена є певною частиною виконання скрипта.

Виконання будь-якого скрипта можна розглядати в різних контекстах, наприклад скрипт який описує похід у ресторан можна розглянути, як спосіб втамування голоду, як варіант проведення вечора, як спосіб зустрічі з друзями, тощо [51]. У зв'язку з чим скрипт повинен викликатися лише в потрібному контексті, в залежності від ситуації [53].

Запуск скрипта на виконання залежить від його заголовків. Виділяють наступні типи заголовків: PH – множина базових умов, які є необхідними для виконання скрипта, IH – можливі контексти в яких може бути виконаний скрипт, LH – множина можливих локалей (місць) де скрипт може бути виконаний, ICH – множина ролей які є частиною скрипта. Кожен тип заголовків визначає міру з якою асоційований зі скриптом контекст може мати місце.

Майже одразу після публікації перших робіт по теорії скриптів, Р. Шанк та його колеги створили систему SAM (система виконання скриптів) для розуміння історій, що побудовані на основі сценаріїв [51, 52, 54, 55]. Ця система була розроблена на снові скриптів та фреймів, оскільки скрипти дозволяють описувати послідовності стереотипних подій в певному контексті, а фрейми стереотипні ситуації, або іншими слова інфраструктуру середовища де виконується той чи інший скрипт. Таке поєднання зробило скрипти більш "зрозумілими", оскільки середовище де вони виконувалися було краще описане за рахунок використання фреймів, які містили додаткову інформацію, яка уточнювала контекст скрипта та розширювала можливості логічного виведення нових знань.

Іноді трапляються нестандартні ситуації, зокрема коли скрипт містить дії, які не мають зв'язку з попередніми діями і можуть "виводити" актора за межі поточного скрипта, у певних випадках, починаючи виконання інших скриптів [53]. У зв'язку з чим виділяють наступні типи подій:

— *втручання* – стани або дії які перешкоджають нормальному виконанню скрипта:

- *перешкоди* – відсутність умов для майбутніх дій,

- *помилки* – завершення дії з неочікуваними та недоречними результатами;

— *відволікання* – неочікувані стани або дії, які ініціюють нові цілі для актора, тимчасово або перманентно виводячи його із поточного скрипта.

Коли актор стикається з перешкодами, він може виконувати певні корегуючі дії спрямовані на створення відсутніх умов для виконання майбутніх дій. Після певної кількості невдалих корегувань, актор мусить покинути поточну сцену. У випадках коли актор стикається із помилками, він може виконувати корегуючі дії у вигляді циклічних повторень певних дій з метою примусового коректного їх завершення.

Різні скрипти можуть взаємодіяти між собою в залежності від відношень між ними. Це дозволяє будувати певні ієрархії скриптів [53, 54, 56]. Виділяють два основних типи відношень між скриптами: *послідовність* (коли скрипти починають та завершують своє виконання один після одного) та *одночасність* (коли скрипти починають свою роботу одночасно і при цьому можуть взаємодіючи або не взаємодіючи між собою). Таким чином скрипти можуть виконуватися послідовно, тобто кожен скрипт використовує результати роботи попереднього та створює передумови для роботи наступного за ним скрипта. Іншими словами, завершення якогось одного скрипта запускає на виконання інший, наступний за ним, скрипт. Також скрипти можуть виконуватися одночасно (паралельно), при цьому деякі скрипти можуть впливати на виконання інших скрипітв, що виконуються в поточний момент часу. Це може призводити до неочікуваних або нестандартних ситуацій (втручань та відволікань).

В залежності від специфіки скриптів, виділяють наступні типи скриптів [53, 54]: *ситуаційні* (для управління ситуаціями та подіями), *персональні* (для представлення власних цілей та планів) та *інструментальні* (для використання та представлення процедурних знань). У виконанні ситуаційних скриптів можуть брати участь багато акторів, в той час як інструментальні скрипти виконуються лише одним актором.

Теорія динамічної пам'яті. Оскільки скрипти описують події в межах кон-

кретної ситуації та містять невеликі порції досвіду в рамках однієї фізичної сцени, Р. Шанк запропонував теорію динамічної пам'яті, яка є розширеною модифікацією теорії скриптів [57–59].

У початковій версії теорії скриптів скрипти є упорядкованими послідовностями сцен [51–53], де поняття сцени визначається наступним чином.

Означення 1.3.2. Сцена – це структурована група дій, що відбуваються в один і той же час, спрямованих на досягнення спільної мети.

В теорії динамічної пам'яті Р. Шанк вводить поняття спогадів (невеликих частин пам'яті), які зберігаються в рамках сцен, що формують певну ситуацію. Виділяють наступні типи сцен: *фізичні* (описують як відбуваються певні події), *соціальні* (описують соціальні відносини між акторами для досягнення певної мети) та *персональні* (описують власні плани акторів для досягнення певних цілей).

Співвідношення між сценами і скриптами, в рамках теорії динамічної пам'яті, можна сформулювати наступним чином.

Означення 1.3.3. Скрипт – це конкретна реалізація деяких узагальнень в рамках конкретної сцени.

Наприклад сцена "вечеря в ресторані" може бути реалізована за допомогою певної кількості специфічних скриптів, таких як "вечеря у французькому ресторані", "вечеря в мексиканському ресторані", "романтична вечеря для двох", "сімейна вечеря в ресторані" тощо. Оскільки кожен скрипт визначає певний контекст, який визначає семантику дії, то будь-яка сцена може отримувати різні контексти (уточнення) за рахунок вибору відповідного скрипта для її реалізації.

Набори сцен та скриптів формують так звані пакети організації пам'яті, які визначаються наступним чином.

Означення 1.3.4. Пакет організації пам'яті (MOP) – це впорядкована група сцен, спрямованих на досягнення певної мети.

Пакети організації пам'яті містять інформацію про зв'язки між різними сценами та організовані навколо певних об'єктів та їх ролей в середовищі.

Існує певна різниця між сценами в рамках пакетів організації пам'яті та сце-

нами в рамках скриптів, вона полягає у тому, що у випадку пакетів сцени – це структури, які можуть бути доступними для інших пакетів, у той час як сцени в рамках скриптів є спеціалізованими для кожного конкретного скрипта і не можуть бути доступними за його межами [57, 58].

Пакети організації пам'яті мають типи, які визначаються типами сцен, які до них входять. Однак існують пакети, які сформовані із сцен, що не належать до жодного з вище зазначених типів сцен. Такі сцени і пакети є абстрактними шаблонами, на основі яких можна визначати конкретні сцени та пакети. У зв'язку з цим Р. Шанк ввів поняття мета-пакету організації пам'яті.

Означення 1.3.5. Мета-пакет організації пам'яті (Meta-MOP) – це впорядкована група узагальнених сцен, спрямованих на досягнення певної узагальненої мети.

Не зважаючи на загальність пакетів та мета-пакетів організації пам'яті, у порівнянні із сценами та скриптами, вони також є контекстно-обмеженими. У зв'язку з цим Р. Шанк запропонував поняття пакету (точки) тематичної організації, яке дозволяє створювати нові структури для опису інформації, яка не є залежною від конкретної предметної області [57, 58].

Означення 1.3.6. Пакет (точка) тематичної організації (TOP) – це високорівнева структура знань, що не залежить від конкретної предметної області і описує подібність між ситуаціями в різних предметних областях.

Іншими словами, пакети тематичної організації описують аналогії між досвідом у різних предметних областях, групуючи абсолютно різні спогади в термінах спільної але абстрактної мети. Пакет тематичної організації пам'яті складається з типу мети, планів та умов, однакових низькорівневих особливостей. Перші два елементи формують основу пакету, а третій – виступає внутрішнім індексом. Оскільки пакети тематичної організації є досить абстрактними, то низькорівневі особливості або індекси виступають деякими асоціаціями, які зв'язують пакети тематичної організації з коректними пакетами організації пам'яті.

Логічне виведення. Теорія динамічної пам'яті дозволяє описувати послідов-

ності стереотипних подій, які можна представити у вигляді певної послідовності сцен та скриптів. В свою чергу, кожну сцену та скрипт можна розглядати, як певну множину фактів, що дозволяє проводити логічний аналіз та отримувати нову, неявну інформацію шляхом пошуку відповідей на питання. Р. Шанк вважав, що будь-яка інформаційна система на основі динамічної пам'яті повинна розуміти ситуацію та інформацію, які вона опрацьовує, де розуміння трактується як формування очікувань, виведення неявної інформації та модифікація пам'яті (навчання) [57].

Висновки. Теорія скриптів та динамічної пам'яті, як і інші моделі подання знань, має свої переваги та недоліки.

Переваги та можливості:

— можливість опису послідовностей стереотипних подій [51, 53, 57];
— можливість опису поведінкових шаблонів стереотипних ситуацій [54];
— можливість прогнозування майбутніх подій в рамках певної множини скриптів [54];
— можливість складання планів на основі скриптів [53, 56];
— можливості порівняння та пошуку аналогій між сценаріями з різних предметних областей [57, 58].

Недоліки та обмеження:

— проблема вибору необхідного скрипта, MOP, Meta-MOP та TOP [54, 55, 57, 58];
— проблема часткової відповідності скриптів до реальних сценаріїв [54];
— проблема можливих змін стереотипних сценаріїв [51, 52, 54];
— проблема деталізації сценаріїв [54, 60, 61];
— проблема розпізнавання подібних сцен, що є частинами різних скриптів [61];
— проблема побудови причинно-наслідкового ланцюга [54, 55];
— проблема розпізнавання різних контекстів скриптів [60];
— проблема генерації нових скриптів, MOP та TOP на основі досвіду [57, 58, 60];

— проблема обробки втручань та перешкод під час виконання скриптів [53];
— проблема семантичної неоднозначності деяких скриптів [53];
— проблема розпізнавання кінця та початку виконання скриптів що виконуються паралельно [53];
— проблема одночасного виконання та взаємодії декількох скриптів [53, 54, 60].

Не зважаючи на свої недоліки та обмеження, системи на основі теорії скриптів та динамічної пам'яті мають застосування в таких сферах, як планування [51, 53, 56, 62], обробка природної мови [52, 54–56, 60, 61], клінічна психологія [53, 63–65], навчання (за допомогою комп'ютера) [58, 66], енергозбереження [67].

1.4. Об'єктно-орієнтоване програмування

Парадигма об'єктно-орієнтованого програмування (ООП) прийшла на зміну процедурному програмуванню, коли об'єми кодів програмних систем почали стрімко зростати, що значно ускладнило оптимізацію, верифікацію, тестування та налагодження таких систем [68]. На відміну від процедурного підходу, де основою були підпрограми, функції, процедури, записи, мітки та переходи між частинами коду, основу ООП складають концепції *об'єктів*, *класів*, *інкапсуляції*, *поліморфізму* та *успадкування*.

ООП є не лише парадигмою програмування, а й моделлю подання знань, оскільки більшість програм написаних в об'єктно-орієнтованому стилі, передбачають опис класових ієрархій, ініціалізацію та маніпулювання об'єктами, що є нічим іншим, як моделюванням предметних областей. Це дозволяє розглядати ООП на рівні з іншими об'єктно-орієнтованими моделями подання знань [69].

На сьогодні існує два основних підходи до ООП – *ООП на основі класів* та *ООП на основі прототипів* [68, 70–72].

1.4.1. Об'єктно-орієнтоване програмування на основі класів. Найперші об'єктно-орієнтовані мови програмування Simula67 та Smalltalk базувалися на концепціях *об'єктів* та *класів* [68, 70, 73, 74].

Об'єкти та класи. Будь-яку предметну область можна охарактеризувати за допомогою певних концептів та сутностей, які є її складовими частинами. В рамках ООП на основі класів концепти визначаються за допомогою класів, у той час як екземпляри кожного з концептів моделюються об'єктами відповідних класів [68, 74]. Розглянемо концепції об'єктів та класів більш детально.

Очевидно, що будь-який об'єкт має певний набір характеристичних властивостей, які визначають його як деяку сутність. Формально довільну властивість об'єкта можна представити наступним чином.

Означення 1.4.1. Властивість (поле або атрибут) $p(A)$ об'єкта A – це кортеж виду $p(A) = (t(p(A)), v(p(A)))$, де $t(p(A))$ – це тип властивості $p(A)$, а $v(p(A))$ – це її значення.

Оскільки зазвичай об'єктам характерна певна кількість властивостей, – вводять поняття специфікації.

Означення 1.4.2. Специфікація $P(A)$ довільного об'єкта A – це множина виду $P(A) = \{p_1(A), \ldots, p_n(A)\}$, де $p_i(A) = (t(p_i(A)), v(p_i(A)))$ – це i-та властивість об'єкта A, де $i = \overline{1, n}$.

На основі специфікації визначають поняття об'єкта.

Означення 1.4.3. Об'єкт – це кортеж виду $A = (P(A))$, де A – це ім'я (ідентифікатор) об'єкта, а $P(A) = \{p_1(A), \ldots, p_n(A)\}$ – це специфікація, що описує його природу.

Властивості об'єктів дозволяють представляти деяку інформацію та знання про об'єкти у термінах певної об'єктно-орієнтованої мови програмування, що дозволяє програмній системі певним чином оперувати об'єктами. Одним із інструментів оперування об'єктами є методи (функції).

Означення 1.4.4. Метод (функція) $f(A)$ довільного об'єкта A – це кортеж виду $f(A) = (t(f(A)), Args)$, де $t(f(A))$ – це тип результату, який повертає метод у результаті свого виконання, а $Args = \{(t(X_1), X_1), \ldots, (t(X_n), X_n)\}$ – це множина вхідних аргументів, де $t(X_i)$ – це тип i-того аргументу, а X_i – це його значення, де $i = \overline{1, n}$.

Методи є певними підпрограмами (функціями), аргументами яких виступають певні властивості об'єктів або самі об'єкти. За допомогою методів можна отримувати та модифікувати значення властивостей об'єктів. Оскільки для об'єктів можна визначити певну кількість різних методів, – вводять поняття сигнатури.

Означення 1.4.5. Сигнатура $F(A)$ довільного об'єкта A – це множина виду $F(A) = \{f_1(A), \ldots, f_m(A)\}$, де $f_i(A) = (t(f_i(A)), Args_i)$ – це i-тий метод об'єкта A, де $i = \overline{1, m}$.

Оскільки певна кількість різних об'єктів може мати однакові (еквівалентні) властивості, у зв'язку з чим до них можна застосувати одні і ті ж методи, вводять поняття класу об'єктів. Однак, на сьогодні в рамках ООП на основі класів це поняття має багато різних інтерпретацій [68, 71, 74, 75].

Означення 1.4.6. Клас – це множина об'єктів; програмний модуль; шаблон для створення об'єктів (фабрика об'єктів); абстрактний тип даних.

Формально клас можна визначити наступним чином.

Означення 1.4.7. Клас – це кортеж виду $T = (P(T), F(T))$, де T – ім'я класу, $P(T) = \{p_1(T), \ldots, p_n(T)\}$ – його специфікація, що визначає його природу, а $F(T) = \{f_1(T), \ldots, f_m(T)\}$ – сигнатура, що визначає його поведінку.

Основна ідея ООП на основі класів полягає у створенні та маніпулюванні конкретними об'єктами класів замість маніпулювання самими класами [68, 71, 74]. Таким чином спочатку визначаються та описуються класи і відношення між ними (формування концептуальних ієрархій), а в ході виконання програм, в залежності від їх логіки, створюється певна кількість об'єктів кожного з описаних класів, над якими і проводяться необхідні маніпуляції. Маніпулювання об'єктами відбувається через доступ до їхніх властивостей та методів. Виділяють три основних типи доступу: *відкритий* (властивості та(або) методи доступні для усіх об'єктів будь-якого класу), *захищений* (властивості та(або) методи доступні лише для об'єктів даного класу та об'єктів усіх класів нащадків) та *закритий* (властивості та(або) методи доступні лише в рамках методів даного класу).

Існують випадки коли властивості та(або) методи класу є спільними для усіх

його об'єктів. У деяких мовах, що реалізують ООП на основі класів, їх називають статичними. Такі властивості визначаються лише один раз при визначенні класу і використовуються усіма його об'єктами. Таким чином довільні зміни значень таких властивостей призведуть до зміни значення властивості одночасно для усіх об'єктів класу. Також до статичних властивостей та методів класу можна здійснювати доступ безпосередньо через інтерфейс класу навіть без створення об'єктів.

Деякі мови що підтримують ООП на основі класів надають можливості визначення незмінних властивостей та методів класів. Таким чином значення таких властивостей залишається незмінним протягом усього процесу виконання програми, а методи не можливо перевизначити в рамках класів нащадків.

Інкапсуляція. Оскільки класи можуть містити атрибути та методи, доступ до яких здійснюється через інтерфейс класу з врахуванням відповідних типів доступу, вводять поняття інкапсуляції [68, 76].

Означення 1.4.8. Інкапсуляція – це властивість об'єктів та класів зберігати деяку внутрішню інформацію, доступ до якої відбувається через інтерфейс класу з врахуванням типів доступу.

Поліморфізм. Іноді трапляються ситуації, коли над об'єктами різних типів потрібно виконувати одну і ту ж операцію. У результаті чого цю операцію потрібно реалізовувати окремо для кожного з типів. Такі операції є прикладами поліморфізму [68, 74, 77].

Означення 1.4.9. Поліморфізм – це використання спільного інтерфейсу для обробки різних типів даних.

В ООП на основі класів виділяють *ad hoc поліморфізм*, *параметричний поліморфізм* та *поліморфізм підтипів*. Ad hoc поліморфізм реалізується за допомогою перевантаження функцій (коли функції з однаковою назвою реалізовують різну логіку для різних типів вхідних параметрів) та приведення типів (перетворення значення одного типу в значення іншого типу. Параметричний поліморфізм реалізується за допомогою узагальненого програмування (підпрограми без визначення конкретного типу параметрів, що дозволяє застосовувати такі підпрограми до

параметрів різного типу). Поліморфізм підтипів реалізується за допомогою обмеження діапазону поліморфних типів (можливість виклику поліморфної функції, що визначена для деякої ієрархії класів).

Успадкування. Існують випадки коли декілька різних типів можуть містити однакові (еквівалентні) властивості або(та) методи. Зазвичай такі типи є семантично близькими або спорідненими, однак якщо їх описувати у вигляді окремих класів, то таке представлення буде неефективним через дублювання еквівалентних властивостей та методів. Для уникнення цієї проблеми нові класи визначають на основі раніше визначених за допомогою механізму успадкування [68, 70, 71, 74, 76, 77].

Означення 1.4.10. Успадкування – це механізм визначення (утворення) нових класів на основі раніше визначених класів (батьківських класів).

Таким чином новоутворені класи можуть використовувати всі атрибути та методи, що визначені у батьківських класах, визначати нові (власні) атрибути та методи, а також за потреби перевизначати деякі методи батьківських класів. Використання успадкування дозволяє будувати концептуальні (класові) ієрархії, де класи пов'язані відношеннями *узагальнення* та *спеціалізації*. В рамках ООП на основі класів виділяють два основних типи успадкування: *одиничне успадкування* (ситуація коли один клас успадковує властивості та(або) методи іншого класу) та *множинне успадкування* (ситуація коли один клас успадковує властивості та(або) методи одразу кількох інших класів).

Не зважаючи на всі переваги, які надає такий механізм як успадкування, при побудові концептуальних ієрархій він породжує проблеми надлишковості, винятків, неоднозначності та несумісності [4, 46, 47, 68, 71]. У більшості випадків розробники мов програмування перекладають вирішення цих проблем на розробників програмного забезпечення або функціонально обмежують реалізацію механізмів успадкування в рамках мови програмування [68].

Абстрактні класи. В рамках багатьох предметних областей існують такі концепти, для яких може не існувати конкретних сутностей. Такі концепти є різного

роду абстракціями, для подання яких в об'єктно-орієнтованому стилі використовують *абстрактні класи*. Для таких класів відсутня можливість створення об'єктів, однак вони використовуються для побудови концептуальних ієрархій, і тому можуть мати, як суперкласи, так і підкласи [68, 71]. Зазвичай абстрактні класи використовуються для організації розгалужень в концептуальних ієрархіях.

На сьогодні об'єктно-орієнтоване програмування на основі класів підтримують такі мови як Squeak, VisualWorks, Common Lisp, C++, C#, Java, Objective-C, Python, Ruby, PHP, Scala.

Висновки. Об'єктно-орієнтоване програмування на основі класів, як і інші моделі подання знань, має свої переваги та недоліки.

Переваги та можливості:

— повторне використання коду класів при створені об'єктів [68, 74, 76, 77];

— інкапсуляція даних в рамках класів та об'єктів [68, 74, 76];

— поліморфізм методів класів [68, 74, 77];

— можливості для побудови концептуальних (класових) ієрархій [68, 70, 71, 74];

— регламентований доступ до полів та методів класів [68, 71, 74];

— представлення декларативних та процедурних знань у вигляді специфікацій та сигнатур класів відповідно [68, 74].

Недоліки та обмеження:

— використання лише однорідних класів об'єктів [2, 21, 22];

— використання лише одиничного та множинного успадкування [4, 68, 74, 76, 77];

— виникнення проблем надлишковості, винятків, неоднозначності та несумісності при проектуванні концептуальних ієрархій з використання механізмів одиничного та множинного успадкування [4, 46, 47, 68];

— статичність (незмінність) структури об'єктів та класів об'єктів [68, 74];

— відсутність можливості створення нових класів об'єктів під час виконання програм [21].

1.4.2. Об'єктно-орієнтоване програмування на основі прототипів. Даний підхід до об'єктно-орієнтованого програмування на сьогодні є менш популярним ніж підхід на основі класів, оскільки він перебуває у процесі становлення та розвитку. Однак на сьогодні існують деякі доволі популярні об'єктно-орієнтовані мови програмування які реалізують такий стиль ООП [68].

Прототипи. На відміну від ООП на основі класів, в рамках підходу на основі прототипів об'єкти що описують певні сутності можуть бути створені без попереднього опису їх класу [68, 70, 72]. Для цього замість класів використовуються *прототипи*.

Означення 1.4.11. Прототип – це репрезентативний загальноприйнятий приклад певного концепту.

Таким чином прототипи та класи мають як певну схожість, так і відмінності. Прототипи є стереотипними ініціалізованими об'єктами в рамках певної предметної області, тоді як класи є абстрактними прототипами для реальних об'єктів певного типу. Таким чином прототип – це завжди один конкретний об'єкт, тоді як клас – це абстракція для створення певної множини конкретних однотипних об'єктів. Отже в рамках ООП на основі прототипів усі об'єкти є прототипами для створення нових об'єктів.

За своєю структурою прототипи подібні до об'єктів в рамках ООП на основі класів, однак окрім полів (властивостей) вони мають ще й методи. Формально прототип можна визначити наступним чином.

Означення 1.4.12. Прототип – це кортеж виду $Pr = (P(Pr), F(Pr))$, де Pr – це ім'я (ідентифікатор) прототипу, $P(Pr) = \{p_1(Pr), \ldots, p_n(Pr)\}$ – це специфікація, що описує його природу, а $F(Pr) = \{f_1(Pr), \ldots, f_m(Pr)\}$ – це сигнатура, яка визначає його поведінку.

Важливою особливістю прототипів є те, що для деяких концептів може не існувати чіткого прототипу, у зв'язку з чим класифікація в рамках ООП на основі прототипів базується на стереотипних апроксимаціях [68]. Іншою важливою особливістю ООП на основі прототипів є те, що концепти в таких системах виникають

у процесі, а не визначаються згідно чітко заданого шаблону (класу). На відміну від ООП на основі класів в рамках підходу на основі прототипів не можливо визначити абстрактні концепти, які описуються за допомогою абстрактних класів, для яких не можливо створювати об'єкти.

Подібно до ООП на основі класів, прототипи можуть мати *відкриті* та *закриті* поля та методи. Відкриті властивості та(або) методи доступні для усіх прототипів, у той час як закриті властивості та(або) методи доступні лише в рамках інших методів та полів цього прототипу [68].

Означення 1.4.13. Делегування – це процес в рамках якого одні об'єкти делегують свою поведінку іншим об'єктами.

У результаті чого нові об'єкти створюються шляхом копіювання та модифікації значень потів та(або) структури копій вже існуючих (раніше створених) об'єктів, які є прототипами для створення нових, похідних від них, об'єктів. Усі об'єкти створені таким чином будуть у певному сенсі близькими або подібними до своїх прототипів. Таким чином механізм делегування задає *відношення близькості* на множині прототипів утворених з його використанням [68, 70–72].

На відміну від успадкування, делегування дозволяє уникати проблем надлишковості, винятків, неоднозначності та несумісності при побудові концептуальних (класових) ієрархій [68, 70].

Сьогодні об'єктно-орієнтоване програмування на основі прототипів підтримують такі мови як Self, Omega, Common Lisp, Object Lisp, JavaScript, JScript.

Висновки. Об'єктно-орієнтоване програмування на основі прототипів, як і інші моделі подання знань, має свої переваги та недоліки.

Переваги та можливості:

— можливість динамічного прототипування [68, 70];

— динамічна зміна відношень між прототипами [68];

— можливість множинного делегування [68];

— можливості для побудови концептуальних ієрархій прототипів [68, 70, 71];

— можливість уникати проблем надлишковості, винятків, неоднозначності та

несумісності під час прототипування [68, 70];

— можливість динамічної зміни структури існуючих прототипів [68, 72].

Недоліки та обмеження:

— відсутні можливості опису абстрактних концептів [68];

— використання лише одиничного та множинного делегування [68, 71];

— класифікація концептів на основі апроксимацій [68].

1.5. Інтелектуальні програмні системи на основі знань

На сьогодні інтелектуальні програмні системи на основі знань є одним з найбільших класів інтелектуальних систем [78]. Основною особливістю таких систем у порівнянні з системами інших класів є те що вони використовують знання про предметну область для вирішення покладених на них завдань і цим самим імітують (симулюють) певний рівень людської інтелектуальності [79].

Структура. Більшість сучасних інтелектуальних програмних систем на основі знань складаються з наступних модулів: інтерфейси користувачів, інтерфейси експертів, моделі подання знань, база знань, база даних, система управління знаннями, система внутрішнього видобування знань, система зовнішнього видобування знань [30, 69, 78–80].

Інтерфейси користувачів дають змогу користувачам взаємодіяти із системою використовуючи її можливості відповідно до її призначення. Інтерфейси експертів дозволяють експертам додавати в систему нові знання про предметну область на яку орієнтована система. Моделі подання знань в залежності від своєї специфіки визначають формати знань які підтримує система та форми інтерфейсів користувачів і експертів. База знань відіграє роль централізованого сховища знань у передбачених моделями подання знань форматах. База даних є фізичним сховищем бази знань. В залежності від специфіки моделей подання знань реалізованих в рамках системи, можуть використовуватися різні види баз даних. Систем управління знаннями дозволяє виконувати певні операції (пошук, редагування, інтеграція нових знань, тощо) над базою знань. Система внутрішнього видобу-

вання знань дозволяє аналізувати наявні знання в системі та видобувати з них нові (неявні) знання у форматах передбачених моделями подання знань. Система зовнішнього видобування знань дозволяє системі видобувати знання у форматах передбачених моделями подання знань із зовнішніх джерел знань, які не є частиною системи.

Таким чином моделі подання знань, які використовуються в рамках системи відіграють ключову роль та безпосередньо визначають специфіку інших модулів інтелектуальних програмних систем на основі знань.

Різновиди систем. На сьогодні виділяють наступні типи інтелектуальних програмних систем на основі знань: експертні системи, системи розпізнавання, рекомендаційні системи, пошукові системи, мультиагентні системи [78, 79, 81–85].

Інтелектуальні програмні системи на основі знань мають багато застосувань в різних сферах людської діяльності. На сьогодні більшість таких програмних систем як консультаційні системи, медичні діагностичні системи, навчальні системи, системи контролю та моніторингу, системи прогнозування, системи планування, пошуково-агрегаційні системи є інтелектуальними програмними системами на основі знань [33, 78].

Висновки до розділу 1

В даному розділі наведений огляд літератури з основними результатами за темою дисертаційного дослідження, в рамках якого були розглянуті та проаналізовані формулювання поняття *знання* та задачі *подання знань*, а також наведена класифікація відомих моделей подання знань.

Основна увага була зосереджена на аналізі таких об'єктно-орієнтованих моделей подання знань, як системи фреймів, скрипти та динамічна пам'ять, об'єктно-орієнтоване програмування на основі класів та на основі прототипів. Проведений аналіз зазначених моделей подання знань дозволяє зробити висновок, що в рамках об'єктно-орієнтованого підходу до подання знань декларативні знання зазвичай подаються у вигляді об'єктів, класів (прототипів) та відношень між ними, у той

час як процедурні знання подаються у вигляді методів класів (прототипів) або сценаріїв (скриптів).

Окрім цього була розглянута та проаналізована загальна структура сучасних інтелектуальних програмних систем на основі знань, в рамках якої моделі подання знань відіграють ключову роль в проектуванні, розробці, функціонуванні та подальшому розвитку цих систем. Після чого були розглянуті деякі відомі застосування моделей подання знань та основні класи інтелектуальних програмних систем на основі знань.

Джерела використані у розділі 1

Для написання даного розділу було використано 62 джерела [2, 4, 21, 22, 28–85], посилання на які зазначені в тексті розділу.

РОЗДІЛ 2
ОБ'ЄКТНО-ОРІЄНТОВАНІ ДИНАМІЧНІ МЕРЕЖІ

З аналізу таких об'єктно-орієнтованих моделей подання знань, як системи фреймів, скрипти та динамічна пам'ять, а також об'єктно-орієнтоване програмування на основі класів та на основі прототипів, можна зробити висновок, що в рамках цих моделей декларативні знання про ту чи іншу предметну область формалізуються в термінах класів, об'єктів та відношень між ними, а процедурні – у термінах методів класів або сценаріїв (скриптів). У зв'язку з цим в рамках декларативної частини нової моделі подання знань визначимо поняття об'єктів, класів об'єктів та відношень між ними, а в рамках процедурної частини – операції над об'єктами та класами об'єктів.

2.1. Об'єкти

Розглянемо такий об'єкт, як натуральне число. Очевидно, що кожне натуральне число повинне бути цілим та додатнім, що є характеристичними властивостями натуральних чисел. Неважко переконатися, що 5 – це дійсно натуральне число, а -2 або 6.3 не є такими. Аналізуючи вище наведені факти, можна зробити висновок, що будь-який об'єкт має певні властивості, які є характеристичними для нього і визначають його як деяку сутність та дозволяють відрізняти його від інших об'єктів [1, 2, 21, 22].

Об'єкти та їх властивості нерозривно пов'язані між собою та не можуть існувати окремо один від одного. З одного боку, об'єкт не може існувати окремо від своїх властивостей, оскільки без них неможливо уявити, описати або створити цей об'єкт. З іншого боку, властивості не можуть існувати окремо від об'єкта, оскільки без нього їх неможливо уявити, побачити, відчути або сприйняти. Таким чином, жоден довільний об'єкт та його властивості не можуть існувати окремо, внаслідок

чого неможливо визначити поняття об'єкта без визначення його властивостей та навпаки [1]. Тому спершу визначимо властивості об'єктів, які глобально можна поділити на два типи – кількісні та якісні [21, 22].

Означення 2.1.1. Кількісна властивість $p_i(A)$ об'єкта A – це кортеж виду

$$p_i(A) = ((v_1(p_i(A)), u_1(p_i(A))), \ldots, (v_q(p_i(A)), u_q(p_i(A)))),$$

де $v_j(p_i(A))$ – це кількісне значення властивості $p_i(A)$, $u_j(p_i(A))$ – це одиниця в якій вимірюється значення $v_j(p_i(A))$, де $j = \overline{1,q}$, $i = \overline{1,n}$, де n – це кількість властивостей об'єкта A.

Розглянемо деякі приклади кількісних властивостей об'єктів.

Приклад 2.1.1. Розглянемо такий об'єкт як яблуко. Будь-яке яблуко має масу, яку можна представити як $p_w(A) = (v(p_w(A)), u(p_w(A)))$. Якщо яблуко має масу 0.15 кг, то властивість $p_w(A)$ матиме наступний вигляд $p_w(A) = (0.15, \text{кг})$.

Приклад 2.1.2. Розглянемо такий об'єкт як слово української мови. Більшість слів української мови містять певну кількість літер, які позначають голосні звуки (г.з.). Що можна представити як $p_{vl}(W) = (v(p_{vl}(W)), u(p_{vl}(W)))$. Якщо $W =$ "математика", тоді властивість $p_{vl}(W)$ матиме наступний вигляд $p_{vl}(W) = ((\text{а}, \text{е}, \text{а}, \text{и}, \text{а}), \text{г.з.})$.

Приклад 2.1.3. Розглянемо таку геометричну фігуру як прямокутник. Однією з важливих властивостей будь-якого прямокутника є довжини його сторін, які можна представити наступним чином

$$p_{ss}(R) = ((v_1(p_{ss}(R)), u_1(p_{ss}(R))), (v_2(p_{ss}(R)), u_2(p_{ss}(R))),$$
$$(v_3(p_{ss}(R)), u_3(p_{ss}(R))), (v_4(p_{ss}(R)), u_4(p_{ss}(R)))).$$

Якщо прямокутник має сторони довжини 2см, 3см, 2см, 3см, то властивість $p_{ss}(R)$ виглядатиме наступним чином $p_{ss}(R) = ((2, \text{см}), (3, \text{см}), (2, \text{см}), (3, \text{см}))$.

Для можливості порівняння різних кількісних властивостей, визначимо поняття їх еквівалентності.

Означення 2.1.2. Дві кількісні властивості $p_1(A_1)$ і $p_1(A_2)$ довільних об'єктів A_1 та A_2 є еквівалентними, тобто $Eq(p_1(A_1), p_1(A_2)) = 1$, тоді і тільки тоді коли

$$(v_1(p_1(A_1)), u_1(p_1(A_1))) = (v_1(p_1(A_2)), u_1(p_1(A_2))), \ldots, (v_n(p_1(A_1)), u_n(p_1(A_1))) = \\ = (v_m(p_1(A_2)), u_m(p_1(A_2))).$$

Тепер визначимо поняття якісної властивості об'єкта.

Означення 2.1.3. Якісна властивість $p_i(A)$ об'єкта A – це k-місна функція верифікації $vf_i(A)$, що визначається наступним чином

$$p_i(A) = vf_i(A) : (p_{j_1}(A), \ldots, p_{j_k}(A)) \to \{0, 1\},$$

де $i = \overline{1, n}$, де n – це кількість властивостей об'єкта A, $(p_{j_1}(A), \ldots, p_{j_k}(A))$ – це кількісні або якісні властивості об'єкта A і $1 \leqslant j_1 \leqslant \cdots \leqslant j_k \leqslant n$, а $1 \leqslant k \leqslant n-1$.

Розглянемо деякі приклади якісних властивостей об'єктів.

Приклад 2.1.4. Розглянемо такий об'єкт, як ціле число n та таку його кількісну властивість, як значення, тобто $p_1(n) = (n, \text{значення})$. Відомо що деяким цілим числам характерна властивість "бути додатнім", яку можна представити наступним чином $p_2(n) = vf_2(n) : p_1(n) \to \{0, 1\}$. Якщо $n > 0$, тоді $vf_2(n) = 1$, якщо $n < 0$ то $vf_2(n) = 0$. Наведена функція верифікації $vf_2(n)$ є булевою функцією, однак вона може мати різні реалізації, наприклад

```
bool f(int n) {
    if ((n * (-1)) > 0) {
        return 0;
    } else {
        return 1;
    }
}
```

```
bool f(int n) {
    if ((n+n) > 0) {
        return 1;
    } else {
        return 0;
    }
}
```

```
bool f(int n) {
    if (pow(n,3) > 0) {
        return 1;
    } else {
        return 0;
    }
}
```

кожна з яких реалізує певний алгоритм перевірки властивості $p_2(n)$.

Приклад 2.1.5. Розглянемо такий об'єкт як трикутник T, розміри сторін якого становлять v_1 см, v_2 см та v_3 см, тобто $p_1(T) = ((v_1, \text{см}), (v_2, \text{см}), (v_3, \text{см}))$. Однією з основних властивостей будь-якого трикутника є "нерівність трикутника",

яку можна представити як $p_{ti}(T) = vf_{ti}(T) : p_1(T) \to \{0, 1\}$. Якщо розміри сторін трикутника T становлять 3 см, 5 см та 7 см тоді $vf_{ti}(T) = 1$, якщо ж трикутник T має сторони довжиною 2 см, 3 см та 5 см, то $vf_{ti}(T) = 0$.

Функція верифікації $vf_{ti}(T)$ може бути реалізована наступним чином

```
bool f(int a, b, c) {
    if ((a + b > c) && (b + c > a) && (a + c > b)){
        return 1;
    } else {
        return 0;
    }
}
```

де a, b та c – довжини сторін трикутника T.

Для можливості порівняння різних якісних властивостей, визначимо поняття їх еквівалентності.

Означення 2.1.4. Дві якісні властивості $vf_1(A_1)$ та $vf_1(A_2)$ довільних об'єктів A_1 та A_2, що визначаються як $vf_1(p_{i_1}(A_1), \ldots, p_{i_k}(A_1))$ та $vf_1(p_{j_1}(A_2), \ldots, p_{j_w}(A_2))$, є еквівалентними, тобто $Eq(vf_1(A_1), vf_1(A_2)) = 1$, тоді і тільки тоді, коли

$$(k = w) \wedge \left(vf_1^{A_1}(p_{i_1}(A_1), \ldots, p_{i_k}(A_1)) = vf_1^{A_2}(p_{i_1}(A_1), \ldots, p_{i_k}(A_1))\right) \wedge$$
$$\wedge \left(vf_1^{A_1}(p_{j_1}(A_2), \ldots, p_{j_w}(A_2)) = vf_1^{A_2}(p_{j_1}(A_2), \ldots, p_{j_w}(A_2))\right).$$

Оскільки об'єкти можуть мати певну кількість властивостей, що визначають їх як деякі сутності, введемо поняття специфікації та розмірності об'єкта.

Означення 2.1.5. Специфікація об'єкта A – це вектор виду

$$P(A) = (p_1(A), \ldots, p_n(A)),$$

такий що $\forall p_i(A), p_j(A),\ i, j = \overline{1, n},\ i \neq j \mid Eq(p_i(A), p_j(A)) = 0$, де $p_i(A)$ та $p_j(A)$ є кількісними або якісними властивостями об'єкта A.

Означення 2.1.6. Розмірність об'єкта A – це натуральне число n, що виражає кількість властивостей, які входять до його специфікації та позначається як $D(A) = |P|$, де P – це множина, що складається з елементів вектора $P(A)$.

Використовуючи поняття еквівалентності кількісних та якісних властивостей об'єктів, введемо поняття еквівалентності специфікацій об'єктів.

Означення 2.1.7. Дві довільні специфікації $P(A) = (p_1(A), \ldots, p_n(A))$ та $P(B) = (p_1(B), \ldots, p_m(B))$ об'єктів A та B є еквівалентними, тобто $P(A) = P(B)$, тоді і тільки тоді, коли

$$(D(A) = D(B)) \wedge (\forall p_i(A)\ \exists! p_j(B)) \wedge (\forall p_j(B)\ \exists! p_i(A)) \mid Eq(p_i(A), p_j(B)) = 1,$$

де $i = \overline{1, n}$, а $j = \overline{1, m}$.

Тепер, використовуючи визначення властивостей та специфікації об'єктів, визначимо поняття об'єкта.

Означення 2.1.8. Об'єкт – це пара виду $A/P(A)$, де A – це ідентифікатор об'єкта, а $P(A) = (p_1(A), \ldots, p_n(A))$ – його специфікація.

Іншими словами, об'єкт – це носій деякої сукупності (множини) властивостей та ознак, що визначають його як деяку сутність.

Означення 2.1.9. Два довільних об'єкта A та B є еквівалентними, тоді і тільки тоді, коли $P(A) = P(B)$, де $P(A)$ та $P(B)$ – специфікації об'єктів A та B відповідно.

Окрім властивостей об'єктів, існують і методи (операції), які можна застосовувати до об'єктів, враховуючи особливості їх специфікацій.

Означення 2.1.10. Метод об'єкта або операція над об'єктом A – це функція $f(A)$, яку можна застосувати до об'єкта з врахуванням особливостей його специфікації.

Розглянемо деякі приклади методів об'єктів.

Приклад 2.1.6. Розглянемо такий об'єкт як натуральне число $n = 5$, який визначається специфікацією $P(n) = (p_1(n), p_2(n), p_3(n)) = (5, 1, 1)$, де $p_1(n)$ – це значення числа, $p_2(n)$ – це властивість числа "бути цілим", а $p_3(n)$ – це властивість числа "бути додатнім".

Однією з операцій, які можна виконати над будь-яким цілим числом, є піднесення числа до певного степеня, тому як приклад методу для об'єкта n можна

розглянути метод $f_1(n) = v(p_1(n))^3 = 5^3 = 125$, який обчислює значення числа n в 3-му степені.

Приклад 2.1.7. Розглянемо такий об'єкт як прямокутник A, який визначається наступною специфікацією

$$P(A) = (p_1(A), p_2(A), p_3(A), p_4(A), p_5(A)) =$$
$$= ((4 \text{ сторони}), (2 \text{ см},\ 3 \text{ см},\ 2 \text{ см},\ 3 \text{ см}), (4 \text{ кути}), (90^\circ, 90^\circ, 90^\circ, 90^\circ), 1),$$

де $p_1(A)$ – це кількість сторін фігури, $p_2(A)$ – розміри сторін фігури, $p_3(A)$ – кількість кутів фігури, $p_4(A)$ – градусні міри кутів фігури, $p_5(A)$ – це рівність протилежних сторін фігури. Якщо відомі довжини сторін прямокутника, то можна обчислити його периметр та площу. Такі обчислення також є методами $f_1(A) = 2(a+b)$, $f_2(A) = ab$, які можна застосувати до об'єкта A, в результаті чого отримаємо що $f_1(A) = 2(2+3) = (10, \text{см})$, а $f_2(A) = 2 \cdot 3 = (6, \text{см}^2)$.

Тепер визначимо поняття еквівалентності двох довільних методів об'єктів. З означення методу об'єкта випливає, що еквівалентність методів повинна визначатися як еквівалентність двох довільних функцій, проте, відомо, що така проблема в загальному випадку нерозв'язна. У зв'язку з чим визначимо поняття еквівалентності двох довільних методів об'єктів наступним чином.

Означення 2.1.11. Два довільні методи $f_1(A)$ та $f_2(B)$ довільних об'єктів A та B еквівалентні, тобто $Eq(f_1(A), f_2(B)) = 1$, тоді і тільки тоді, коли

$$(f_1(A) = f_2(A)) \wedge (f_2(B) = f_1(B)).$$

Іншими словами, два методи $f_1(A)$ та $f_2(B)$ можуть бути абсолютно різними функціями, проте, якщо вони повертають однаковий результат для двох різних об'єктів A та B, то ці функції є еквівалентними в контексті цих об'єктів.

Оскільки для кожного об'єкта може існувати певна кількість методів, які можуть бути застосовані до нього з врахуванням особливостей його специфікації, визначимо поняття сигнатури та функціональності об'єкта.

Означення 2.1.12. Сигнатура об'єкта A – це вектор виду

$$F(A) = (f_1(A), \ldots, f_m(A)),$$

де $f_i(A)$, $i = \overline{1,m}$ є i-м методом об'єкта A.

Означення 2.1.13. Функціональність $func(A)$ об'єкта A – це натуральне число m, що виражає кількість методів, які входять до його сигнатури.

Використовуючи поняття еквівалентності методів об'єктів, введемо поняття еквівалентності їхніх сигнатур.

Означення 2.1.14. Дві довільні сигнатури $F(A) = (f_1(A), \ldots, f_n(A))$ та $F(B) = (f_1(B), \ldots, f_m(B))$ довільних об'єктів A та B є еквівалентними, тобто $F(A) = F(B)$, тоді і тільки тоді, коли

$$(func(A) = func(B)) \wedge (\forall f_i(A)\ \exists! f_j(B)) \wedge (\forall f_j(B)\ \exists! f_i(A)) \mid Eq(f_i(A), f_j(B)) = 1,$$

де $i = \overline{1,n}$, а $j = \overline{1,m}$.

2.2. Класи об'єктів

В загальному об'єкти можна поділити на конкретні (реально матеріально існуючі) та абстрактні. Кожен конкретний об'єкт, незалежно від того, коли і як він був створений, є матеріальною реалізацією свого абстрактного образу – *прототипу*. Відповідно, кожен прототип є деякою абстрактною специфікацією для створення майбутніх реальних об'єктів [1, 2].

Згідно з Означенням об'єкта 2.1.8, кожен об'єкт має певну специфікацію, яка визначає його як деяку сутність. Проте, існують специфікації, які одночасно визначають деяку кількість об'єктів, як і сигнатури, які можна застосовувати до певної кількості об'єктів. Таким чином, об'єкти, що визначаються однією специфікацію і до яких можна застосувати одну і ту ж сигнатуру, є *однотипними* і належать до одного класу об'єктів [1, 2, 21].

2.2.1. Однорідні класи об'єктів. В рамках об'єктно-орієнтованого програмування (ООП), програмісти оперують специфікаціями та методами об'єктів, абстрагуючись від конкретних реалізацій об'єктів, називаючи їх *типами* або *класами об'єктів* [68, 70, 74–77]. Клас складається із *полів* (властивостей) та *методів*

(операцій), які формують його специфікацію та сигнатуру відповідно. Аналізуючи концепцію об'єктно-орієнтованого класу, можна зробити висновок, що клас є прототипом для створення конкретних об'єктів. Всі об'єкти певного класу матимуть специфікацію, аналогічну до специфікації їхнього класу, і до них можна буде застосовувати лише ті методи, які визначені в рамках їх класу. Таким чином, використовуючи один об'єктно-орієнтований клас, неможливо створити об'єкти різних типів. Це пов'язано з тим, що всі класи в рамках ООП є *однорідними*, тобто такими, що описують лише однотипні об'єкти. Використовуючи введене раніше поняття об'єкта та концепцію класу в рамках ООП, визначимо поняття однорідного класу об'єктів.

Означення 2.2.1. Однорідний клас об'єктів T – це кортеж виду

$$T = (P(T), F(T)) = ((p_1(T), \ldots, p_n(T)), (f_1(T), \ldots, f_m(T))),$$

де $P(T)$ – це специфікація, що визначає певну кількість об'єктів, а $F(T)$ – сигнатура, яку до них можна застосовувати.

Таким чином, клас об'єктів можна розглядати як деяку узагальнену форму розгляду властивостей об'єктів та операцій над об'єктами без самих об'єктів або як деякий абстрактний об'єкт чи прототип для певної кількості конкретних об'єктів.

Важливим моментом у визначенні класів об'єктів є задання певних значень для кількісних властивостей, оскільки, якщо для певної кількісної властивості $p(T_1)$ класу T_1 визначене конкретне значення $v(P(T_1))$, це означає, що для всіх об'єктів цього класу ця властивість матиме таке ж значення. Іншими словами, ініціалізація кількісних властивостей класу визначає степінь загальності або конкретності класу. Якщо для деякої кількісної властивості $p(T_2)$ класу T_2 не визначене конкретне значення, то це означає, що різні об'єкти цього класу можуть мати різні значення цієї властивості. Таким чином, клас об'єктів, у якого не задані конкретні значення кількісних властивостей, є більш загальним, ніж клас об'єктів з тією ж специфікацією, але для якої визначене конкретне значення хоча б для однієї кількісної властивості. У зв'язку з цим введемо наступні два означення.

Означення 2.2.2. Кількісна властивість $p_1(T)$ класу T є строго визначеною,

тоді і тільки тоді, коли вона має чітко визначене кількісне значення $v_i(p_1(T))$ та одиниці вимірювання цього значення $u_i(p_1(T))$.

Означення 2.2.3. Кількісна властивість $p_1(T)$ класу T є слабко визначеною, тоді і тільки тоді, коли вона не має чітко визначеного кількісного значення $v_i(p_1(T))$, але має чітко визначені одиниці вимірювання цього значення $u_i(p_1(T))$.

Означення 2.2.4. Якісна властивість $p_1(T)$ класу T є строго визначеною, тоді і тільки тоді, коли вона визначена на строго визначених кількісних та(або) якісних властивостях.

Означення 2.2.5. Якісна властивість $p_1(T)$ класу T є слабко визначеною, тоді і тільки тоді, коли вона визначена на слабко визначених кількісних та(або) якісних властивостях.

Для зручності формулювання подальших означень, введемо поняття розмірності та функціональності однорідного класу об'єктів.

Означення 2.2.6. Розмірність $D(T)$ однорідного класу об'єктів T – це натуральне число n, що виражає кількість властивостей, що входять до його специфікації.

Означення 2.2.7. Функціональність $func(T)$ однорідного класу об'єктів T – це натуральне число m, що виражає кількість методів, що входять до його сигнатури.

Розглянемо деякі приклади однорідних класів об'єктів.

Приклад 2.2.1. Розглянемо клас об'єктів, що описує яблука та визначається наступним чином

$$T_A = (p_1(T_A) = (v, \text{сорт}),\ p_2(T_A) = (v, \text{колір}),\ p_3(T_A) = (v, \text{кг}),$$
$$p_4(T_A) = (v, \text{грн/кг}),\ f_1(T_A) = (v(p_3(A)) \cdot v(p_4(A)), \text{грн})),$$

де $p_1(T_A)$ – це назва сорту яблука, $p_2(T_A)$ – це колір яблука, $p_3(T_A)$ – це маса яблука, $p_4(T_A)$ – це ціна за 1 кг, а $f_1(T_A)$ – це метод визначення ціни яблука.

Тепер розглянемо об'єкти A_1 та A_2 класу T_A, які визначаються наступним

чином

$$A_1 = (p_1(A_1) = ((\text{мунтуан, сорт}),\ p_2(A_1) = (\text{червоно-жовтий, колір}),$$
$$p_3(A_1) = (0.1, \text{кг}),\ p_4(A_1) = (45, \text{грн/кг})).$$
$$A_2 = (p_1(A_2) = ((\text{стракримсон, сорт}),\ p_2(A_2) = (\text{темно-червоний, колір}),$$
$$p_3(A_2) = (0.13, \text{кг}),\ p_4(A_2) = (75, \text{грн/кг})).$$

Якщо до об'єктів A_1 та A_2 застосувати метод $f_1(T_A)$, то отримаємо що вартість яблука A_1 становить 4.5 грн, а яблуко A_2 коштує 9.75 грн. Очевидно, що об'єкти A_1 та A_2 є однорідними, оскільки вони визначені в рамках одного класу об'єктів та мають однакові властивості. Проте, згідно з Означенням 2.1.2

$$p_1(A_1) \neq p_1(A_2), \ldots, p_4(A_1) \neq p_4(A_2),$$

оскільки $v(p_1(A_1)) \neq v(p_1(A_2)), \ldots, v(p_4(A_1)) \neq v(p_4(A_2))$, але при цьому

$$u(p_1(A_1)) = u(p_1(A_2)), \ldots, u(p_4(A_1)) = u(p_4(A_2)).$$

Таким чином, з Означень 2.1.7 та 2.1.9 випливає, що об'єкти A_1 та A_2 не є еквівалентними.

Враховуючи вище зазначений приклад, визначимо поняття еквівалентності кількісних та якісних властивостей об'єкта та однорідного класу об'єктів.

Означення 2.2.8. Кількісна властивість $p_1(A)$ довільного об'єкта A та кількісна властивість $p_1(T)$ довільного однорідного класу об'єктів T є еквівалентними, тобто $Eq(p_1(A), p_1(T)) = 1$, тоді і тільки тоді, коли

$$\begin{cases} (v(p_1(A)) = v(p_1(T))) \wedge (u(p_1(A)) = u(p_1(T))), & p_1(T) \text{ строго визначена}, \\ u(p_1(A)) = u(p_1(T)), & p_1(T) \text{ слабко визначена}. \end{cases}$$

Означення 2.2.9. Якісна властивість $vf_1(A)$ довільного об'єкта A та якісна властивість $vf_1(T)$ довільного однорідного класу об'єктів T є еквівалентними, тобто $Eq(vf_1(A), vf_1(T)) = 1$, тоді і тільки тоді, коли

$$\left(vf_1^A(A) = vf_1^B(A)\right) \wedge \left(vf_1^A(B) = vf_1^B(B)\right).$$

Тепер визначимо поняття належності об'єкта до деякого однорідного класу об'єктів.

Означення 2.2.10. Об'єкт $A/(p_1(A), \ldots, p_n(A))$ є об'єктом однорідного класу об'єктів $T = (p_1(T), \ldots, p_m(T), f_1(T), \ldots, f_k(T))$, тобто $A \in T$, тоді і тільки тоді, коли $(D(A) = D(T)) \wedge (\forall p_i(A)\ \exists! p_j(T)) \wedge (\forall p_j(T)\ \exists! p_i(A))|\ Eq(p_i(A), p_j(T)) = 1$, де $i = \overline{1,n}$, а $j = \overline{1,m}$.

Приклад 2.2.2. Використовуючи поняття однорідного класу, можна визначити примітивні типи даних для деяких мов програмування. Для прикладу визначимо тип Int в мові програмування C++. Таким чином, клас Int матиме наступну специфікацію

$$P(Int) = (p_1(Int) = (v, \text{значення}),\ p_2(Int) = vf_2(Int) = (1),$$
$$p_3(Int) = vf_3(Int) = (1)),$$

де $p_1(Int)$ – це значення числа, $vf_2(Int) : p_1(Int) \to \{0, 1\}$ – функція верифікації що визначає властивість "бути цілим числом" і $vf_3(Int) : p_1(Int) \to \{0, 1\}$ – функція верифікації що визначає властивість "бути не меншим ніж -2147336148 та не більшим ніж 2147336147". Функції $vf_2(Int)$ та $vf_3(Int)$ можуть бути реалізовані наступним чином.

```
bool vf_2(int n) {
    if (n == (int)n) {
        return 1;
    } else {
        return 0;
    }
}
```

```
bool vf_3(int n) {
    if ((n >= -2147336148) &&
        (n <= 2147336148)) {
        return 1;
    } else {
        return 0;
    }
}
```

Очевидно, що параметром n функцій $vf_2(Int)$ та $vf_3(Int)$ буде значення властивості $p_1(Int)$, тобто $v(p_1(Int))$. Тепер визначимо для класу Int деякі елементарні операції над цілими числами:

$$F(Int) = (f_1(Int) = (v(p_1(Int)) + n, \text{значення}),$$

$$f_2(Int) = (v(p_1(Int)) \cdot n, \text{значення})),$$

де $f_1(Int)$ – це операція, що додає до об'єкта класу Int деяке ціле число n, а $f_2(Int)$ – це операція, що множить об'єкт класу Int на деяке ціле число n.

Розглянемо об'єкти $A_1, \ldots, A_4$, які визначаються наступним чином

$$A_1 = (p_1(A_1) = (5, \text{значення}),\ p_2(A_1) = vf_2(A_1) = (1),\ p_3(A_1) = vf_3(A_1) = (1)),$$
$$A_2 = (p_1(A_2) = (23, \text{значення}),\ p_2(A_2) = vf_2(A_2) = (1),\ p_3(A_2) = vf_3(A_2) = (1)),$$
$$A_3 = (p_1(A_3) = (2.5, \text{значення}),\ p_2(A_3) = vf_2(A_3) = (0),\ p_3(A_3) = vf_3(A_3) = (1)),$$
$$A_4 = (p_1(A_4) = (8, \text{значення}),\ p_2(A_4) = vf_2(A_4) = (1),\ p_3(A_4) = vf_3(A_4) = (1),$$
$$p_4(A_4) = vf_4(A_4) = (1)),$$

де $p_1(A_1)$, $p_1(A_2)$, $p_1(A_3)$, $p_1(A_4)$ – це значення чисел, $vf_2(A_1)$, $vf_2(A_2)$, $vf_2(A_3)$, $vf_2(A_4)$ – це функції верифікації, що визначають властивість "бути цілим числом", тобто $vf_2(A_i) : p_2(A_i) \to \{0, 1\}$, $i = \overline{1, 4}$; $vf_3(A_1)$, $vf_3(A_2)$, $vf_3(A_3)$, $vf_3(A_4)$ – це функції верифікації, що визначають властивість "бути не меншим, ніж -2147336148 та не більшим, ніж 2147336147", тобто $vf_3(A_i) : p_3(A_i) \to \{0, 1\}$, $i = \overline{1, 4}$ і $vf_4(A_4)$ – це функція верифікації, що визначає властивість "бути парним числом".

Аналізуючи специфікації однорідного класу об'єктів Int та об'єктів $A_1, \ldots, A_4$, можна бачити, що об'єкти $A_1, A_2 \in Int$, а об'єкти $A_3, A_4 \notin Int$, що випливає з Означення 2.2.10. Проте, виникають деякі запитання, зокрема, чому $A_4 \notin Int$, незважаючи на те, що це слідує з Означення 2.2.10. Тому розглянемо перевірку належності об'єкта A_4 до класу Int більш детально. Для того, щоб показати, що об'єкт $A_4 \in Int$ або навпаки $A_4 \notin Int$ потрібно показати чи спростувати еквівалентність їхніх специфікацій, використовуючи Означення 2.2.10. Аналізуючи специфікації $P(A_4)$ та $P(Int)$, можна бачити, що

$$u(p_1(A_4)) = u(p_1(Int)),\ vf_2(A_4) = vf_2(Int),\ vf_3(A_4) = vf_3(Int).$$

Проте, специфікація $P(A_4)$ містить 4 властивості, в той час як специфікація $P(Int)$ – лише 3, тому $A_4 \notin Int$. Але, з іншого боку, число 8 є об'єктом класу

Int, оскільки для нього виконуються усі властивості, які зазначені у специфікації $P(Int)$. Однак, об'єкт A_4, який описує число 8 має більший рівень деталізації, ніж наведена в даному прикладі специфікація класу Int, за рахунок додаткової властивості $vf_4(A_4)$. Якщо проаналізувати належність об'єктів A_1 та A_2 до класу Int, то можна помітити, що їхні специфікації містять однакову кількість властивостей і є еквівалентними, що дозволяє говорити про однаковий рівень деталізації цих об'єктів та класу, до якого вони належать. Таким чином, необхідною умовою належності об'єкта до деякого класу об'єктів є однакова кількість властивостей у їхніх специфікаціях.

2.2.2. Одноядерні неоднорідні класи об'єктів. Однорідні класи об'єктів мають широке практичне застосування в об'єктно-орієнтованому програмуванні, однак, незважаючи на всі їх переваги, вони мають деякі обмеження у контексті представлення реальних та абстрактних об'єктів з різних предметних областей у термінах тієї чи іншої об'єктно-орієнтованої мови програмування. Ці обмеження полягають у тому, що всі об'єкти однорідного класу мають лише ті властивості, які визначені в специфікації їх класу і до них можна застосовувати лише методи, визначені в сигнатурі цього ж класу. Проте, існує багато об'єктів, що належать до різних класів одночасно, і якщо виникає потреба працювати з їх моделями в рамках однієї програми, що написана з використанням деякої об'єктно-орієнтованих мови програмування, то потрібно описувати окремий клас для кожного нового типу об'єктів. Якщо кількість таких класів та об'єктів, створених в їх рамках, є не дуже великою, то для їх представлення можна використовувати концепцію однорідних класів. Однак, коли мова йде про велику кількість різних класів, то процес їх опису більш тривалим, не кажучи вже про розміри програмних кодів, складність їх верифікації, тестування, а також швидкодію таких програм.

Коли класи, які потрібно визначити в рамках певної програми, не мають однакових властивостей та методів, тоді не залишається нічого іншого, як використовувати однорідні класи. Проте, якщо деякі класи мають такі властивості та методи, то їх представлення може бути більш оптимальним, зокрема у контекс-

ті витраченої пам'яті. Для цього в рамках об'єктно-орієнтованого програмування існує такий потужний інструмент як *успадкування*, який дозволяє визначати нові класи об'єктів на основі інших, раніше визначених класів [68, 70, 74–77]. Однак, незважаючи на всі переваги успадкування, воно породжує такі проблеми: *проблема винятків*, *проблема надлишковості*, *проблема неоднозначності* та *проблема несумісності* [4, 46, 47]. Таким чином, покращуючи представлення об'єктів та класів в контексті пам'яті, успадкування породжує проблеми, що ускладнюють розробку програмних систем, їх верифікацію та відладку. Проте, незважаючи на це, успадкування також надає великі можливості для проектування архітектури програмних систем, тому успадкування, як ключовий механізм об'єктно-орієнтованого програмування з усіма його перевагами та недоліками, буде розглянуте в наступних розділах.

З Означення 2.1.8 та Означення 2.2.1 випливає, що кожен об'єкт належить як мінімум до одного класу об'єктів. Однак, існують такі об'єкти, що належать до декількох класів одночасно [2, 21, 22]. Розглянемо деякі приклади таких ситуацій.

Приклад 2.2.3. Відомо, що довільні натуральні числа $n_1, \ldots, n_m$ одночасно належать до цілих, раціональних та дійсних чисел, тобто $n_1, \ldots, n_m \in N$, $n_1, \ldots, n_m \in Z$, $n_1, \ldots, n_m \in Q$ та $n_1, \ldots, n_m \in R$. Також відомо, що серед зазначених класів чисел найбільшим є клас дійсних чисел R, оскільки він включає в себе, як натуральні, так і цілі, так і раціональні числа. Іншими словами, клас R визначає числа різних типів, що суперечить концепції однорідного класу, оскільки такі класи дозволяють визначати лише однотипні об'єкти.

Визначимо клас T_R за допомогою наступної специфікації

$$P(T_R) = (p_1(T_R) = vf_1(T_R) = (1),\ p_2(T_R) = vf_2(T_R) = (1),$$
$$p_3(T_R) = vf_3(T_R) = (1),\ p_4(T_R) = vf_4(T_R) = (1),\ p_5(T_R) = vf_5(T_R) = (1)),$$

де $p_1(T_R)$ – це функція верифікації, що визначає властивість "бути цілим числом", $p_2(T_R)$ – це функція верифікації, що визначає властивість "бути натуральним числом", $p_3(T_R)$ – це функція верифікації, що визначає властивість "бути раціональним числом", $p_4(T_R)$ – це функція верифікації, що визначає властивість "бути ці-

лим від'ємним числом", $p_5(T_R)$ – це функція верифікації, що визначає властивість "бути парним числом". Очевидно, що специфікація класу R може містити більшу кількість різних властивостей, проте, у даному випадку достатньо розглянути лише деякі з них.

Тепер розглянемо наступні числа 3, 2.75, -16, 4, -7.48. Очевидно, що всі ці числа належать до різних типів, проте, всі вони є об'єктами класу T_R, тому їх специфікації мають містити ті ж самі властивості, що й специфікація $P(T_R)$ або, іншими словами, всі вони мають задовольняти специфікацію $P(T_R)$. Очевидно, що кожне з наведених чисел задовольняє специфікацію класу T_R по різному, що відображено в Таблиці 2.1.

Таблиця 2.1

Відповідність чисел 3, 2.75, -16, 4 **та** -7.48 **до специфікації класу** T_R

$p_j(A_i)/A_i$	**3**	**2.75**	**−16**	**4**	**−7.48**
$p_1(A_i)$= "бути цілим числом"	1	0	1	1	0
$p_2(A_i)$= "бути натуральним числом"	1	0	0	1	0
$p_3(A_i)$= "бути раціональним числом"	0	1	0	0	1
$p_4(A_i)$= "бути від'ємним цілим числом"	0	0	1	0	1
$p_5(A_i)$= "бути парним числом"	0	0	1	1	1

З точки зору математики, числа 3, 2.75, -16, 4 та -7.48 є об'єктами класу T_R, проте, аналізуючи Таблицю 2.1, можна бачити, що клас T_R не може бути описаний з використанням концепції однорідного класу, оскільки він описує об'єкти різних, іноді взаємовиключаючих або непоєднуваних типів. Іншими словами, не існує жодного числа з класу T_R, для якого б одночасно мали місце властивості $p_1(T_R), \ldots, p_5(T_R)$.

Приклад 2.2.4. Розглянемо клас опуклих чотирикутників T_{CP}, який визначається наступним чином

$$P(T_{CP}) = (p_1(T_{CP}) = (4, \text{сторони}),\ p_2(T_{CP}) = ((v_1, \text{см}), (v_2, \text{см}), (v_3, \text{см}), (v_4, \text{см})),$$
$$p_3(T_{CP}) = (4, \text{кути}),\ p_4(T_{CP}) = ((v_1, ^\circ), (v_2, ^\circ), (v_3, ^\circ), (v_4, ^\circ)),$$
$$p_5(T_{CP}) = vf_5(T_{CP}) = (1)),$$

де $p_1(T_{CP})$ – це кількість сторін фігури, $p_2(T_{CP})$ – це розміри сторін фігури, $p_3(T_{CP})$ – це кількість кутів фігури, $p_4(T_{CP})$ – це градусні міри внутрішніх кутів фігури, а $vf_5(T_{CP})$ – це функція верифікації, яка визначає властивість, що сума всіх внутрішніх кутів чотирикутника рівна 360°, тобто $vf_5(T_{CP}) : p_4(T_{CP}) \to \{0, 1\}$, де $vf_5(T_{CP}) = (v_1(p_4(T_{CP})) + v_2(p_4(T_{CP})) + v_3(p_4(T_{CP})) + v_4(p_4(T_{CP})) = 360)$.

Розглянемо такі об'єкти як квадрат S та прямокутник R, що визначаються наступним чином

$$S = (p_1(S) = (4, \text{сторони}),\ p_2(S) = ((2, \text{см}), (2, \text{см}), (2, \text{см}), (2, \text{см})),$$
$$p_3(S) = (4, \text{кути}),\ p_4(S) = ((90°), (90°), (90°), (90°)),\ p_5(S) = vf_5(S) = (1),$$
$$p_6(S) = vf_6(S) = (1),\ p_7(S) = vf_7(S) = (1)).$$
$$R = (p_1(R) = (4, \text{сторони}),\ p_2(R) = ((2, \text{см}), (3, \text{см}), (2, \text{см}), (3, \text{см})),$$
$$p_3(R) = (4, \text{кути}),\ p_4(R) = ((90°), (90°), (90°), (90°)),\ p_5(R) = vf_5(R) = (1),$$
$$p_6(R) = vf_6(R) = (1),\ p_7(R) = vf_7(R) = (1)),$$

де $p_1(S)$, $p_1(R)$ – це кількість сторін фігури, $p_2(S)$, $p_2(R)$ – це розміри сторін фігури, $p_3(S)$, $p_3(R)$ – це кількість внутрішніх кутів фігури, $p_4(S)$, $p_4(R)$ – це градусні міри внутрішніх кутів фігури, $vf_5(S)$, $vf_5(R)$ – це функції верифікації, які визначають властивість, що сума всіх внутрішніх кутів фігури дорівнює 360°, тобто $vf_5(S) : p_4(S) \to \{0, 1\}$, де

$$vf_5(S) = (v_1(p_4(S)) + v_2(p_4(S)) + v_3(p_4(S)) + v_4(p_4(S)) = 360),$$

і $vf_5(R) : p_4(R) \to \{0, 1\}$, де

$$vf_5(R) = (v_1(p_4(R)) + v_2(p_4(R)) + v_3(p_4(R)) + v_4(p_4(R)) = 360),$$

$vf_6(S)$ – це функція верифікації, яка визначає властивість, що всі сторони фігури рівні по довжині, тобто $vf_6(S) : p_2(S) \to \{0, 1\}$, де

$$vf_6(S) = (v_1(p_2(S)) = v_2(p_2(S)) = v_3(p_2(S)) = v_4(p_2(S))),$$

$vf_6(R)$ – це функція верифікації, яка визначає властивість, що протилежні сто-

рони фігури є рівними, тобто $vf_6(R) : p_2(R) \to \{0, 1\}$, де

$$vf_6(R) = ((v_1(p_2(R)) = v_3(p_2(R))) \wedge (v_2(p_2(R)) = v_4(p_2(R)))),$$

$vf_7(S)$, $vf_7(R)$ – це функції верифікації, які визначають властивість, що всі внутрішні кути фігури мають градусну міру 90°, тобто $vf_7(S) : p_4(S) \to \{0, 1\}$, де

$$vf_7(S) = (v_1(p_4(S)) = v_2(p_4(S)) = v_3(p_4(S)) = v_4(p_4(S)) = 90),$$

і $vf_7(R) : p_4(R) \to \{0, 1\}$, де

$$vf_7(R) = (v_1(p_4(R)) = v_2(p_4(R)) = v_3(p_4(R)) = v_4(p_4(R)) = 90).$$

Очевидно, що будь-які квадрат та прямокутник відносяться до класу опуклих чотирикутників. Проте, проаналізувавши специфікації об'єктів S, R та класу T_{CP}, можна бачити, що, згідно з Означенням 2.2.10, об'єкти S та R не належать до класу T_{CP}, оскільки специфікації $P(S)$, $P(R)$ містять різну кількість властивостей і мають більший рівень деталізації, ніж $P(T_{CP})$. Також варто зауважити, що квадрат – це прямокутник, сторони якого мають однакову довжину, тому для квадрата S виконуються усі ті ж властивості, що й для прямокутника R, проте, для прямокутника R властивість $p_6(S)$ не виконується. Аналогічно для квадрата S та прямокутника R виконуються усі властивості із специфікації $P(T_{CP})$, однак, зворотні твердження не мають місця. Таким чином, клас опуклих чотирикутників також не може бути описаний з використанням концепції однорідного класу, оскільки вона не дозволяє описати різні типи опуклих чотирикутників в рамках одного класу.

Аналізуючи вище наведені приклади, можна бачити, що існує два різних види класів об'єктів – *однорідні*, які описують об'єкти одного типу, та *неоднорідні*, які описують об'єкти різних типів. У зв'язку з цим визначимо концепцію неоднорідного класу об'єктів [2, 21, 22].

Означення 2.2.11. Неоднорідний клас об'єктів T – це кортеж виду

$$T = (Core(T), pr_1(A_1), \ldots, pr_l(A_l)),$$

де $Core(T) = (P(T), F(T))$ – це ядро класу T, що містить властивості та методи, що є спільними для об'єктів $A_1, \ldots, A_l$, а $pr_i(A_i) = (P(A_i), F(A_i))$, де $i = \overline{1, r}$, $r \leqslant l$, – це їх проекції, що містять властивості та методи, які є характерними лише для цих об'єктів.

Концепція неоднорідного класу об'єктів дозволяє описати певну кількість різних типів в рамках об'єктно-орієнтованого підходу, використовуючи для цього лише один клас об'єктів, в той час як представлення кожного нового типу об'єктів в рамках об'єктно-орієнтованого програмування вимагає визначення нового класу або використання механізму успадкування, якщо класи мають спільні властивості та методи.

Аналізуючи Означення 2.2.1 та 2.2.11, можна бачити, що будь-який однорідний клас об'єктів визначає лише один тип об'єктів, тому у цьому випадку поняття *клас* і *тип* є еквівалентними. Проте, неоднорідний клас об'єктів визначає, як мінімум, два різних типи об'єктів в рамках одного класу, тому у цьому випадку поняття *клас* і *тип* мають різні значення. Іншими словами неоднорідний клас об'єктів описує кілька різних типів об'єктів. У зв'язку з цим визначимо поняття типу в рамках неоднорідного класу об'єктів.

Означення 2.2.12. Тип t_i неоднорідного класу об'єктів $T_{t_1,\ldots,t_n}$ – це однорідний клас об'єктів виду $t_i = (Core(T_{t_1,\ldots,t_n}), pr_j(A_j))$, де $Core(T_{t_1,\ldots,t_n})$ – це ядро класу $T_{t_1,\ldots,t_n}$, а $pr_j(A_j)$ – його j-та проекція, де $i = \overline{1, n}$, $j = \overline{1, m}$, $m \leqslant n$.

Розглянемо приклад неоднорідного класу об'єктів [5].

Приклад 2.2.5. Відомо, що такі геометричні фігури як квадрат, прямокутник та ромб відносяться до класу опуклих чотирикутників. Визначимо неоднорідний клас об'єктів T_{SRRb}, що описує ці типи опуклих чотирикутників наступним чином

$$T_{SRRb} = (p_1(T_{SRRb}) = (4, \text{сторони}),\ p_2(T_{SRRb}) = (4, \text{кути}),$$
$$p_3(T_{SRRb}) = vf_3(T_{SRRb}) = (1),$$
$$f_1(T_{SRRb}) = (v_1(p_2(t_i)) + v_2(p_2(t_i)) + v_3(p_2(t_i)) + v_4(p_2(t_i)), \text{см}),\ i \in \{S, R, Rb\},$$
$$p_1(t_S) = ((2, \text{см}), (2, \text{см}), (2, \text{см}), (2, \text{см})),\ p_2(t_S) = ((90^\circ), (90^\circ), (90^\circ), (90^\circ)),$$

$$p_3(t_S) = vf_3(t_S) = (1),\ p_4(t_S) = vf_4(t_S) = (1),\ f_1(t_S) = \left((v_1(p_1(t_S)))^2, \text{см}^2\right),$$
$$p_1(t_R) = ((2, \text{см}), (3, \text{см}), (2, \text{см}), (3, \text{см})),\ p_2(t_R) = ((90^\circ), (90^\circ), (90^\circ), (90^\circ)),$$
$$p_3(t_R) = vf_3(t_R) = (1),\ p_4(t_R) = vf_4(t_R) = (1),$$
$$f_1(t_R) = \left(v_1(p_1(t_R)) \cdot v_2(p_1(t_R)), \text{см}^2\right),\ p_1(t_{Rb}) = ((3, \text{см}), (3, \text{см}), (3, \text{см}), (3, \text{см})),$$
$$p_2(t_{Rb}) = ((80^\circ), (100^\circ), (80^\circ), (100^\circ)),\ p_3(t_{Rb}) = vf_3(t_{Rb}) = (1),$$
$$p_4(t_{Rb}) = vf_4(t_{Rb}) = (1),\ f_1(t_{Rb}) = \left((v_1(p_1(t_{Rb})))^2 \cdot \sin(v_1(p_4(t_{Rb}))), \text{см}^2\right)),$$

де $p_1(T_{SRRb})$ – це кількість сторін фігури, $p_2(T_{SRRb})$ – це кількість внутрішніх кутів фігури, $vf_3(T_{SRRb})$ – це функція верифікації, яка визначає властивість, що сума всіх внутрішніх кутів фігури дорівнює 360°, тобто $vf_3(T_{SRRb}) : p_2(T_{SRRb}) \to \{0, 1\}$, де $vf_3(T_{SRRb}) = (v_1(p_4(t_i)) + v_2(p_4(t_i)) + v_3(p_4(t_i)) + v_4(p_4(t_i)) = 360)$, $i \in \{S, R, Rb\}$; $f_1(T_{SRRb})$ – це метод обрахунку периметра фігури, $p_1(t_S)$, $p_1(t_R)$, $p_1(t_{Rb})$ – це розміри сторін фігур, $p_2(t_S)$, $p_2(t_R)$, $p_2(t_{Rb})$ – це градусні міри внутрішніх кутів фігур, $vf_3(t_S)$ – це функція верифікації, яка визначає властивість, що всі сторони фігури рівні по довжині, тобто $vf_3(t_S) : p_1(t_S) \to \{0, 1\}$, де

$$vf_3(t_S) = (v_1(p_1(t_S)) = v_2(p_1(t_S)) = v_3(p_1(t_S)) = v_4(p_1(t_S))),$$

$vf_4(t_S)$ – це функція верифікації, яка визначає властивість, що всі внутрішні кути фігури мають градусну міру 90°, тобто $vf_4(t_S) : p_2(t_S) \to \{0, 1\}$, де

$$vf_4(t_S) = (v_1(p_2(t_S)) = v_2(p_2(t_S)) = v_3(p_2(t_S)) = v_4(p_2(t_S)) = 90),$$

$f_1(t_S)$ – це метод обчислення площі фігури, $vf_3(t_R)$ – це функція верифікації, яка визначає властивість, що протилежні сторони фігури рівні по довжині, тобто $vf_3(t_R) : p_1(t_R) \to \{0, 1\}$, де

$$vf_3(t_R) = ((v_1(p_1(t_R)) = v_3(p_1(t_R))) \wedge (v_2(p_1(t_R)) = v_4(p_1(t_R)))),$$

$vf_4(t_R)$ – це функція верифікації, яка визначає властивість, що всі внутрішні кути фігури мають градусну міру 90°, тобто $vf_4(t_R) : p_1(t_R) \to \{0, 1\}$, де

$$vf_4(t_R) = (v_1(p_2(t_R)) = v_2(p_2(t_R)) = v_3(p_2(t_R)) = v_4(p_2(t_R)) = 90),$$

$f_1(t_R)$ – це метод обчислення площі фігури, $vf_3(t_{Rb})$ – це функція верифікації, яка визначає властивість рівності усіх сторін фігури, тобто $vf_3(t_{Rb}) : p_1(t_{Rb}) \to \{0, 1\}$, де $vf_3(t_{Rb}) = (v_1(p_1(t_{Rb})) = v_2(p_1(t_{Rb})) = v_3(p_1(t_{Rb})) = v_4(p_1(t_{Rb})))$; $vf_4(t_{Rb})$ – це функція верифікації, яка визначає властивість рівності протилежних внутрішніх кутів фігури, тобто $vf_4(t_{Rb}) : p_2(t_{Rb}) \to \{0, 1\}$, де

$$vf_4(t_{Rb}) = ((v_1(p_2(t_{Rb})) = v_3(p_2(t_{Rb}))) \wedge (v_2(p_2(t_{Rb})) = v_4(p_2(t_{Rb})))),$$

і $f_1(t_{Rb})$ – це метод обчислення площі фігури.

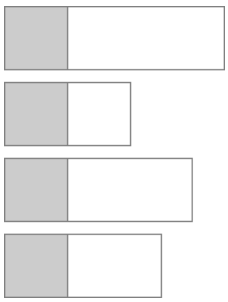
Таким чином, неоднорідний клас об'єктів T_{SRRb} одночасно описує три типи опуклих чотирикутників t_S, t_R та t_{Rb}. Отже, концепція неоднорідного класу об'єктів дозволяє описувати класи, що визначають два і більше різних типів об'єктів. Такий підхід дозволяє більш ефективно представляти знання за рахунок побудови ядра неоднорідного класу.

Дійсно, аналізуючи наведений приклад, можна бачити, що для представлення кожного з типів, які описують квадрати, прямокутники та ромби, потрібно описати по 7 властивостей та 2 методи, тобто в сумі 21 властивість та 6 методів. Використання концепції неоднорідного класу дозволяє представити ці типи, описавши лише 3 властивості та 1 метод для ядра, та по 4 властивості та 1 методу для кожної з проекцій, тобто сумарно 15 властивостей та 4 методи. Таким чином, запропонований підхід дозволяє уникнути дублювання властивостей та методів у представленнях типів, зменшує розміри програмного коду та дозволяє більш ефективно представляти інформацію у базах даних.

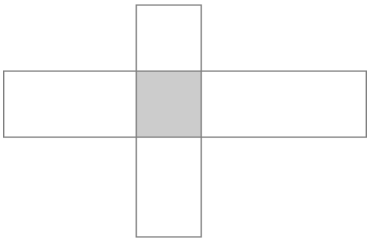
Структура неоднорідного класу об'єктів може бути представлена графічно (див. Рис. 2.1).

На Рис. 2.1a зображені чотири однорідні класи об'єктів, кожен з яких описує певний тип. Квадратом сірого кольору позначені властивості та методи, які є спільними (еквівалентними) для цих класів; квадрат та прямокутники білого кольору позначають властивості та методи, які є характерними лише для конкретного типу об'єктів.

Аналізуючи Рис. 2.1b, можна бачити, що концепція одноядерного неоднорідно-



(a) Однорідні класи об'єктів



(b) Одноядерний неоднорідний клас об'єктів

Рис. 2.1: Визначення типів об'єктів за допомогою однорідних та одноядерного неоднорідного класів об'єктів

го класу об'єктів дозволяє уникнути зайвого дублювання еквівалентних властивостей та методів у специфікаціях та сигнатурах класів. Окрім цього, такий підхід дозволяє визначати мінімум два різних типи об'єктів в рамках одного неоднорідного класу об'єктів. Походження назви цього виду класів об'єктів обумовлене тим, що об'єкти таких класів, на відміну від однорідних, можуть відрізнятися між собою за типом.

2.2.3. Багатоядерні неоднорідні класи об'єктів. Згідно з Означенням 2.2.11, ядро неоднорідного класу об'єктів містить властивості та методи, які є спільними для всіх типів класу, а проекції класу містять властивості та методи, які є характерними лише для конкретних типів. Однак, іноді трапляються ситуації, коли декілька проекцій можуть містити еквівалентні властивості та методи, які не потрапили до ядра класу, оскільки є характерними не для всіх типів класу. У такому випадку буде відбуватися дублювання цих властивостей та методів. Для того, щоб запобігти цьому та дещо оптимізувати структуру класу, введемо поняття *ядра рівня* m [18].

Означення 2.2.13. Ядро рівня m неоднорідного класу $T_{t_1,\ldots,t_n}$ – це кортеж виду $Core^m\left(T_{t_1,\ldots,t_n}\right) = \left(P\left(T_{t_{i_1},\ldots,t_{i_m}}\right), F\left(T_{t_{i_1},\ldots,t_{i_m}}\right)\right)$, де $t_{i_1},\ldots,t_{i_m}$ – це довільні m типів об'єктів з множини типів $\{t_1,\ldots,t_n\}$, де $1 \leqslant m \leqslant n$ і $1 \leqslant i_1 < \cdots < i_m \leqslant n$, а $P\left(T_{t_{i_1},\ldots,t_{i_m}}\right)$ та $F\left(T_{t_{i_1},\ldots,t_{i_m}}\right)$ – це специфікація та сигнатура ядра неоднорідного класу $T_{t_{i_1},\ldots,t_{i_m}}$, які містять строго визначені властивості та методи спільні для усіх

об'єктів типів $t_{i_1}, \ldots, t_{i_m}$.

Оскільки не всі типи неоднорідного класу можуть мати спільні властивості або (та) методи, то багатоядерний неоднорідний клас об'єктів, який визначає n типів може мати k ядер рівня m, де $0 \leqslant k \leqslant C_n^m$.

Тепер узагальнимо Означення 2.2.11 з урахуванням Означень 2.2.12 та 2.2.13, визначивши поняття *багатоядерного неоднорідного класу об'єктів* [18].

Означення 2.2.14. Багатоядерний неоднорідний клас об'єктів $T_{t_1,\ldots,t_n}$ – це кортеж виду

$$T_{t_1,\ldots,t_n} = \Big(Core_1^n(T_{t_1,\ldots,t_n}), Core_1^{n-1}(T_{t_1,\ldots,t_n}), \ldots, Core_{k_{n-1}}^{n-1}(T_{t_1,\ldots,t_n}), \ldots,$$
$$Core_1^1(T_{t_1,\ldots,t_n}), \ldots, Core_{k_1}^1(T_{t_1,\ldots,t_n}), pr_1(t_1), \ldots, pr_n(t_n)\Big),$$

де $Core_1^n(T_{t_1,\ldots,t_n})$ – це ядро рівня n класу $T_{t_1,\ldots,t_n}$, $Core_{i_{n-1}}^{n-1}(T_{t_1,\ldots,t_n})$ – це i_{n-1}-ше ядро рівня $n-1$ класу $T_{t_1,\ldots,t_n}$, де $i_{n-1} = \overline{1, k_{n-1}}$ і $k_{n-1} \leqslant C_n^{n-1}$, $Core_{i_1}^1(T_{t_1,\ldots,t_n})$ – це i_1-ше ядро рівня 1 класу $T_{t_1,\ldots,t_n}$, де $i_1 = \overline{1, k_1}$ і $k_1 \leqslant C_n^1$, $pr_i(t_i)$ – це i-та проекція класу $T_{t_1,\ldots,t_n}$, яка містить слабко визначені властивості та методи, характерні лише для типу t_i, де $i = \overline{1, n}$.

Розглянемо приклад багатоядерного неоднорідного класу об'єктів [5].

Приклад 2.2.6. Відомо, що такі геометричні фігури як квадрат, прямокутник та ромб відносяться до класу опуклих чотирикутників. Визначимо багатоядерний неоднорідний клас об'єктів T_{SRRb}, що описує ці типи опуклих чотирикутників наступним чином

$$T_{SRRb} = (p_1(T_{SRRb}) = (4, \text{сторони}),\ p_2(T_{SRRb}) = (4, \text{кути}),$$
$$p_3(T_{SRRb}) = vf_3(T_{SRRb}) = (1),$$
$$f_1(T_{SRRb}) = (v_1(p_2(t_i)) + v_2(p_2(t_i)) + v_3(p_2(t_i)) + v_4(p_2(t_i)), \text{см}),\ i \in \{S, R, Rb\},$$
$$p_1(T_{SR}) = ((90^\circ), (90^\circ), (90^\circ), (90^\circ)),\ p_2(T_{SR}) = vf_2(t_i) = (1),\ i \in \{S, R\}$$
$$p_1(T_{SRb}) = vf_1(t_i) = (1),\ i \in \{S, Rb\},\ f_1(T_S) = \left(v_1(p_1(t_S)))^2, \text{см}^2\right),$$
$$p_1(T_R) = vf_1(t_R) = (1),\ f_1(T_R) = \left(v_1(p_1(t_R)) \cdot v_2(p_1(t_R)), \text{см}^2\right),$$
$$p_1(T_{Rb}) = vf_1(t_{Rb}) = (1),\ f_1(T_{Rb}) = \left((v_1(p_1(t_{Rb})))^2 \cdot \sin(v_1(p_4(t_{Rb}))), \text{см}^2\right),$$

$$p_1(t_S) = ((2, \text{см}), (2, \text{см}), (2, \text{см}), (2, \text{см})),\ p_1(t_R) = ((2, \text{см}), (3, \text{см}), (2, \text{см}), (3, \text{см})),$$
$$p_1(t_{Rb}) = ((3, \text{см}), (3, \text{см}), (3, \text{см}), (3, \text{см})),\ p_2(t_{Rb}) = ((80^\circ), (100^\circ), (80^\circ), (100^\circ))),$$

де $p_1(T_{SRRb})$ – це кількість сторін фігури, $p_2(T_{SRRb})$ – це кількість внутрішніх кутів фігури, $vf_3(T_{SRRb})$ – це функція верифікації, яка визначає властивість, що сума всіх внутрішніх кутів фігури рівна 360°, тобто $vf_3(T_{SRRb}) : p_3(T_{SRRb}) \to \{0, 1\}$, де $vf_3(T_{SRRb}) = (v_1(p_4(t_i)) + v_2(p_4(t_i)) + v_3(p_4(t_i)) + v_4(p_4(t_i)) = 360)$, $i \in \{S, R, Rb\}$; $f_1(T_{SRRb})$ – це метод обрахунку периметра фігури, $p_1(T_{SR})$ – це градусні міри внутрішніх кутів фігури, $vf_2(T_{SR})$ – це функція верифікації, яка визначає властивість, що всі внутрішні кути фігури мають градусну міру 90°, тобто $vf_2(T_{SR}) : p_1(T_{SR}) \to \{0, 1\}$, де

$$vf_2(T_{SR}) = (v_1(p_1(T_{SR})) = v_2(p_1(T_{SR})) = v_3(p_1(T_{SR})) = v_4(p_1(T_{SR})) = 90),$$

де $i \in \{S, R\}$, $f_1(T_{SR})$ – це метод обчислення площі фігури, $vf_1(T_{SRb})$ – це функція верифікації, яка визначає властивість, що всі сторони фігури мають однакову довжину, тобто $vf_1(T_{SRb}) : p_1(T_{SRb}) \to \{0, 1\}$, де

$$vf_1(T_{SRb}) = (v_1(p_1(t_i)) = v_2(p_1(t_i)) = v_3(p_1(t_i)) = v_4(p_1(t_i))),\ i \in \{S, Rb\},$$

$f_1(T_S)$ – це метод обчислення площі фігури, $vf_1(T_R)$ – це функція верифікації, яка визначає властивість, що протилежні сторони фігури мають однакову довжину, тобто $vf_1(T_R) : p_1(t_R) \to \{0, 1\}$, де

$$vf_1(t_R) = ((v_1(p_1(t_R)) = v_3(p_1(t_R))) \wedge (v_2(p_1(t_R)) = v_4(p_1(t_R)))),$$

$vf_1(T_{Rb})$ – це функція верифікації, яка визначає властивість рівності протилежних внутрішніх кутів фігури, тобто $vf_1(T_{Rb}) : p_1(T_{Rb}) \to \{0, 1\}$, де

$$vf_1(t_{Rb}) = ((v_1(p_2(t_{Rb})) = v_3(p_2(t_{Rb}))) \wedge (v_2(p_2(t_{Rb})) = v_4(p_2(t_{Rb})))),$$

$f_1(T_{Rb})$ – це метод обчислення площі фігури, $p_1(t_S)$, $p_1(t_R)$, $p_1(t_{Rb})$ – це розміри сторін фігур, $p_2(t_{Rb})$ – це градусні міри внутрішніх кутів фігури.

Таблиця 2.2

Структура ядер неоднорідного класу T_{SRRb}

Ядро	**Властивість / Метод**	**Спільні для типів**
$Core_1^3(T_{SRRb})$	$p_1(T_{SRRb})$, $p_2(T_{SRRb})$, $p_3(T_{SRRb})$, $f_1(T_{SRRb})$	t_S, t_R, t_{Rb}
$Core_1^2(T_{SRRb})$	$p_1(T_{SR})$, $p_2(T_{SR})$, $f_1(T_{SR})$	t_S, t_R
$Core_2^2(T_{SRRb})$	$p_1(T_{SRb})$	t_S, t_{Rb}
$Core_1^1(T_{SRRb})$	$f_1(T_S)$	t_S
$Core_2^1(T_{SRRb})$	$p_1(T_R)$, $f_1(T_R)$	t_R
$Core_3^1(T_{SRRb})$	$p_1(T_{Rb})$, $f_1(T_{Rb})$	t_{Rb}

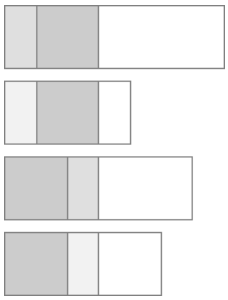
Аналізуючи структуру неоднорідного класу T_{SRRb}, можна бачити, що він дійсно є багатоядерним, оскільки має 1 ядро рівня 3 та по 2 ядра рівнів 2 та 1. Структура усіх ядер класу T_{SRRb} наведена у Таблиці 2.2.

Таким чином, багатоядерний неоднорідний клас об'єктів T_{SRRb} одночасно описує три типи опуклих чотирикутників t_S, t_R та t_{Rb}. Отже, концепція багатоядерного неоднорідного класу об'єктів також дозволяє описувати класи, що визначають два і більше різних типів об'єктів, причому такий підхід дозволяє більш ефективно представляти знання за рахунок побудови ядер рівня m неоднорідного класу.

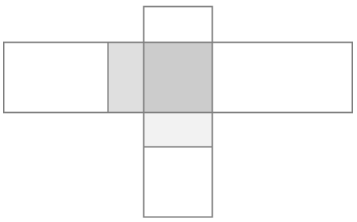
Дійсно, аналізуючи наведений приклад, можна бачити, що для представлення кожного з типів, які описують квадрати, прямокутники та ромби, потрібно описати по 7 властивостей та 2 методи, тобто в сумі 21 властивість та 6 методів. Використання концепції багатоядерних неоднорідних класів дозволяє представити ці типи, описавши лише 3 властивості і 1 метод для ядра $Core_1^3(T_{SRRb})$, 2 властивості і 1 метод для ядра $Core_1^2(T_{SRRb})$, 1 властивість для ядра $Core_2^2(T_{SRRb})$, 1 метод для ядра $Core_1^1(T_{SRRb})$, 1 властивість та 1 метод для ядра $Core_2^1(T_{SRRb})$, 1 властивість та 1 метод для ядра $Core_3^1(T_{SRRb})$, 1 властивість для проекції $pr_1(t_S)$, 1 властивість для проекції $pr_1(t_R)$ та 2 властивості для проекції $pr_1(t_{Rb})$, тобто сумарно 12 властивостей та 4 методи. Таким чином, запропонований підхід дозволяє уникнути дублювання властивостей та методів у представленнях типів, ще більше зменшити розміри програмного коду та ще більш ефективно представляти інформацію у базах даних.

Структура багатоядерного неоднорідного класу об'єктів, може бути представ-

лена графічно (див. Рис. 2.2).



(a) Однорідні класи об'єктів



(b) Трьох'ядерний неоднорідний клас об'єктів

Рис. 2.2: Визначення типів об'єктів за допомогою однорідних та багатоядерного неоднорідного класів об'єктів

На Рис. 2.2a зображені чотири однорідних класи об'єктів, кожен з яких описує певний тип об'єктів. Квадратом сірого кольору позначені властивості та методи, які є спільними (еквівалентними) для цих класів; сірі прямокутники у першого та третього згори класів позначають властивості та методи, спільні для цих класів; сірі прямокутники у другого та четвертого згори класів позначають властивості та методи, спільні для цих класів; а білий квадрат та прямокутники позначають властивості та методи, які є характерними лише для конкретного типу об'єктів.

Аналізуючи Рис. 2.2b, можна бачити, що концепція багатоядерного неоднорідного класу об'єктів дозволяє уникнути зайвого дублювання еквівалентних властивостей та методів у специфікаціях та сигнатурах класів і у певних випадках є ефективнішою за концепцію одноядерного неоднорідного класу.

Застосування одноядерних та багатоядерних неоднорідних класів об'єктів є особливо ефективним у випадку, коли типи, які визначаються в рамках неоднорідного класу, мають спільні (еквівалентні) властивості або (та) методи. Однак, ці концепції також можуть бути використані і у випадку, коли типи неоднорідного класу не мають спільних властивостей та методів. У цьому випадку неоднорідний клас не матиме ядер і складатиметься лише з проекцій типів.

Розглянемо відповідний приклад.

Приклад 2.2.7. Визначимо неоднорідний клас об'єктів T_{CB}, що описуватиме такі типи об'єктів як автомобіль t_{Car} та книга t_{Book}, наступним чином

$$T_{CB} = (p_1(Car) = (v(p_1(Car)), \text{км/год}),\ p_2(Car) = (v(p_2(Car)), \text{т}),$$
$$p_3(Car) = (v(p_3(Car)), \text{л}),\ p_4(Car) = (v(p_4(Car)), \text{марка автомобіля}),$$
$$p_1(Book) = (v(p_1(Book)), \text{жанр книги}),\ p_2(Book) = (v(p_2(Book)), \text{с}),$$
$$p_3(Book) = (v(p_3(Book)), \text{розділи(ів)}),\ p_4(Book) = (v(p_4(Book)), \text{р}),$$
$$p_5(Book) = (v(p_5(Book)), \text{видавництво})),$$

де $p_1(Car)$ – це швидкість автомобіля, $p_2(Car)$ – це маса автомобіля, $p_3(Car)$ – це об'єм двигуна автомобіля, $p_4(Car)$ – це марка автомобіля, $p_1(Book)$ – це жанр книги, $p_2(Book)$ – це кількість сторінок книги, $p_3(Book)$ – це кількість розділів книги, $p_4(Book)$ – це рік видання книги, $p_5(Book)$ – це назва видавництва, де видана книга.

Аналізуючи специфікації типів t_{Car} та t_{Book}, можна бачити, що вони не мають спільних властивостей, тому неоднорідний клас об'єктів T_{CB} немає ядра. У цьому випадку використання концепції неоднорідних класів об'єктів не дає змоги економніше представити типи об'єктів t_{Car} та t_{Book} в рамках неоднорідного класу T_{CB} у порівнянні з представленням цих типів за допомогою однорідних класів об'єктів.

Зауваження 2.2.1. Аналізуючи специфікації типів об'єктів t_{Car} та t_{Book}, наведених у прикладі 2.2.7, можна бачити, що їх рівень деталізації може бути збільшений за рахунок додавання інших властивостей, які характерні цим типам об'єктів. Наприклад, якщо до специфікацій типів t_{Car} та t_{Book} додати таку властивість як вартість, то специфікація неоднорідного класу об'єктів T_{CB} міститиме ядро виду $Core(T_{CB}) = (p_{price}(T_{CB}))$.

Зауваження 2.2.2. З прикладу 2.2.7 та зауваження 2.2.1 виникає запитання, чи існують хоча б два типи об'єктів, які б при максимально можливому рівні деталізації їх представлень не мали б спільних властивостей?

Вище наведені приклади демонструють особливості застосування концепцій

одноядерних та багатоядерних неоднорідних класів об'єктів та їх переваги в контексті представлення знань. Враховуючи це, визначимо поняття належності об'єкта до багатоядерного неоднорідного класу об'єктів.

Означення 2.2.15. Об'єкт $A/(p_1(A), \ldots, p_m(A))$ є об'єктом неоднорідного класу об'єктів

$$T_{t_1,\ldots,t_n} = \Big(Core_1^n(T_{t_1,\ldots,t_n}), Core_1^{n-1}(T_{t_1,\ldots,t_n}), \ldots, Core_{k_{n-1}}^{n-1}(T_{t_1,\ldots,t_n}), \ldots,$$
$$Core_1^1(T_{t_1,\ldots,t_n}), \ldots, Core_{k_1}^1(T_{t_1,\ldots,t_n}), pr_1(t_1), \ldots, pr_n(t_n)\Big),$$

тоді і тільки тоді, коли він є об'єктом типу t_i, тобто $A \in t_i$, де

$$t_i = \Big(Core_1^n(T_{t_1,\ldots,t_n}), Core_1^{n-1}(T_{t_1,\ldots,t_n}), \ldots, Core_{k_{n-1}}^{n-1}(T_{t_1,\ldots,t_n}), \ldots,$$
$$Core_1^1(T_{t_1,\ldots,t_n}), \ldots, Core_{k_1}^1(T_{t_1,\ldots,t_n}), pr_i(t_i)\Big),$$

де $Core_1^n(T_{t_1,\ldots,t_n})$, $Core_{i_{n-1}}^{n-1}(T_{t_1,\ldots,t_n})$ та $Core_{i_1}^1(T_{t_1,\ldots,t_n})$ – це ядро рівня n, i_{n-1}-те та i_1-те ядро рівнів $n-1$ та 1 класу $T_{t_1,\ldots,t_n}$ відповідно, що містить властивості та методи характерні для типу t_i, де $i_{n-1} = \overline{1, k_{n-1}}$, $i_1 = \overline{1, k_1}$ де $k_{n-1} \leqslant C_n^{n-1}$, $k_1 \leqslant C_n^1$, а $pr_i(t_i)$ – це i-та проекція класу $T_{t_1,\ldots,t_n}$, яка містить властивості та методи характерні лише для типу t_i, де $i = \overline{1, n}$.

Іншими словами, об'єкт A є об'єктом багатоядерного неоднорідного класу об'єктів $T_{t_1,\ldots,t_n}$ тоді і тільки тоді, коли він є об'єктом одного з його типів, тобто $A \in t_i$, де $i = \overline{1, n}$.

2.2.4. Порівняльний аналіз. У процесі розвитку об'єктно-орієнтованого програмування, була створена технологія об'єктно-реляційного відображення (ORM) [86–89], яка дозволяє представляти властивості об'єктів та класів у реляційних базах даних, а також перетворювати інформацію з бази даних у об'єкти певних класів, що полегшує розробку об'єктно-орієнтованих програмних систем, що є підтвердженням того, що представлення об'єктів та класів об'єктів у базах даних є актуальною задачею. Однак, збереження в реляційних базах даних лише властивостей об'єктів та класів накладає певні обмеження на передачу та обмін такими базами даних між різними інформаційними системами, оскільки

структура класів, окрім властивостей, також передбачає методи, які зазвичай не зберігаються у базі даних.

Для того, щоб перевірити, як впливає використання концепції одноядерних та багатоядерних неоднорідних класів на ефективність представлення об'єктів та їх типів у реляційних базах даних, був проведений наступний експеримент [5].

Експеримент 2.1. В рамках експерименту було створено чотири реляційні бази даних: перша – з використанням концепції однорідних класів (ОК), друга – з використанням концепції однорідних класів та механізму одиничного успадкування (ОК+ОУ), третя – з використанням концепції одноядерних неоднорідних класів (НОК) та четверта – з використанням концепції багатоядерних неоднорідних класів (БНОК). В якості класів були обрані типи t_S, t_R та t_{Rb} з Прикладу 2.2.5 та Прикладу 2.2.6.

База даних на основі ОК містила 6 таблиць: три таблиці для полів типів t_S, t_R та t_{Rb}, які мали по 28 колонок кожна, і ще три – для методів типів t_S, t_R та t_{Rb}, кожна з яких мала по 5 колонок. Таблиці для збереження полів типів t_S, t_R та t_{Rb} були зв'язані з відповідними таблицями для збереження методів.

База даних на основі ОК+ОУ містила 11 таблиць: одну – для класу t_{SRRb}, яка мала 24 колонки, три – для класів t_S, t_R та t_{Rb}, які мали по 6 колонок кожна, одну – для методів класу t_{SRRb}, яка мала 3 колонки, три – для методів класів t_S, t_R та t_{Rb}, які мали по 3 колонки кожна, і три таблиці – для зв'язків між класом t_{SRRb} та класами t_S, t_R, t_{Rb}, які мали по 3 колонки кожна. Таблиці для збереження полів класів t_{SRRb}, t_S, t_R та t_{Rb} були зв'язані з відповідними таблицями для збереження методів.

База даних на основі НОК містила 7 таблиць: одну – для ядра одноядерного неоднорідного класу $Core(T_{SRRb})$, яка мала 9 колонок, три – для його проекцій $pr_1(S)$, $pr_2(R)$ та $pr_3(Rb)$, кожна з яких мала по 23 колонки, і три таблиці – для методів типів t_S, t_R та t_{Rb}, які мали по 3 колонки кожна. Проекції типів t_S, t_R та t_{Rb} були зв'язані з ядром класу та відповідними таблицями для збереження методів.

База даних на основі БНОК містила 9 таблиць: одну – для ядра $Core_1^3(T_{SRRb})$,

яка мала 9 колонок, одну – для ядра $Core_1^2(T_{SRRb})$, яка мала 3 колонки, одну – для ядра $Core_2^2(T_{SRRb})$, яка мала 11 колонок, одну – для ядра $Core_1^1(T_{SRRb})$, яка мала 3 колонки, одну – для ядра $Core_2^1(T_{SRRb})$, яка мала 5 колонок, одну – для ядра $Core_3^1(T_{SRRb})$, яка мала 5 колонок, і три таблиці – для проекцій $pr_1(S)$, $pr_2(R)$ та $pr_3(Rb)$, які мали 13, 12 та 20 колонок відповідно. Проекції типів t_S, t_R та t_{Rb} були зв'язані з відвідними таблицями для збереження ядер класу T_{SRRb}.

Експеримент проводився в середовищі операційної системи Debian GNU/Linux 9 (Stretch). В якості сервера баз даних було обрано MariaDB 10.1.26. Метою експерименту було порівняння об'ємів фізичної пам'яті, які займають бази даних розгорнуті на сервері та їхні експортовані *.sql-файли.

В ході експерименту було проведено 20 вимірювань. Спершу до кожної бази даних було додано по 4000 об'єктів кожного типу (тобто сумарно 12000 об'єктів за одну вставку), після чого було проведене перше вимірювання. Потім процедура була повторена ще 19 разів. В кінці експерименту бази даних містили по 80000 об'єктів кожного типу (тобто сумарно 240000 об'єктів у кожній БД). Для спрощення процесу автоматичної генерації баз даних та їх подальшого аналізу, усі об'єкти одного типу були проініціалізовані однаково. Результати проведених вимірювань наведені у Таблиці А.1 та Таблиці А.2.

За отриманими результатами були побудовані наступні залежності між розмірами баз даних та кількістю об'єктів, які вони містять:

$$S(DB_{\text{ОК}}) = 0.00029 \cdot Q + 1.6347, \quad S(DB_{\text{ОК+ОУ}}) = 0.000446 \cdot Q + 3.6413,$$
$$S(DB_{\text{НОК}}) = 0.000244 \cdot Q + 1.4151, \quad S(DB_{\text{БНОК}}) = 0.000162 \cdot Q + 0.8177,$$

де $S(DB_i)$ – це розмір бази даних i-го типу, де $i \in \{\text{ОК}, \text{ОК+ОУ}, \text{НОК}, \text{БНОК}\}$, а Q – це кількість об'єктів, які містить база даних. Графічна інтерпретація вимірювань з Таблиці А.1 та встановлених залежностей і їхніх лінійних апроксимацій зображені на Рис. 2.3.

Також були побудовані наступні залежності між розмірами експортованих *.sql-файлів баз даних та кількістю об'єктів, які вони містять:

$$S(FDB_{\text{ОК}}) = 0.000293 \cdot Q - 0.0186, \quad S(FDB_{\text{ОК+ОУ}}) = 0.000328 \cdot Q - 0.1673,$$

$$S(FDB_{\text{НОК}}) = 0.0002 \cdot Q - 0.0182, \quad S(FDB_{\text{БНОК}}) = 0.0000748 \cdot Q - 0.0166,$$

де $S(FDB_i)$ – це розмір експортованого *.sql-файлу бази даних i-го типу, де $i \in \{\text{ОК}, \text{ОК+ОУ}, \text{НОК}, \text{БНОК}\}$, а Q – це кількість об'єктів, які містить база даних. Графічна інтерпретація вимірювань з Таблиці А.2 зображені на Рис. 2.4.

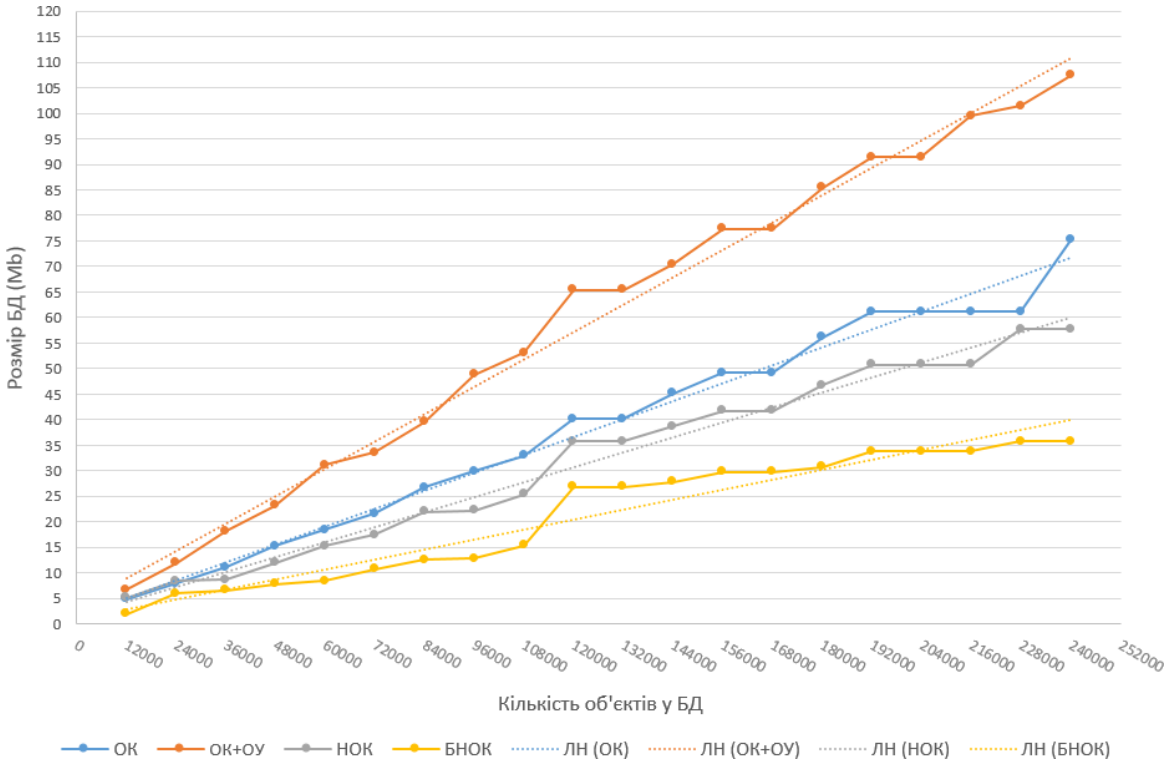


Рис. 2.3: Порівняння об'ємів пам'яті, які використовують бази даних, розгорнуті на сервері

Для порівняння ефективності концепції НОК з концепціями ОК та ОК+ОУ, а також концепції БНОК з концепціями ОК, ОК+ОУ та НОК у контексті збереження інформації в реляційних базах даних та експортування *.sql-файлів цих БД були обчислені відповідні коефіцієнти ефективності (Таблиця А.3 та Таблиця А.4), розрахунки яких були виконані за наступною формулою

$$E(T_1/T_2) = 1 - \frac{T_1}{T_2},$$

де T_1 та T_2 – це об'єми пам'яті необхідні для збереження інформації у реляційній базі даних на основі концепції T_1 та T_2 відповідно. Розрахунки коефіцієнтів були виконані для випадків коли БД містили $n \in \{12000, \ldots, 240000\}$ об'єктів. Після

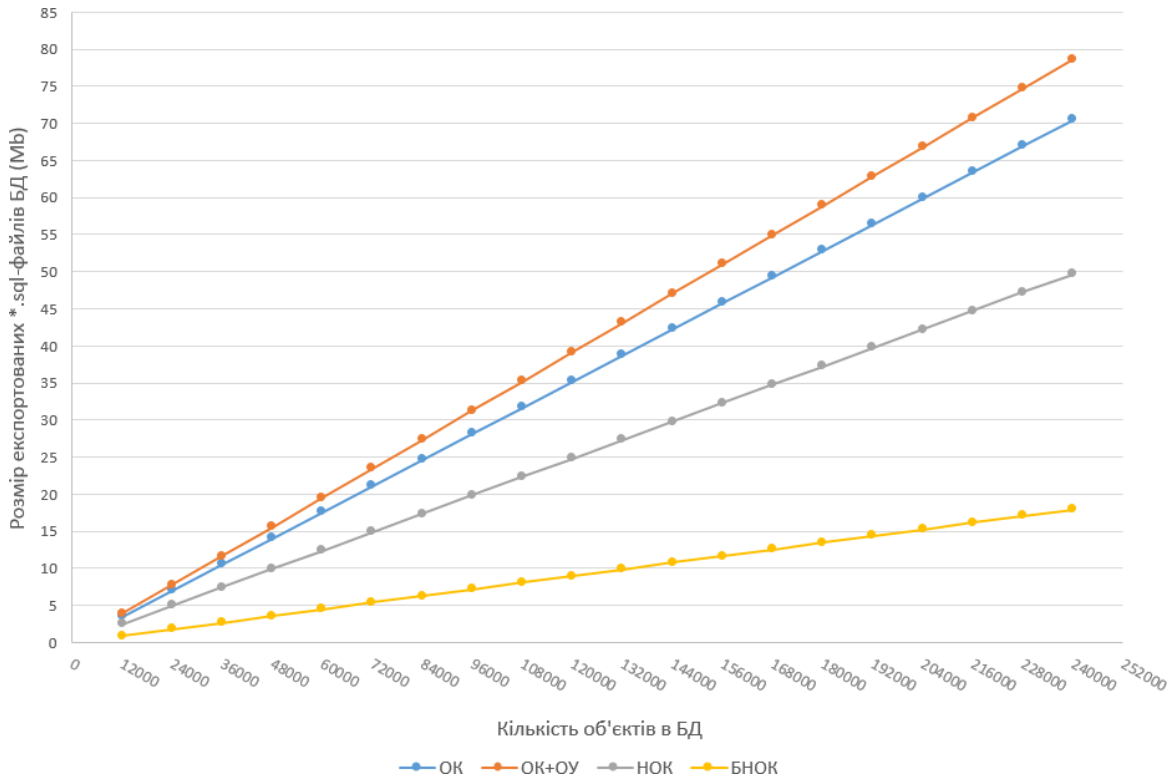


Рис. 2.4: Порівняння об’ємів пам’яті, які використовують експортовані *.sql-файли баз даних

чого були обчислені середні коефіцієнти ефективності для кожного з випадків що аналізувалися.

Підводячи підсумки проведеного експерименту, можна прийти до наступних висновків:

1. використання концепції НОК для збереження у реляційній базі даних n об’єктів, де $n \in \{12000, \ldots, 240000\}$ у середньому зменшує розмір:
   а) бази даних:
      — на 15% у порівнянні з використанням концепції ОК;
      — на 45.3% у порівнянні з використанням концепції ОК+ОУ;
   б) експортованого *.sql-файлу бази даних:
      — на 29.5% у порівнянні з використанням концепції ОК;
      — на 36.6% у порівнянні з використанням концепції ОК+ОУ;
2. SQL-запити до таблиць бази даних на основі НОК виконуються швидше, ніж у випадках використання концепцій ОК та ОК+ОУ;
3. використання концепції БНОК для збереження у реляційній базі даних n

об'єктів, де $n \in \{12000, \ldots, 240000\}$ у середньому зменшує розмір:

а) бази даних:

— на 45% у порівнянні з використанням концепції ОК;

— на 64.8% у порівнянні з використанням концепції ОК+ОУ;

— на 35.2% у порівнянні з використанням концепції НОК;

б) експортованого *.sql-файлу бази даних:

— на 74.6% у порівнянні з використанням концепції ОК;

— на 77% у порівнянні з використанням концепції ОК+ОУ;

— на 64% у порівнянні з використанням концепції НОК;

4. SQL-запити до таблиць бази даних на основі БНОК виконуються швидше, ніж у випадках використання концепцій ОК, ОК+ОУ та НОК;

Отже, використання концепції БНОК у деяких випадках є значно ефективнішим за використання концепції НОК, ОК та ОК+ОУ.

2.3. Відношення між класами об'єктів

Між класами об'єктів можуть існувати певні відношення, які задають деякі зв'язки між об'єктами та їх класами та між різними класи об'єктів. Виділяють відношення *еквівалентності*, *агрегації*, *узагальнення* та *спеціалізації*, які дозволяють певним чином порівнювати класи об'єктів між собою та будувати концептуальні (класові) ієрархії [40, 68, 70, 71, 74, 76]. Тому визначимо ці типи відношень для однорідних та неоднорідних класів об'єктів.

2.3.1. Еквівалентність. Відношення еквівалентності дозволяє проводити порівняння класів об'єктів між собою та ідентифікувати еквівалентні або відмінні між собою класи об'єктів. Для визначення відношення еквівалентності між однорідними та неоднорідними класами об'єктів, введемо поняття еквівалентності їхніх специфікацій та сигнатур. Оскільки класи об'єктів можна розглядати, як деякі абстрактні об'єкти або прототипи для конкретних об'єктів, то еквівалентність кількісних та якісних властивостей класів об'єктів вводиться подібно до еквівалентності відповідних властивостей об'єктів.

Означення 2.3.1. Дві кількісні властивості $p_1(T_1)$ та $p_1(T_2)$ класів об'єктів T_1 та T_2 є еквівалентними, тобто $Eq(p_1(T_1), p_1(T_2)) = 1$, тоді і тільки тоді, коли

1. $(v_i(p_1(T_1)) = v_j(p_1(T_2))) \wedge (u_i(p_1(T_1)) = u_j(p_1(T_2)))$, де $i = \overline{1,n}$, $j = \overline{1,m}$, якщо $n = m$ і $p_1(T_1)$ та $p_1(T_2)$ – строго визначені;
2. $u_i(p_1(T_1)) = u_j(p_1(T_2))$, де $i = \overline{1,n}$, $j = \overline{1,m}$, якщо $n = m$ і $p_1(T_1)$ та $p_1(T_2)$ – слабко визначені.

Означення 2.3.2. Дві якісні властивості $vf_1(T_1)$ та $vf_1(T_2)$ класів об'єктів T_1 та T_2, що визначаються як $vf_1(p_{i_1}(T_1), \ldots, p_{i_k}(T_1))$ та $vf_1(p_{j_1}(T_2), \ldots, p_{j_w}(T_2))$, є еквівалентними, тобто $Eq(vf_1(T_1), vf_1(T_2)) = 1$, тоді і тільки тоді, коли

$$(k = w) \wedge \left(vf_1^{T_1}(p_{i_1}(T_1), \ldots, p_{i_k}(T_1)) = vf_1^{T_2}(p_{i_1}(T_1), \ldots, p_{i_k}(T_1))\right) \wedge$$
$$\wedge \left(vf_1^{T_1}(p_{j_1}(T_2), \ldots, p_{j_w}(T_2)) = vf_1^{T_2}(p_{j_1}(T_2), \ldots, p_{j_w}(T_2))\right).$$

Для зручності формулювання подальших означень, введемо поняття міри неоднорідності класів об'єктів.

Означення 2.3.3. Міра неоднорідності $h(T)$ класу об'єктів T – це натуральне число, яке обчислюється наступним чином

$$h(T) = \begin{cases} 0, & n = 1; \\ n - 1, & n > 1; \end{cases}$$

де n – це кількість типів, які визначає клас T.

Таким чином, згідно з Означенням 2.3.3, якщо $h(T) = 0$, то клас T є однорідним, якщо $h(T) > 0$, – неоднорідним.

Оскільки специфікації однорідних та неоднорідних класів об'єктів мають різну структуру, то сформулюємо поняття еквівалентності специфікацій окремо для кожного виду класів об'єктів.

Означення 2.3.4. Специфікації $P(T_1)$ та $P(T_2)$ однорідних класів об'єктів T_1 та T_2 є еквівалентними, тобто $P(T_1) = P(T_2)$, тоді і тільки тоді, коли $D(T_1) = D(T_2)$ і $(\forall p_i(T_1)\ \exists! p_j(T_2)) \wedge (\forall p_j(T_2)\ \exists! p_i(T_1))\ \big|\ Eq(p_i(T_1), p_j(T_2)) = 1$, де $p_i(T_1)$ та $p_j(T_2)$ – це i-та та j-та властивості класів T_1 та T_2 відповідно, де $i = \overline{1, D(T_1)}$ і $j = \overline{1, D(T_2)}$.

Означення 2.3.5. Специфікації $P(T_1)$ та $P(T_2)$ неоднорідних класів об'єктів T_1 та T_2 є еквівалентними, тобто $P(T_1) = P(T_2)$, тоді і тільки тоді, коли $h(T_1) = h(T_2)$ і

$$\left(\forall p_{w_i}\left(t_i^1\right)\ \exists! p_{k_j}\left(t_j^2\right)\right) \wedge \left(\forall p_{k_j}\left(t_j^2\right)\ \exists! p_{w_i}\left(t_i^1\right)\right) \wedge \left(D\left(t_i^1\right) = \right.$$
$$\left. = D\left(t_j^2\right)\right)\ \Big|\ Eq\left(p_{w_i}\left(t_i^1\right), p_{k_j}\left(t_j^2\right)\right) = 1\Big),$$

де $p_{w_i}\left(t_i^1\right)$, $w_i = \overline{1, D\left(t_i^1\right)}$ – це w_i-та властивість типу t_i^1, $i = \overline{1, h(T_1)+1}$, який визначається неоднорідним класом T_1, а $p_{k_j}\left(t_j^2\right)$, $k_j = \overline{1, D\left(t_j^2\right)}$, – це k_j-та властивість типу t_j^2, $j = \overline{1, h(T_2)+1}$, який визначається неоднорідним класом T_2.

Тепер введемо поняття еквівалентності методів класів об'єктів, що визначається подібно до еквівалентності методів об'єктів.

Означення 2.3.6. Два методи $f_1(T_1)$ та $f_2(T_2)$ класів об'єктів T_1 та T_2 еквівалентні, тобто $Eq(f_1(T_1), f_2(T_2)) = 1$, тоді і тільки тоді, коли

$$(f_1(T_1) = f_2(T_1)) \wedge (f_1(T_2) = f_2(T_2)).$$

Оскільки сигнатури однорідних та неоднорідних класів об'єктів, як і їхні специфікації, мають різну структуру, сформулюємо поняття еквівалентності сигнатур для кожного виду класів об'єктів.

Означення 2.3.7. Сигнатури $F(T_1)$ та $F(T_2)$ однорідних класів об'єктів T_1 та T_2 є еквівалентними, тобто $F(T_1) = F(T_2)$, тоді і тільки тоді, коли $func(T_1) = func(T_2)$ і $(\forall f_i(T_1)\ \exists! f_j(T_2)) \wedge (\forall f_j(T_2)\ \exists! f_i(T_1))\ \Big|\ Eq(f_i(T_1), f_j(T_2)) = 1$, де $f_i(T_1)$, $i = \overline{1, func(T_1)}$ – це i-тий метод класу T_1, а $f_j(T_2)$, $j = \overline{1, func(T_2)}$ – це j-тий метод класу T_2.

Означення 2.3.8. Сигнатури $F(T_1)$ та $F(T_2)$ неоднорідних класів об'єктів T_1 та T_2 є еквівалентними, тобто $F(T_1) = F(T_2)$, тоді і тільки тоді, коли $h(T_1) = h(T_2)$ і

$$\left(\forall f_{w_i}\left(t_i^1\right)\ \exists! f_{k_j}\left(t_j^2\right)\right) \wedge \left(\forall f_{k_j}\left(t_j^2\right)\ \exists! f_{w_i}\left(t_i^1\right)\right) \wedge \left(func\left(t_i^1\right) = \right.$$
$$\left. = func\left(t_j^2\right)\right)\ \Big|\ Eq\left(f_{w_i}\left(t_i^1\right), f_{k_j}\left(t_j^2\right)\right) = 1\Big),$$

де $f_{w_i}\left(t_i^1\right)$, $w_i = \overline{1, func\left(t_i^1\right)}$ – це w_i-тий метод типу t_i^1, $i = \overline{1, h(T_1)+1}$, який визначається класом T_1, а $f_{k_j}\left(t_j^2\right)$, $k_j = \overline{1, func\left(t_j^2\right)}$ – це k_j-тий метод типу t_j^2, $j = \overline{1, h(T_2)+1}$, який визначається класом T_2.

Тепер, використовуючи вище сформульовані означення, визначимо поняття еквівалентності для однорідних та неоднорідних класів об'єктів.

Означення 2.3.9. Два однорідні класи об'єктів T_1 та T_2 є еквівалентними, тобто $Eq(T_1, T_2) = 1$, тоді і тільки тоді, коли $(P(T_1) = P(T_2)) \wedge (F(T_1) = F(T_2))$, де $P(T_1)$, $P(T_2)$ – це специфікації класів T_1 та T_2, а $F(T_1)$, $F(T_2)$ – їх сигнатури.

Означення 2.3.10. Два неоднорідні класи об'єктів T_1 та T_2 є еквівалентними, тобто $Eq(T_1, T_2) = 1$, тоді і тільки тоді, коли $\left(\forall t_i^1\ \exists! t_j^2\right) \wedge \left(\forall t_j^2\ \exists! t_i^1\right) \mid Eq\left(t_i^1, t_j^2\right) = 1$, де t_i^1, $i = \overline{1, h(T_1)+1}$ – це i-тий тип класу T_1, а t_j^2, $j = \overline{1, h(T_2)+1}$ – це j-тий тип класу T_2.

Тепер покажемо що відношення Eq визначені в Означенні 2.3.9 та 2.3.10, є відношеннями еквівалентності.

Твердження 2.3.1. Відношення $Eq(T_1, T_2) \Leftrightarrow (T_1 \equiv T_2)$, де T_1 та T_2 – це довільні однорідні класи об'єктів, є відношенням еквівалентності визначеним на множині класів об'єктів $C = \{T_1, T_2, T_3 \dots\}$.

Доведення. Відомо що відношення еквівалентності R, яке задане на деякій множині X, є рефлексивним (aRa), симетричним $(aRb \Rightarrow bRa)$ та транзитивним $((aRb) \wedge (bRc) \Rightarrow (aRc))$ бінарним відношенням $(\forall a, b, c \in X)$ [90–92]. Отже $\forall T_1, T_2, T_3 \in C$ повинні виконуватися рефлексивність $(T_1 \equiv T_1)$, симетричність $((T_1 \equiv T_2) \Rightarrow (T_2 \equiv T_1))$ та транзитивність $((T_1 \equiv T_2) \wedge (T_2 \equiv T_3) \Rightarrow (T_1 \equiv T_3))$.

Рефлексивність. Припустимо, що $\forall T_1 \in C$ має місце $T_1 \not\equiv T_1$, тоді згідно з Означенням 2.3.4 та 2.3.7 приходимо до суперечності, оскільки один і той же однорідний клас об'єктів не може мати дві відмінні специфікації та сигнатури, у результаті чого отримуємо що $\forall T_1 \in C,\ T_1 \equiv T_1$.

Симетричність. Якщо $\forall T_1, T_2 \in C$ має місце $T_1 \equiv T_2$, то має місце й $T_2 \equiv T_1$, оскільки Означення 2.3.4 та 2.3.7 встановлюють бієкцію між специфікаціями та сигнатурами однорідних класів об'єктів T_1 та T_2, у результаті чого отримуємо що

$(T_1 \equiv T_2) \Rightarrow (T_2 \equiv T_1)$.

Транзитивність. Припустимо, що $\forall T_1, T_2, T_3 \in C$ має місце $T_1 \equiv T_2$ та $T_2 \equiv T_3$ і $T_1 \not\equiv T_3$. З того що $T_1 \not\equiv T_3$, слідує те що $T_1 \neq T_3$. Однак з $T_2 \equiv T_3$ слідує що $T_2 = T_3$, як із $T_1 \equiv T_2$ слідує що $T_1 = T_2$, що призводить до суперечності оскільки $(T_1 = T_2 = T_3) \not\Rightarrow (T_1 \neq T_3)$, тому $(T_1 \equiv T_2) \wedge (T_2 \equiv T_3) \Rightarrow (T_1 \equiv T_3)$.

В результаті чого отримуємо що $Eq(T_1, T_2) \Leftrightarrow (T_1 \equiv T_2)$, де T_1 та T_2 – це довільні однорідні класи об'єктів, є відношенням еквівалентності визначеним на множині класів об'єктів $C = \{T_1, T_2, T_3 \dots\}$, що й треба було довести. □

Твердження 2.3.2. Відношення $Eq(T_1, T_2) \Leftrightarrow (T_1 \equiv T_2)$, де T_1 та T_2 – це довільні неоднорідні класи об'єктів, є відношенням еквівалентності визначеним на множині класів об'єктів $C = \{T_1, T_2, T_3 \dots\}$.

Доведення. Відомо що відношення еквівалентності R, яке задане на деякій множині X, є рефлексивним (aRa), симетричним $(aRb \Rightarrow bRa)$ та транзитивним $((aRb) \wedge (bRc) \Rightarrow (aRc))$ бінарним відношенням $(\forall a, b, c \in X)$ [90–92]. Отже $\forall T_1, T_2, T_3 \in C$ повинні виконуватися рефлексивність $(T_1 \equiv T_1)$, симетричність $((T_1 \equiv T_2) \Rightarrow (T_2 \equiv T_1))$ та транзитивність $((T_1 \equiv T_2) \wedge (T_2 \equiv T_3) \Rightarrow (T_1 \equiv T_3))$.

Рефлексивність. Припустимо, що $\forall T_1 \in C$ має місце $T_1 \not\equiv T_1$, тоді згідно з Означенням 2.3.5 та 2.3.8 приходимо до суперечності, оскільки кожен тип одного і того ж неоднорідного класу об'єктів не може мати дві відмінні специфікації та сигнатури, у результаті чого отримуємо що $\forall T_1 \in C,\ T_1 \equiv T_1$.

Симетричність. Якщо $\forall T_1, T_2 \in C$ має місце $T_1 \equiv T_2$, то має місце й $T_2 \equiv T_1$, оскільки Означення 2.3.5 та 2.3.8 встановлюють бієкцію між специфікаціями та сигнатурами типів неоднорідних класів об'єктів T_1 та T_2, у результаті чого отримуємо що $(T_1 \equiv T_2) \Rightarrow (T_2 \equiv T_1)$.

Транзитивність. Припустимо, що $\forall T_1, T_2, T_3 \in C$ має місце $T_1 \equiv T_2$ та $T_2 \equiv T_3$ і $T_1 \not\equiv T_3$. З того що $T_1 \not\equiv T_3$, слідує те що $T_1 \neq T_3$. Однак з $T_2 \equiv T_3$ слідує що $T_2 = T_3$, як із $T_1 \equiv T_2$ слідує що $T_1 = T_2$, що призводить до суперечності оскільки $(T_1 = T_2 = T_3) \not\Rightarrow (T_1 \neq T_3)$, тому $(T_1 \equiv T_2) \wedge (T_2 \equiv T_3) \Rightarrow (T_1 \equiv T_3)$.

В результаті чого отримуємо що $Eq(T_1, T_2) \Leftrightarrow (T_1 \equiv T_2)$, де T_1 та T_2 – це

довільні неоднорідні класи об'єктів, є відношенням еквівалентності визначеним на множині класів об'єктів $C = \{T_1, T_2, T_3 \dots\}$, що й треба було довести. □

2.3.2. Агрегація. Відношення агрегації (*is-a-part-of*, *part-of-whole*, *has-a-part*) відіграє важливу роль у проектуванні та побудові концептуальних (класових) ієрархій, оскільки воно дозволяє описувати ситуації коли один клас об'єктів є складовою частиною іншого класу об'єктів [40, 68, 70, 71, 74, 76]. Враховуючи що поняття одноядерного та багатоядерного неоднорідного класу об'єктів створюють різницю між поняттями *клас* та *тип* об'єктів, перш ніж перейти до визначення відношення агрегації (включення) між однорідними та неоднорідними класами об'єктів, визначимо для них поняття підспецифікації, підсигнатури та підтипу.

Оскільки специфікації однорідних та неоднорідних класів об'єктів мають різну структуру, сформулюємо поняття включення специфікації або підспецифікації окремо для кожного виду класів об'єктів.

Означення 2.3.11. Специфікація $P(T_1)$ однорідного класу об'єктів T_1 є підспецифікацією специфікації $P(T_2)$ однорідного класу об'єктів T_2, тобто $P(T_1) \subseteq P(T_2)$, тоді і тільки тоді, коли $D(T_1) \leqslant D(T_2)$ і

$$\forall p_i(T_1)\ \exists! p_j(T_2) \mid Eq(p_i(T_1), p_j(T_2)) = 1,$$

де $p_i(T_1)$ та $p_j(T_2)$ – це i-та та j-та властивості класів T_1 та T_2 відповідно, де $i = \overline{1, D(T_1)}$ і $j = \overline{1, D(T_2)}$.

Означення 2.3.12. Специфікація $P(T_1)$ неоднорідного класу об'єктів T_1 є підспецифікацією специфікації $P(T_2)$ неоднорідного класу об'єктів T_2, тобто $P(T_1) \subseteq P(T_2)$, тоді і тільки тоді, коли $h(T_1) \leqslant h(T_2)$ і

$$\left(\forall p_{w_i}\left(t_i^1\right)\ \exists! p_{k_j}\left(t_j^2\right)\right) \wedge \left(D\left(t_i^1\right) \leqslant D\left(t_j^2\right)\right) \mid Eq\left(p_{w_i}\left(t_i^1\right), p_{k_j}\left(t_j^2\right)\right) = 1,$$

де $p_{w_i}\left(t_i^1\right)$, $w_i = \overline{1, D\left(t_i^1\right)}$ – це w_i-та властивість типу t_i^1, $i = \overline{1, h(T_1) + 1}$, який визначається класом T_1, а $p_{k_j}\left(t_j^2\right)$, $k_j = \overline{1, D\left(t_j^2\right)}$ – це k_j-та властивість типу t_j^2, $j = \overline{1, h(T_2) + 1}$, який визначається класом T_2.

Тепер визначимо поняття підсигнатури однорідного та неоднорідного класів об'єктів. Оскільки сигнатури однорідних та неоднорідних класів об'єктів, як і їх специфікації, мають різну структуру, сформулюємо поняття включення сигнатури або підсигнатури окремо для кожного виду класів об'єктів.

Означення 2.3.13. Сигнатура $F(T_1)$ однорідного класу об'єктів T_1 є підсигнатурою сигнатури $F(T_2)$ однорідного класу об'єктів T_2, тобто $F(T_1) \subseteq F(T_2)$, тоді і тільки тоді, коли $func(T_1) \leqslant func(T_2)$ і $\forall f_i(T_1)\ \exists! f_j(T_2) \mid Eq(f_i(T_1), f_j(T_2)) = 1$, де $f_i(T_1)$, $i = \overline{1, func(T_1)}$ – це i-тий метод класу T_1, а $f_j(T_2)$, $j = \overline{1, func(T_2)}$ – це j-тий метод класу T_2.

Означення 2.3.14. Сигнатура $F(T_1)$ неоднорідного класу об'єктів T_1 є підсигнатурою сигнатури $F(T_2)$ неоднорідного класу об'єктів T_2, тобто $F(T_1) \subseteq F(T_2)$, тоді і тільки тоді, коли $h(T_1) \leqslant h(T_2)$ і

$$\left(\forall f_{w_i}\left(t_i^1\right)\ \exists! f_{k_j}\left(t_j^2\right)\right) \wedge \left(func\left(t_i^1\right) \leqslant func\left(t_j^2\right)\right) \mid Eq\left(f_{w_i}\left(t_i^1\right), f_{k_j}\left(t_j^2\right)\right) = 1,$$

де $f_{w_i}\left(t_i^1\right)$, $w_i = \overline{1, func\left(t_i^1\right)}$ – це w_i-тий метод типу t_i^1, $i = \overline{1, h(T_1)+1}$, який визначається класом T_1, а $f_{k_j}\left(t_j^2\right)$, $k_j = \overline{1, func\left(t_j^2\right)}$ – це k_j-тий метод типу t_j^2, $j = \overline{1, h(T_2)+1}$, який визначається класом T_2.

Тепер, використовуючи поняття підспецифікації та підсигнатури однорідного та неоднорідного класів об'єктів, визначимо поняття підтипу та підкласу.

Означення 2.3.15. Тип об'єктів t_1 є підтипом типу об'єктів t_2, тобто $t_1 \subseteq t_2$, тоді і тільки тоді, коли $(P(t_1) \subseteq P(t_2)) \wedge (F(t_1) \subseteq F(t_2))$, де $P(t_1)$, $P(t_2)$ – це специфікації типів t_1 та t_2, а $F(t_1)$, $F(t_2)$ – їх сигнатури.

Означення 2.3.16. Клас об'єктів T_1 є підкласом класу об'єктів T_2, тобто $T_1 \subseteq T_2$, тоді і тільки тоді, коли $\forall t_i^1\ \exists! t_j^2 \mid t_i^1 \subseteq t_j^2$, де t_i^1, $i = \overline{1, h(T_1)+1}$ – це i-тий тип класу T_1, а t_j^2, $j = \overline{1, h(T_2)+1}$ – це j-тий тип класу T_2.

Тепер покажемо що відношення $\subseteq$, визначене в Означенні 2.3.16, є відношенням часткового порядку (включення).

Твердження 2.3.3. Відношення $Sub(T_1, T_2) \Leftrightarrow (T_1 \subseteq T_2)$, де T_1 та T_2 – це довільні класи об'єктів, є відношенням часткового порядку визначеним на множині

класів об'єктів $C = \{T_1, T_2, T_3 \dots\}$.

Доведення. З означення відношення часткового порядку слідує що відношення R, яке задане на деякій множині X, є рефлексивним (aRa), антисиметричним $((aRb) \wedge (bRa) \Rightarrow (a = b))$ та транзитивним $((aRb) \wedge (bRc) \Rightarrow (aRc))$ бінарним відношенням $(\forall a, b, c \in X)$ [90–93]. Отже $\forall T_1, T_2, T_3 \in C$ повинні виконуватися рефлексивність $(T_1 \subseteq T_1)$, антисиметричність $((T_1 \subseteq T_2) \wedge (T_2 \subseteq T_1) \Rightarrow (T_1 = T_2))$ та транзитивність $((T_1 \subseteq T_2) \wedge (T_2 \subseteq T_3) \Rightarrow (T_1 \subseteq T_3))$.

Рефлексивність. З доведень Тверджень 2.3.1 та 2.3.2 слідує, що $T_1 \equiv T_1$, тому згідно з Означенням 2.3.16 отримуємо що $T_1 \subseteq T_1$.

Антисиметричність. Якщо має місце $T_1 \subseteq T_2$, то з Означення 2.3.16 слідує що $\forall t_i^1,\ \exists! t_j^2 \mid t_i^1 \subseteq t_j^2$ і $h(T_1) \leqslant h(T_2)$. Якщо має місце $T_2 \subseteq T_1$, то з Означення 2.3.16 слідує що $\forall t_j^2,\ \exists! t_i^1 \mid t_j^2 \subseteq t_i^1$ і $h(T_2) \leqslant h(T_1)$. Якщо має місце $(T_1 \subseteq T_2) \wedge (T_2 \subseteq T_1)$, то з Означення 2.3.16 слідує що

$$\left(\forall t_i^1,\ \exists! t_j^2\right) \wedge \left(\forall t_j^2,\ \exists! t_j^1\right) \ \Big|\ \left(t_i^1 \subseteq t_j^2\right) \wedge \left(t_j^2 \subseteq t_i^1\right) \wedge (h(T_1) \leqslant h(T_2)) \wedge (h(T_2) \leqslant h(T_1)),$$

звідки отримуємо що $(h(T_1) = h(T_2)) \wedge (T_1 \equiv T_2) \Rightarrow (T_1 = T_2)$.

Транзитивність. Припустимо що $(T_1 \subseteq T_2) \wedge (T_2 \subseteq T_3) \Rightarrow (T_1 \not\subseteq T_3)$. Тоді з $(T_1 \subseteq T_2) \wedge (T_2 \subseteq T_3)$ та Означення 2.3.16 слідує що

$$\left(\left(\forall t_i^1,\ \exists! t_j^2 \mid t_i^1 \subseteq t_j^2\right) \wedge (h(T_1) \leqslant h(T_2))\right) \wedge \left(\left(\forall t_j^2,\ \exists! t_k^3 \mid t_j^2 1 \subseteq t_k^3\right) \wedge (h(T_2) \leqslant h(T_3))\right).$$

Однак з $T_1 \not\subseteq T_3$ слідує що $\forall t_i^1,\ \nexists t_k^3 \mid t_i^1 \subseteq t_k^3$, а отже має місце $\forall t_i^1,\ \nexists t_j^2 \mid t_i^1 \subseteq t_j^2$ і $\forall t_j^2,\ \nexists t_k^3 \mid t_j^2 \subseteq t_k^3$, як наслідок отримуємо суперечність. Таким чином має місце $(T_1 \subseteq T_2) \wedge (T_2 \subseteq T_3) \Rightarrow (T_1 \subseteq T_3)$.

В результаті чого отримуємо що $Sub(T_1, T_2) \Leftrightarrow (T_1 \subseteq T_2)$, де T_1 та T_2 – це довільні класи об'єктів, є відношенням часткового порядку (включення) визначеним на множині класів об'єктів $C = \{T_1, T_2, T_3 \dots\}$, що й треба було довести. □

2.3.3. Узагальнення та спеціалізація. Відношення узагальнення (*is-a*, *is-an*) і обернене до нього відношення спеціалізації (*a-kind-of*, *an-instance-of*) також відіграють важливу роль у проектуванні та побудові концептуальних (класових)

ієрархій, оскільки вони дозволяють зв'язувати між собою класи різного рівня абстракції [40, 74, 76]. Більш загальні класи об'єктів завжди мають меншу кількість властивостей та(або) методів ніж класи які їх спеціалізують (уточнюють, конкретизують). Як наслідок суперкласи будуть підкласами усіх класів які їх спеціалізують, у зв'язку з чим відношення узагальнення та спеціалізації можна визначити на основі раніше визначеного відношення агрегації.

В рамках об'єктно-орієнтованого програмування на основі класів відрізняють поняття підтипів на агрегації, у тому сенсі що агрегована частина одного класу є наслідком успадкування іншого класу, коли ж мова йде про підтипи, то маються на увазі вкладені класи. Однак не зважаючи на ці відмінності, як і внутрішні типи в рамках класу так і агреговані частини зовнішніх класів задовольняють Означення 2.3.15 та Означення 2.3.16.

Оскільки однорідні та неоднорідні класи об'єктів мають різну структуру, визначимо відношення узагальнення та спеціалізації для усіх можливих випадків.

Означення 2.3.17. Однорідний клас об'єктів T_1 узагальнює однорідний клас об'єктів T_2 тоді і тільки тоді коли $t_1 \subset t_2$, де t_1, t_2 – це типи, які визначають класи T_1 та T_2 відповідно.

Означення 2.3.18. Однорідний клас об'єктів T_2 спеціалізує однорідний клас об'єктів T_1 тоді і тільки тоді коли $t_1 \subset t_2$, де t_1, t_2 – це типи, які визначають класи T_1 та T_2 відповідно.

Означення 2.3.19. Неоднорідний клас об'єктів T_1 узагальнює неоднорідний клас об'єктів T_2 тоді і тільки тоді коли $\forall t_i^1,\ \exists! t_j^2 \mid \left(t_i^1 \subseteq t_j^2\right) \wedge \left(h(T_1) < h(T_2)\right)$, де t_i^1, $i = \overline{1, h(T_1)+1}$ – це i-тий тип класу T_1, а t_j^2, $j = \overline{1, h(T_2)+1}$ – це j-тий тип класу T_2.

Означення 2.3.20. Неоднорідний клас об'єктів T_2 спеціалізує неоднорідний клас об'єктів T_1 тоді і тільки тоді коли $\forall t_i^1,\ \exists! t_j^2 \mid \left(t_i^1 \subseteq t_j^2\right) \wedge \left(h(T_1) < h(T_2)\right)$, де t_i^1, $i = \overline{1, h(T_1)+1}$ – це i-тий тип класу T_1, а t_j^2, $j = \overline{1, h(T_2)+1}$ – це j-тий тип класу T_2.

Означення 2.3.21. Однорідний клас об'єктів T_1 узагальнює неоднорідний

клас об'єктів T_2 тоді і тільки тоді коли $t_1 \subseteq Core^n(T_2)$, де t_1 – це тип, який визначає клас T_1, а $Core^n(T_2)$ – це ядро рівня n класу T_2, де n – це кількість типів, які визначає клас T_2.

Означення 2.3.22. Однорідний клас об'єктів T_2 спеціалізує неоднорідний клас об'єктів T_1 тоді і тільки тоді коли $t_1 \subseteq Core^n(T_2)$, де t_1 – це тип, який визначає клас T_1, а $Core^n(T_2)$ – це ядро рівня n класу T_2, де n – це кількість типів, які визначає клас T_2.

Оскільки усі вище зазначені відношення узагальнення та спеціалізації визначені на основі відношення агрегації, яке є відношенням часткового порядку (включення), то вони також є відношеннями часткового порядку (включення).

2.4. Успадкування

Успадкування є одним з найважливіших структурних елементів об'єктно-орієнтованої парадигми програмування, оскільки є ключовим засобом для побудови класових (концептуальних) ієрархій [46, 47, 68, 71, 74–77, 94–96]. Основна ідея успадкування полягає у передачі деяким класам частини функціональності інших класів, при цьому кожний клас який отримує певну функціональність від інших класів розширює її, додаючи певний набір властивостей та методів, в наслідок чого кожен наступний клас у ланцюгу успадкування отримуватиме доповнену (розширену) функціональність найбільш загального класу. У цьому випадку більш загальні класи називаються суперкласами, а менш загальні або більш спеціалізовані – підкласами.

Однак така термінологія може ввести в оману, оскільки за визначенням суперклас має меншу кількість властивостей та методів у порівнянні із його підкласами. Таким чином, якщо деякий клас T_3 успадковує клас T_2, який у свою чергу успадковує клас T_1, то ці класи перебувають у наступному відношенні $T_1 \subseteq T_2 \subseteq T_3$, а не у відношенні $T_3 \subseteq T_2 \subseteq T_1$, яке є семантичною інтерпретацією включення. Іншими словами клас T_2 спеціалізує клас T_1, а клас T_3 спеціалізує (розширює, конкретизує) клас T_2 і як наслідок клас T_1 [96]. Отже відношення успадкування

для класів визначається через обернене відношення включення класів.

2.4.1. Механізми успадкування. В рамках об'єктно-орієнтованого програмування на основі класів розглядають два основних типи успадкування: *одиничне* та *множинне* [68, 70, 74–77, 94, 96]. У випадку одиничного успадкування, клас наслідує лише один суперклас, у той час як у випадку множинного успадкування, клас одночасно наслідує $n \geqslant 2$ різних суперкласів.

Означення 2.4.1. Одиничне успадкування (Sg) – це процес передачі батьківським класом властивостей та(або) методів одному дочірньому класу.

Означення 2.4.2. Множинне успадкування (M) – це процес передачі кількома батьківськими класами властивостей та(або) методів одному дочірньому класу.

Використання механізму успадкування (як одиничного так і множинного) дозволяє ефективну та гнучку побудову класових (концептуальних) ієрархій, за рахунок повторного використання спільних частин класів (частин програмного коду). Що дозволяє уникати дублювання властивостей та методів класів і як наслідок – нераціонального використання пам'яті. Наявність побудованої ієрархії класів визначає між класами відношення включення $\subseteq$, яке є відношенням часткового порядку.

Однак існує й інша класифікація механізмів успадкування, в рамках якої виділяють *сильне* та *слабке* успадкування [73, 97, 98]. У першому випадку більш спеціалізований клас успадковує усі властивості та методи більш загального класу з мірою $\mu = 1$, у той час як у другому випадку – відбувається успадкування з мірою $0 < \mu \leqslant 1$. Така класифікація дозволяє розглядати процес успадкування у контексті того, з якою мірою у більш спеціалізованому класі будуть виражені успадковані ним властивості та методи.

Означення 2.4.3. Сильне успадкування (St) – це процес передачі батьківським класом властивостей та(або) методів дочірнім класам з мірою $\mu = 1$.

Означення 2.4.4. Слабке успадкування (W) – це процес передачі батьківським класом властивостей та(або) методів дочірнім класам з мірою $0 < \mu \leqslant 1$.

2.4.2. Проблеми успадкування. Не зважаючи на всі переваги використання такого потужного механізму, як успадкування, іноді саме це, може призвести до серйозних проблем пов'язаних з архітектурою концептуальних ієрархій побудованих з використанням успадкування. На сьогодні відомі чотири основних проблеми, які виникають у наслідок використання механізму успадкування, відомі як *проблеми надлишковості*, *винятків*, *неоднозначності* та *семантичних конфліктів* [4, 46, 47, 68, 71].

Проблема 1 (Надлишковість). Проблема надлишковості полягає у тому, що успадкування в концептуальних ієрархіях передбачає транзитивну передачу властивостей та методів від найбільш загальних класів до найбільш конкретних, в результаті чого найбільш спеціалізовані класи можуть успадкувати надлишкові властивості та методи і як наслідок усі об'єкти таких класів матимуть структури аналогічні до структури своїх класів, що може призвести до появи у них додаткових можливостей, які не передбачені їхньою семантикою, а також до нераціонального використання пам'яті.

Проблема 2 (Винятки). Проблема винятків полягає у тому, що суперкласи можуть містити деякі властивості та методи, які не притаманні усім їх підкласам, в результаті чого підкласи можуть успадкувати певні властивості, які суперечать їхній семантиці, що може призвести до неоднозначності бази знань та процесу логічного виведення з її використанням.

Проблема 3 (Неоднозначність). Проблема неоднозначності полягає у тому, що множинне успадкування в концептуальних ієрархіях передбачає ситуації коли один клас може успадковувати властивості та методи від $n \geqslant 2$ непов'язаних між собою суперкласів одного або різних рівнів, які можуть містити властивості та методи з однаковими іменами, в результаті чого, для підкласу виникає проблема вибору таких властивостей та методів.

Проблема 4 (Несумісність). Проблема несумісності полягає у тому, що множинне успадкування в концептуальних ієрархіях передбачає ситуації коли один клас може успадкувати від своїх суперкласів семантично непоєднувані (вза-

ємовиключні, несумісні) властивості.

На даний час не існує єдиного підходу до вирішення зазначених проблем у загальному випадку, а більшість запропонованих рішень несумісні з парадигмою об'єктно-орієнтованого програмування та представлення знань. У зв'язку з чим, методи вирішення цих проблем, зокрема в рамках об'єктно-орієнтованого представлення знань є досить актуальною задачею.

2.4.3. Класифікація механізмів успадкування. Однак окрім двох вище зазначених класифікацій механізмів успадкування існує ще одна. У випадку одиничного та множинного успадкування більш спеціалізовані класи наслідують усі властивості та методи більш загальних класів і це є причиною багатьох проблем пов'язаних з використанням механізму успадкування. Якщо класи не будуть успадковувати усі без винятку властивості та методи батьківських класів, то це допоможе уникати поява надлишковості, винятків та неоднозначності у класових ієрархіях. Таким чином можна виділити ще такі два типи успадкування, як *повне* та *часткове*. У першому випадку більш спеціалізовані класи успадковують усі властивості та методи більш загальних класів, а у другому – відбувається успадкування лише певної частини властивостей та методів.

Означення 2.4.5. Повне успадкування (F) – це процес передачі батьківським класом усіх своїх властивостей та(або) методів дочірнім класам.

Означення 2.4.6. Часткове успадкування (P) – це процес передачі батьківським класом певних частин своїх властивостей та(або) методів дочірнім класам.

Процес часткового успадкування має певні особливості, пов'язані із внутрішніми структурними зв'язками класів, що успадковуються. Існують такі класи об'єктів, деякі властивості та методи яких визначені через інші властивості та методи цих класів, відповідно. У зв'язку з цим визначимо поняття структурних компонент зв'язності типу L_p та L_f.

Означення 2.4.7. Структурна компонента зв'язності $L_p(T)$ класу об'єктів T – це підмножина властивостей $L_p(T) = \{p_{i_1}(T), \ldots, p_{i_n}(T)\}$ класу T, така що $1 \leqslant i_1 \leqslant \cdots \leqslant i_n \leqslant D(T)$ і для якої $\exists! p(T) \in L_p(T)$ таке що визначається через

$|L_p(T)| - 1$ властивість класу T.

Означення 2.4.8. Структурна компонента зв'язності $L_f(T)$ класу об'єктів T – це підмножина методів $L_f(T) = \{f_{j_1}(T), \ldots, f_{j_m}(T)\}$ класу T, така що $1 \leqslant j_1 \leqslant \cdots \leqslant j_m \leqslant func(T)$ і для якої $\exists! f(T) \in L_f(T)$ таке що визначається через $|L_f(T)| - 1$ метод класу T.

Взаємозв'язки які утворюють структурні компоненти зв'язності типу L_p та L_f впливають на процес успадкування, оскільки якщо один клас частково успадковує властивості та(або) методи від іншого класу, то в ході успадкування він може отримати від батьківського класу лише деякі частини компонент типу L_p та(або) L_f. У результаті чого виникає розрив внутрішніх структурних зв'язків між властивостями та методами класу в рамках дочірніх класів. Для запобігання виникненню таких ситуацій у процесі часткового успадкування, необхідно формувати перелік успадкованих властивостей та методів за рахунок структурно незалежних властивостей та методів або структурних компонент зв'язності типу L_p та L_f.

Усі три вище розглянуті класифікації механізмів успадкування, можна об'єднати в одну загальну класифікацію (Таблиця 2.3) [4].

Таблиця 2.3

Класифікація механізмів успадкування

Успадкування							
Одиничне				Множинне			
Повне		Часткове		Повне		Часткове	
Сильне	Слабке	Сильне	Слабке	Сильне	Слабке	Сильне	Слабке

Така класифікація не лише об'єднує в собі різні механізми успадкування, а й дозволяє утворювати більш складні його типи. Розглянемо Таблицю 2.3, як бінарне дерево та розглянемо два його піддерева, де коренем лівого піддерева є одиничне успадкування, а правого – множинне. Усі найдовші ланцюги цих дерев утворюватимуть наступних вісім, найбільш загальних типів успадкування:

1. Одиничне повне сильне (SgFSt – Single Full Strong)

2. Одиничне повне слабке (SgFW – Single Full Weak)
3. Одиничне часткове сильне (SgPSt – Single Partial Strong)
4. Одиничне часткове слабке (SgPW – Single Partial Weak)
5. Множинне повне сильне (MFSt – Multiple Full Strong)
6. Множинне повне слабке (MFW – Multiple Full Weak)
7. Множинне часткове сильне (MPSt – Multiple Partial Strong)
8. Множинне часткове слабке (MPW – Multiple Partial Weak)

Оскільки на даний час парадигма об'єктно-орієнтованого програмування (зокрема її конкретні реалізації в рамках об'єктно-орієнтованих мов програмування) не адаптована до представлення нечітких об'єктів, класів та відношень між ними, розглянемо більш детально лише "сильні" типи успадкування.

2.4.3.1. SgFSt-успадкування. Основна ідея SgFSt-успадкування полягає у тому, що будь-який клас ієрархії може мати не більше одного суперкласу, від якого він успадковує з мірою $\mu = 1$ усі властивості та методи. Саме цей тип успадкування використовується у всіх сучасних об'єктно-орієнтованих мовах програмування під назвою "одиничне успадкування".

Приклад 2.4.1. Розглянемо класичний приклад ієрархій успадкування де через використання SgFSt-успадкування виникає проблема винятків.

```
class Birds{
  public:
    bool has_wings = true;
    bool can_fly = true;
}

class Penguins : public Birds{
  public:
    eats = 'fish';
}
```

```
class Eagles : public Birds{
  public:
    eats = 'mouse';
}
```

Аналізуючи ієрархію, можна бачити, що клас `Birds` є суперкласом для класів `Eagles` та `Penguins` від якого вони успадковують властивості `has_wings` та `can_fly`. В результаті чого, усі об'єкти класу `Penguins` успадкують властивість

`can_fly = true`, що є наглядним прикладом проблеми винятків, оскільки пінгвіни не можуть літати на відміну від орлів. Причиною проблеми є значення властивості `can_fly`, оскільки воно є різним для різних представників класу `Birds`.

2.4.3.2. SgPSt-успадкування. Основна ідея SgPSt-успадкування полягає у тому, що будь-який клас ієрархії може мати не більше одного суперкласу, від якого він успадковує з мірою $\mu = 1$ лише певну частину властивостей та методів.

Приклад 2.4.2. Розглянемо ієрархію успадкування із попереднього прикладу та додамо до неї використання SgPSt-успадкування.

```
class Birds{
  public:
    bool has_wings = true;
    bool can_fly = true;
}

class Penguins : public Birds(has_wings){
  public:
    eats = 'fish';
    bool can_fly = false;
}
```

```
class Eagles : public Birds{
  public:
    eats = 'mouse';
}
```

Аналізуючи ієрархію, можна бачити, що класи `Eagles` та `Penguins` наслідують клас `Birds`, використовуючи SgFSt-успадкування та SgPSt-успадкування, відповідно, у результаті чого клас `Penguins` успадковує від класу `Birds` лише властивість `has_wings`, у той час як клас `Eagles` – усі властивості. Таким чином використання SgPSt-успадкування, дозволяє уникати успадкування більш конкретними класами властивостей та методів більш загальних класів, які не є властивими для усіх їхніх підкласів, чим самим уникаючи проблеми винятків. Також SgPSt-успадкування дозволяє уникати проблеми надлишковості, оскільки дозволяє підкласам успадковувати лише необхідні властивості та(або) методи.

2.4.3.3. MFSt-успадкування. Основна ідея MFSt-успадкування полягає у тому, що будь-який клас ієрархії може мати $n \geqslant 2$ суперкласів, від яких він успадковує з мірою $\mu = 1$ усі властивості та методи. Саме цей тип успадкування

використовується у всіх[1] сучасних об'єктно-орієнтованих мовах програмування під назвою "множинне успадкування".

Приклад 2.4.3. Розглянемо класичний приклад ієрархій успадкування де через використання MFSt-успадкування виникають проблеми неоднозначності та несумісності.

```
class Pacifist{
  public:
    use_physical_violence = false;
    against_war = true;
}

class Soldier{
  public:
    use_physical_violence = true;
    can_kill_enemy = true;
}
```

```
class Christian : public Pacifist{
  public:
    believe_in_god = true;
    can_kill_enemy = false;
}

class Ukrainian_Soldier :
      public Soldier, public Christian{
  public:
    nationality = 'ukrainian';
}
```

Аналізуючи наведену ієрархію, можна бачити що клас `Ukrainian_Soldier` має два суперкласи `Soldier` та `Christian`. Однак у результаті створення об'єктів класу `Ukrainian_Soldier` виникає проблема неоднозначності, оскільки вони успадкують властивості `can_kill_enemy = true` та `use_physical_violence = true` від класу `Soldier`, а також та властивості `can_kill_enemy = false` та `use_physical_violence = false` від класу `Christian`, який у свою чергу, по транзитивності успадковує властивість `use_physical_violence` від класу `Pacifist`. В наслідок чого виникає питання які саме властивості і від якого із суперкласів мають успадкувати об'єкти класу `Ukrainian_Soldier`.

При створенні об'єктів класу `Ukrainian_Soldier`, усі вони успадкують властивості `believe_in_god = true` та `against_war = true` від класу `Christian`, а також властивості `use_physical_violence = true` та `can_kill_enemy = true` від класу `Soldier`. В наслідок чого виникає проблема несумісності, оскільки усі об'єкти класу `Ukrainian_Soldier` одночасно успадковують несумісні властивості

[1]Маються на увазі лише ті об'єктно-орієнтовані мови програмування, які підтримують механізм множинного успадкування.

`can_kill_enemy = true` та `against_war = true`.

2.4.3.4. MPSt-успадкування. Основна ідея MPSt-успадкування полягає у тому, що будь-який клас ієрархії може мати $n \geqslant 2$ суперкласів, від яких він успадковує з мірою $\mu = 1$ усі властивості та методи або лише їх певну частину.

Приклад 2.4.4. Розглянемо ієрархію успадкування із попереднього прикладу та додамо до неї використання MPSt-успадкування.

```
class Pacifist{
  public:
    use_physical_violence = false;
    against_war = true;
}

class Soldier{
  public:
    use_physical_violence = true;
    can_kill_enemy = true;
}
```

```
class Christian : public Pacifist{
  public:
    believe_in_god = true;
    can_kill_enemy = false;
}

class Ukrainian_Soldier :
      public Soldier,
      public Christian(believe_in_god){
  public:
    nationality = 'ukrainian';
}
```

Аналізуючи ієрархію, можна бачити що клас `Ukrainian_Soldier` наслідує класи `Soldier` та `Christian` використовуючи MPSt-успадкування, у результаті чого клас `Ukrainian_Soldier` успадковує від класу `Christian` лише властивість `believe_in_god`. Таким чином, MPSt-успадкування дозволяє уникати проблем неоднозначності та несумісності, оскільки клас `Ukrainian_Soldier` успадковує від класу `Christian` лише властивість `believe_in_god = true`.

2.5. Операції над об'єктами та класами об'єктів

Однією з характерних особливостей більшості об'єктно-орієнтованих мов програмування є можливість оперувати тільки об'єктами класів, які описані в коді програми на момент її виконання [68, 74]. Це означає, що створення нових класів та їх об'єктів, протягом виконання програми, можливе лише тоді, коли це передбачає логіка програми. Іншими словами, програмний код визначає та фіксує

певну *базову множину класів*, які програма може використовувати під час свого виконання.

Протягом останніх десятиліть з активним розвитком досліджень в галузі штучного інтелекту виникла потреба у розробці так званих адаптивних програмних систем, тобто систем, які пристосовуються або адаптуються до специфіки середовища в якому вони функціонують. Одним із шляхів створення такого роду програмних систем є створення *механізму самореконструкції програм*, за допомогою якого програми зможуть самостійно вносити корективи у свою структуру та логіку, шляхом динамічного редагування програмних кодів, під час власного виконання.

Однією з підзадач для реалізації цього механізму для об'єктно-орієнтованих мов програмування, є генерація класів або коду під час виконання програми (RTCG) [99–103]. Однак, варто розрізняти динамічну генерацію кодів на основі певних шаблонів, які також входять до так званої множини базових класів програми і генерацію "нових класів", які явно не описані в коді програми навіть у вигляді шаблонів.

Генерація нових класів об'єктів також має важливе значення для інтелектуального аналізу даних, зокрема для процесу видобування *нових* або *неявних знань* з множини *базових знань* [104–106]. Такий процес, у певному сенсі, є подібним до процесу логічного виведення теорем на основі певної системи аксіом у логічних моделях представлення знань. Видобуті неявні знання можуть використовуватися інтелектуальними інформаційними системами для прийняття певних рішень.

На сьогодні існують деякі підходи до реалізації механізмів RTCG для деяких об'єктно-орієнтованих мов програмування, зокрема C++ [102, 107], Java [108] та C# [109]. Однак у випадку Java та C#, генерація кодів під час виконання програми зводиться до маніпуляцій з байт-кодом і є можливою лише для платформ Java та .NET відповідно. Альтернативним підходом до практичної реалізації механізму RTCG в об'єктно-орієнтованих мовах програмування є введення так званих *конструкторів класів*, які даватимуть змогу утворювати (генерувати) нові класи, шляхом певних операцій над базовими класами. Це дозволить утворювати нові

класи об'єктів під час виконання програмного коду, не виконуючи ніяких проміжних маніпуляцій.

На сьогодні, парадигма ООП дозволяє використовувати класи об'єктів лише, як абстрактні шаблони для створення конкретних об'єктів (екземплярів), а в якості операції над класами можна розглядати хіба що наслідування. Ситуація з об'єктами класів трохи краща, оскільки до них можна викликати методи, які визначені у батьківських класах, однак ці методи дозволяють лише оперувати із значеннями властивостей об'єктів. У зв'язку з цим, визначимо деякі операції над об'єктами та класами об'єктів, результатами яких будуть *нові* об'єкти, класи та множини об'єктів. В залежності від характеру дії, операції над класами глобально можна поділити на два типи – *експлуататори* та *модифікатори* [21, 22].

2.5.1. Експлуататори. У загальному поняття експлуататора можна визначити наступним чином.

Означення 2.5.1. Експлуататор – це функція (метод), яка (-ий) використовує об'єкти та класи об'єктів у якості незмінних аргументів для створення нових об'єктів, класів, множин та мультимножин об'єктів.

В рамках сучасного ООП більшість методів класів є *локально-замкнутими* по відношенню до об'єктів класу і не можуть бути застосованими до об'єктів різних типів, за винятком випадків викликів методів батьківських класів та перевантаження методів класів [68, 74, 75, 94, 95]. Більшість експлуататорів також є локально замкненими на певний тип чи множину типів об'єктів. Однак існують універсальні експлуататори, які можна застосовувати до будь-яких об'єктів та класів об'єктів.

2.5.1.1. Універсальні експлуататори класів об'єктів. Тепер використовуючи Означення 2.2.1, 2.2.11 та 2.2.14, визначимо універсальні експлуататори об'єднання, однорідного та неоднорідного перетину, різниці, симетричної різниці та клонування для класів об'єктів.

Означення 2.5.2. Об'єднання $T_1 \cup \cdots \cup T_n$ класів об'єктів $T_1, \ldots, T_n$, $n \geqslant 2$, які визначають $l_1, \ldots, l_n$ типів об'єктів відповідно, $l_1 \geqslant 1, \ldots, l_n \geqslant 1$, – це клас об'єктів

$T_{1...m}$, що визначає типи $t_1, \ldots, t_m$, де $\forall t_{w_1}, t_{w_2} \mid w_1 \neq w_2$, має місце $Eq(t_{w_1}, t_{w_2}) = 0$, де $w_1, w_2 = \overline{1,m}$, $1 \leqslant m \leqslant l_1 + \cdots + l_n$, і для якого має місце наступне твердження $\left(\forall t_i^k,\ \exists! t_j^{1...m}\right) \wedge \left(\forall t_j^{1...m},\ \exists t_i^k\right) \ \Big|\ Eq\left(t_i^k, t_j^{1...m}\right) = 1$, де t_i^k – це i-й тип класу T_k, де $i = \overline{1,l_k}$, $k = \overline{1,n}$, а $t_j^{1...m}$ – це j-й тип класу $T_{1...m}$, $j = \overline{1,m}$.

В залежності від еквівалентності та ступеня неоднорідності класів $T_1, \ldots, T_n$, клас $T_{1...m}$ буде однорідним або неоднорідним.

Означення 2.5.3. Однорідний перетин $T_1 \cap \cdots \cap T_n$ класів об'єктів $T_1, \ldots, T_n$, $n \geqslant 2$, які визначають $l_1, \ldots, l_n$ типів об'єктів відповідно, $l_1 \geqslant 1, \ldots, l_n \geqslant 1$, – це однорідний клас об'єктів T_A, який визначає тип t_A, для якого має місце наступне твердження $(\forall t_i,\ t_A \subseteq t_i) \wedge \left(\nexists t_B \ \Big|\ (t_A \subset t_B) \wedge (t_B \subseteq t_i)\right)$, де t_i – це тип що визначається класом T_i, де $i = \overline{1,n}$. Однорідний перетин класів об'єктів $T_1 \cap \cdots \cap T_n$ існує тоді і тільки тоді, коли $\exists p_{i_1}(t_1), \ldots, \exists p_{i_n}(t_n) \mid Eq(p_{i_1}(t_1), \ldots, p_{i_n}(t_n)) = 1$, де $p_{i_k}(t_k)$ – це i_k-та властивість типу t_k, де $i_k = \overline{1, D(t_k)}$ і $k = \overline{1,n}$.

Операція однорідного перетину класів об'єктів дозволяє утворювати нові класи об'єктів лише за умови існування властивостей та(або) методів, які є спільними одночасно для усіх типів кожного з класів. Проте іноді трапляються ситуації, коли таких властивостей та(або) методів не існує для n типів об'єктів, але вони можуть існувати для m типів об'єктів, де $1 \leqslant m < n$. Також можливі випадки, коли спільні властивості та(або) методи можуть існувати одночасно для певної кількості різних груп типів об'єктів. У зв'язку з чим, є доцільним визначити експлуататор неоднорідного перетину класів об'єктів.

Означення 2.5.4. Неоднорідний перетин $T_1 \Cap \cdots \Cap T_n$ класів об'єктів $T_1, \ldots, T_n$, $n \geqslant 2$, які визначають типи $t_1^1, \ldots, t_{l_1}^1, \ldots, t_1^n, \ldots, t_{l_n}^n$ відповідно, $l_1 \geqslant 1, \ldots, l_n \geqslant 1$, – це багатоядерний неоднорідний клас об'єктів T, що має наступну структуру

$$T = \left(Core_1^n(T), Core_1^{n-1}(T), \ldots, Core_{w_{n-1}}^{n-1}(T), \ldots, Core_1^1(T), \ldots, Core_{w_1}^1(T)\right),$$

де кожне ядро $Core_{i_m}^m(T)$, $1 \leqslant m \leqslant n$, $i_m = \overline{1, w_m}$, де $w_m \leqslant C_n^m$ визначає підтип $t_{i_m}^{1,\ldots,m}$ такий що $\forall t_{i_m}^{1,\ldots,m},\ \exists (t_1, \ldots, t_m) \ \Big|\ \left(t_{i_m}^{1,\ldots,m} = t_1 \cap \cdots \cap t_m\right)$. Неоднорідний перетин класів об'єктів $T_1 \Cap \cdots \Cap T_n$ існує тоді і тільки тоді, коли

$\exists\left(p_{i_1}(t_1)\right),\ldots,\exists\left(p_{i_m}(t_m)\right)\ \mid\ Eq(p_{i_1}(t_1),\ldots,p_{i_m}(t_m))=1$, де $2\leqslant m\leqslant n$, $p_{i_k}(t_k)$ – це i_k-та властивість типу t_k, де $i_k=\overline{1,D(t_k)}$ і $k=\overline{1,n}$.

Таким чином, у результаті застосування експлуататора неоднорідного перетину до класів об'єктів $T_1,\ldots,T_n$ буде утворений багатоядерний неоднорідний клас об'єктів T, який у загальному випадку визначатиме множину типів об'єктів $\left\{t_1^n,t_1^{n-1},\ldots,t_{w_{n-1}}^{n-1},\ldots,t_1^1,\ldots,t_{w_1}^1\right\}$, де $0\leqslant w_i\leqslant C_n^i$, а $i=\overline{1,n-1}$.

Означення 2.5.5. Різниця $T_1\setminus T_2$ між класом об'єктів T_1 та класом об'єктів T_2, що визначають типи $t_1^1,\ldots,t_n^1$ та $t_1^2,\ldots,t_m^2$ відповідно, де $n,m\geqslant 1$, – це клас об'єктів $T_{1\setminus 2}$, що визначає типи $t_1^{1\setminus 2},\ldots,t_k^{1\setminus 2}$, такі що $k\leqslant n+m$ і має місце наступне твердження

$$\forall t_i^{1\setminus 2},t_w^2,\ \exists t_j^1\ \mid\ \left(t_i^{1\setminus 2}\subset t_j^1\right)\wedge\nexists\left(t_i^{1\setminus 2}\cap t_w^2\right)\wedge\left(\nexists t^{1\setminus 2}\ \mid\ \left(t_i^{1\setminus 2}\subset t^{1\setminus 2}\right)\wedge\right.$$
$$\left.\wedge\left(t^{1\setminus 2}\subseteq t_j^1\right)\wedge\nexists\left(t^{1\setminus 2}\cap t_w^2\right)\right),$$

де $i=\overline{1,k}$, $j=\overline{1,n}$, $w=\overline{1,m}$. Різниця між класом об'єктів T_1 та класом об'єктів T_2 існує тоді і тільки тоді, коли $\exists p_{i_1}\left(t_j^1\right),\exists p_{i_2}\left(t_w^2\right)\ \mid\ Eq\left(p_{i_1}\left(t_j^1\right),p_{i_2}\left(t_w^2\right)\right)=0$, де $p_{i_1}\left(t_j^1\right)$ – це i_1 властивість типу t_j^1, де $i_1=\overline{1,D\left(t_j^1\right)}$, а $p_{i_2}\left(t_w^2\right)$ – це i_2 властивість типу t_w^2, де $i_2=\overline{1,D\left(t_w^2\right)}$.

Отже, у результаті застосування експлуататора різниці до класів об'єктів T_1 та T_2 буде утворений клас об'єктів $T_{1\setminus 2}$, який визначатиме типи об'єктів $t_1^{1\setminus 2},\ldots,t_k^{1\setminus 2}$, які міститимуть властивості та(або) методи, які є характерними для типів $t_1^1,\ldots,t_n^1$ і не є такими для типів $t_1^2,\ldots,t_m^2$. В залежності від ступеня неоднорідності класів об'єктів T_1 та T_2, клас $T_{1\setminus 2}$ буде однорідним або неоднорідним.

Означення 2.5.6. Симетрична різниця $T_1\div T_2$ між класами об'єктів T_1 та T_2, що визначають типи $t_1^1,\ldots,t_n^1$ та $t_1^2,\ldots,t_m^2$ відповідно, де $n,m\geqslant 1$, – це неоднорідний клас об'єктів $T_{1\div 2}$, що визначає типи об'єктів $t_1^{1\setminus 2},\ldots,t_w^{1\setminus 2}$ та $t_1^{2\setminus 1},\ldots,t_q^{2\setminus 1}$, такі що $w+q\leqslant n+m$. Симетрична різниця між класами об'єктів T_1 та T_2 існує тоді і тільки тоді, коли $\exists p_{i_1}\left(t_i^1\right),\exists p_{i_2}\left(t_j^2\right)\ \mid\ Eq\left(p_{i_1}\left(t_i^1\right),p_{i_2}\left(t_j^2\right)\right)=0$, де $p_{i_1}\left(t_i^1\right)$ – це i_1 властивість типу t_i^1, де $i_1=\overline{1,D\left(t_i^1\right)}$, $i=\overline{1,n}$ а $p_{i_2}\left(t_j^2\right)$ – це i_2 властивість типу t_j^2, де $i_2=\overline{1,D\left(t_j^2\right)}$, $j=\overline{1,m}$.

Таким чином, у результаті застосування експлуататора симетричної різниці до класів об'єктів T_1 та T_2 буде утворений неоднорідний клас об'єктів $T_{1\div 2}$, який визначає типи об'єктів $t_1^{1\backslash 2}, \ldots, t_w^{1\backslash 2}$ та $t_1^{2\backslash 1}, \ldots, t_q^{2\backslash 1}$, які містять властивості та(або) методи, які характерні лише для типів об'єктів $t_1^1, \ldots, t_n^1$ та $t_1^2, \ldots, t_m^2$ відповідно.

Означення 2.5.7. Клон $Clone_i(T)$ довільного класу об'єктів T, що визначає типи $t_1, \ldots, t_n$, де $n \geqslant 1$, – це клас об'єктів $T_i/(P(T), F(T))$, де $P(T)$ та $F(T)$ – це специфікація та сигнатура класу об'єктів T відповідно, а i – це номер його клону.

Таким чином, на відміну від застосування до класів об'єктів усіх вище зазначених операцій, експлуататор клонування створює нову, пронумеровану копію класу об'єктів T.

2.5.1.2. Універсальні експлуататори об'єктів. Головною особливістю об'єктних експлуататорів є використання в якості аргументів для створення нових класів, не самих об'єктів, а їхніх типів. Оскільки між об'єктами та їх типами існує різниця, яка полягає у тому що всі об'єкти мають строго визначені властивості, зокрема кількісні властивості, в той час як типи об'єктів можуть мати, як строго, так і слабко визначені кількісні властивості. Тому використовуючи Означення 2.2.1, 2.2.11, 2.2.14 та вище зазначені експлуататори класів об'єктів, визначимо універсальні експлуататори об'єднання, мультиоб'єднання, однорідного та неоднорідного перетину, різниці, симетричної різниці та клонування об'єктів [2, 7–10, 21, 26].

Означення 2.5.8. Об'єднання об'єктів $O_1 \cup \cdots \cup O_n$ типів $t_1, \ldots, t_n$ відповідно, де $n \geqslant 2$, – це множина об'єктів S, яка визначається наступним чином

$$O_1/t_{O_1} \cup \cdots \cup O_n/t_{O_n} = \{O_1, \ldots, O_m\}/T_S = S/T_S,$$

де t_{O_l} – це тип об'єкта O_l, $l = \overline{1,n}$; $m = |S|$, де $1 \leqslant m \leqslant n$ і $\forall O_i, O_j \in S,\ i \neq j$ має місце $Eq(O_i, O_j) = 0$, де $i, j = \overline{1,m}$; $T_S = t_{O_1} \cup \cdots \cup t_{O_n}$ – це клас новоутвореної множини об'єктів S, що визначає типи $t_1, \ldots, t_k$, де $\forall t_{w_1}, t_{w_2},\ w_1 \neq w_2$ має місце $Eq(t_{w_1}, t_{w_2}) = 0$, де $w_1, w_2 = \overline{1,k}$, $1 \leqslant k \leqslant n$, і для яких виконується умова $\left(\forall t_i^{O_i},\ \exists! t_j^S\right) \wedge \left(\forall t_j^S,\ \exists t_i^{O_i}\right) \mid Eq\left(t_i^{O_i}, t_j^S\right) = 1$, де $t_i^{O_i}$ – це тип об'єкта O_i, де $i = \overline{1,n}$,

а t_j^S – це j-ий тип класу T_S, де $j = \overline{1,k}$.

Таким чином, в залежності від однотипності об'єктів $O_1, \ldots, O_n$, клас T_S буде однорідним або неоднорідним. Експлуататор об'єднання є однією з ключових операцій над об'єктами, оскільки він пов'язує між собою концепції об'єкта, типу, класу та множини об'єктів, а також є конструктивним способом створення множин та класів множин об'єктів.

Означення 2.5.9. Мультиоб'єднання об'єктів $O_1 \sqcup \cdots \sqcup O_n$ типів $t_1, \ldots, t_n$ відповідно, де $n \geqslant 2$, – це мультимножина об'єктів M, яка визначається наступним чином $O_1/t_{O_1} \sqcup \cdots \sqcup O_n/t_{O_n} = \{(O_1, m_1) \ldots, (O_k, m_k)\}/T_M = M/T_M$, де t_{O_i} – це тип об'єкта O_l, $l = \overline{1,n}$; m_i – це кратність об'єкта O_i у мультимножині об'єктів M, де $i = \overline{1,k}$, де $1 \leqslant k \leqslant n$, і має місце $\left(\forall O_i,\ \exists O_j^M\right) \wedge \left(\forall O_j^M,\ \exists O_i\right) \mid Eq\left(O_i, O_j^M\right) = 1$; $T_M = t_{O_1} \cup \cdots \cup t_{O_n}$ – це клас новоутвореної мультимножини об'єктів M, що визначає типи $t_1, \ldots, t_k$, де $\forall t_{w_1}, t_{w_2},\ w_1 \neq w_2 \mid Eq(t_{w_1}, t_{w_2}) = 0$, де $w_1, w_2 = \overline{1,k}$, $1 \leqslant k \leqslant n$, і має місце $\left(\forall t_i^{O_i},\ \exists! t_j^M\right) \wedge \left(\forall t_j^M,\ \exists t_i^{O_i}\right) \mid Eq\left(t_i^{O_i}, t_j^M\right) = 1$, де $t_i^{O_i}$ – це тип об'єкта O_i, де $i = \overline{1,n}$, а t_j^M – це j-ий тип класу T_M, де $j = \overline{1,k}$.

Таким чином, в залежності від однотипності об'єктів $O_1, \ldots, O_n$, клас T_S буде однорідним або неоднорідним. Експлуататор мультиоб'єднання є однією з ключових операцій над об'єктами, оскільки він пов'язує між собою концепції об'єкта, типу, класу та мультимножини об'єктів, а також є конструктивним способом створення мультимножин та класів мультимножин об'єктів.

Означення 2.5.10. Однорідний перетин об'єктів $O_1 \cap \cdots \cap O_n$ типів $t_1, \ldots, t_n$ відповідно, де $n \geqslant 2$, – це однорідний клас об'єктів T_A, який визначає тип t_A, для якого має місце $\left(\forall t_i^{O_i},\ t_A \subseteq t_i^{O_i}\right) \wedge \left(\nexists t_B \mid (t_A \subset t_B) \wedge \left(t_B \subseteq t_i^{O_i}\right)\right)$, де $t_i^{O_i}$ – це тип об'єкта O_i, де $i = \overline{1,n}$. Однорідний перетин об'єктів $O_1 \cap \cdots \cap O_n$ існує тоді і тільки тоді, коли $\exists p_{i_1}(t_1), \ldots, \exists p_{i_n}(t_n) \mid Eq(p_{i_1}(t_1), \ldots, p_{i_n}(t_n)) = 1$, де $p_{i_k}(t_k)$ – це i_k-та властивість типу t_k, де $i_k = \overline{1, D(t_k)}$ і $k = \overline{1,n}$.

Таким чином, у результаті застосування експлуататора однорідного перетину до об'єктів $O_1, \ldots, O_n$, буде утворений однорідний клас об'єктів T_A, який визначатиме тип t_A, такий що міститиме лише ті властивості, які є спільними для об'єктів

$O_1, \ldots, O_n$.

Однорідний перетин об'єктів дозволяє утворювати нові класи об'єктів, лише за умови існування властивостей, які є спільними одночасно для усіх об'єктів. Проте іноді трапляються ситуації, коли таких властивостей не існує для n об'єктів, але вони можуть існувати для m об'єктів, де $1 \leqslant m < n$. Також можливі випадки, коли спільні властивості можуть існувати одночасно для певної кількості різних груп об'єктів. У зв'язку з чим, є доцільним визначити експлуататор неоднорідного перетину об'єктів.

Означення 2.5.11. Неоднорідний перетин об'єктів $O_1 \Cap \cdots \Cap O_n$ типів $t_1, \ldots, t_n$ відповідно, де $n \geqslant 2$, – це багатоядерний неоднорідний клас об'єктів T_A, що має наступну структуру

$$T_A = \big(Core_1^n(T_A), Core_1^{n-1}(T_A), \ldots, Core_{w_{n-1}}^{n-1}(T_A), \ldots, Core_1^1(T_A), \ldots, Core_{w_1}^1(T_A)\big),$$

де кожне ядро $Core_{i_m}^m(T_A)$, $1 \leqslant m \leqslant n$, $i_m = \overline{1, w_m}$, де $w_m \leqslant C_n^m$ визначає підтип $t_{i_m}^{1,\ldots,m}$ для якого має місце $\forall t_{i_m}^{1,\ldots,m}$, $\exists (t_1, \ldots, t_m) \mid \left(t_{i_m}^{1,\ldots,m} = t_1 \cap \cdots \cap t_m\right)$. Неоднорідний перетин об'єктів $O_1 \Cap \cdots \Cap O_n$ існує тоді і тільки тоді, коли $\exists \left(p_{i_1}(t_1)\right), \ldots, \exists \left(p_{i_m}(t_m)\right) \mid Eq(p_{i_1}(t_1), \ldots, p_{i_m}(t_m)) = 1$, де $2 \leqslant m \leqslant n$, $p_{i_k}(t_k)$ – це i_k-та властивість типу t_k, де $i_k = \overline{1, D(t_k)}$ і $k = \overline{1, n}$.

Отже, у результаті застосування експлуататора неоднорідного перетину до об'єктів $O_1, \ldots, O_n$ буде утворений багатоядерний неоднорідний клас об'єктів T_A, який у загальному випадку визначатиме наступну множину типів об'єктів $\left\{t_1^n, t_1^{n-1}, \ldots, t_{w_{n-1}}^{n-1}, \ldots, t_1^1, \ldots, t_{w_1}^1\right\}$, де $0 \leqslant w_i \leqslant C_n^i$, а $i = \overline{1, n-1}$.

Означення 2.5.12. Різниця $O_1 \setminus O_2$ між об'єктом O_1 типу t_1 і об'єктом O_2 типу t_2, – це однорідний клас об'єктів T_A, що визначає тип $t_{1\setminus 2}$ для якого має місце $(t_{1\setminus 2} \subseteq t_1) \wedge \nexists (t_{1\setminus 2} \cap t_2) \wedge \left(\nexists t'_{1\setminus 2} \mid \left(t_{1\setminus 2} \subset t'_{1\setminus 2}\right) \wedge \left(t'_{1\setminus 2} \subseteq t_1\right) \wedge \nexists \left(t'_{1\setminus 2} \cap t_2\right)\right)$. Різниця між об'єктом O_1 та об'єктом O_2 існує тоді і тільки тоді, коли

$$\exists p_{i_1}(t_1), \exists p_{i_2}(t_2) \mid Eq(p_{i_1}(t_1), p_{i_2}(t_2)) = 0,$$

де $p_{i_k}(t_k)$ – це i_k-та властивість типу t_k, де $i_k = \overline{1, D(t_k)}$ і $k = \overline{1, 2}$.

Таким чином, у результаті застосування експлуататора різниці до об'єктів O_1 та O_2 буде утворений однорідний клас об'єктів T_A, який визначатиме тип об'єктів $t_{1\setminus 2} \subseteq t_1$, який міститиме властивості, які є характерними лише для об'єкта O_1.

Означення 2.5.13. Симетрична різниця $O_1 \div O_2$ між об'єктом O_1 типу t_1 та об'єктом O_2 типу t_2, – це неоднорідний клас об'єктів T_A, що визначає типи $t_{1\setminus 2}$ та $t_{2\setminus 1}$. Симетрична різниця між об'єктами O_1 та O_2 існує тоді і тільки тоді, коли $\exists p_{i_1}(t_1), \exists p_{i_2}(t_2) \mid Eq(p_{i_1}(t_1), p_{i_2}(t_2)) = 0$, де $p_{i_k}(t_k)$ – це i_k-та властивість типу t_k, де $i_k = \overline{1, D(t_k)}$ і $k = \overline{1, 2}$.

Таким чином, у результаті застосування експлуататора симетричної різниці до об'єктів O_1 та O_2 буде утворений неоднорідний клас об'єктів T_A, який визначає типи об'єктів $t_{1\setminus 2}$ і $t_{2\setminus 1}$ такі що містять властивості, які характерні лише для об'єктів O_1 та O_2 відповідно.

Означення 2.5.14. Клон $Clone_i(A)$ довільного об'єкта A типу t – це об'єкт $A_i/P(A)$, де $P(A)$ – це специфікація об'єкта A, а i – це номер його клону.

Таким чином, на відміну від застосування до об'єктів усіх вище зазначених операцій, експлуататор клонування створює нову, пронумеровану копію об'єкта A.

2.5.2. Модифікатори. В рамках сучасного ООП поняття класу та об'єкта є в певному сенсі статичними, оскільки після компіляції (інтерпретації) та запуску програми структура класу та об'єктів не може бути змінена. Можливі лише зміни значень деяких властивостей під дією певних операторів або викликів методів класів [68, 74, 75].

Усі на сьогодні існуючі об'єктно-орієнтовані моделі представлення знань мають мережеву або ієрархічну структуру та можуть бути графічно представлені за допомогою зв'язного орієнтованого графа або дерева, вершинам якого можуть відповідати об'єкти або класи об'єктів, а ребрам – зв'язки між ними. Більшість таких моделей є статичними і відображають лише певну чітко визначену (сформовану) ієрархію відношень між об'єктами та класами об'єктів. Тому за допомогою таких МПЗ не можливо представити знання, що можуть бути змінені з часом,

оскільки це призведе до перебудови структури моделі предметної області. Однак, зміни певних знань з часом (еволюція) є також знаннями, які у свою чергу можуть змінювати уявлення про предметну область, що вимагає перебудови (адаптацію до змін) її моделі.

У зв'язку з чим визначимо поняття модифікаторів об'єктів та класів об'єктів, які дозволятимуть модифікацію знань з часом, а також дозволяють будувати новий тип зв'язків *modification-of* між концептами, який по суті описує можливий стан концепту після його трансформації або ж його трансформаційну історію. Такий підхід дозволятиме моделювати зміни або еволюцію знань з часом. У загальному поняття модифікатора можна визначити наступним чином [14, 27].

Означення 2.5.15. Модифікатор об'єкта (класу об'єктів) – це вектор функцій модифікації, які змінюють об'єкт (клас об'єктів), шляхом зміни його специфікації (та(або) сигнатури).

В залежності від характеру зміни сутності об'єкту або класу об'єктів, можна виділити наступні типи модифікації: *повна*, *часткова*, *породжуюча*, *знищуючя*, *замінна*, *комбінована*. Принципи дії перших п'яти типів модифікації графічно зображені на Рис.2.5. Усі типи модифікації продемонстровані на специфікації деякого об'єкта A, яка складається з 5 властивостей, де кожній властивості відповідає один квадрат.

У випадку повної модифікації об'єкта $M^F(A)$ відбувається модифікація усіх властивостей об'єкта A. Властивості, які модифікуються позначені сірими квадратами. У випадку часткової модифікації $M^P(A)$ відбувається модифікація лише властивостей $p_1(A)$, $p_2(A)$ та $p_4(A)$. У випадку породжувальної модифікації $M^G(A)$, до специфікації об'єкта додається нова властивість $p_6(A)$. У випадку знищувальної модифікації $M^D(A)$, знищується властивість $p_4(A)$. І у випадку замінної модифікації $M^R(A)$, властивість $p_3(A)$ замінюється на властивість $p(A)$.

Перш ніж перейти до визначення та розгляду модифікаторів об'єктів та класів об'єктів, розглянемо наступний приклад.

Приклад 2.5.1. Розглянемо клас T_S, який описує такий тип об'єктів, як ква-

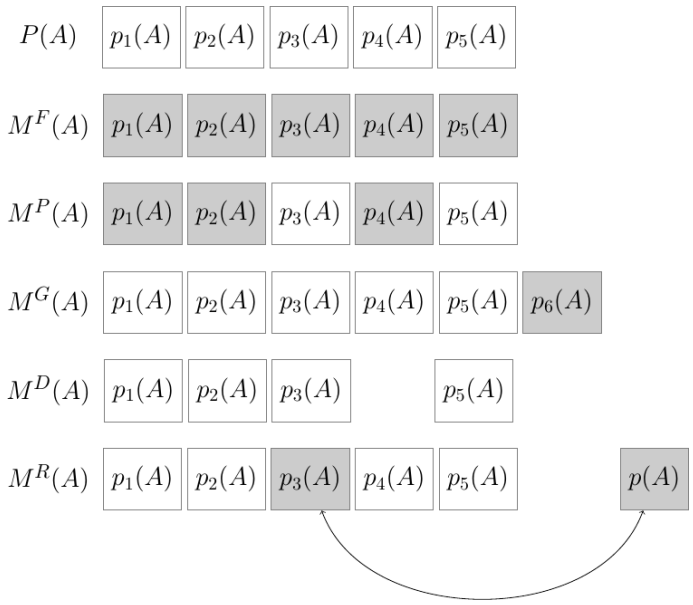


Рис. 2.5: Графічна інтерпретація принципів дії різних типів модифікації: повна M^F, часткова M^P, породжувальна M^G, знищувальна M^D та замінна M^R

драти та має наступну структуру:

$$T_S = (p_1(T_S) = (4, \text{сторони}),\ p_2(T_S) = ((v_1, \text{см}), (v_2, \text{см}), (v_3, \text{см}), (v_4, \text{см})),$$
$$p_3(T_S) = (4, \text{кути}),\ p_4(T_S) = ((90,^\circ), (90,^\circ), (90,^\circ), (90,^\circ)),\ p_5(T_S) = vf_5(T_S) = 1,$$
$$p_6(T_S) = vf_6(T_S) = 1,\ p_7(T_S) = vf_7(T_S) = 1,\ f_1(T_S) = (v_1(p_2(T_S)) \cdot 4, \text{см}),$$
$$f_2(T_S) = \left(v_1(p_2(T_S))^2, \text{см}^2\right),$$

де $p_1(T_S)$ – це кількість сторін фігури, $p_2(T_S)$ – це розміри сторін фігури, $p_3(T_S)$ – це кількість внутрішніх кутів фігури, $p_4(T_S)$ – це градусні міри внутрішніх кутів фігури, $vf_5(T_S)$ – це функція верифікації, що визначає властивість "сума градусних мір усіх внутрішніх кутів фігури рівна 360°", тобто $vf_5(T_S) : p_4(T_S) \to \{0, 1\}$, де $vf_5(T_S) = (v_1(p_4(T_S)) + v_2(p_4(T_S)) + v_3(p_4(T_S)) + v_4(p_4(T_S)) = 360)$, $vf_6(T_S)$ – це функція верифікації, що визначає властивість "усі сторони фігури рівні по довжині", тобто $vf_6(T_S) : p_2(T_S) \to \{0, 1\}$, де

$$vf_6(T_S) = (v_1(p_2(T_S)) = v_2(p_2(T_S)) = v_3(p_2(T_S)) = v_4(p_2(T_S))),$$

$vf_7(T_S)$ – це функція верифікації, що визначає властивість "усі внутрішні кути фігури мають градусну міру 90°", тобто $vf_7(T_S) : p_4(T_S) \to \{0, 1\}$, де

$$vf_7(T_S) = (v_1(p_4(T_S)) = v_2(p_4(T_S)) = v_3(p_4(T_S)) = v_4(p_4(T_S)) = 90),$$

$f_1(T_S)$ – це метод обчислення периметру фігури, а $f_2(T_S)$ – це метод обчислення площі фігури.

Аналізуючи специфікацію класу T_S, можна помітити наступні залежності між властивостями та методами класу: кількість сторін квадрата $p_1(T_S)$ та кількість його внутрішніх кутів $p_3(T_S)$ пропорційно пов'язані між собою; розміри сторін квадрата $p_2(T_S)$ та градусні міри його внутрішніх кутів $p_4(T_S)$ також пропорційно пов'язані між собою; властивість $p_5(T_S)$ залежить від градусних мір внутрішніх кутів фігури $p_4(T_S)$; рівність усіх сторін фігури $p_6(T_S)$ залежить від розмірів сторін фігури $p_2(T_S)$; властивість $p_7(T_S)$ залежить від градусних мір внутрішніх кутів фігури $p_4(T_S)$; метод обчислення периметру фігури $f_1(T_S)$ залежить від розмірів сторін фігури $p_2(T_S)$; метод обчислення площі фігури $f_2(T_S)$ також залежить від розмірів сторін фігури $p_2(T_S)$. Усі зазначені залежності можна представити у вигляді структурної діаграми класу T_S (див. Рис. 2.6).

Аналізуючи Рис. 2.6, можна зробити висновок, що при модифікації будь-яких властивостей фігури повинні змінюватися інші, залежні від них, властивості та методи. Наприклад, якщо модифікувати значення властивості $p_4(T_S)$ таким чином, що градусні міри внутрішніх кутів фігури стають відмінними від $90°$, то властивість $p_7(T_S)$ перестане виконуватися і прийме значення 0, а метод для обчислення площі $f_2(T_S)$ буде давати некоректний результат. Якщо ж модифікувати значення властивості $p_1(T_S)$ – кількість сторін фігури, то зміниться уся специфікація та частково сигнатура класу, в результаті чого сам клас T_S перестане визначати квадрати.

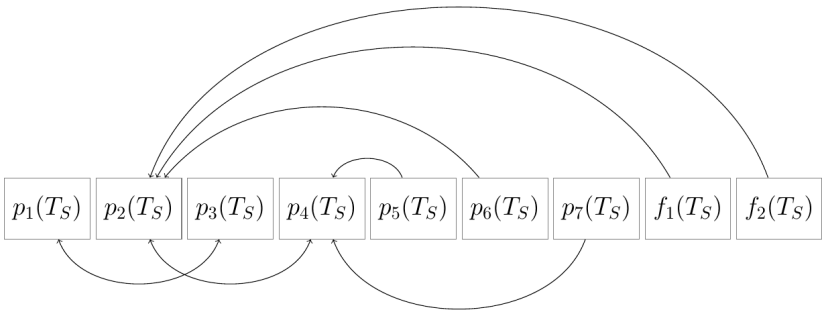


Рис. 2.6: Структурна діаграма класу T_S

Приклад 2.5.1 наглядно демонструє, що між властивостями об'єктів та кла-

сів об'єктів існують певні залежності, такі що при модифікації якоїсь однієї властивості об'єкта чи класу об'єктів, тягнуть за собою зміни усіх інших, залежних від неї властивостей та методів. Такі зв'язки між властивостями об'єктів та між властивостями та методами класів об'єктів, утворюють ніщо інше як структурні компоненти зв'язності, які також необхідно враховувати у процесах часткового успадкування. В реальному світі при модифікації якоїсь конкретної властивості будь-якого об'єкта, усі інші залежні від неї властивості змінюються "автоматично". Деякі із них модифікуються, деякі зникають, а іноді можуть з'являються нові властивості. Якщо модифікація конкретної властивості реального існуючого об'єкта автоматично тягне за собою зміни усіх інших залежних від неї властивостей та методів, то у випадку з абстрактним представленням реальних об'єктів природна реакція (рефлексія) на зміни якихось властивостей відсутня. Тому виникає потреба в штучній підтримці такої рефлексії, оскільки без неї абстрактні моделі реальних об'єктів після їхньої модифікації перестануть імітувати (моделювати) реальність.

2.5.2.1. Модифікатори об'єктів. Перш ніж визначити поняття модифікаторів об'єктів, визначимо поняття функцій модифікації кількісних та якісних властивостей об'єктів [14, 27].

Означення 2.5.16. Функція модифікації кількісної властивості $p_i(A)$ об'єкта A – це функція, що є еквівалентною будь-якій із наступних функцій

$$m^v(p_i(A)) : (v_1(p_i(A)), \ldots, v_n(p_i(A))) \to (v_1, \ldots, v_n), \tag{2.1}$$

$$m^u(p_i(A)) : (u_1(p_i(A)), \ldots, u_n(p_i(A))) \to (u_1, \ldots, u_n), \tag{2.2}$$

$$m^p(p_i(A)) : ((v_1(p_i(A)), u_1(p_i(A))), \ldots, (v_n(p_i(A)), u_n(p_i(A)))) \to \to ((v_1, u_1), \ldots, (v_n, u_n)) \tag{2.3}$$

де $v_j(p_i(A))$ – це кількісне значення властивості $p_i(A)$, $u_j(p_i(A))$ – це одиниці вимірювання значення властивості $p_i(A)$ об'єкта A, де $j = \overline{1,n}$, $i = \overline{1, D(A)}$, $(v_1, \ldots, v_n)$ – це нове кількісне значення властивості $p_i(A)$, а $(u_1, \ldots, u_n)$ – це одиниці його вимірювання.

Таким чином існує три варіанти модифікації кількісних властивостей об'єктів:

1. модифікація значення кількісної властивості (2.1),
2. модифікація одиниць вимірювання кількісного значення властивості (2.2),
3. модифікація кількісного значення властивості та одиниць його вимірювання (2.3).

Означення 2.5.17. Функція модифікації якісної властивості $p_i(A) = vf_i(A)$ об'єкта A – це функція виду $m(p_i(A)) : vf_i(A) \to vf(A)$, де $vf_i(A)$ – це функція верифікації, що визначає i-ту якісну властивість об'єкта A, де $i = \overline{1, D(A)}$, а $vf(A)$ – це функція верифікації, що визначає нову якісну властивість об'єкта A.

Тепер, використовуючи Означення 2.5.16 та Означення 2.5.17, визначимо поняття повного M^F та часткового M^P модифікаторів об'єктів.

Означення 2.5.18. Повний модифікатор об'єкта A – це вектор виду

$$M^F(A) = (m_1(p_1(A)), \ldots, m_n(p_n(A))),$$

де $m_i(p_i(A))$ – це функція модифікації i-ї кількісної або якісної властивості об'єкта A, де $i = \overline{1, n}$, а $n = D(A)$.

Означення 2.5.19. Частковий модифікатор об'єкта A – це вектор виду

$$M^P(A) = (m_1(p_{j_1}(A)), \ldots, m_n(p_{j_2}(A))),$$

де $m_i(p_j(A))$ – це функція модифікації j-ї кількісної або якісної властивості об'єкта A, де $i = \overline{1, n}$, $1 \leqslant n < D(A)$, $j = \overline{j_1, j_2}$ де $1 \leqslant j_1 \leqslant \cdots \leqslant j_2 \leqslant D(A)$.

Тепер визначимо поняття породжувального M^G, знищувального M^D та замінного M^R модифікаторів об'єктів.

Означення 2.5.20. Функція породження властивості об'єкта A – це функція виду $m^G(A) : P(A) \to (P(A), p_{n+1}(A)) = (p_1(A), \ldots, p_n(A), p_{n+1}(A))$, де $P(A)$ – це специфікація об'єкта A, $n = D(A)$, а $p_{n+1}(A)$ – це нова кількісна або якісна властивість, що додається до специфікації об'єкта A.

Означення 2.5.21. Породжувальний модифікатор об'єкта A – це вектор виду $M^G(A) = (m_{i_1}(p_{i_1}(A)), \ldots, m_{i_2}(p_{i_2}(A)))$, де $m_i(p_i(A))$ – це функція породження i-

ї кількісної або якісної властивості об'єкта A, де $i = \overline{i_1, i_2}$, $i_1 = D(A) + 1$ та $i_1 \leqslant \cdots \leqslant i_2$.

Означення 2.5.22. Функція знищення властивості об'єкта A – це функція виду $m^D(A) : P(A) \to (p_1(A), \ldots, p_{k-1}(A), p_{k+1}(A), \ldots, p_{n-1}(A))$, де $P(A)$ – це специфікація об'єкта A, $p_k(A)$ – це кількісна або якісна властивість, що видаляється із його специфікації, де $n = D(A)$.

Означення 2.5.23. Знищувальний модифікатор об'єкта A – це вектор виду $M^D(A) = (m_{i_1}(p_{i_1}(A)), \ldots, m_{i_2}(p_{i_2}(A)))$, де $m_i(p_i(A))$ – це функція знищення i-ї кількісної або якісної властивості об'єкта A, де $i = \overline{i_1, i_2}$ та $1 \leqslant i_1 \leqslant \cdots \leqslant i_2 \leqslant D(A)$.

Означення 2.5.24. Функція заміни властивості об'єкта A – це функція виду $m^R(A) : p_i(A) \to p(A)$, де $p_i(A)$ – це i-та кількісна або якісна властивість об'єкта A, яка заміняється іншою кількісною або якісною властивістю $p(A)$, де $i = \overline{1, D(A)}$.

Означення 2.5.25. Замінний модифікатор об'єкта A – це вектор виду

$$M^R(A) = (m_{i_1}(p_{i_1}(A)), \ldots, m_{i_2}(p_{i_2}(A))),$$

де $m_i(p_i(A))$ – це функція заміни i-ї кількісної або якісної властивості об'єкта A, де $i = \overline{i_1, i_2}$ та $1 \leqslant i_1 \leqslant \cdots \leqslant i_2 \leqslant D(A)$.

Використовуючи модифікатори об'єктів M^F, M^P, M^G, M^D та M^R можна визначити поняття комбінованого модифікатора об'єктів.

Означення 2.5.26. Комбінований модифікатор степеня n об'єкта A – це суперпозиція n модифікаторів об'єктів

$$M^{X_1,\ldots,X_n}(A) = \left(M_n^{X_n}\left(M_{n-1}^{X_{n-1}}\left(\ldots\left(M_1^{X_1}(A)\right)\ldots\right)\right)\right),$$

де $X_1, \ldots, X_n \in \{F, P, G, D, R\}$.

Таким чином комбінована модифікація об'єктів складається з почергового застосування до об'єктів, що модифікуються, певної послідовності модифікаторів з множини $\left\{M^F, M^P, M^G, M^D, M^R\right\}$.

2.5.2.2. Модифікатори класів об'єктів. Перш ніж визначити поняття модифікаторів класів об'єктів, аналогічно до випадку модифікаторів об'єктів, визначимо поняття функцій модифікації кількісних та якісних властивостей класів об'єктів [14, 27]. Враховуючи те що клас у певному сенсі є абстрактним об'єктом, то Означення 2.5.16 та 2.5.17 можна переформулювати наступним чином.

Означення 2.5.27. Функція модифікації кількісної властивості $p_i(T)$ класу об'єктів T – це функція, що є еквівалентною будь-якій із наступних функцій

$$m^v(p_i(T)) : (v_1(p_i(T)), \ldots, v_n(p_i(T))) \to (v_1, \ldots, v_n), \tag{2.4}$$

$$m^u(p_i(T)) : (u_1(p_i(T)), \ldots, u_n(p_i(T))) \to (u_1, \ldots, u_n), \tag{2.5}$$

$$m^p(p_i(T)) : ((v_1(p_i(T)), u_1(p_i(T))), \ldots, (v_n(p_i(T)), u_n(p_i(T)))) \to \tag{2.6}$$
$$\to ((v_1, u_1), \ldots, (v_n, u_n))$$

де $v_j(p_i(T))$ – це кількісне значення властивості $p_i(T)$, $u_j(p_i(T))$ – це одиниці вимірювання значення властивості $p_i(T)$ класу об'єктів T, де $j = \overline{1, n}$, $i = \overline{1, D(T)}$, $(v_1, \ldots, v_n)$ – це нове кількісне значення властивості $p_i(T)$, а $(u_1, \ldots, u_n)$ – це одиниці його вимірювання.

Означення 2.5.28. Функція модифікації якісної властивості $p_i(T) = vf_i(T)$ класу об'єктів T – це функція виду $m(p_i(T)) : vf_i(T) \to vf(T)$, де $vf_i(T)$ – це функція верифікації, що визначає i-ту якісну властивість класу T, де $i = \overline{1, D(T)}$, а $vf(T)$ – це функція верифікації, що визначає нову якісну властивість класу T.

Тепер сформулюємо поняття функції модифікації методів класу.

Означення 2.5.29. Функція модифікації методу $f_i(T)$ класу об'єктів T – це функція виду $m(f_i(T)) : f_i(T) \to f(T)$, де $f_i(T)$ – це i-й метод класу об'єктів T, де $i = \overline{1, D(T)}$, а $f(T)$ – це новий метод класу об'єктів T.

Тепер, використовуючи Означення 2.5.27, 2.5.28 та 2.5.29, визначимо поняття повного M^F та часткового M^P модифікаторів класів об'єктів.

Означення 2.5.30. Повний модифікатор класу об'єкта TA – це вектор виду $M^F(T) = (m_1(p_1(T)), \ldots, m_n(p_n(T)), m_1(f_1(T)), \ldots, m_k(f_k(T)))$, де $m_i(p_i(T))$ – це функція модифікації i-ї кількісної або якісної властивості класу T, де $i = \overline{1, n}$,

$n = D(T)$, а $m_j(f_j(T))$ – це функція модифікації j-го методу класу T, де $j = \overline{1,k}$, $k = func(T)$.

Означення 2.5.31. Частковий модифікатор класу об'єктів T – це вектор виду $M^P(T) = (m_1(p_{j_1}(T)), \ldots, m_n(p_{j_2}(T)), m_1(f_{w_1}(T)), \ldots, m_k(f_{w_2}(T)))$, де $m_i(p_j(T))$ – це функція модифікації j-ї кількісної або якісної властивості класу T, де $i = \overline{1,n}$, $1 \leqslant n < D(T)$, $j = \overline{j_1, j_2}$ де $0 \leqslant j_1 \leqslant \cdots \leqslant j_2 \leqslant D(T)$, а $m_v(f_w(T))$ – це функція модифікації w-го методу класу T, де $v = \overline{1,k}$, $1 \leqslant k < func(T)$, $w = \overline{w_1, w_2}$ де $0 \leqslant w_1 \leqslant \cdots \leqslant w_2 \leqslant func(T)$, при цьому $n + k \geqslant 1$.

Тепер визначимо поняття породжувального M^G, знищувального M^D та замінного M^R модифікаторів класів об'єктів.

Означення 2.5.32. Функція породження властивості класу об'єктів T – це функція виду $m^G(T) : P(T) \to (P(T), p_{n+1}(T)) = (p_1(T), \ldots, p_n(T), p_{n+1}(T))$, де $P(T)$ – це специфікація класу T, $n = D(T)$, а $p_{n+1}(T)$ – це нова кількісна або якісна властивість, що додається до специфікації класу T.

Означення 2.5.33. Функція породження методу класу об'єктів T – це функція виду $m^G(T) : F(T) \to (F(T), f_{n+1}(T)) = (f_1(T), \ldots, f_n(T), f_{n+1}(T))$, де $F(T)$ – це сигнатура класу T, $n = func(T)$, а $f_{n+1}(T)$ – це новий метод, що додається до сигнатури класу T.

Означення 2.5.34. Породжувальний модифікатор класу об'єктів T – це вектор виду $M^G(T) = (m_{i_1}(p_{i_1}(T)), \ldots, m_{i_2}(p_{i_2}(T)), m_{j_1}(f_{j_1}(T)), \ldots, m_{j_2}(f_{j_2}(T)))$, де $m_i(p_i(T))$ – це функція породження i-ї кількісної або якісної властивості класу T, де $i = \overline{i_1, i_2}$, $i_1 = D(T) + 1$ та $i_1 \leqslant \cdots \leqslant i_2$, а $m_j(f_j(T))$ – це функція породження j-го методу класу T, де $j = \overline{j_1, j_2}$, $j_1 = func(T) + 1$ та $j_1 \leqslant \cdots \leqslant j_2$.

Означення 2.5.35. Функція знищення властивості класу об'єктів T – це функція виду $m^D(T) : P(T) \to (p_1(T), \ldots, p_{k-1}(T), p_{k+1}(T), \ldots, p_{n-1}(T))$, де $P(T)$ – це специфікація класу T, $p_k(T)$ – це кількісна або якісна властивість, що видаляється із його специфікації, де $n = D(T)$.

Означення 2.5.36. Функція знищення методу класу об'єктів T – це функція виду $m^D(T) : F(T) \to (f_1(T), \ldots, f_{k-1}(T), f_{k+1}(T), \ldots, f_{n-1}(T))$, де $F(T)$ – це

сигнатура класу T, $f_k(T)$ – це метод, що видаляється із його специфікації, де $n = func(T)$.

Означення 2.5.37. Знищувальний модифікатор класу об'єктів T – це вектор виду $M^D(T) = (m_{i_1}(p_{i_1}(T)), \ldots, m_{i_2}(p_{i_2}(T)), m_{j_1}(f_{j_1}(T)), \ldots, m_{j_2}(f_{j_2}(T)))$, де $m_i(p_i(T))$ – це функція знищення i-ї кількісної або якісної властивості класу T, де $i = \overline{i_1, i_2}$ та $1 \leqslant i_1 \leqslant \cdots \leqslant i_2 \leqslant D(T)$, а $m_j(f_j(T))$ – це функція знищення j-го методу класу T, де $j = \overline{j_1, j_2}$ та $1 \leqslant j_1 \leqslant \cdots \leqslant j_2 \leqslant func(T)$.

Означення 2.5.38. Функція заміни властивості класу об'єктів T – це функція виду $m^R(T) : p_i(T) \to p(T)$, де $p_i(T)$ – це i-та кількісна або якісна властивість класу T, яка заміняється іншою кількісною або якісною властивістю $p(T)$, де $i = \overline{1, D(T)}$.

Означення 2.5.39. Функція заміни методу класу об'єктів T – це функція виду $m^R(T) : f_i(T) \to f(T)$, де $f_i(T)$ – це i-й метод класу T, який заміняється іншим методом $f(T)$, де $i = \overline{1, func(T)}$.

Означення 2.5.40. Замінний модифікатор класу об'єктів T – це вектор виду $M^R(T) = (m_{i_1}(p_{i_1}(T)), \ldots, m_{i_2}(p_{i_2}(T)), m_{j_1}(p_{j_1}(T)), \ldots, m_{j_2}(p_{j_2}(T)))$, де $m_i(p_i(T))$ – це функція заміни i-ї кількісної або якісної властивості класу T, де $i = \overline{i_1, i_2}$ та $1 \leqslant i_1 \leqslant \cdots \leqslant i_2 \leqslant D(T)$, а $m_j(f_j(T))$ – це функція заміни j-го методу класу T, де $j = \overline{j_1, j_2}$ та $1 \leqslant j_1 \leqslant \cdots \leqslant j_2 \leqslant func(T)$.

Використовуючи модифікатори класів об'єктів M^F, M^P, M^G, M^D та M^R можна визначити поняття комбінованого модифікатора.

Означення 2.5.41. Комбінований модифікатор степеня n класу об'єктів T – це суперпозиція n модифікаторів класів об'єктів

$$M^{X_1,\ldots,X_n}(T) = \left(M_n^{X_n}\left(M_{n-1}^{X_{n-1}}\left(\ldots\left(M_1^{X_1}(T)\right)\ldots\right)\right)\right),$$

де $X_1, \ldots, X_n \in \{F, P, G, D, R\}$.

2.5.2.3. Відношення модифікації. На відміну від експлуататорів, які використовують об'єкти та класи об'єктів як незмінні параметри для утворення

нових об'єктів та класів, модифікатори змінюють структуру об'єктів та класів об'єктів. Якщо до деякого об'єкта A_1 застосувати експлуататор клонування $Clone_1(A_1)$, то в результаті утвориться клон цього об'єкту A_2. Застосувавши до об'єкта A_2 деякий модифікатор[2] $M^X(A_2)$, отримаємо деяку його модифікацію A_3, яка також буде модифікацією об'єкта A. Таким чином між об'єктами A_1 та A_3 утвориться відношення модифікації $A_3 \sim A_1$. Таке відношення дозволяє моделювати трансформації об'єктів та класів об'єктів та дозволяє встановлювати неочевидні зв'язки між різними предметними областями. Визначимо відношення модифікації для об'єктів та класів об'єктів [22].

Означення 2.5.42. Об'єкти O_1 та O_3 перебувають у відношенні модифікації, тобто $O_3 \sim O_1$, тоді і тільки тоді коли

$$\exists O_2,\ \exists M^X \ \Big|\ (O_2 = Clone_i(O_1)) \wedge \left(O_3 = M^X(O_2)\right),$$

де O_2 – це клон об'єкта O_1, а M^X – це довільний модифікатор, який можна застосувати до об'єкта O_2 враховуючи його структуру.

Означення 2.5.43. Класи об'єктів T_1 та T_3 перебувають у відношенні модифікації, тобто $T_3 \sim T_1$, тоді і тільки тоді коли

$$\exists T_2,\ \exists M^X \ \Big|\ (T_2 = Clone_i(T_1)) \wedge \left(T_3 = M^X(T_2)\right),$$

де T_2 – це клон класу об'єктів T_1, а M^X – це довільний модифікатор, який можна застосувати до класу об'єктів T_2 враховуючи його структуру.

Тепер покажемо що відношення $\sim$, визначене в Означеннях 2.5.43 та 2.5.42, є відношеннями строго часткового порядку.

Твердження 2.5.1. Відношення $Modf(O_2, O_1) \Leftrightarrow (O_2 \sim O_1)$, де O_2 та O_1 – це довільні об'єкти, є відношенням строго часткового порядку визначеним на множині об'єктів $O = \{O_1, O_2, O_3 \dots\}$.

Доведення. З означення відношення строго часткового порядку слідує що відношення R, задане на множині X, є іррефлексивним $(\neg(aRa))$, транзитивним

[2]Маються на увазі лише ті модифікатори, які можна застосувати до цього об'єкту враховуючи особливості його структури.

$((aRb) \wedge (bRc) \Rightarrow (aRc))$ та асиметричним $(aRb \Rightarrow \neg(bRa))$ бінарним відношенням $(\forall a, b, c \in X)$ [90, 93]. Отже $\forall O_1, O_2, O_3 \in O$ повинні виконуватися іррефлексивність $(\neg(O_1 \sim O_1))$, транзитивність $((O_2 \sim O_1) \wedge (O_3 \sim O_2) \Rightarrow (O_3 \sim O_1))$ та асиметричність $((O_2 \sim O_1) \Rightarrow \neg(O_1 \sim O_2))$.

Іррефлексивність. Згідно з Означенням 2.5.15, модифікатори змінюють структуру об'єктів, тому $M^X(O_1) \neq O_1$, звідки слідує що $\neg(O_1 \sim O_1)$.

Транзитивність. Припустимо, що $(O_2 \sim O_1) \wedge (O_3 \sim O_2) \Rightarrow \neg(O_3 \sim O_1)$. З Означення 2.5.42 слідує що $\exists O_1', \ \exists M^X \mid \big(O_1' = Clone_1(O_1)\big) \wedge \big(O_2 = M^X(O_1')\big)$ і $\exists O_2', \ \exists M^X \mid \big(O_2' = Clone_1(O_2)\big) \wedge \big(O_3 = M^X(O_2')\big)$, а отже $O_2 = M^X(Clone_1(O_1))$ та $O_3 = M^X(Clone_1(O_2))$ тобто $O_3 = M^X(Clone_1(M^X(Clone_1(O_1))))$, з чого слідує що $O_3 \sim O_1$, у результаті чого отримуємо суперечність. Таким чином має місце $(O_2 \sim O_1) \wedge (O_3 \sim O_2) \Rightarrow (O_3 \sim O_1)$.

Асиметричність. Припустимо що $(O_2 \sim O_1) \Rightarrow (O_1 \sim O_2)$. З іррефлексивності $\neg(O_1 \sim O_1)$ слідує що $M^X(O_1) \neq O_1$, а отже $\exists O_2 \mid M^X(Clone_i(O_1)) = O_2$, тобто $O_2 \sim O_1$. З іншого боку з $O_1 \sim O_2$ слідує що $M^X(Clone_i(O_2)) = T_1$, однак при цьому має місце $\forall O_1, O_2 \nexists M^X \mid \big(M^X(Clone_i(O_1)) = O_2\big) \wedge \big(M^X(Clone_i(O_2)) = O_1\big)$, таким чином приходимо до суперечності. Отже $(O_2 \sim O_1) \Rightarrow \neg(O_1 \sim O_2)$.

В результаті чого отримуємо що $Modf(O_2, O_1) \Leftrightarrow (O_2 \sim O_1)$, де O_2 та O_1 – це довільні об'єкти, є відношенням строго часткового порядку визначеним на множині об'єктів $O = \{O_1, O_2, O_3 \dots\}$, що й треба було довести. □

Твердження 2.5.2. Відношення $Modf(T_2, T_1) \Leftrightarrow (T_2 \sim T_1)$, де T_2 та T_1 – це довільні класи об'єктів, є відношенням строго часткового порядку визначеним на множині класів об'єктів $C = \{T_1, T_2, T_3 \dots\}$.

Доведення. З означення відношення строго часткового порядку слідує що відношення R, задане на множині X, є іррефлексивним $(\neg(aRa))$, транзитивним $((aRb) \wedge (bRc) \Rightarrow (aRc))$ та асиметричним $(aRb \Rightarrow \neg(bRa))$ бінарним відношенням $(\forall a, b, c \in X)$ [90, 93]. Отже $\forall T_1, T_2, T_3 \in C$ повинні виконуватися іррефлексивність $(\neg(T_1 \sim T_1))$, транзитивність $((T_2 \sim T_1) \wedge (T_3 \sim T_2) \Rightarrow (T_3 \sim T_1))$ та асиметричність $((T_2 \sim T_1) \Rightarrow \neg(T_1 \sim T_2))$

Іррефлексивність. Згідно з Означенням 2.5.15, модифікатори змінюють структуру класів об'єктів, тому $M^X(T_1) \neq T_1$, звідки слідує що $\neg(T_1 \sim T_1)$.

Транзитивність. Припустимо, що $(T_2 \sim T_1) \wedge (T_3 \sim T_2) \Rightarrow \neg(T_3 \sim T_1)$. З Означення 2.5.43 слідує що $\exists T_1',\ \exists M^X \mid \big(T_1' = Clone_1(T_1)\big) \wedge \big(T_2 = M^X(T_1')\big)$ і $\exists T_2',\ \exists M^X \mid \big(T_2' = Clone_1(T_2)\big) \wedge \big(T_3 = M^X(T_2')\big)$, а отже $T_2 = M^X(Clone_1(T_1))$ та $T_3 = M^X(Clone_1(T_2))$ тобто $T_3 = M^X(Clone_1(M^X(Clone_1(T_1))))$, з чого слідує що $T_3 \sim T_1$, у результаті чого отримуємо суперечність. Таким чином має місце $(T_2 \sim T_1) \wedge (T_3 \sim T_2) \Rightarrow (T_3 \sim T_1)$.

Асиметричність. Припустимо що $(T_2 \sim T_1) \Rightarrow (T_1 \sim T_2)$. З іррефлексивності $\neg(T_1 \sim T_1)$ слідує що $M^X(T_1) \neq T_1$, а отже $\exists T_2 \mid M^X(Clone_i(T_1)) = T_2$, тобто $T_2 \sim T_1$. З іншого боку з $T_1 \sim T_2$ слідує що $M^X(Clone_i(T_2)) = T_1$, однак при цьому має місце $\forall T_1, T_2\ \nexists M^X \mid \big(M^X(Clone_i(T_1)) = T_2\big) \wedge \big(M^X(Clone_i(T_2)) = T_1\big)$, таким чином приходимо до суперечності. Отже $(T_2 \sim T_1) \Rightarrow \neg(T_1 \sim T_2)$.

В результаті чого отримуємо що $Modf(T_2, T_1) \Leftrightarrow (T_2 \sim T_1)$, де T_2 та T_1 – це довільні класи об'єктів, є відношенням строго часткового порядку визначеним на множині класів об'єктів $C = \{T_1, T_2, T_3 \dots\}$, що й треба було довести. □

2.5.3. Генератори. Поняття "множина" та "мультимножина" є центральними поняттями в теорії множин та мультимножин і одними із фундаментальних понять для математики та комп'ютерних наук в цілому. Однак крім цього ці поняття також мають важливе та глобальне значення, в життєдіяльності кожної людини, оскільки люди повсякденно використовують множини та мультимножини у своїй розумовій діяльності, у процесі сприйняття, аналізу, порівняння, пошуку, класифікації, тощо [2, 6–11, 25]. Кожна людина свідомо або підсвідомо створює множини та мультимножини, оперує ними, застосовуючи до них всі відомі теоретико-множинні операції і не тільки. Таким чином поняття "множина" та "мультимножина" є фундаментальними категоріями (абстракціями) людського інтелекту.

Означення 2.5.44. Генератор або конструктор множини (мультимножини) об'єктів – це довільна суперпозиція експлуататорів об'єктів, що обов'язково мі-

стить хоча б один універсальний експлуататор об'єднання (мультиоб'єднання), яка дозволяє утворити множину (мультимножину) об'єктів та її клас.

2.5.3.1. Універсальні конструктори множин об'єктів. З появою перших робіт по теорії множин [110], розпочалися активні дослідження у даному напрямку. У результаті чого, досить швидко цю теорію стали відносити до основ математики. Протягом всієї історії розвитку даної теорії було запропоновано багато її варіантів, це в першу чергу було пов'язано з парадоксами, які були знайдені у Канторівській теорії множин. Після чого відбулися деякі спроби побудувати теорію множин використовуючи підходи відмінні від запропонованого Кантором, так з'явилася теорія типів Рассела (T) [111, 112], яку пізніше розвинув Куайн (NF), (ML) [112, 113]. Згодом відбулися спроби аксіоматизації теорії множин, які відомі в історії, як аксіоматичні теорії множин Цермело-Френкеля (ZF) та фон Неймана-Бернайса-Гєделя (NBG) [112–114]. Пізніше з'явилася альтернативна теорія множин, яку запропонував П. Вопенка (AST) [115, 116].

Незважаючи на розмаїття підходів до побудови теорії множин, одним з найважливіших критеріїв для усіх її версій є конструктивність побудованої теорії, оскільки саме конструктивність є гарантією існування алгоритмів побудови різного роду об'єктів та вирішення задач в її рамках. Однак аналізуючи вище згадані версії теорії множин, виникають питання про походження так званих "конкретних множин". Виходячи з основ класичної теорії множин [117], можна зробити висновок, що "нові множини" можна отримувати шляхом застосування теоретико-множинних операцій до "вже існуючих множин" і це дійсно так. Проте не зникають питання про походження цих так званих "вже існуючих множин", їхню кількість, їх види і т. д. [2, 7–11]. Виникають питання про можливості автоматизації процесу створення множин для обчислювальних машин (комп'ютерів); автоматизації класифікації та ідентифікації елементів множин; автоматичної генерації множин, що відносяться до певного класу. Іншими словами виникає питання про можливість практичної реалізації для комп'ютерів здатності оперувати такою базовою категорією людського мислення як "множина".

Перш ніж перейти до розгляду процесів створення множин об'єктів, розглянемо деякі важливі твердження сучасної теорії множин. Згідно з [117], поняття множини можна визначити наступним чином.

Означення 2.5.45. Множина – це довільна сукупність визначених і відмінних між собою, об'єктів нашої інтуїції або інтелекту (елементи множини), які розглядаються як єдине ціле.

З наведеного означення слідує: сукупність елементів розглядається як один об'єкт; повинен існувати механізм порівняння об'єктів; множина однозначно визначається (задається) своїми елементами. Аналізуючи Означення2.5.45, можна прийти до висновку, що воно не є конструктивним, оскільки не дає прямої відповіді на те, як саме створювати (будувати) множини. Однак згідно з [114, 117] виділяють два способи визначення (задання) множин:

1. Таблична форма – безпосереднє визначення сукупності елементів множини, тобто $S = \{a, b, \ldots, z\}$.
2. За ознакою – визначення характерних властивостей для елементів множини, тобто $S = \{x \mid p(x)\}$, де $p(x)$ – це властивість, яка виконується для усіх елементів множини S.

Одними з широко використовуваних у теорії множин понять, є поняття порожньої $\varnothing$ та одноелементної $\{a\}$ множини.

Означення 2.5.46. Порожня множина – це множина, що не містить елементів і визначається наступним чином $\varnothing = \{x \mid x \neq x\}$.

Означення 2.5.47. Одноелементна множина – це множина, що містить в точності лише один елемент і визначається як $S_1 = \{x\}$, де $|S_1| = 1$.

Аналізуючи Означення 2.5.45, 2.5.46, 2.5.47 та вище зазначені способи визначення множин, можна прийти до наступних неоднозначностей:

1. Якщо будь-яка множина однозначно визначається сукупністю своїх елементів, то порожня множина є щонайменше *невизначеною множиною*.
2. Якщо будь-яка множина це сукупність відмінних між собою елементів, то порожня множина не є множиною згідно з Означенням 2.5.45, оскільки

порожня множина не містить жодного елементу.

3. Аналогічно, якщо будь-яка множина це сукупність відмінних між собою елементів, то одноелементна множина не є множиною згідно з Означенням 2.5.45, оскільки будь-яка одноелементна множина складається з одного елемента, який є тотожним самому собі.
4. Немає сенсу вводити означення, згідно з яким одиничний об'єкт розглядатиметься як одне ціле, оскільки він уже є ним, інакше виникає питання у чому принципова різниця між x та $\{x\}$.
5. Із семантичного значення слова *сукупність* слідує, що сукупність складається з двох і більше елементів. Таким чином порожня та одноелементні множини не є сукупностями елементів, а значить і множинами згідно з Означенням 2.5.45.
6. Згідно з другим способом визначення множин, порожню множину можна визначити через визначення певної властивості такої множини, наприклад $S_1 = \{x \mid x \neq x\} = \{\varnothing\}$, однак знову приходимо до пунктів 1, 2 та 5.
7. Аналогічно, згідно з другим способом визначення множин, будь-яку одноелементну множину можна визначити через певну властивість такої множини, наприклад $S_1 = \{x \mid x = 5 - 5\} = \{0\}$, однак знову приходимо до пунктів 3, 4 та 5.

Аналізуючи усі вище сформульовані твердження, можна зробити висновок, що порожня та одноелементні множини не є множинами згідно з Означенням 2.5.45, а спосіб визначення множин через визначення їх властивостей дозволяє визначати, як порожню так і одноелементні множини. Таким чином спосіб задання множин через визначення їх властивостей, потребує наступних уточнень

$$S = \{x_1, \ldots, x_n \mid (n \geqslant 2) \wedge (\forall x_i, x_j, i, j = \overline{1,n}, i \neq j \mid Eq(x_i, x_j) = 0) \wedge$$
$$\wedge (\forall x_i, i = \overline{1,n} \mid p_1(x_i)) \wedge \cdots \wedge (\forall x_i, i = \overline{1,n} \mid p_m(x_i)))\},$$

де $x_1, \ldots, x_n$ – елементи множини S, $Eq(x_i, x_j)$ – це функція, яка реалізує механізм порівняння елементів x_i та x_j, а $p_1(x_i), \ldots, p_m(x_i)$ – це властивості, які виконуються для елементів множини S.

Тепер враховуючи усі вище зазначені зауваження, Означення2.1.8, 2.5.8 та означення об'єднання множин [91, 117] сформулюємо конструктивне означення поняття *множина об'єктів*.

Означення 2.5.48. Множина об'єктів S – це результат застосування універсального експлуататора об'єднання до довільних $n \geqslant 2$ об'єктів, або $m \geqslant 2$ множин об'єктів, або $q \geqslant 1$ об'єктів та $w \geqslant 1$ множин об'єктів.

Використовуючи Означення2.5.48, можна визначити три універсальних конструктори (генератори) множин об'єктів.

Означення 2.5.49. Універсальний конструктор (генератор) множин об'єктів USC1 – це схема побудови множин об'єктів, яка визначається наступним чином

$$O_1/T_{O_1} \cup \cdots \cup O_n/T_{O_n} = \{O_1, \ldots, O_{i_1}\}/T_S = S/T_S,$$

де O_i, $i = \overline{1,n}$ – це довільні об'єкти, T_{O_i} – це їх класи, S – це новоутворена множина об'єктів, а T_S – це її клас, $n \geqslant 2$, $i_1 \leqslant n$.

Означення 2.5.50. Універсальний конструктор (генератор) множин об'єктів USC2 – це схема побудови множин об'єктів, яка визначається наступним чином

$$S_1/T_{S_1} \cup \cdots \cup S_m/T_{S_m} = \{O_1, \ldots, O_{i_2}\}/T_S = S/T_S,$$

S_i, $i = \overline{1,m}$ – це множини довільних об'єктів, T_{S_i} – це їх класи, S – це новоутворена множина об'єктів, T_S – це її клас, $m \geqslant 2$, $i_2 \leqslant |S_1| + \cdots + |S_m|$.

Означення 2.5.51. Універсальний конструктор (генератор) множин об'єктів USC3 – це схема побудови множин об'єктів, яка визначається наступним чином

$$O_1/T_{O_1} \cup \cdots \cup O_q/T_{O_q} \cup S_1/T_{S_1} \cup \cdots \cup S_w/T_{S_w} = \{O_1, \ldots, O_{i_3}\}/T_S = S/T_S,$$

де O_i, $i = \overline{1,q}$ – це довільні об'єкти, T_{O_i} – це їх класи, S_j, $j = \overline{1,w}$ – це множини довільних об'єктів, T_{S_j} – це їх класи, S – це новоутворена множина об'єктів, а T_S – це її клас, $q, w \geqslant 1$, $i_3 \leqslant q + |S_1| + \cdots + |S_w|$.

Конструктор USC1 побудований на основі універсального експлуататора об'єднання об'єктів (див. Означення2.5.8). Конструктор USC2 побудований на основі

універсального експлуататора об'єднання множин об'єктів, який певною формою універсального експлуататора об'єднання об'єктів. Конструктор USC3 побудований на використанні універсальних експлуататорів об'єднання об'єктів та об'єднання множин об'єктів, що також є деякою формою першого з них.

В залежності від різнотипності об'єктів, однорідності та неоднорідності класів множин об'єктів результуюча множина об'єктів буде однорідною (міститиме лише однотипні об'єкти) або неоднорідною (міститиме різнотипні об'єкти) [2, 7–11]. У зв'язку з чим, визначимо поняття однорідної та неоднорідної множини об'єктів.

Означення 2.5.52. Множина об'єктів $S = \{O_1, \ldots, O_n\}/T_S$ є однорідною (неоднорідною) тоді і тільки тоді, коли клас T_S є однорідним (неоднорідним).

2.5.3.2. Універсальні конструктори мультимножин об'єктів. Через певний час після появи аксіоматичних версій теорії множин ZF та NBG, з'явилися перші роботи присвячені теорії мультимножин [118–122], яка у певному сенсі є узагальненням класичної теорії множин.

Неформально поняття мультимножини можна визначити наступним чином.

Означення 2.5.53. Мультимножина – це множина, для будь-якого елементу якої може існувати $n \geqslant 1$ еквівалентних йому елементів.

Таким чином будь-яку мультимножину можна визначити на основі деякої множини, у зв'язку з чим, для мультимножин вводять поняття *базової множини*.

Означення 2.5.54. Базова (породжуюча) множина довільної мультимножини $M = \{x_1, \ldots, x_n\}$ – це така множина $S_b = \{x_1, \ldots, x_m\}$, що

$$(\forall x_i \in M,\ \exists! x_j \in S_b) \wedge (\forall x_j \in S_b,\ \exists x_i \in M) \mid x_i = x_j,$$

де $i = \overline{1, n}$, $j = \overline{1, m}$ і $n \geqslant m$.

Використовуючи Означення 2.5.54, поняття мультимножини можна визначити наступним чином.

Означення 2.5.55. Мультимножина – це множина пар виду

$$M(S) = \{(x_1, m_1), \ldots, (x_n, m_n)\},$$

де $S = \{x_1, \ldots, x_n\}$ – це породжуюча множина, а $m_1, \ldots, m_n$ – це кратності (кількості входжень) елементів $x_1, \ldots, x_n$ до мультимножини $M(S)$, які визначаються як $m_i : x_i \to N$, де $i = \overline{1, n}$, а N – це множина натуральних чисел.

Означення 2.5.56. Потужність довільної мультимножини M – це сума кратностей усіх її елементів, тобто

$$|M| = \sum_{i=1}^{n} m_i,$$

де n – це кількість пар виду (x_i, m_i).

Тепер враховуючи усі вище зазначені зауваження, Означення2.1.8, 2.5.9 та означення об'єднання мультимножин [118–122] сформулюємо конструктивне означення поняття *мультимножина об'єктів*.

Означення 2.5.57. Мультимножина об'єктів M – це результат мультиоб'єднання довільних $n \geqslant 2$ об'єктів, або $m \geqslant 2$ множин об'єктів, або $k \geqslant 2$ мультимножин об'єктів, або $q \geqslant 1$ об'єктів та $w \geqslant 1$ множин об'єктів, або $q \geqslant 1$ об'єктів та $l \geqslant 1$ мультимножин об'єктів, або $w \geqslant 1$ множин та $l \geqslant 1$ мультимножин об'єктів, або $q \geqslant 1$ об'єктів та $w \geqslant 1$ множин і $l \geqslant 1$ мультимножин об'єктів.

Використовуючи Означення2.5.57, можна визначити сім універсальних конструкторів (генераторів) мультимножин об'єктів.

Означення 2.5.58. Універсальний конструктор (генератор) мультимножин об'єктів UMC1 – це схема побудови мультимножин об'єктів, яка визначається наступним чином $O_1/T_{O_1} \sqcup \cdots \sqcup O_{n_1}/T_{O_{n_1}} = \{O_1, \ldots, O_{n_1}\}/T_M = M/T_M$, де O_i – це довільні об'єкти, T_{O_i} – це їх класи, де $i = \overline{1, n_1}$, M – це новоутворена мультимножина об'єктів, а T_M – це її клас, $n_1 \geqslant 2$.

Означення 2.5.59. Універсальний конструктор (генератор) мультимножин об'єктів UMC2 – це схема побудови мультимножин об'єктів, яка визначається наступним чином $S_1/T_{S_1} \sqcup \cdots \sqcup S_{n_2}/T_{S_{n_2}} = \{O_1, \ldots, O_{i_2}\}/T_M = M/T_M$, де S_i – це множини довільних об'єктів, T_{S_i} – це їх класи, де $i = \overline{1, n_2}$, M – це новоутворена мультимножина об'єктів, T_M – це її клас, $n_2 \geqslant 2$, $i_2 = |S_1| + \cdots + |S_{n_2}|$.

Означення 2.5.60. Універсальний конструктор (генератор) мультимножин об'єктів UMC3 – це схема побудови мультимножин об'єктів, яка визначається наступним чином $M_1/T_{M_1} \sqcup \cdots \sqcup M_{n_3}/T_{M_{n_3}} = \{O_1, \ldots, O_{i_3}\}/T_M = M/T_M$, де M_i – це мультимножини довільних об'єктів, T_{M_i} – це їх класи, же $i = \overline{1, n_3}$, M – це новоутворена мультимножина об'єктів, T_M – це її клас, $n_3 \geqslant 2$, $i_3 = |M_1| + \cdots + |M_{n_3}|$.

Означення 2.5.61. Універсальний конструктор (генератор) мультимножин об'єктів UMC4 – це схема побудови мультимножин об'єктів, яка визначається наступним чином

$$O_1/T_{O_1} \sqcup \cdots \sqcup O_{n_4}/T_{O_{n_4}} \sqcup S_1/T_{S_1} \sqcup \cdots \sqcup S_{n_5}/T_{S_{n_5}} = \{O_1, \ldots, O_{i_4}\}/T_M = M/T_M,$$

де O_i, $i = \overline{1, n_4}$ – це довільні об'єкти, T_{O_i} – це їх класи, S_j, $j = \overline{1, n_5}$ – це множини довільних об'єктів, T_{S_j} – це їх класи, M – це новоутворена мультимножина об'єктів, а T_M – це її клас, $n_4, n_5 \geqslant 1$, $i_4 = n_4 + |S_1| + \cdots + |S_{n_5}|$.

Означення 2.5.62. Універсальний конструктор (генератор) мультимножин об'єктів UMC5 – це схема побудови мультимножин об'єктів, яка визначається наступним чином

$$O_1/T_{O_1} \sqcup \cdots \sqcup O_{n_6}/T_{O_{n_6}} \sqcup M_1/T_{M_1} \sqcup \cdots \sqcup M_{n_7}/T_{M_{n_7}} = \\ = \{O_1, \ldots, O_{i_5}\}/T_M = M/T_M,$$

де O_i, $i = \overline{1, n_6}$ – це довільні об'єкти, T_{O_i} – це їх класи, M_j, $j = \overline{1, n_7}$ – це мультимножини довільних об'єктів, T_{M_j} – це їх класи, M – це новоутворена мультимножина об'єктів, а T_M – це її клас, $n_6, n_7 \geqslant 1$, $i_5 = n_6 + |M_1| + \cdots + |M_{n_7}|$.

Означення 2.5.63. Універсальний конструктор (генератор) мультимножин об'єктів UMC6 – це схема побудови мультимножин об'єктів, яка визначається наступним чином

$$S_1/T_{S_1} \sqcup \cdots \sqcup S_{n_8}/T_{S_{n_8}} \sqcup M_1/T_{M_1} \sqcup \cdots \sqcup M_{n_9}/T_{M_{n_9}} = \{O_1, \ldots, O_{i_6}\}/T_M = M/T_M,$$

де S_i, $i = \overline{1, n_8}$ – це множини довільних об'єктів, T_{S_i} – це їх класи, M_j – це мультимножини довільних об'єктів, T_{M_j} – це їх класи, де $j = \overline{1, n_9}$, M –

це новоутворена мультимножина об'єктів, а T_M – це її клас, $n_8, n_9 \geqslant 1$, $i_6 = |S_1| + \cdots + |S_{n_8}| + |M_1| + \cdots + |M_{n_9}|$.

Означення 2.5.64. Універсальний конструктор (генератор) мультимножин об'єктів UMC7 – це схема побудови мультимножин об'єктів, яка визначається наступним чином

$$O_1/T_{O_1} \sqcup \cdots \sqcup O_{n_{10}}/T_{O_{n_{10}}} \sqcup S_1/T_{S_1} \sqcup \cdots \sqcup S_{n_{11}}/T_{S_{n_{11}}} \sqcup$$
$$\sqcup M_1/T_{M_1} \sqcup \cdots \sqcup M_{n_{12}}/T_{M_{n_{12}}} = \{O_1, \ldots, O_{i_7}\}/T_M = M/T_M,$$

де O_i, $i = \overline{1, n_{10}}$ – це довільні об'єкти, T_{O_i} – це їх класи, де S_j, $j = \overline{1, n_{11}}$ – це множини довільних об'єктів, T_{S_j} – це їх класи, M_k, $k = \overline{1, n_{12}}$ – це мультимножини довільних об'єктів, T_{M_k} – це їх класи, M – це новоутворена мультимножина об'єктів, а T_M – це її клас, $n_{10}, n_{11}, n_{12} \geqslant 1$, $i_7 = n_{10} + |S_1| + \cdots + |S_{n_{11}}| + |M_1| + \cdots + |M_{n_{12}}|$.

Конструктор UMC1 побудований на основі універсального експлуататора мультиоб'єднання об'єктів (див. Означення2.5.9), а конструктори UMC2, UMC3, UMC4, UMC5, UMC6 та UMC7 побудовані на основі його різних форм.

Використовуючи Означення 2.5.54, 2.5.9, 2.5.14 можна визначити універсальні конструктори однорідних та неоднорідних мультимножин об'єктів [2, 6, 25].

Означення 2.5.65. Універсальний конструктор однорідних мультимножин об'єктів UHMC – це схема побудови мультимножин об'єктів, що визначається наступним чином

$$A/T_A \sqcup \left(\bigsqcup_{i=1}^{m} Clone_i(A/T_A) \right) = M/T_M,$$

де m – це кратність об'єкта A класу T_A, M – це новоутворена мультимножина об'єктів, а T_M – це її клас.

Означення 2.5.66. Універсальний конструктор неоднорідних мультимножин об'єктів UIMC – це схема побудови мультимножин об'єктів, що визначається наступним чином

$$\bigsqcup_{i=1}^{n} \left(A_i/T_{A_i} \sqcup \left(\bigsqcup_{j=1}^{m_i} Clone_j(A_i/T_{A_i}) \right) \right) = M/T_M,$$

де $m_1, \ldots, m_n$ – це кратності об'єктів $A_1/T_{A_1}, \ldots, A_n/T_{A_n}$ відповідно, M – це новоутворена мультимножина об'єктів, а T_M – це її клас.

2.5.3.3. Спеціалізовані конструктори мультимножин об'єктів. Усі вище визначені конструктори мультимножин об'єктів є універсальними, оскільки з їх допомогою можна побудувати довільні мультимножини об'єктів. Однак використовуючи Означення 2.5.44, можна будувати і більш спеціалізовані конструктори мультимножин об'єктів з певними корисними властивостями [2, 3, 11, 12]. У зв'язку з цим визначимо CP, RCL, PS та D2 конструктори мультимножин об'єктів та покажемо деякі їх корисні властивості.

CP конструктор. Даний конструктор мультимножин об'єктів використовує концепцію декартового добутку множин (Cartesian Product). Однак на відміну від класичного означення декартового добутку множин, замість пар, які є елементами цього добутку, використовуються множини об'єктів. Перш ніж визначити сам конструктор, введемо операцію індексування копій елементів у мультимножинах об'єктів.

Означення 2.5.67. Індексування об'єкта $A_i/(p_1(A_i), \ldots, p_n(A_i))$ – це перевизначення його індексу наступним чином $Ind_w(A_i) = A_{i+w}/(p_1(A_i), \ldots, p_n(A_i))$, де w – це приріст індексу i.

Означення 2.5.68. *CP* конструктор мультимножин об'єктів – це схема побудови мультимножин об'єктів, яка визначається наступним чином

$$CP(S_1, S_2) = \bigsqcup_{i=1}^{n} \bigsqcup_{j=1}^{m} \left(Ind_j(A_i) \sqcup Ind_i(B_j)\right),$$

де S_1, S_2 – це базові множини для мультимножини об'єктів $CP(S_1, S_2)$, $A_i \in S_1$, $B_j \in S_2$ і $n = |S_1|$, $m = |S_2|$.

Тепер покажемо деякі корисні властивості CP конструктора мультимножин об'єктів.

Теорема 2.5.1. *Потужність будь-якої мультимножини об'єктів M, утвореної за допомогою CP конструктора, дорівнює $2nm$, де $n = |S_1|$, $m = |S_2|$, а S_1 та S_2 – це базові множини для мультимножини об'єктів M.*

Доведення. Відомо, що потужність Декартового добутку двох довільних множин S_1 та S_2, можна обчислити наступним чином $|S_1 \times S_2| = |S_1| \cdot |S_2| = nm$. З того, що всі елементи Декартового добутку двох множин є упорядкованими парами, слідує що $|CP(S_1, S_2)| = 2nm$. □

Теорема 2.5.2. *Кратність $m(A_i)$ кожного об'єкта A_i з мультимножини об'єктів M, утвореної за допомогою CP конструктора, можна обчислити за допомогою наступної формули*

$$m(A_i) = \begin{cases} |S_1|, & \exists B_j \in S_1 \mid Eq(A_i, B_j) = 1, \\ |S_2|, & \exists C_k \in S_2 \mid Eq(A_i, C_k) = 1, \end{cases}$$

де $i = \overline{1, 2nm}$, $n = |S_1|$, $m = |S_2|$, $j = \overline{1, n}$, $k = \overline{1, m}$, а S_1, S_2 – базові множини для мультимножини об'єктів M.

Доведення. Доведення слідує з означення Декартового добутку множин. □

Розглянемо приклад можливого застосування CP конструктора мультимножин об'єктів.

Приклад 2.5.2. Припустимо що існує необхідність сконструювати електричну гірлянду з зелених (G), жовтих (Y), оранжевих (O), синіх (B), фіолетових (P) та рожевих (R) лампочок. Однак перш ніж перейти до безпосереднього виготовлення гірлянди необхідно створити для неї певну кольорову схему, а отже необхідно вирішити які кольори і скільки лампочок кожного кольору будуть використані для цього.

Для кращої демонстративності будемо використовувати лампочки усіх шести кольорів з умовою що в кольоровій схемі майбутньої гірлянди лампочки кожного кольору мають зустрічатися більше ніж один раз. Тепер розділимо множину кольорів $S = \{G, Y, O, B, P, R\}$ на дві довільні підмножини S_1 та S_2 потужностей $n \geqslant 2$ та $m \geqslant 2$ відповідно, таких що $S_1 \cap S_2 = \varnothing$, а $n + m = |S|$. Нехай $S_1 = \{G, Y, O\}$, а $S_2 = \{B, P, R\}$.

Використовуючи Теорему 2.5.1 обчислимо потужність мультимножини, що утворюватиме майбутню кольорову схему гірлянди, в результаті чого отримає-

мо що $|CP(S_1, S_2)| = 2nm = 2 \cdot 3 \cdot 3 = 18$. Отже для кольорової схеми майбутньої гірлянди знадобиться 18 лампочок.

Тепер використовуючи Теорему 2.5.2 обчислимо кратність кожного елементу майбутньої мультимножини $CP(S_1, S_2)$, іншими словами кількість ламп кожного кольору, у результаті чого отримаємо що $m(A_i) = 3$ де $A_i \in S$, оскільки $|S_1| = |S_2| = 3$. Таким чином кольорова схема майбутньої гірлянди міститиме по 3 лампи 6 кольорів.

Тепер застосуємо CP конструктор мультимножин об'єктів до множин S_1, S_2 та побудуємо кольорову схему майбутньої гірлянди, у результаті чого отримаємо

$$CP(S_1, S_2) = \bigsqcup_{i=1}^{3} \bigsqcup_{j=1}^{3} (Ind_j(A_i \in S_1) \sqcup Ind_i(B_j \in S_2)) =$$

$$= \{G_1, B_1\}/T_{S_1} \sqcup \{G_2, P_1\}/T_{S_2} \sqcup \{G_3, R_1\}/T_{S_3} \sqcup \{Y_1, B_2\}/T_{S_4} \sqcup \{Y_2, P_2\}/T_{S_5} \sqcup$$

$$\sqcup \{Y_3, R_2\}/T_{S_6} \sqcup \{O_1, B_3\}/T_{S_7} \sqcup \{O_2, P_3\}/T_{S_8} \sqcup \{O_3, R_3\}/T_{S_9} =$$

$$= \{G_1, B_1, G_2, P_1, G_3, R_1, Y_1, B_2, Y_2, P_2, Y_3, R_2, O_1, B_3, O_2, P_3, O_3, R_3\}/T_S.$$

Побудувавши мультимножину $CP(S_1, S_2)$ можна переконатися у тому що вона справді містить по 3 лампи 6 різних кольорів, при цьому кожна з лам має свій унікальний номер. Варіюючи розбиття множини базових кольорів S на дві підмножини S_1 та S_2, що не перетинаються, можна будувати й інші варіанти кольорових схем гірлянди, однак не залежно від цього вона складатиметься з 18 ламп, серед яких будуть лампи усіх 6 кольорів у кількості $2 \leqslant k \leqslant 4$ штук.

RCL конструктор. Основою даного конструктора мультимножин об'єктів є рекурсивне клонування (Recursive Cloning) множин об'єктів. Перш ніж визначити конструктор, введемо операцію клонування множин об'єктів.

Означення 2.5.69. Клон довільної множини об'єктів $S = \{O_1, \ldots, O_n\}/T_S$ – це множина об'єктів $Clone_i(S) = S_i = \{O_{i+i}, \ldots, O_{n+i}\}/T_S$, де T_S – це клас множини об'єктів S, а i – це номер її клону.

Означення 2.5.70. RCL^n конструктор мультимножин об'єктів – це схема

побудови мультимножин об'єктів, яка визначається наступним чином

$$RCL^n(S) = \begin{cases} S, & n = 0, \\ S \sqcup Clone_{2^{n-1}}(S), & n = 1, \\ RCL^{n-1}(S) \sqcup Clone_{2^{n-1}}(RCL^{n-1}(S)), & n \geqslant 2, \end{cases}$$

де S – це базова множина для мультимножини об'єктів $RCL^n(S)$, а n – глибина рекурсії.

Тепер покажемо деякі корисні властивості RCL конструктора мультимножин об'єктів.

Теорема 2.5.3. *Потужність будь-якої мультимножини об'єктів M, утвореної за допомогою RCL^n конструктора, дорівнює $n2^i$, $n = |S_b|$, де S_b – базова множина для мультимножини об'єктів M, а i – це глибина рекурсії.*

Доведення. Згідно з визначенням RCL конструктора, на кожному кроці створення мультимножини об'єктів M виконується мультиоб'єднання двох множин об'єктів однакової потужності. Такими чином, якщо $|S_b| = n$, то на кроці $i = 1$ ми отримаємо мультимножину об'єктів, потужність якої обчислюється наступним чином $n + n = 2n = n2^i$. На кроці $i = 2$ ми отримаємо мультимножину об'єктів потужності $2n + 2n = 4n$, що можна представити як $n2^1 + n2^1 = n2^2$, на кроці $i = 3$ ми отримаємо мультимножину об'єктів потужності $4n + 4n = 8n$, що можна представити як $n2^2 + n2^2 = n2^3$ і так далі. Таким чином на кроці $i = k$ ми отримаємо мультимножину об'єктів потужності $n2^{k-1} + n2^{k-1} = n2^k$, таким чином потужність результуючої мультимножини об'єктів буде рівна $n2^i$, де i – це глибина рекурсії, щой потрібно було довести. □

Теорема 2.5.4. *Кратність $m(A_j)$ довільного об'єкта A_j з мультимножини об'єктів M, утвореної за допомогою RCL^n конструктора, рівна 2^i, де $j = \overline{1, n2^i}$, $n = |S_b|$, де S_b – це базова множина для мультимножини об'єктів M, а i – це глибина рекурсії RCL конструктора.*

Доведення. З означення RCL конструктора слідує, що на кожному кроці i, RCL конструктор однаково збільшує кратність усіх елементів множини об'єктів

S. Відповідно до Теореми 2.5.3, потужність результуючої мультимножин об'єктів $|S| = n2^i$, де i – це глибина рекурсії RCL конструктора. Використовуючи ці факти, можна прийти до висновку що $m(A_j) = n2^i/n = 2^i$, що й потрібно було довести. □

Розглянемо приклад можливого застосування RCL конструктора мультимножин об'єктів.

Приклад 2.5.3. Модифікуємо Приклад 2.5.2 з конструюванням кольорової схеми гірлянди і припустимо що множина базових кольорів складається із зеленого (G), жовтого (Y) та червоного (R) кольорів, тобто $S = \{G, Y, R\}$. Припустимо що необхідна нам глибина рекурсії рівна 2.

Використовуючи Теорему 2.5.3 обчислимо потужність мультимножини, що утворюватиме майбутню кольорову схему гірлянди, в результаті чого отримаємо що $|RCL^2(S)| = n2^i = 3 \cdot 2^2 = 12$. Отже для кольорової схеми майбутньої гірлянди знадобиться 12 лампочок.

Тепер використовуючи Теорему 2.5.4 обчислимо кратність кожного елементу майбутньої мультимножини $RCL^2(S)$, іншими словами кількість ламп кожного кольору, у результаті чого отримаємо що $m(A_i) = 2^i = 2^2 = 4$. Таким чином кольорова схема майбутньої гірлянди міститиме по 4 лампи 3 кольорів.

Тепер застосуємо RCL конструктор мультимножин об'єктів до множини S та побудуємо кольорову схему майбутньої гірлянди, у результаті чого отримаємо

$$RCL^2(S) = RCL^1(S) \sqcup Clone_2\left(RCL^1(S)\right) =$$
$$= \{G, Y, R\}/T_S \sqcup \{G_1, Y_1, R_1\}/T_S \sqcup \{G_2, Y_2, R_2, G_3, Y_3, R_3\}/T_S =$$
$$= \{G, Y, R, G_1, Y_1, R_1, G_2, Y_2, R_2, G_3, Y_3, R_3\}/T_S.$$

Побудувавши мультимножину $RCL^2(S)$ можна переконатися у тому що вона справді містить по 4 лампи 3 різних кольорів, при цьому кожна з лам має свій унікальний номер.

PS конструктор. Даний конструктор мультимножин об'єктів базується на ідеї мультиоб'єднання усіх підмножин деякої множини об'єктів. Однак перш ніж визначити сам конструктор сформулюємо та доведемо наступне твердження.

Твердження 2.5.3. Кількість усіх підмножин довільної множини об'єктів S дорівнює $2^n - n - 1$, де $n = |S|$.

Доведення. Відомо що булеан довільної множини S – це множина елементами якої є усі підмножини S, порожня множина $\varnothing$ та сама множина S [91, 117, 123]. Булеан довільної множини S визначається як $P(S) = \{P \mid P \subseteq S\}$. Таким чином булеан множини $S = \{a, b, c\}$ матиме наступний вигляд

$$P(S) = \{\{\varnothing\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Відомо що потужність булеану $P(S)$ довільної множини S дорівнює 2^n, де $n = |S|$ [91, 117, 123]. Однак, згідно з Означенням 2.5.48, $\{\varnothing\}$, $\{a\}$, $\{b\}$, $\{c\}$ не є множинами об'єктів, тому формула обчислення потужності булеану довільної множини може бути перевизначена для обчислення кількості усіх можливих підмножин множини об'єктів S наступним чином $q(S_w) = 2^n - n - 1$, де S_w – це довільна підмножина S, а $q(S_w)$ – це кількість усіх можливих S_w. Що й потрібно було довести. □

Означення 2.5.71. *PS* конструктор мультимножин об'єктів – це схема побудови мультимножин об'єктів, яка визначається наступним чином

$$PS(S) = \bigsqcup_{w=1}^{2^n-n-1} S_w,$$

де S – це базова множина об'єктів для мультимножини об'єктів $PS(S)$, а S_w – це довільна підмножина S.

Тепер покажемо деякі корисні властивості PS конструктора мультимножин об'єктів.

Теорема 2.5.5. *Потужність будь-якої мультимножини об'єктів M, утвореної за допомогою PS конструктора, дорівнює $n\left(2^{n-1} - 1\right)$, де $n = |S_b|$, а S_b – це базова множина для мультимножини об'єктів M.*

Доведення. Розглянемо множину $S_1 = \{a, b, c\}$ та побудуємо її булеан

$$P(S_1) = \{\{\varnothing\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}.$$

Тепер шляхом об'єднання усіх елементів множини S_1 утворимо нову мультимножину M_1. Очевидно що $M_1 = \{a, b, c, a, b, a, c, b, c, a, b, c\}$, а $|M_1| = 12$.

Розглянемо множину $S_2 = \{a, b, c, d\}$ та побудуємо її булеан

$$P(S_2) = \{\{\varnothing\}, \{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\},$$
$$\{a, b, c\}, \{a, b, d\}, \{a, c, d\}, \{b, c, d\}, \{a, b, c, d\}.\}$$

Тепер аналогічно до випадку з M_1 утворимо нову мультимножину M_2. Очевидно що $M_2 = \{a, b, c, d, a, b, a, c, a, d, b, c, b, d, c, d, a, b, c, a, b, d, a, c, d, b, c, d, a, b, c, d\}$, а $|M_2| = 32$.

Відомо що потужність булеану $|P(S)| = 2^n$, де $n = |S|$. Враховуючи те що $|P(S_1)| = 2^3 = 8$, представимо $|M_1|$ наступним чином $|M_1| = \left(3 \cdot 2^3\right)/2 = 12$, що дійсно має місце. Враховуючи те що $P(S_2) = 2^4 = 16$, аналогічним чином представимо $|M_2|$ як $|M_2| = \left(4 \cdot 2^4\right)/2 = 32$, що також має місце. Використовуючи принцип математичної індукції отримуємо що для мультимножини M_k потужності n, має місце $|M_k| = (n2^n)/2 = n2^{n-1}$.

Із Твердження 2.5.3 слідує, що кількість усіх підмножин довільної множини об'єктів S рівна $2^n - n - 1$, де $n = |S|$. З Означення 2.5.48 слідує що, об'єкти і порожня множина $\varnothing$ не є множинами об'єктів. Враховуючи це і те що $|\varnothing| = 0$, приходимо до висновку що $|M_o| = n2^{n-1} - n = n\left(2^{n-1} - 1\right)$, де M_o – це довільна мультимножина об'єктів утворена за допомогою PS конструктора. Що й треба було довести. □

Теорема 2.5.6. *Кратність $m(A_i)$ довільного об'єкта A_i з мультимножини об'єктів M, утвореної за допомогою PS конструктора, дорівнює $2^{n-1} - 1$, де $n = |S_b|$, а S_b – це базова множина для мультимножини об'єктів M.*

Доведення. Представимо генерацію можливих підмножин об'єктів $S_1, \ldots, S_w$ для множини об'єктів S_b може як комбінацію $k = \overline{2, n}$ різних елементів з множини, що містить n елементів, тобто C_n^k. Підмножини потужності 2 утворюються шляхом комбінування кожного об'єкта A_i з кожним об'єктом множини $S_b \setminus A_i$. Очевидно що можливо утворити лише $n - 1$ таких підмножин, тобто C_{n-1}^1. У

випадку підмножин потужності 3 можна утворити C_{n-1}^{2} таких підмножин, а у випадку потужності k отримаємо C_{n-1}^{k-1}.

Відповідно до означення PS конструктора, кратність кожного об'єкта $A_i \in S$ є сумою кратностей об'єкта A_i в кожній підмножині, тобто

$$m(A_i) = \sum_{w=1}^{2^n-n-1} m_w(A_i),$$

де $m_w(A_i)$ – це кратність об'єкта A_i у підмножині об'єктів $S_w \subseteq S_b$. Це випливає з $S_1 \cup \cdots \cup S_w$, де $w = 2^n - n - 1$. Використовуючи цей факт приходимо до того що $m(A_i) = C_{n-1}^{1} + \cdots + C_{n-1}^{n-1}$, де $n = |S_b|$. Це означає що кожен об'єкт $A_i \in S$ має однакову кратність. З чого слідує що

$$m(A_i) = \frac{|S|}{|S_b|} = \frac{n\left(2^{n-1}-1\right)}{n} = 2^{n-1} - 1,$$

де $n = |S_b|$. Що й потрібно було довести. □

З доведення Теореми 2.5.6 слідує, що кратність кожного об'єкта A_i з мультимножини об'єктів M може бути розрахована як сума відповідних біноміальних коефіцієнтів. Отже таким чином можна побудувати деяку частину трикутника Паскаля, яку можна поєднати з Теоремою 2.5.5 та 2.5.6. У зв'язку з чим сформулюємо наступний наслідок.

Наслідок 2.5.6.1. *Потужність мультимножини об'єктів утвореної за допомогою PS конструктора, кратність кожного її об'єкта та кількість підмножин об'єктів, з яких була створена мультимножина можна обчислити за допомогою наступної матриці*

$m(A_k)$	$\lvert S\rvert$	$q(S_w)$	$\lvert S_b\rvert$	2	3	4	5	6	…
1	2	1	2	1					
3	9	4	3	3	1				
7	28	11	4	6	4	1			,
15	75	26	5	10	10	5	1		
31	186	57	6	15	20	15	6	1	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

де стовпчик $m(A_k)$ – це кратності об'єкта A_k у мультимножині об'єктів S, стовпчик $|S|$ – це потужності мультимножини об'єктів S, стовпчик $q(S_w)$ – це кількості підмножин об'єктів $S_w \subseteq S$, які були використані для утворення мультимножин об'єктів S, стовпчик $|S_b|$ – це потужності базової множини об'єктів, стовпчики 2, 3, 4, 5, 6, ... – це кількості підмножин об'єктів потужності 2, 3, 4, 5, 6, ..., відповідно, які були використані для утворення мультимножини об'єктів S.

Елементи стовпчика $m(A_k)$ можна обчислити за допомогою теореми 2.5.6 або наступної формули

$$a_{i\geqslant 2,1} = \begin{cases} 1, & i = 2, \\ \sum_{i\geqslant 2, j\geqslant 4} a_{i-1,j}, & i \geqslant 2. \end{cases}$$

Елементи стовпчика $|S|$ можна обчислити за допомогою теореми 2.5.5 або наступної формули

$$a_{i\geqslant 2,2} = \sum_{i\geqslant 2, j\geqslant 5} a_{i,j} a_{1,j}.$$

Елементи стовпчика $q(S_w)$ можна обчислити за допомогою твердження 2.5.3 або наступної формули

$$a_{i\geqslant 2,3} = \sum_{i\geqslant 2, j\geqslant 5} a_{i,j}.$$

Елемент $a_{i\geqslant 2, j\geqslant 5}$ можна обчислити за допомогою формули

$$a_{i\geqslant 2, j\geqslant 5} = \begin{cases} 1, & j - i = 1, \\ a_{i-1,j-1} + a_{i-1,j}, & j - i < 3, \end{cases}$$

або формули

$$a_{i\geqslant 2, j\geqslant 5} = \frac{a_{i,4}!}{a_{1,j}!(a_{i,4} - a_{1,j})!}.$$

Розглянемо приклад можливого застосування PS конструктора мультимножин об'єктів.

Приклад 2.5.4. Розглянемо Приклад 2.5.3 з конструюванням кольорової схеми для гірлянди, де множина базових кольорів складається з трьох кольорів $S = \{G, Y, R\}$.

Використовуючи Теорему 2.5.5 обчислимо потужність мультимножини, що утворюватиме майбутню кольорову схему гірлянди, в результаті чого отримаємо що $|PS(S)| = n\left(2^{n-1}-1\right) = 3 \cdot \left(2^2-1\right) = 9$. Отже для кольорової схеми майбутньої гірлянди знадобиться 9 лампочок.

Тепер використовуючи Теорему 2.5.6 обчислимо кратність кожного елементу майбутньої мультимножини $PS(S)$, іншими словами кількість ламп кожного кольору, у результаті чого отримаємо що $m(A_i) = 2^{n-1} - 1 = 2^2 - 1 = 3$. Таким чином кольорова схема майбутньої гірлянди міститиме по 3 лампи 3 кольорів.

На відміну від попередніх конструкторів мультимножин, PS конструктор сам по собі не нумерує копії об'єктів в побудовані мультимножині. У зв'язку з чим введемо наступну процедуру нумерації. Оскільки усі підмножини об'єктів множини S вибираються з цієї множини, то необхідно індексувати обрані копії в цій множині, таким чином нумеруючи елементи відповідним чином для наступних підмножин.

$$S_1 = \{G, Y\},\ S = \{Ind_1(G), Ind_1(Y), R\} = \{G_1, Y_1, R\},$$
$$S_2 = \{G_1, R\},\ S = \{Ind_1(G_1), Y_1, Ind_1(R)\} = \{G_2, Y_1, R_1\},$$
$$S_3 = \{Y_1, R_1\},\ S = \{G_2, Ind_1(Y_1), Ind_1(R_1)\} = \{G_2, Y_2, R_2\},$$
$$S_4 = \{G_2, Y_2, R_2\},\ S = \{Ind_1(G_2), Ind_1(Y_2), Ind_1(R_2)\} = \{G_3, Y_3, R_3\}.$$

Тепер застосуємо PS конструктор мультимножин об'єктів до множини S та побудуємо кольорову схему майбутньої гірлянди, у результаті чого отримаємо

$$PS(S) = \{G, Y\}/TS_1 \sqcup \{G_1, R\}/T_{S_2} \sqcup \{Y_1, R_1\}/T_{S_3} \sqcup \{G_2, Y_2, R_2\}/T_{S_4} =$$
$$= \{G, Y, G_1, R, Y_1, R_1, G_2, Y_2, R_2\}/T_S.$$

Побудувавши мультимножину $PS(S)$ можна переконатися у тому що вона справді містить по 3 лампи 3 різних кольорів, при цьому кожна з лам має свій унікальний номер. Варіюючи порядок вибору підмножин з множини базових кольорів S, можна будувати й інші варіанти кольорових схем гірлянди, однак не залежно від цього вона складатиметься з 9 ламп, по 3 кожного кольору.

D2 конструктор. Даний конструктор мультимножин об'єктів базується на ідеї декомпозиції базової множини на дві підмножини, що не мають перетину, а їх об'єднання утворює базову множину. Після чого будується мультиоб'єднання об'єднань таких пар. Однак перш ніж визначити цей конструктор, доведемо наступне твердження.

Твердження 2.5.4. Кількість усіх можливих підмножин множини об'єктів S, отриманих за допомогою бінарної декомпозиції дорівнює $2^n - 2n - 2$, де $n = |S|$.

Доведення. Відповідно до Твердження 2.5.3, кількість усіх можливих підмножин множини об'єктів S рівна $2^n - n - 1$, де $n = |S|$. Проте результат бінарної декомпозиції множини об'єктів не містить підмножин потужності $n-1$ та n. Відомо що для будь-якої множини об'єктів S потужності n існує лише одна підмножина потужності n. Кількість підмножин потужності $n-1$ можна обчислити наступним чином

$$C_n^{n-1} = \frac{n!}{(n-1)!(n-(n-1))!} = \frac{n!}{(n-1)!1!} = \frac{n!}{(n-1)!} = n,$$

це випливає з доведення Теореми 2.5.6. Враховуючи усі ці факти отримуємо що $q(S_w) = 2^n - n - 1 - n - 1 = 2^n - 2n - 2$, де $n = |S|$. Що й потрібно було довести. □

Означення 2.5.72. *D2* конструктор мультимножин об'єктів – це схема побудови мультимножин об'єктів, яка визначається наступним чином

$$D2(S) = \bigsqcup_{w=1}^{2^n-2n-2} (S_1 \cup S_2),$$

де S – це базова множина для мультимножини об'єктів $D2(S)$, $S_1, S_2 \subseteq S$ – це множини об'єктів, що не мають перетину і для яких має місце $S_1 \cup S_2 = S$, $n = |S|$.

Теорема 2.5.7. *Потужність будь-якої мультимножини об'єктів M, утвореної за допомогою D2 конструктора, дорівнює $n\left(2^{n-1} - n - 1\right)$, де $n = |S_b|$, а S_b – це базова множина для мультимножини об'єктів M.*

Доведення. Відповідно до Теореми 2.5.5, потужність будь-якої мультимножини об'єктів S, утвореної за допомогою PS конструктора дорівнює $n(2^{n-1} - 1)$, де $n = |S_b|$. З доведення Твердження 2.5.4, слідує що результат бінарної декомпозиції множини об'єктів S не містить підмножин потужності $n - 1$ та n. Також

відомо що для довільної множини, потужності n, існує одна n-елементна та n $(n-1)$-елементних підмножин. Враховуючи всі ці факти приходимо до того що $|S| = n\left(2^{n-1}-1\right) - n(n-1) - n = n\left(2^{n-1}-n-1\right)$, де $n = |S_b|$. Що й потрібно було довести. □

Теорема 2.5.8. *Кратність $m(A_i)$ кожного об'єкта A_i з мультимножини об'єктів M, утвореної за допомогою $D2$ конструктора, дорівнює $2^{n-1}-n-1$, де $n = |S_b|$, а S_b – це базова множина для мультимножини об'єктів M.*

Доведення. З доведення Теореми 2.5.6, випливає що для довільної множини об'єктів S_b, потужності n, можливо побудувати лише C_{n-1}^{k-1} підмножин об'єктів потужності k. Також відомо що усі об'єкти $A_i \in S$ мають однакову кратність. Враховуючи ці факти приходимо до того що

$$m(A_i) = \frac{|S|}{|S_b|} = \frac{n\left(2^{n-1}-n-1\right)}{n} = 2^{n-1}-n-1,$$

де $n = |S_b|$. Що й потрібно було довести. □

З доведення Теореми 2.5.8 слідує, що кратність кожного об'єкта A_i з мультимножини об'єктів M може бути розрахована як сума відповідних біноміальних коефіцієнтів. Отже таким чином можна побудувати деяку частину трикутника Паскаля, яку можна поєднати з Теоремою 2.5.7 та 2.5.8. У зв'язку з чим сформулюємо наступний наслідок.

Наслідок 2.5.8.1. *Потужність мультимножини об'єктів утвореної за допомогою $D2$ конструктора, кратність кожного її об'єкта та кількість підмножин об'єктів, з яких була створена мультимножина можна обчислити за до-*

помогою наступної матриці

$m(A_k)$	$\lvert S\rvert$	$q(S_w)$	$\lvert S_b\rvert$	2	3	4	5	6	...
3	12	6	4	6					
10	50	20	5	10	10				
25	150	50	6	15	20	15			,
56	392	112	7	21	35	35	21		
119	952	238	8	28	56	70	56	28	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

де стовпчик $m(A_k)$ – це кратності об'єкта A_k у мультимножині об'єктів S, стовпчик $|S|$ – це потужності мультимножини об'єктів S, стовпчик $q(S_w)$ – це кількості підмножин об'єктів $S_w \subseteq S$, які були використані для утворення мультимножин об'єктів S, стовпчик $|S_b|$ – це потужності базової множини об'єктів, стовпчики 2, 3, 4, 5, 6, ... – це кількості підмножин об'єктів потужності 2, 3, 4, 5, 6, ..., відповідно, які були використані для утворення мультимножини об'єктів S.

Елементи стовпчика $m(A_k)$ можна обчислити за допомогою теореми 2.5.8 або наступної формули

$$a_{i\geqslant 2,1} = \begin{cases} 3, & i = 2, \\ \sum_{i\geqslant 2, j\geqslant 4} a_{i-1,j}, & i > 2. \end{cases}$$

Елементи стовпчика $|S|$ можна обчислити за допомогою теореми 2.5.7 або наступної формули

$$a_{i\geqslant 2,2} = \sum_{i\geqslant 2, j\geqslant 5} a_{i,j} a_{1,j}.$$

Елементи стовпчика $q(S_w)$ можна обчислити за допомогою твердження 2.5.4 або наступної формули

$$a_{i\geqslant 2,3} = \sum_{i\geqslant 2, j\geqslant 5} a_{i,j}.$$

Елемент $a_{i\geqslant 2,j\geqslant 5}$ можна обчислити за допомогою формули

$$a_{i\geqslant 2,j\geqslant 5} = \begin{cases} 6, & i = 2,\ j = 5, \\ a_{i-1,j-1} + a_{1,j+1}, & j > 5,\ j - i = 3, \\ a_{i-1,j-1} + a_{i-1,j}, & j - i < 3, \end{cases}$$

або формули

$$a_{i\geqslant 2,j\geqslant 5} = \frac{a_{i,4}!}{a_{1,j}!(a_{i,4} - a_{1,j})!}.$$

Розглянемо приклад можливого застосування D2 конструктора мультимножин об'єктів.

Приклад 2.5.5. Розглянемо Приклад 2.5.3 з конструюванням кольорової схеми для гірлянди та припустимо що множина базових кольорів складається з чотирьох кольорів $S = \{G, Y, R, B\}$.

Використовуючи Теорему 2.5.7 обчислимо потужність мультимножини, що утворюватиме майбутню кольорову схему гірлянди, в результаті чого отримаємо що $|D2(S)| = n\left(2^{n-1} - n - 1\right) = 4 \cdot \left(2^3 - 4 - 1\right) = 12$. Отже для кольорової схеми майбутньої гірлянди знадобиться 12 лампочок.

Тепер використовуючи Теорему 2.5.8 обчислимо кратність кожного елементу майбутньої мультимножини $D2(S)$, іншими словами кількість ламп кожного кольору, у результаті чого отримаємо що $m(A_i) = 2^{n-1} - n - 1 = 2^3 - 4 - 1 = 3$. Таким чином кольорова схема майбутньої гірлянди міститиме по 3 лампи 4 кольорів.

На відміну від CP та RCL конструкторів мультимножин, D2 конструктор сам по собі не нумерує копії об'єктів в побудованій мультимножині. У зв'язку з чим введемо наступну процедуру нумерації. Оскільки усі підмножини об'єктів множини S вибираються з цієї множини, то необхідно індексувати елементи цих підмножин відповідно до номеру декомпозиції, таким чином нумеруючи елементи відповідним чином для наступних підмножин.

$$S_1 = \{Ind_1(G), Ind_1(Y)\} = \{G_1, Y_1\},\ S_2 = \{Ind_1(R), Ind_1(B)\} = \{R_1, B_1\},$$
$$S_3 = \{Ind_2(G), Ind_2(Y)\} = \{G_2, Y_2\},\ S_4 = \{Ind_2(R), Ind_2(B)\} = \{R_2, B_2\},$$
$$S_5 = \{Ind_3(G), Ind_3(Y)\} = \{G_3, Y_3\},\ S_5 = \{Ind_3(R), Ind_3(B)\} = \{R_3, B_3\},$$

Тепер застосуємо D2 конструктор мультимножин об'єктів до множини S та побудуємо кольорову схему майбутньої гірлянди, у результаті чого отримаємо

$$D2(S) = \{G_1, Y_1\}/T_{S_1} \sqcup \{R_1, B_1\}/T_{S_2} \sqcup \{G_2, Y_2\}/T_{S_3} \sqcup \{R_2, B_2\}/T_{S_4} \sqcup$$
$$\sqcup \{G_3, Y_3\}/T_{S_5} \sqcup \{R_3, B_3\}/T_{S_6} = \{G_1, Y_1, R_1, B_1, G_2, Y_2, R_2, B_2, G_3, Y_3, R_3, B_3\}/T_S$$

Побудувавши мультимножину $D2(S)$ можна переконатися у тому що вона справді містить по 3 лампи 4 різних кольорів, при цьому кожна з лам має свій унікальний номер.

2.6. Об'єктно-орієнтовані динамічні мережі та їх властивості

Враховуючи усі переваги та недоліки відомих об'єктно-орієнтованих моделей подання знань, що були розглянуті та проаналізовані у першому розділі, та використовуючи визначені поняття об'єктів, класів об'єктів, відношень між класами та об'єктами, експлуататорів, модифікаторів об'єктів та класів об'єктів, генераторів множин та мультимножин об'єктів, визначимо наступну об'єктно-орієнтовану динамічну модель подання знань [4, 13, 22].

Означення 2.6.1. Об'єктно-орієнтована динамічна мережа (ООДМ) – це кортеж виду $OODN = (O, C, R, E, M, G)$, де O – це множина об'єктів, C – це множина класів об'єктів, R, E, M та G – це відповідно множини відношень, експлуататорів, модифікаторів та генераторів визначених на множинах O та C.

Глобально запропоновану модель подання знань концептуально можна поділити на дві частини – *декларативну* та *процедурну*. Декларативна частина включає в себе множини O, C та R, які дозволяю формалізувати декларативні знання про ту чи іншу предметну область. Процедурна частина включає в себе множини E, M та G, які дозволяю отримувати нові знання на основі базових знань [4, 17]. Таким чином ООДМ дозволяє формалізацію в об'єктно-орієнтованому стилі як декларативних так і процедурних знань.

Оскільки значна частина людських знань мають нечітку та розмиту природу [124–129], то можливість формалізації знань такого типу є важливим критерієм

для сучасних моделей подання знань. У зв'язку з чим запропонована модель подання знань може бути узагальнена на нечіткий випадок [15, 16, 19, 23, 24].

2.6.1. Видобування знань. Однією з ключових функцій сучасних інтелектуальних програмних систем на основі знань є *видобування знань*, яку згідно з [20, 104–106, 130–134], можна визначити наступним чином.

Означення 2.6.2. Видобування знань – це процес створення або одержання знань з структурованих (наприклад, реляційних баз даних, об'єктно-орієнтованих баз даних, UML, XML та їх нечітких розширень, запропонованих в [135, 136]), напів-структурованих (наприклад, інформаційні блоки) і неструктурованих (наприклад, текст, документи, зображення) джерел даних.

Важливим аспектом цього процесу є видобування знань у термінах певної моделі подання знань, яка використовується в інтелектуальній програмній системі і дозволяє певним чином інтерпретувати (трактувати) видобуту інформацію [20]. Однак специфіка різних моделей подання знань не дозволяє безперешкодно використовувати одні й ті ж методи видобування знань в інтелектуальних програмних системах на основі різних моделей подання знань. У зв'язку з чим кожна модель подання знань потребує розробки для неї відповідних методів видобування знань та адаптації відомих методів для моделей того ж типу.

Таким чином можна сформулювати наступне означення поняття *база знань*.

Означення 2.6.3. База знань – це кортеж $KB = (KRF, DB)$, де KRF – це певна модель подання знань, а DB – це база даних для збереження інформації в термінах KRM.

Аналізуючи Означення 2.6.2, можна виділити наступні типи джерел інформації (даних) для видобування знань:

1. **Група експертів.** Типове джерело інформації для експертних систем, коли група експертів $Expert_1, \ldots, Expert_n$ у певній предметній області, формалізують свої знання про цю область у термінах певної моделі подання знань KRF. У цьому випадку видобування знань виконують експерти.
2. **Бази даних.** Типове джерело інформації для багатьох програмних си-

стем. Інформаційне наповнення відокремлених баз даних $DB_1, \ldots, DB_m$ може відбуватися як за допомогою експертів так і автоматично. У випадку автоматичного наповнення видобування знань виконує програмна система.

3. **Вебсторінки.** Типове джерело інформації для більшості користувачів мережі Internet. Інформаційне наповнення вебсторінок $wp_1, \ldots, wp_k$ може відбуватися двома шляхами: за допомогою баз даних або експертів (формування статичних сторінок або наповнення баз даних). У цьому випадку видобування знань виконують експерти або окремі модулі програмних систем.

4. **Файли.** Типове джерело інформації як для користувачів програмних систем так і для самих систем. Інформаційне наповнення файлів $f_1, \ldots, f_l$ також може відбуватися двома шляхами: за допомогою експертів або автоматично. У цьому випадку видобування знань виконують експерти або окремі модулі програмних систем.

5. **Бази знань.** Типове джерело інформації для інтелектуальних програмних систем. Інформаційне наповнення баз знань $KB_1, \ldots, KB_w$ може відбуватися як за допомогою експертів та і в автоматичному режимі коли робить сама система. У такому випадку видобування знань можуть виконувати як експерти так і окремі модулі програмних систем.

Усі зазначені типи джерел інформації та процес видобування знань інтелектуальною програмною системою зображено на Рис. 2.7 та Рис. 2.8.

Глобально можна виділити два типи видобування знань – *зовнішнє* та *внутрішнє*. У випадку зовнішнього видобування, програмна система шляхом інтелектуального аналізу знаходить нову інформацію та нові знання із зовнішніх джерел, які не є частиною системи. У випадку внутрішнього видобування, програмна система аналізує знання що наявні у базі знань системи та намагається отримати нові (неявні) знання на основі базових знань.

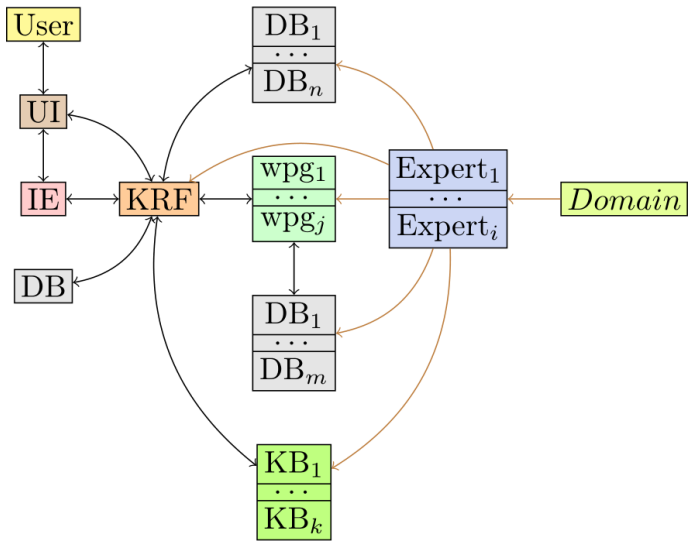


Рис. 2.7: Процес видобування знань інтелектуальною програмною системою з різнотипних джерел інформації

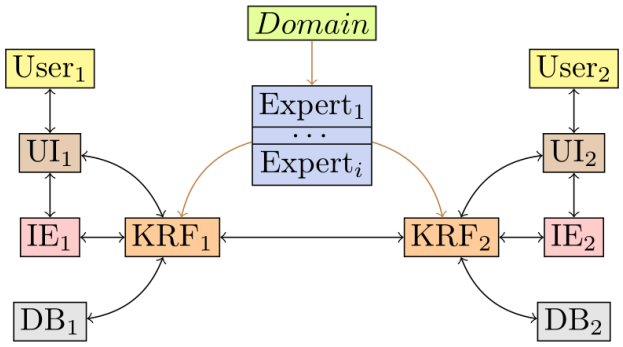


Рис. 2.8: Процес взаємного видобування знань однією інтелектуальною програмною системою з іншої інтелектуальної програмної системи

2.6.2. Видобування знань в ООДМ. В рамках фреймових систем та об'єктно-орієнтованого програмування зовнішнє видобування знань відбувається шляхом *парсингу за зразком*, коли програмна система намагається знайти об'єкти або класи з відомою їй структурою, у той час як внутрішнє видобування знань відбувається шляхом *якісної фільтрації*, коли об'єкти та класи аналізуються за певними характеристиками, що дозволяє отримувати нові (неявні) знання про них.

У попередніх підрозділах були визначені універсальні експлуататори об'єктів та класів об'єктів, які дозволяють будувати нові об'єкти та класи на основі базових. Такий підхід дозволяє розширювати множину класів об'єктів ООДМ за

рахунок утворення нових класів. Розглянемо процеси розширення підмножини однорідних класів множини класів ООДМ за допомогою універсальних експлуататорів об'єднання і однорідного перетину, та доведемо деякі корисні властивості таких розширень [17, 20].

Теорема 2.6.1. *Для будь-якої ООДМ виду*

$$OODN = (O, C = \{T_1, \dots, T_n, T_{n+1}, \dots, T_m\}, R, E = \{\cup, \dots\}, M),$$

де $T_1, \dots, T_n$ *– однорідні класи об'єктів, що визначають попарно-нееквівалентні типи* $t_1, \dots, t_n$*, такі що* $\forall t_i, t_j \in \{t_1, \dots, t_n\}, i, j = \overline{1, n}, i \neq j, \nexists t \mid (t \subset t_i) \wedge (t \subset t_j)$*, усі можливі застосування експлуататора об'єднання (включаючи усі можливі суперпозиції) до класів об'єктів з множини* C *і класів об'єктів отриманих за допомогою експлуататора об'єднання, завжди генерують* $2^n - n - 1$ *нових унікальних класів об'єктів, де* $n = |C|$*.*

Доведення. Згідно з Означенням 2.5.2, результатом об'єднання n довільних класів об'єктів $T_1, \dots, T_n$, що визначають типи об'єктів $t_1^1, \dots, t_{m_1}^1, \dots, t_1^n, \dots, t_{m_n}^n$ відповідно, є клас об'єктів $T_{1,\dots,n}$, який визначає у точності всі типи об'єктів, що визначають класи $T_1, \dots, T_n$. Якщо для $\forall T_1, \dots, T_k \in \{T_1, \dots, T_n\}$, де $k = \overline{2, n}$ не існує типу $t \mid t \subset t_1, \dots, t \subset t_k$, то клас $T_{1,\dots,n}$ складатиметься лише з проекцій типів, тобто $T_{1,\dots,n} = (pr_1(t_1), \dots, pr_n(t_n))$.

Відомо, що кількість усіх можливих унікальних класів об'єктів створених на основі базової множини класів $C = \{T_1, \dots, T_n\}$ з використанням експлуататора об'єднання $\cup$, може бути представлена у вигляді суми $k = \overline{2, n}$ комбінацій різних класів об'єктів з множини класів C. Відомо що

$$\sum_{n=0}^{k} C_n^k = 2^n.$$

Однак застосовуючи експлуататор об'єднання до різних класів об'єктів з множини C, неможливо утворити класи об'єктів, які б визначали 1 та 0 різних типів об'єктів, оскільки клас об'єктів T_{12}, який утвориться в результаті об'єднання $\forall T_1, T_2 \in C$, визначатиме, що найменше 1 тип об'єктів, однак за умовою теореми,

класи об'єктів $T_1, \ldots, T_n$ не мають спільних властивостей та методів, тому клас T_{12} визначатиме щонайменше 2 різних типи об'єктів, отже

$$q(C_{\cup}) = \sum_{k=0}^{n} C_n^k - C_n^1 - C_n^0 = \sum_{k=2}^{n} C_n^k = 2^n - n - 1,$$

що й потрібно було довести. □

Теорема 2.6.2. *Для будь-якої ООДМ, для якої виконуються усі умови Теореми 2.6.1, підмножина однорідних класів $C_{JSL} = \{T_1, \ldots, T_n\}$ множини класів об'єктів $C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_m\}$, розширена відповідно до Теореми 2.6.1, разом з універсальним експлуататором об'єднання $\cup$, утворюють верхню напів-ґратку з одиницею $JSL = (C_{JSL} = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}\}, E_{JSL} = \{\cup, 1\})$, де клас $T_{JSL} = T_1 \cup \cdots \cup T_n$ – її найбільша верхня грань, тобто* 1.

Доведення. Згідно з з означеннями, наведеними у [90–92, 123, 137–140], верхня напівґратка це система $JSM = (A, \Omega = \{\vee, 1\})$, де A – це частково упорядкована множина (носій напів-ґратки), $\vee$ – це бінарна ідемпотентна, комутативна та асоціативна операція і 1 – це унарна операція, що визначені на множині A. Окрім цього $\forall a \in A,\ 1$ виконується властивість $(L_1) : a \vee 1 = 1$.

Відповідно до теореми, носієм напівґратки є множина класів об'єктів C, а множина експлуататорів E містить бінарну операцію $\cup$ та унарну операцію 1, визначені на множині C, тобто $JSL = (C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}\}, E = \{\cup, 1\})$.

Відповідно до Означення 2.5.2, результатом об'єднання n довільних класів об'єктів $T_1, \ldots, T_n$, що визначають типи об'єктів $t_1^1, \ldots, t_{m_1}^1, \ldots, t_1^n, \ldots, t_{m_n}^n$ відповідно, є клас об'єктів $T_{1,\ldots,n}$, який визначає у точності всі типи об'єктів, що визначають класи $T_1, \ldots, T_n$, з чого випливає, що усі вище зазначені властивості для операції $\vee$ виконуються і для експлуататора об'єднання класів об'єктів $\cup$, тобто $\forall T_1, T_2, T_3 \in C$ має місце ідемпотентність $(T_1 \cup T_1 = T_1)$, комутативність $(T_1 \cup T_2 = T_2 \cup T_1)$ та асоціативність $(T_1 \cup (T_2 \cup T_3) = (T_1 \cup T_2) \cup T_3)$. Також Означення 2.5.2 слідує, що $\forall T \in C,\ T \cup T_{JSL} = T_{JSL}$, де $T_{JSL} = T_1 \cup \cdots \cup T_n$.

Тепер покажемо, що множина C є частково упорядкованою множиною. Для цього потрібно визначити відношення $\forall T_1, T_2 \in C \mid T_1 \subseteq T_2 \Leftrightarrow T_1 \cup T_2 = T_2$ та

показати що воно є відношенням часткового порядку на множині C. Таке відношення було визначене раніше за допомогою Означення 2.3.16. Тепер доведемо, що воно є відношенням часткового порядку, тобто є рефлексивним, антисиметричним та транзитивним.

Рефлексивність: $T_1 \subseteq T_1 \Leftrightarrow T_1 \cup T_1 = T_1$ слідує з ідемпотентності експлуататора об'єднання.

Антисиметричність: $T_1 \subseteq T_2 \Leftrightarrow T_1 \cup T_2 = T_2$, $T_2 \subseteq T_1 \Leftrightarrow T_2 \cup T_1 = T_1$. З комутативності експлуататора об'єднання випливає, що $T_1 = T_2$.

Транзитивність:

$$\begin{array}{l} T_1 \subseteq T_2 \Leftrightarrow T_1 \cup T_2 = T_2 \\ T_2 \subseteq T_3 \Leftrightarrow T_2 \cup T_3 = T_3 \end{array} \Longrightarrow (T_1 \cup T_2) \cup T_3 = T_1 \cup (T_2 \cup T_3) =$$

$$= T_1 \cup T_3 = T_3 \Rightarrow T_1 \cup T_3 = T_3 \Leftrightarrow T_1 \subseteq T_3.$$

Отже $JSL = (C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}\}, E = \{\cup, 1\})$ є верхньою напівґраткою з одиницею, що й потрібно було довести. □

Теорема 2.6.3. *Для будь-якої ООДМ виду*

$$OODN = (O, C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_m\}, R, E = \{\cup, \ldots\}, M),$$

де $T_1, \ldots, T_n$ – однорідні класи об'єктів, що визначають попарно-нееквівалентні типи $t_1, \ldots, t_n$, для яких існує тип t, такий що $t \subset t_1, \ldots, t \subset t_n$, усі можливі застосування експлуататора об'єднання (включаючи усі можливі суперпозиції) до класів об'єктів з множини C і класів об'єктів, отриманих за допомогою експлуататора об'єднання, завжди генерують $2^n - n - 1$ нових унікальних класів об'єктів, де $n = |C|$.

Доведення. Відповідно до Означення 2.5.2, результатом об'єднання n довільних класів об'єктів $T_1, \ldots, T_n$, що визначають типи $t_1^1, \ldots, t_{m_1}^1, \ldots, t_1^n, \ldots, t_{m_n}^n$ відповідно, є клас об'єктів $T_{1,\ldots,n}$, який визначає у точності всі типи об'єктів, що визначають класи $T_1, \ldots, T_n$. Якщо існує тип t такий що $t \subset t_1, \ldots, t \subset t_n$, то клас $T_{1,\ldots,n}$ матиме наступну структуру $T_{1,\ldots,n} = (Core(T_{1,\ldots,n}), pr_1(t_1), \ldots, pr_2(t_n))$.

Відомо, що кількість усіх можливих унікальних класів об'єктів створених на основі базової множини класів $C = \{T_1, \ldots, T_n\}$ з використанням експлуататора об'єднання $\cup$, може бути представлена у вигляді суми $k = \overline{2,n}$ комбінацій різних класів об'єктів з множини класів C. Відомо що

$$\sum_{n=0}^{k} C_n^k = 2^n.$$

Однак застосовуючи експлуататор об'єднання до різних класів об'єктів з множини C, неможливо утворити класи об'єктів, які б визначали 1 та 0 різних типів об'єктів, оскільки клас об'єктів T_{12}, який утвориться в результаті об'єднання $\forall T_1, T_2 \in C$, визначатиме, що найменше 1 тип об'єктів, однак за умовою теореми, для класів об'єктів $T_1, \ldots, T_n$ існує спільний підтип t, такий що $t \subset t_1, \ldots, t \subset t_n$, а отже $D(t) + func(t) < D(T_1) + D(T_2) + func(T_1) + func(T_2)$, тому клас T_{12} визначатиме щонайменше 2 різних типи об'єктів, у результаті чого

$$q(C_E) = \sum_{k=0}^{n} C_n^k - C_n^1 - C_n^0 = \sum_{k=2}^{n} C_n^k = 2^n - n - 1,$$

що й потрібно було довести. □

Теорема 2.6.4. *Для будь-якої ООДМ, для якої виконуються усі умови Теореми 2.6.3, підмножина однорідних класів $C_{JSL} = \{T_1, \ldots, T_n\}$ множини класів об'єктів $C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_m\}$, розширена відповідно до Теореми 2.6.3, разом з експлуататором об'єднання $\cup$, утворюють верхню напівґратку з одиницею $JSL = (C_{JSL} = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}\}, E_{JSL} = \{\cup, 1\})$, де клас $T_{JSL} = T_1 \cup \cdots \cup T_n$ – її найбільша верхня грань, тобто* 1.

Доведення. Згідно з означеннями, наведеними у [90–92, 123, 137–140], верхня напівґратка це система $JSM = (A, \Omega = \{\vee, 1\})$, де A – це частково упорядкована множина (носій напівґратки), $\vee$ – це бінарна ідемпотентна, комутативна та асоціативна операція і 1 – це унарна операція, що визначені на множині A. Окрім цього $\forall a \in A,\ 1$ виконується властивість $(L_1) : a \vee 1 = 1$.

Відповідно до теореми, носієм напівґратки є множина класів об'єктів C, а множина експлуататорів E містить бінарну операцію $\cup$ та унарну операцію 1, визначені на множині C, тобто $JSL = (C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}\}, E = \{\cup, 1\})$.

Відповідно до Означення 2.5.2, результатом об'єднання n довільних класів об'єктів $T_1, \ldots, T_n$, що визначають типи об'єктів $t_1^1, \ldots, t_{m_1}^1, \ldots, t_1^n, \ldots, t_{m_n}^n$ відповідно, є клас об'єктів $T_{1,\ldots,n}$, який визначає у точності всі типи об'єктів, що визначають класи $T_1, \ldots, T_n$, з чого випливає, що усі вище зазначені властивості для операції $\vee$, виконуються і для експлуататора об'єднання класів об'єктів $\cup$, тобто $\forall T_1, T_2, T_3 \in C$ має місце ідемпотентність $(T_1 \cup T_1 = T_1)$, комутативність $(T_1 \cup T_2 = T_2 \cup T_1)$ та асоціативність $(T_1 \cup (T_2 \cup T_3) = (T_1 \cup T_2) \cup T_3)$. Також Означення 2.5.2 слідує, що $\forall T \in C,\ T \cup T_{JSL} = T_{JSL}$, де $T_{JSL} = T_1 \cup \cdots \cup T_n$.

З доведення Теореми 2.6.2, слідує що множина C є частково упорядкованою множиною, на якій визначене відношення включення $\subseteq$, яке є відношенням часткового порядку, тому $JSL = (C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}\}, E = \{\cup, 1\})$ є верхньою напівґраткою з одиницею, що й потрібно було довести. □

Теорема 2.6.5. *Для будь-якої ООДМ виду*

$$OODN = (O, C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_m\}, R, E = \{\cap, \ldots\}, M),$$

де $T_1, \ldots, T_n$ – однорідні класи об'єктів, що визначають попарно-нееквівалентні типи $t_1, \ldots, t_n$, для яких існує тип t, такий що

$$((t \subset t_1) \wedge \cdots \wedge (t \subset t_n)) \wedge (\nexists t' \mid (t \subset t') \wedge (t' \subset t_1) \wedge \cdots \wedge (t' \subset t_n)),$$

усі можливі застосування експлуататора однорідного перетину (включаючи усі можливі суперпозиції) до класів об'єктів з множини C і класів об'єктів, отриманих за допомогою експлуататора однорідного перетину, завжди генерують $2^n - n - 1$ нових унікальних класів об'єктів, де $n = |C|$.

Доведення. Відповідно до Означення 2.5.3, результатом застосування універсального експлуататора однорідного перетину n довільних класів об'єктів $T_1, \ldots, T_n$, які визначають типи об'єктів $t_1^1, \ldots, t_{m_1}^1, \ldots, t_1^n, \ldots, t_{m_n}^n$ відповідно, є клас об'єктів $T_{1\cap\cdots\cap n}$, який визначає тип об'єктів $t \mid t \subset t_1, \ldots, t \subset t_n$.

Відомо, що кількість усіх можливих унікальних класів об'єктів створених на основі базової множини класів $C = \{T_1, \ldots, T_n\}$ з використанням експлуататора

однорідного перетину $\cap$, може бути представлена у вигляді суми $k = \overline{2,n}$ комбінацій різних класів об'єктів з множини класів C. Відомо що

$$\sum_{n=0}^{k} C_n^k = 2^n.$$

Однак, оскільки експлуататор однорідного перетину класів об'єктів є n-арною операцією, де $n \geqslant 2$, то у результаті однорідного перетину $\forall T_1, \ldots, T_k \in \{T_1, \ldots, T_n\}$, де $k = \overline{2,n}$ не можливо утворити клас, який визначає підтип t, що є спільним для 0 класів об'єктів, оскільки відповідно до теореми, для усіх типів $t_1, \ldots, t_n$, які визначені класами $T_1, \ldots, T_n$, існує спільний підтип $t \mid t \subset t_1, \ldots, t \subset t_n$. Також згідно з теоремою, не можливо утворити жодного нового класу об'єктів, який б визначав підтип t, який є спільним для 1 класу об'єктів, оскільки усі такі класи будуть збігатися з множиною базових класів $T_1, \ldots, T_n$, так як вони можуть бути утворені лише як $T_i \cap T_i = T_i$, де $i = \overline{1,n}$. В результаті чого комбінації C_n^0 та C_n^1 не можуть враховуватися, тому

$$q(C_\cap) = \sum_{k=0}^{n} C_n^k - C_n^1 - C_n^0 = \sum_{k=2}^{n} C_n^k = 2^n - n - 1,$$

що й потрібно було довести. □

Теорема 2.6.6. *Для будь-якої ООДМ, для якої виконуються усі умови Теореми 2.6.5, підмножина однорідних класів $C_{MSL} = \{T_1, \ldots, T_n\}$ множини класів об'єктів $C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_m\}$, розширена відповідно до Теореми 2.6.5, разом з експлуататором однорідного перетину $\cap$, утворюють нижню напівґратку з нулем $MSL = (C_{MSL} = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}\}, E_{MSL} = \{\cap, 0\})$, де клас $T_\cap = T_1 \cap \cdots \cap T_n$ – її найменша нижня грань, тобто* 0.

Доведення. Згідно з означеннями, наведеними у [90–92, 123, 137–140], нижня напівґратка це система $MSM = (A, \Omega = \{\wedge, 0\})$, де A – це частково упорядкована множина (носій напівґратки), $\wedge$ – це бінарна ідемпотентна, комутативна та асоціативна операція, а 0 – це унарна операція, що визначені на множині A. Окрім цього $\forall a \in A,\ 0$ виконується властивість $(L_1) : a \wedge 0 = 0$.

Відповідно до теореми, носієм напів-ґратки є множина класів об'єктів C, а множина експлуататорів E містить бінарну операцію $\cap$ та унарну операцію 0, визначені на множині C, тобто $MSL = (C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}\}, E = \{\cap, 0\})$.

Відповідно до Означення 2.5.3, результатом однорідного перетину n довільних класів об'єктів $T_1, \ldots, T_n$, що визначають типи об'єктів $t_1^1, \ldots, t_{m_1}^1, \ldots, t_1^n, \ldots, t_{m_n}^n$ відповідно, є клас об'єктів $T_{1\cap\ldots\cap n}$, який визначає тип об'єктів $t \mid t \subset t_1, \ldots, t \subset t_n$, з чого випливає, що усі вище зазначені властивості для операції $\wedge$ виконуються і для експлуататора однорідного перетину класів об'єктів $\cap$, тобто $\forall T_1, T_2, T_3 \in C$ має місце ідемпотентність $(T_1 \cap T_1 = T_1)$, комутативність $(T_1 \cap T_2 = T_2 \cap T_1)$ та асоціативність $(T_1 \cap (T_2 \cap T_3) = (T_1 \cap T_2) \cap T_3)$. Також Означення 2.5.3 слідує, що $\forall T \in C,\ T \cap T_{MSL} = T_{MSL}$, де $T_{MSL} = T_1 \cap \cdots \cap T_n$.

Тепер покажемо, що множина C є частково упорядкованою множиною. Для цього потрібно визначити відношення $\forall\ T_1, T_2 \in C \mid T_1 \subseteq T_2 \Leftrightarrow T_1 \cap T_2 = T_1$ та показати що воно є відношенням часткового порядку на множині C. Таке відношення було визначене раніше за допомогою Означення 2.3.16. Тепер доведемо, що воно є відношенням часткового порядку, тобто є рефлексивним, антисиметричним та транзитивним.

Рефлексивність: $T_1 \subseteq T_1 \Leftrightarrow T_1 \cap T_1 = T_1$ слідує з ідемпотентності експлуататора однорідного перетину $\cap$.

Антисиметричність: $T_1 \subseteq T_2 \Leftrightarrow T_1 \cap T_2 = T_1$, $T_2 \subseteq T_1 \Leftrightarrow T_2 \cap T_1 = T_2$. З комутативності експлуататора однорідного перетину випливає, що $T_1 = T_2$.

Транзитивність:

$$\left.\begin{array}{l} T_1 \subseteq T_2 \Leftrightarrow T_1 \cap T_2 = T_2 \\ T_2 \subseteq T_3 \Leftrightarrow T_2 \cap T_3 = T_3 \end{array}\right\} \Longrightarrow (T_1 \cap T_2) \cap T_3 = T_1 \cap (T_2 \cap T_3) =$$

$$= T_1 \cap T_3 = T_3 \Rightarrow T_1 \cap T_3 = T_3 \Leftrightarrow T_1 \subseteq T_3.$$

Отже $MSL = (C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}\}, E = \{\cap, 0\})$ є нижньою напівґраткою з нулем, що й потрібно було довести. □

Теорема 2.6.7. *Для будь-якої ООДМ виду*

$$OODN = (O, C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_m\}, R, E = \{\cup, \cap, \ldots\}, M),$$

де $T_1, \ldots, T_n$ – однорідні класи об'єктів, що визначають попарно-нееквівалентні типи $t_1, \ldots, t_n$, для яких існує тип t, такий що

$$((t \subset t_1) \wedge \cdots \wedge (t \subset t_n)) \wedge (\nexists t' \mid (t \subset t') \wedge (t' \subset t_1) \wedge \cdots \wedge (t' \subset t_k)),$$

де $k = \overline{2,n}$, усі можливі застосування експлуататорів об'єднання $\cup$ та однорідного перетину $\cap$ (включаючи усі можливі суперпозиції) до класів об'єктів з множини C і класів об'єктів, отриманих за допомогою універсального експлуататора об'єднання та однорідного перетину відповідно, завжди генерують $2^{n+1} - 2(n+1)$ нових унікальних класів об'єктів, де $n = |C|$.

Доведення. З Теореми 2.6.3 слідує, що кількість $q(C_\cup)$ нових унікальних класів об'єктів отриманих шляхом усіх можливих застосувань експлуататора об'єднання $\cup$ (включаючи усі можливі суперпозиції) до класів об'єктів з множини C і класів об'єктів отриманих за допомогою експлуататора об'єднання, є скінченною та дорівнює $2^n - n - 1$, де $n = |C|$.

З Теореми 2.6.5 слідує, що кількість $q(C_\cap)$ нових унікальних класів об'єктів отриманих шляхом усіх можливих застосувань експлуататора однорідного перетину $\cap$ (включаючи усі можливі суперпозиції) до класів об'єктів з множини C і класів об'єктів отриманих за допомогою експлуататора однорідного перетину, є скінченною та дорівнює $2^n - n - 1$, де $n = |C|$.

Враховуючи ці два факти та те, що процеси генерації об'єднань та однорідних перетинів базових класів об'єктів з множини C є незалежними, то сумарна кількість $q(C_{\cup\cap})$ нових унікальних класів об'єктів, яку можна утворити за допомогою експлуататора об'єднання та однорідного перетину обчислюється за наступною формулою

$$q(C_{\cup\cap}) = 2\sum_{k=0}^{n} C_n^k - C_n^1 - C_n^0 = 2\sum_{k=2}^{n} C_n^k = 2^{n+1} - 2(n+1),$$

де $n = |C|$, що й потрібно було довести. □

Теорема 2.6.8. *Для будь-якої ООДМ, для якої виконуються усі умови Теореми 2.6.7, підмножина однорідних класів $C_L = \{T_1, \ldots, T_n\}$ множини класів*

об'єктів $C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_m\}$*, розширена відповідно до Теореми 2.6.7, разом з універсальними експлуататорами об'єднання* $\cup$ *та однорідного перетину* $\cap$*, утворює повну обмежену дистрибутивну модулярну класову ґратку з умовою обриву строго зростаючих та спадаючих ланцюгів*

$$L = (C_L = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}, T_{2^n}, \ldots, T_{2^{n+1}-n-2}\}, E_L = \{\cup, \cap, 1, 0\}),$$

де клас $T_\cup = T_1 \cup \cdots \cup T_n$ *– її найбільша верхня грань, а клас* $T_\cap = T_1 \cap \cdots \cap T_n$ *– її найменша нижня грань, тобто* 1 *та* 0 *відповідно.*

Доведення. Згідно з означеннями, наведеними у [90–92, 123, 137–140], повна обмежене дистрибутивна модулярна ґратка з умовою обриву спадаючих та зростаючих ланцюгів це система $L = (A, \Omega = \{\vee, \wedge, 1, 0\})$, де A – це частково упорядкована множина (носій ґратки), $\vee$, $\wedge$ – це бінарні ідемпотентні, комутативні та асоціативні операції, а 1, 0 – це унарні операції, що визначені на множині A. Окрім цього для операцій $\cup$, $\cap$, 1 та 0 і $\forall a, b, c \in A,\ 0$ виконується ідемпотентність $(a \vee a = a;\ a \wedge a = a)$, комутативність $(a \vee b = b \vee a;\ a \wedge b = b \wedge a)$, асоціативність $((a \vee b) \vee c = a \vee (b \vee c);\ (a \wedge b) \wedge c = a \wedge (b \wedge c))$, дистрибутивність $(a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c);\ a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c))$, поглинання $(a \vee (a \wedge b) = a;\ a \wedge (a \vee b) = a)$, властивості 0 та 1 $(a \vee 0 = a;\ a \wedge 1 = a;\ a \wedge 0 = 0;\ a \vee 1 = 1)$, а також $\forall a, b, c \in A$, таких що $a \leqslant c$, має місце модулярність $(a \vee (b \wedge c) = (a \vee b) \wedge c)$.

Відповідно до теореми, носієм ґратки є множина C, а множина експлуататорів E містить бінарні операції $\cup$, $\cap$ та унарні операції 1 та 0, визначені на множині C, тобто $L = (C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}, T_{2^n}, \ldots, T_{2^{n+1}-n-2}\}, E = \{\cup, \cap, 1, 0\})$.

Покажемо що експлуататор об'єднання $\cup$ та однорідного перетину $\cap$ є ідемпотентними, комутативними та асоціативними операціями замкнутими на множині класів C.

Відповідно до Означення 2.5.2, результатом об'єднання n довільних класів об'єктів $T_1, \ldots, T_n$, що визначають типи об'єктів $t_1^1, \ldots, t_{m_1}^1, \ldots, t_1^n, \ldots, t_{m_n}^n$ відповідно, є клас об'єктів $T_{1,\ldots,n}$, який визначає у точності всі типи об'єктів, що визначають класи $T_1, \ldots, T_n$.

Відповідно до Теореми 2.6.7, для типів $t_1, \ldots, t_n$, які визначають класи

$T_1, \ldots, T_n \in C$, існує тип t, такий що

$$((t \subset t_1) \wedge \cdots \wedge (t \subset t_n)) \wedge (\nexists t' \mid (t \subset t') \wedge (t' \subset t_1) \wedge \cdots \wedge (t' \subset t_k)),$$

де $k = \overline{2, n}$ з чого слідує, що усі вище зазначені властивості для операції $\vee$ виконуються і для експлуататора об'єднання класів об'єктів $\cup$, тобто $\forall T_1, T_2, T_3 \in C$ має місце ідемпотентність ($T_1 \cup T_1 = T_1$), комутативність ($T_1 \cup T_2 = T_2 \cup T_1$) та асоціативність ($T_1 \cup (T_2 \cup T_3) = (T_1 \cup T_2) \cup T_3$).

Замкненість експлуататора об'єднання $\cup$ на множині C слідує з того, що максимально та мінімально-можливе об'єднання довільних класів з множини C дорівнює класу $T_\cup = T_1 \cup \cdots \cup T_n$ та $T_\cap = T_1 \cap \cdots \cap T_n$ відповідно, де $T_\cup, T_\cap \in C$ за умовою теореми. Всі інші можливі результати застосування універсального експлуататора об'єднання до довільних класів з множини C збігатимуться з класами множини C, оскільки це слідує з Теореми 2.6.7.

Відповідно до Означення 2.5.3, результатом однорідного перетину n довільних класів об'єктів $T_1, \ldots, T_n$, що визначають типи об'єктів $t_1^1, \ldots, t_{m_1}^1, \ldots, t_1^n, \ldots, t_{m_n}^n$ відповідно, є клас об'єктів $T_{1 \cap \ldots \cap n}$, який визначає тип об'єктів $t \mid t \subset t_1, \ldots, t \subset t_n$.

Відповідно до Теореми 2.6.7, для типів $t_1, \ldots, t_n$, які визначають класи $T_1, \ldots, T_n \in C$, існує тип t, такий що

$$((t \subset t_1) \wedge \cdots \wedge (t \subset t_n)) \wedge (\nexists t' \mid (t \subset t') \wedge (t' \subset t_1) \wedge \cdots \wedge (t' \subset t_k)),$$

де $k = \overline{2, n}$ з чого випливає, що усі вище зазначені властивості для операції $\wedge$ виконуються і для експлуататора однорідного перетину класів об'єктів, тобто $\forall T_1, T_2, T_3 \in C$ має місце ідемпотентність ($T_1 \cap T_1 = T_1$), комутативність ($T_1 \cap T_2 = T_2 \cap T_1$) та асоціативність ($T_1 \cap (T_2 \cap T_3) = (T_1 \cap T_2) \cap T_3$).

Замкненість експлуататора однорідного перетину $\cap$ на множині C випливає з того, що для типів $t_1, \ldots, t_n$, які визначають класи $T_1, \ldots, T_n \in C$, існує тип t, такий що $((t \subset t_1) \wedge \cdots \wedge (t \subset t_n)) \wedge (\nexists t' \mid (t \subset t') \wedge (t' \subset t_1) \wedge \cdots \wedge (t' \subset t_k))$, де $k = \overline{2, n}$, що гарантує що в результаті однорідного перетину довільних класів з множини C будуть отримані класи об'єктів, які є елементами множини C.

Тепер покажемо що множина C є частково упорядкованою множиною. Для

цього потрібно визначити відношення

$$\forall\, T_1, T_2 \in C \mid (T_1 \subseteq T_2 \Leftrightarrow T_1 \cup T_2 = T_2) \wedge (T_1 \subseteq T_2 \Leftrightarrow T_1 \cap T_2 = T_1)$$

та показати що воно є відношенням часткового порядку на множині C. Таке відношення було визначене раніше за допомогою Означення 2.3.16. Тепер доведемо, що воно є відношенням часткового порядку, тобто є рефлексивним, антисиметричним та транзитивним.

Рефлексивність: $T_1 \subseteq T_1 \Leftrightarrow (T_1 \cup T_1 = T_1) \wedge (T_1 \cap T_1 = T_1)$ слідує з ідемпотентності експлуататорів об'єднання $\cup$ та однорідного перетину $\cap$.

Антисиметричність: $T_1 \subseteq T_2 \Leftrightarrow T_1 \cup T_2 = T_2$, $T_2 \subseteq T_1 \Leftrightarrow T_2 \cup T_1 = T_1$ та $T_1 \subseteq T_2 \Leftrightarrow T_1 \cap T_2 = T_1$, $T_2 \subseteq T_1 \Leftrightarrow T_2 \cap T_1 = T_2$. З комутативності експлуататорів об'єднання та однорідного перетину випливає, що $T_1 = T_2$.

Транзитивність:

$$\left.\begin{array}{l} T_1 \subseteq T_2 \Leftrightarrow T_1 \cup T_2 = T_2 \\ T_2 \subseteq T_3 \Leftrightarrow T_2 \cup T_3 = T_3 \end{array}\right\} \Longrightarrow (T_1 \cup T_2) \cup T_3 = T_1 \cup (T_2 \cup T_3) =$$

$$= T_1 \cup T_3 = T_3 \Rightarrow T_1 \cup T_3 = T_3 \Leftrightarrow T_1 \subseteq T_3.$$

$$\left.\begin{array}{l} T_1 \subseteq T_2 \Leftrightarrow T_1 \cap T_2 = T_2 \\ T_2 \subseteq T_3 \Leftrightarrow T_2 \cap T_3 = T_3 \end{array}\right\} \Longrightarrow (T_1 \cap T_2) \cap T_3 = T_1 \cap (T_2 \cap T_3) =$$

$$= T_1 \cap T_3 = T_3 \Rightarrow T_1 \cap T_3 = T_3 \Leftrightarrow T_1 \subseteq T_3.$$

Отже $L = (C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}, T_{2^n}, \ldots, T_{2^{n+1}-n-2}\}, E = \{\cup, \cap\})$ є повною ґраткою.

З Теореми 2.6.7 та замкнутості експлуататорів об'єднання та однорідного перетину на множині класів C слідує що класи $T_\cup = T_1 \cup \cdots \cup T_n$ та $T_\cap = T_1 \cap \cdots \cap T_n$ є максимально-можливим об'єднанням та мінімально-можливим перетином довільних класів з множини C відповідно, а отже є 1 (одиницею) та 0 (нулем) ґратки L, отже $L = (C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}, T_{2^n}, \ldots, T_{2^{n+1}-n-2}\}, E = \{\cup, \cap, 1, 0\})$ є повною обмеженою ґраткою.

Тепер покажемо, що для повної обмеженої ґратки L виконуються закони дистрибутивності та модулярності, а також умови обриву строго зростаючих та

строго спадаючих ланцюгів. Для чого покажемо що для ґратки L має місце закон дистрибутивності, тобто $\forall T_1, T_2, T_3 \in C$ має місце $T_1 \cap (T_2 \cup T_3) = (T_1 \cap T_2) \cup (T_1 \cap T_3)$. Нехай $\exists t \mid t \subseteq T_1 \cap (T_2 \cup T_3)$ з чого випливає що $(t \subseteq T_1) \wedge (t \subseteq (T_2 \cup T_3))$ звідки слідує що $(t \subseteq T_1) \wedge ((t \subseteq T_2) \vee (t \subseteq T_3))$, а отже $t \subseteq (T_1 \cap T_2) \cup (T_1 \cap T_3)$. Тепер покажемо що рівність є справедливою в зворотньому випадку.

Нехай $\exists t \mid t \subseteq (T_1 \cap T_2) \cup (T_1 \cap T_3)$ з чого випливає що $(t \subseteq (T_1 \cap T_2)) \vee (t \subseteq (T_1 \cap T_3))$ звідки слідує що $((t \subseteq T_1) \wedge (t \subseteq T_2)) \vee ((t \subseteq T_1) \wedge (t \subseteq T_3))$, а отже $t \subseteq T_1 \cap (T_2 \cup T_3)$. Таким чином ґратка L є дистрибутивною ґраткою.

Тепер покажемо що для ґратки L має місце закон модулярності, тобто $\forall T_1, T_2, T_3 \in C \mid T_1 \subseteq T_3$ має місце $T_1 \cup (T_2 \cap T_3) = (T_1 \cup T_2) \cap T_3$. Нехай $\exists t \mid t \subseteq T_1 \cup (T_2 \cap T_3)$ з чого випливає що $(t \subseteq T_1) \vee (t \subseteq (T_2 \cap T_3))$ звідки слідує що $(t \subseteq T_1) \vee ((t \subseteq T_2) \wedge (t \subseteq T_3))$, а отже $t \subseteq (T_1 \cup T_2) \cap (T_1 \cup T_3)$, а оскільки $T_1 \subseteq T_3$, отримуємо $t \subseteq (T_1 \cup T_2) \cap T_3$. Тепер покажемо що рівність є справедливою в зворотньому випадку.

Нехай $\exists t \mid t \subseteq (T_1 \cup T_2) \cap T_3$ з чого випливає що $(t \subseteq (T_1 \cup T_2)) \wedge (t \subseteq T_3)$ звідки слідує що $((t \subseteq T_1) \vee (t \subseteq T_2)) \wedge (t \subseteq T_3)$, а отже $t \subseteq (T_1 \cap T_3) \cup (T_2 \cap T_3)$, оскільки $T_1 \subseteq T_3$, отримуємо $t \subseteq T_1 \cup (T_2 \cap T_3)$. Таким чином ґратка L є модулярною ґраткою.

Виконання умов обриву строго зростаючих та строго спадаючих ланцюгів для ґратки L випливає з її обмеженості, зокрема існування її 1 (одиниці) та 0 (нуля).

Таким чином ґратка

$$L = (C = \{T_1, \ldots, T_n, T_{n+1}, \ldots, T_{2^n-1}, T_{2^n}, \ldots, T_{2^{n+1}-n-2}\}, E = \{\cup, \cap, 1, 0\})$$

є повною обмеженою дистрибутивною модулярною ґраткою з умовами обриву строго зростаючих та строго спадаючих ланцюгів, що й потрібно було довести. □

Побудова верхніх та нижніх обмежених напівґраток дозволяє видобувати нові знання з множини базових класів ООДМ у вигляді нових класів об'єктів та відношень між ними, які є відношеннями часткового порядку. Також побудова верхньої обмеженої напівґратки дозволяє проводити ефективну реструктуризацію баз даних та знань за рахунок збереження одиниці цієї напівґратки, яка є неоднорідним

класом, замість множини базових однорідних класів об'єктів (атомів напівґратки).

Теорема 2.6.9. *За допомогою найбільшої верхньої грані (одиниці) верхніх напівґраток утворених відповідно до Теореми 2.6.2, Теореми 2.6.4 та Теореми 2.6.8 відповідно, завжди можна відновити усі елементи цих напівґраток.*

Доведення. Згідно з формулюваннями Теорем 2.6.2, 2.6.4 та 2.6.8, верхні обмежені класові напівґратки утворюються шляхом розширення підмножини однорідних класів множини базових класів ООДМ за допомогою універсального експлуататора об'єднання. З Означення 2.5.2 та формулювання Теорем 2.6.2, 2.6.4, 2.6.8 слідує що всі утворені класи об'єктів будуть неоднорідними класами, оскільки кожен з них визначатиме $n \geqslant 2$ типів об'єктів. Найбільш загальними з новоутворених класів будуть класи об'єктів, які є найбільшими верхніми гранями утворених напівґраток, оскільки вони визначатимуть усі типи з підмножин однорідних класів множин базових класів ООДМ.

Таким чином, якщо найбільші верхні грані верхніх обмежених класових напівґраток матимуть структуру $T_{1,\ldots,n} = (Core(T_{1,\ldots,n}), pr_1(t_1), \ldots, pr_n(t_n))$, то згідно з Означенням 2.2.12, можна виділити клас $t_i = (Core(T_{1,\ldots,n}), pr_i(t_i))$, який є i-м типом класу $T_{1,\ldots,n}$, такий що $(\forall t_i,\ \exists! T_i \in C) \wedge (\forall T_i \in C,\ \exists! t_i) \mid Eq(t_i, T_i) = 1$. Таким чином, відновивши атоми верхніх обмежених класових напівґраток на основі їх найбільших верхніх граней, за допомогою Теорем 2.6.1, 2.6.3 та 2.6.7 можна відновити усі інші елементи цих напівґраток, що й потрібно було довести. □

Висновки до розділу 2

В даному розділі представлена нова об'єктно-орієнтована динамічна модель подання знань – "Об'єктно-орієнтовані динамічні мережі". В рамках запропонованої моделі подання знань були визначені поняття об'єктів та їх властивостей, після чого були введені поняття однорідних, одноядерних та багатоядерних неоднорідних класів об'єктів, для яких були визначені відношення еквівалентності, агрегації, узагальнення та спеціалізації.

Для різних типів класів об'єктів був проведений експеримент, метою якого бу-

ло порівняння об'ємів фізичної пам'яті, які займають об'єктно-орієнтовані бази даних розгорнуті на сервері та їхні експортовані *.sql-файли. Результати експерименту показали що неоднорідні класи об'єктів дозволяють зменшити розміри баз даних і їх експортованих *.sql-файлів та пришвидшити виконання SQL-запитів, у порівнянні з однорідні класи об'єктів та однорідні класи об'єктів зв'язані одиничним успадкуванням.

Для більш ефективного проектування концептуальних (класових) ієрархій була розроблена класифікація механізмів успадкування, яка включає в себе 6 базових та 8 комбінованих механізмів успадкування. На основі розробленої класифікації було запропоновано підхід до вирішення проблем надлишковості, винятків, неоднозначності та несумісності.

Для об'єктів та класів об'єктів були визначені три типи операцій: експлуататори, модифікатори та генератори. Експлуататори дозволяють будувати нові об'єкти, класи, множини та мультимножини об'єктів, модифікатори дозволяють змінювати структуру об'єктів та класів об'єктів, а генератори дозволяють конструювати довільні множини та мультимножини об'єктів.

На основі застосування універсальних експлуататорів об'єднання та однорідного перетину був запропонований універсальний підхід до видобування неявних знань на основі базових однорідних класів об'єктів, який дозволяє видобувати неявні знання у вигляді обмежених верхніх (нижніх) класових напівґраток. Що дозволило розробити процедуру ефективної реструктуризації баз даних та знань, яка дозволяє зберігати ті ж об'єми даних, але використовувати для цього менше пам'яті.

Джерела використані у розділі 2

Для написання даного розділу було використано 93 джерела [1–27, 40, 46, 47, 68, 70, 71, 73–77, 86–102, 102–140], посилання на які зазначені в тексті розділу.

РОЗДІЛ 3
ЗАСТОСУВАННЯ ОБ'ЄКТНО-ОРІЄНТОВАНИХ ДИНАМІЧНИХ МЕРЕЖ

3.1. Вибір предметної області

Для практичної та більш наглядної демонстрації можливостей об'єктно-орієнтованих динамічних мереж, що були представлені у попередньому розділі, необхідно побудувати конкретну мержу для певної предметної області. У зв'язку з цим в якості предметної області розглянемо такий розділ геометрії, як опуклі чотирикутники, оскільки цей домен є відносно простим та не потребує додаткових знань. Побудуємо для цього домену відповідну об'єктно-орієнтовану динамічну мережу $ConvexQuadrangles = (O, C, R, E, M, G)$, на основі таких типів опуклих чотирикутників як квадрат, прямокутник, паралелограм, ромб та трапеція.

3.2. Визначення множини класів

Визначимо множину базових класів ООДМ $ConvexQuadrangles$ на основі типів t_{Sq}, t_{Rt}, t_{Pr}, t_{Rh}, t_{Tr}, де t_{Sq} – це тип квадратів, t_{Rt} – це тип прямокутників, t_{Pr} – це тип паралелограмів, t_{Rh} – це тип ромбів і t_{Tr} – це тип трапецій. Для усіх цих типів існують еквівалентні властивості та методи, у зв'язку з чим їх можна визначати різними способами, використовуючи при цьому як однорідні (Означення 2.2.1), так і неоднорідні (Означення 2.2.11, 2.2.14) класи об'єктів. У випадку використання однорідних класів об'єктів їх доцільно зв'язувати за допомогою механізмів успадкування (Таблиця 2.3). Однак враховуючи те що багатоядерні неоднорідні класи об'єктів у цьому випадку будуть значно ефективнішими за однорідні та одноядерні неоднорідні класи об'єктів (Експеримент 2.1), опишемо багатоядерний неоднорідний клас об'єктів T_{CQ}, що визначає типи t_{Sq}, t_{Rt}, t_{Pr}, t_{Rh}, t_{Tr} наступним

чином

$$T_{CQ} = (p_1(T_{CQ}) = (4, \text{сторони}),\ p_2(T_{CQ}) = (4, \text{кути}),\ p_3(T_{CQ}) = vf_3(T_{CQ}) = (1),$$
$$f_1(T_{CQ}) = (v_1(p_2(t_i)) + v_2(p_2(t_i)) + v_3(p_2(t_i)) + v_4(p_2(t_i)), \text{см}),\ i \in \{t_{Sq}, \ldots, t_{Tr}\});$$
$$T_{RhPr} = (p_1(T_{RhPr}) = vf_1(T_{RhPr}) = (1),\ p_2(T_{RhPr}) = vf_2(T_{RhPr}) = (1),$$
$$f_1(T_{RhPr}) = \left(v_1(p_1(t_i)) \cdot v_2(p_1(t_i)) \cdot \sin(v_2(p_2(t_i))), \text{см}^2\right),\ i \in \{Rh, Pr\}\right);$$
$$T_{SqRt} = (p_1(T_{SqRt}) = vf_1(T_{SqRt}) = (1),$$
$$p_2(T_{SqRt}) = ((90,^\circ), (90,^\circ), (90,^\circ), (90,^\circ)),$$
$$f_1(T_{SqRt}) = \left(v_1(p_1(t_i)) \cdot v_2(p_1(t_i)), \text{см}^2\right),\ i \in \{Sq, Rt\}\right);$$
$$T_{SqRh} = (p_1(T_{SqRh}) = vf_1(T_{SqRh}) = (1)),$$
$$T_{RtPr} = (p_1(T_{RtPr}) = vf_1(T_{RtPr}) = (1)),$$
$$T_{Tr} = (p_1(T_{Tr}) = vf_1(T_{Tr}) = (1),$$
$$f_1(T_{Tr}) = \frac{v_2(p_1(T_{Tr})) + v_4(p_1(T_{Tr}))}{2} \cdot$$
$$\cdot\sqrt{v_1(p_1(T_{Tr}))^2 - \left(\frac{(v_2(p_1(T_{Tr})) - v_4(p_1(T_{Tr}))^2 + v_1(p_1(T_{Tr}))) - v_3(p_1(T_{Tr}))}{2 \cdot (v_2(p_1(T_{Tr}))) - v_4(p_1(T_{Tr})))}\right)^2}\Bigg);$$
$$t_{Sq} = (p_1(t_{Sq}) = ((v_1, \text{см}), (v_2, \text{см}), (v_3, \text{см}), (v_4, \text{см})));$$
$$t_{Rt} = (p_1(t_{Rt}) = ((v_1, \text{см}), (v_2, \text{см}), (v_3, \text{см}), (v_4, \text{см})));$$
$$t_{Pr} = (p_1(t_{Pr}) = ((v_1, \text{см}), (v_2, \text{см}), (v_3, \text{см}), (v_4, \text{см})),$$
$$p_2(t_{Pr}) = ((v_1,^\circ), (v_2,^\circ), (v_3,^\circ), (v_4,^\circ)));$$
$$t_{Rh} = (p_1(t_{Rh}) = ((v_1, \text{см}), (v_2, \text{см}), (v_3, \text{см}), (v_4, \text{см})),$$
$$p_2(t_{Rh}) = ((v_1,^\circ), (v_2,^\circ), (v_3,^\circ), (v_4,^\circ)));$$
$$t_{Tr} = (p_1(t_{Tr}) = ((v_1, \text{см}), (v_2, \text{см}), (v_3, \text{см}), (v_4, \text{см})),$$
$$p_2(t_{Tr}) = ((v_1,^\circ), (v_2,^\circ), (v_3,^\circ), (v_4,^\circ)));$$

де $p_1(T_{CQ})$ – це кількість сторін фігури; $p_2(T_{CQ})$ – це кількість внутрішніх кутів фігури; $vf_3(T_{CQ})$ – це функція верифікації, яка визначає властивість "сума всіх внутрішніх кутів фігури рівна 360°", тобто

$$vf_3(T_{CQ}) : p_2(t_i) \to \{0, 1\},\ i \in \{t_{Sq}, \ldots, t_{Tr}\},$$

$$vf_3(T_{CQ}) = (v_1(p_2(t_i)) + v_2(p_2(t_i)) + v_3(p_2(t_i)) + v_4(p_2(t_i)) = 360);$$

$f_1(T_{CQ})$ – це метод обрахунку периметра фігури; $p_1(T_{RhPr})$ – це функція верифікації, яка визначає властивість рівності протилежних кутів фігури, тобто

$$vf_1(T_{RhPr}) : p_2(t_i) \to \{0, 1\},\ i = \{Rh, Pr\},$$
$$vf_1(T_{RhPr}) = ((v_1(p_2(t_i)) = v_3(p_2(t_i))) \wedge (v_2(p_2(t_i)) = v_4(p_2(t_i))));$$

$p_2(T_{RhPr})$ – це функція верифікації, яка визначає властивість "протилежні сторони фігури паралельні", тобто

$$vf_2(T_{RhPr}) : p_1(t_i) \to \{0, 1\},\ i \in \{Rh, Pr\},$$
$$vf_2(T_{RhPr}) = ((v_1(p_1(t_i)) \parallel v_3(p_1(t_i))) \wedge (v_2(p_1(t_i)) \parallel v_4(p_1(t_i))));$$

$f_1(T_{RhPr})$ – це метод обрахунку площі фігури; $p_1(T_{SqRt})$ – це функція верифікації, яка визначає властивість "усі внутрішні кути фігури рівні 90°", тобто

$$vf_1(T_{SqRt}) : p_2(t_i) \to \{0, 1\},\ i \in \{Sq, Rt\},$$
$$vf_1(T_{SqRt}) = (v_1(p_2(t_i)) = v_2(p_2(t_i)) = v_3(p_2(t_i)) = v_4(p_2(t_i)) = 90);$$

$p_2(T_{SqRt})$ – це градусні міри внутрішніх кутів фігури; $f_1(T_{SqRt})$ – це метод обрахунку площі фігури; $p_1(T_{SqRh})$ – це функція верифікації, яка визначає властивість рівності усіх сторін фігури, тобто

$$vf_1(T_{SqRh}) : p_1(t_i) \to \{0, 1\},\ i \in \{Sq, Rh\},$$
$$vf_1(T_{SqRh}) = (v_1(p_1(t_i)) = v_2(p_1(t_i)) = v_3(p_1(t_i)) = v_4(p_1(t_i)));$$

$p_1(T_{RtPr})$ – це функція верифікації, яка визначає властивість рівності протилежних сторін фігури, тобто

$$vf_1(T_{RtPr}) : p_1(t_i) \to \{0, 1\},\ i \in \{Rt, Pr\},$$
$$vf_1(T_{RtPr}) = ((v_1(p_1(t_i)) = v_3(p_1(t_i))) \wedge (v_2(p_1(t_i)) = v_4(p_1(t_i))));$$

$p_1(T_{Tr})$ – це функція верифікації, яка визначає властивість "основи фігури паралельні, а бічні сторони непаралельні", тобто

$$vf_1(T_{Tr}) : p_1(t_{Tr}) \to \{0, 1\},$$

$$vf_1(T_{Tr}) = ((v_2(p_1(t_{Tr})) \parallel v_4(p_1(t_{Tr}))) \wedge (v_1(p_1(t_{Tr})) \nparallel v_3(p_1(t_{Tr}))));$$

$f_1(T_{Tr})$ – це метод обрахунку площі фігури; $p_1(t_{Sq})$, $p_1(t_{Rt})$, $p_1(t_{Rh})$, $p_1(t_{Tr})$, $p_1(t_{Pr})$ – це розміри сторін фігур; $p_2(t_{Rh})$, $p_2(t_{Tr})$, $p_2(t_{Pr})$ – це градусні міри внутрішніх кутів фігур.

Таким чином множина класів ООДМ має вигляд $C = \{T_{CQ}\}$, де T_{CQ} – це багатоядерний неоднорідний клас об'єктів, що одночасно визначає п'ять різних типів опуклих чотирикутників. Структура класу T_{CQ} наведена в Таблиці Б.1.

3.3. Визначення множини об'єктів

Тепер на основі множини класів $C = \{T_{CQ}\}$ ООДМ *ConvexQuadrangles* визначимо множину об'єктів, для чого створимо по 2 об'єкти кожного з типів t_{Sq}, t_{Rt}, t_{Rh}, t_{Tr}, t_{Pr} та визначимо їх наступним чином.

$$Sq_1 = ((4, \text{сторони}), (4, \text{кути}), (1), (1), (1), ((90,^\circ), (90,^\circ), (90,^\circ), (90,^\circ)),$$
$$((2, \text{см}), (2, \text{см}), (2, \text{см}), (2, \text{см})));$$

$$Sq_2 = ((4, \text{сторони}), (4, \text{кути}), (1), (1), (1), ((90,^\circ), (90,^\circ), (90,^\circ), (90,^\circ)),$$
$$((3, \text{см}), (3, \text{см}), (3, \text{см}), (3, \text{см})));$$

$$Rt_1 = ((4, \text{сторони}), (4, \text{кути}), (1), (1), (1), ((90,^\circ), (90,^\circ), (90,^\circ), (90,^\circ)),$$
$$((2, \text{см}), (3, \text{см}), (2, \text{см}), (3, \text{см})));$$

$$Rt_2 = ((4, \text{сторони}), (4, \text{кути}), (1), (1), (1), ((90,^\circ), (90,^\circ), (90,^\circ), (90,^\circ)),$$
$$((4, \text{см}), (2, \text{см}), (4, \text{см}), (2, \text{см})));$$

$$Rh_1 = ((4, \text{сторони}), (4, \text{кути}), (1), (1), (1), (1), ((80,^\circ), (100,^\circ), (80,^\circ), (100,^\circ)),$$
$$((4, \text{см}), (4, \text{см}), (4, \text{см}), (4, \text{см})));$$

$$Rh_2 = ((4, \text{сторони}), (4, \text{кути}), (1), (1), (1), (1), ((110,^\circ), (70,^\circ), (110,^\circ), (70,^\circ)),$$
$$((4, \text{см}), (4, \text{см}), (4, \text{см}), (4, \text{см})));$$

$$Tr_1 = ((4, \text{сторони}), (4, \text{кути}), (1), (1), ((75.5,^\circ), (104.5,^\circ), (75.5,^\circ), (104.5,^\circ)),$$
$$((2, \text{см}), (3, \text{см}), (2, \text{см}), (4, \text{см})));$$

$$Tr_2 = ((4, \text{сторони}), (4, \text{кути}), (1), (1), ((90,^\circ), (90,^\circ), (124.5,^\circ), (55.5,^\circ)),$$

$$((2, \text{см}), (3, \text{см}), (2.8, \text{см}), (5, \text{см})));$$

$$Pr_1 = ((4, \text{сторони}), (4, \text{кути}), (1), (1), (1), (1), ((65,^\circ), (115,^\circ), (65,^\circ), (115,^\circ)),$$
$$((2, \text{см}), (4, \text{см}), (2, \text{см}), (4, \text{см})));$$

$$Pr_2 = ((4, \text{сторони}), (4, \text{кути}), (1), (1), (1), (1), ((120,^\circ), (60,^\circ), (120,^\circ), (60,^\circ)),$$
$$((3, \text{см}), (1, \text{см}), (3, \text{см}), (1, \text{см}))).$$

Таким чином множина об'єктів ООДМ *ConvexQuadrangles* матиме вигляд $O = \{Sq_1, Sq_2, Rt_1, Rt_2, Rh_1, Rh_2, Tr_1, Tr_2, Pr_1, Pr_2\}$.

3.4. Визначення множини відношень

Тепер визначимо множину відношень ООДМ *ConvexQuadrangles* на основі множини об'єктів O та множини класів об'єктів C. Оскільки усі об'єкти з множини O мають певний тип, ромб та прямокутник є різновидами паралелограма, а квадрат є різновидом ромба та прямокутника, то множина відношень ООДМ матиме наступний вигляд

$$R = \Big\{ Rh \xrightarrow{is-a} Pr,\ Rt \xrightarrow{is-a} Pr,\ Sq \xrightarrow{is-a} Rh,\ Sq \xrightarrow{is-a} Rt,\ Sq_1 \xrightarrow{instance-of} t_{Sq},$$
$$Sq_2 \xrightarrow{instance-of} t_{Sq},\ Rt_1 \xrightarrow{instance-of} t_{Rt},\ Rt_2 \xrightarrow{instance-of} t_{Rt},\ Rh_1 \xrightarrow{instance-of} t_{Rh},$$
$$Rh_2 \xrightarrow{instance-of} t_{Rh},\ Tr_1 \xrightarrow{instance-of} t_{Tr},\ Tr_2 \xrightarrow{instance-of} t_{Tr},$$
$$Pr_1 \xrightarrow{instance-of} t_{Pr},\ Pr_2 \xrightarrow{instance-of} t_{Pr} \Big\}.$$

3.5. Побудова концептуальних ієрархій

Множина класів об'єктів ООДМ містить багатоядерний неоднорідний клас T_{CQ}, який визначає типи опуклих чотирикутників t_{Sq}, t_{Rt}, t_{Pr}, t_{Rh}, t_{Tr}, тому в даному випадку концептуальна ієрархія матиме лише один рівень (Рис. 3.1). Ієрархія зображеної на Рис. 3.1 побудована з використанням SgPSt механізму успадкування, тобто

$$T_{CQ} \xrightarrow{SgPSt} Sq_1,\ T_{CQ} \xrightarrow{SgPSt} Sq_2,\ T_{CQ} \xrightarrow{SgPSt} Rt_1,\ T_{CQ} \xrightarrow{SgPSt} Rt_2,$$

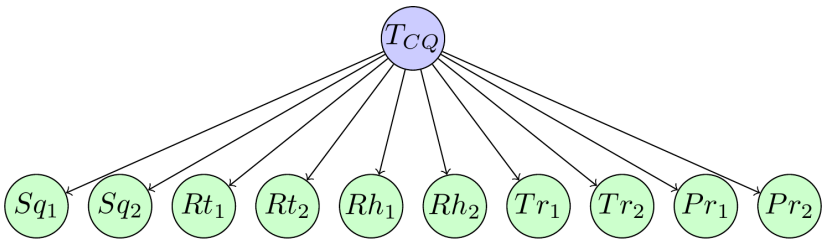


Рис. 3.1: Концептуальна ієрархія ООДМ *ConvexQuadrangles* на основі багатоядерного неоднорідного класу об'єктів

$$T_{CQ} \xrightarrow{SgPSt} Rh_1,\ T_{CQ} \xrightarrow{SgPSt} Rh_2,\ T_{CQ} \xrightarrow{SgPSt} Tr_1,\ T_{CQ} \xrightarrow{SgPSt} Tr_2,$$
$$T_{CQ} \xrightarrow{SgPSt} Pr_1,\ T_{CQ} \xrightarrow{SgPSt} Pr_2.$$

Однак якщо використовувати однорідні класи об'єктів для подання окремих типів опуклих чотирикутників, то можна визначити клас T_{CPg}, який визначатиме спільні властивості та методи для усіх опуклих чотирикутників. Потім властивості та методи класу T_{CPg} успадкують класи T_{Sq}, T_{Rt}, T_{Tr} та T_{RhPr}, після чого можна визначити класи T_{Rh} та T_{Pr}, які успадкують властивості та методи класу T_{RhPr}, після чого можна створювати опуклі чотирикутники усіх 5 типів (Рис. 3.2).

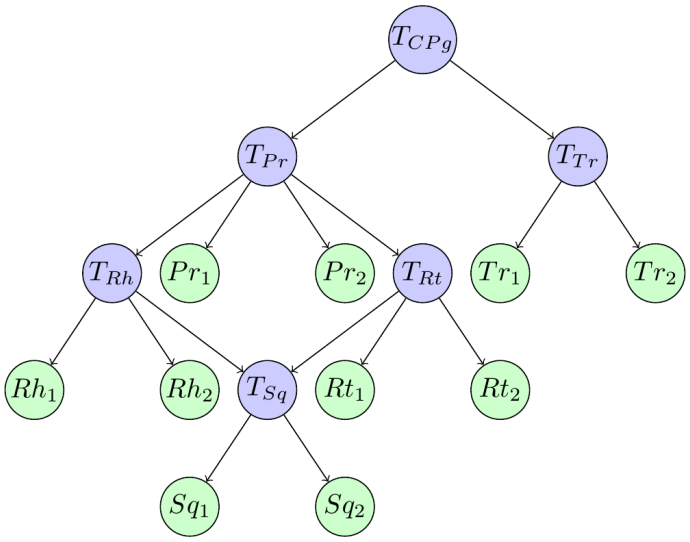


Рис. 3.2: Концептуальна ієрархія ООДМ *ConvexQuadrangles* на основі однорідних класів об'єктів

Перший рівень ієрархії зображеної на Рис. 3.2 побудований з використанням SgFSt механізму успадкування, тобто

$$T_{CPg} \xrightarrow{SgFSt} T_{Pr},\ T_{CPg} \xrightarrow{SgFSt} T_{Tr}.$$

Другий рівень ієрархії побудований з використанням SgPSt та SgFSt механізмів успадкування, тобто

$$T_{Pr} \xrightarrow{SgPSt} T_{Rh},\ T_{Pr} \xrightarrow{SgPSt} T_{Rt},\ T_{Pr} \xrightarrow{SgFSt} Pr_1,\ T_{Pr} \xrightarrow{SgFSt} Pr_2,$$
$$T_{Tr} \xrightarrow{SgFSt} Tr_1,\ T_{Tr} \xrightarrow{SgFSt} Tr_2.$$

Третій рівень ієрархії побудований з використанням MPSt та SgFSt механізмів успадкування, тобто

$$T_{Rh}, T_{Rt} \xrightarrow{MPSt} T_{Sq},\ T_{Rh} \xrightarrow{SgFSt} Rh_1,\ T_{Rh} \xrightarrow{SgFSt} Ph_2,\ T_{Rt} \xrightarrow{SgFSt} Rt_1,$$
$$T_{Rt} \xrightarrow{SgFSt} Rt_2,\ T_{Sq} \xrightarrow{SgFSt} Sq_1,\ T_{Sq} \xrightarrow{SgFSt} Sq_2.$$

При конструюванні концептуальних ієрархій важливу роль грає набір базових класів та механізми успадкування, які використовуються для її побудови. Оскільки використання класичних механізмів успадкування (одиничне та множинне), які використовуються майже у всіх об'єктно-орієнтованих моделях подання знань, може призвести до виникнення проблем надлишковості, винятків, неоднозначності та несумісності. У зв'язку з цим варто за необхідності використовувати механізми SgFSt, SgPSt, MFSt, MPSt у яких присутній ген часткового успадкування P (Таблиця 2.3).

Враховуючи те що ядра та проекції класу об'єктів T_{CQ} є однорідними класами, то застосовуючи до них механізми SgFSt, SgPSt, MFSt, MPSt успадкування, можна побудувати відповідну концептуальну ієрархію (Рис. 3.3).

Перший рівень ієрархії зображеної на Рис. 3.3 побудований з використанням SgFSt механізму успадкування, тобто

$$T_{CQ} \xrightarrow{SgFSt} T_{RhPr},\ T_{CQ} \xrightarrow{SgFSt} T_{SqRt},\ T_{CQ} \xrightarrow{SgFSt} T_{SqRh},$$
$$T_{CQ} \xrightarrow{SgFSt} T_{RtPr},\ T_{CQ} \xrightarrow{SgFSt} T_{Tr}.$$

Другий рівень ієрархії побудований з використанням MFSt та SgFSt механізмів успадкування, тобто

$$T_{SqRt}, T_{SqRh} \xrightarrow{MFSt} t_{Sq},\ T_{SqRt}, T_{RtPr} \xrightarrow{MFSt} t_{Rt},\ T_{RhPr}, T_{SqRh} \xrightarrow{MFSt} t_{Rh},$$

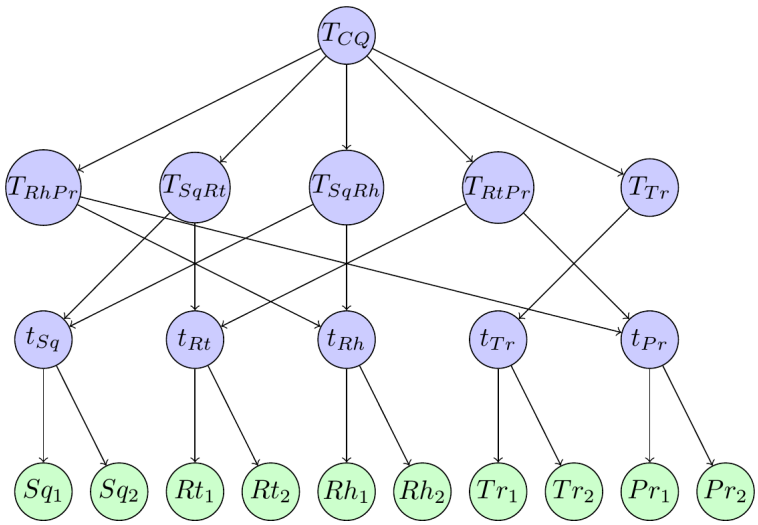


Рис. 3.3: Концептуальна ієрархія ООДМ *ConvexQuadrangles* на основі однорідних підкласів багатоядерного неоднорідного класу об'єктів

$$T_{RhPr}, T_{RtPr} \xrightarrow{MFSt} t_{Pr},\ T_{Tr} \xrightarrow{SgFSt} t_{Tr}.$$

Третій рівень ієрархії побудований з використанням SgFSt механізму успадкування, тобто

$$t_{Sq} \xrightarrow{SgFSt} Sq_1,\ t_{Sq} \xrightarrow{SgFSt} Sq_2,\ t_{Rt} \xrightarrow{SgFSt} Rt_1,\ t_{Rt} \xrightarrow{SgFSt} Rt_2,\ t_{Rh} \xrightarrow{SgFSt} Rh_1,$$
$$t_{Rh} \xrightarrow{SgFSt} Rh_2,\ t_{Tr} \xrightarrow{SgFSt} Tr_1,\ t_{Tr} \xrightarrow{SgFSt} Tr_2,\ t_{Pr} \xrightarrow{SgFSt} Pr_1,\ t_{Pr} \xrightarrow{SgFSt} Pr_2.$$

Концептуальні ієрархії зображені на Рис. 3.1 та Рис. 3.3 є більш оптимальними ніж Рис. 3.2, оскільки у цих випадках різні класи об'єктів одно рівня не мають еквівалентних властивостей та методів.

3.6. Застосування експлуататорів

Визначимо множину експлуататорів ООДМ *ConvexQuadrangles* за рахунок універсальних експлуататорів об'єднання, мультиоб'єднання, однорідного перетину, неоднорідного перетину, різниці, симетричної різниці та клонування об'єктів та класів об'єктів. Таким чином множина експлуататорів ООДМ матиме наступний вигляд $E = \{\cup, \sqcup, \cap, \Cap, \setminus, \div, Clone_i\}$.

3.6.1. Застосування експлуататорів класів об'єктів. Користуючись Означенням 2.2.12 виділимо усі 5 типів багатоядерного неоднорідного класу об'єктів T_{CQ} та застосуємо до них по черзі усі універсальні експлуататори класів об'єктів.

Об'єднання. Використовуючи Означення 2.5.2, обчислимо результат об'єднання типів t_{Sq}, t_{Rt} та t_{Rh}, тобто $t_{Sq} \cup t_{Rt} \cup t_{Rh} = T^{\cup}_{SqRtRh}$, де клас $T^{\cup}_{SqRtRh}$ – це неоднорідний багатоядерний клас об'єктів, що визначає типи t_{Sq}, t_{Rt} та t_{Rh}, і має наступну структуру

$$T^{\cup}_{SqRtRh} = \big(Core^3_1(T_{SqRtRh}), Core^2_1(T_{SqRt}), Core^2_2(T_{SqRh}), Core^1_1(T_{Rt}), Core^1_2(T_{Rh}),$$
$$pr_1(t_{Sq}), pr_2(t_{Rt}), pr_3(t_{Rh})) = ((p_1(T_{SqRtRh}), p_2(T_{SqRtRh}), p_3(T_{SqRtRh}), f_1(T_{SqRtRh})),$$
$$(p_1(T_{SqRt}), p_2(T_{SqRt}), f_1(T_{SqRt})), (p_1(T_{SqRh})), (p_1(T_{Rt})), (p_1(T_{Rh}), p_2(T_{Rh}),$$
$$f_1(T_{Rh})), (p_1(t_{Sq})), (p_1(t_{Rt})), (p_1(t_{Rh}), p_2(t_{Rh}))).$$

Однорідний перетин. Використовуючи Означення 2.5.3, обчислимо результат однорідного перетину типів t_{Sq}, t_{Rt} та t_{Rh}, тобто $t_{Sq} \cap t_{Rt} \cap t_{Rh} = T^{\cap}_{SqRtRh}$, де клас $T^{\cap}_{SqRtRh}$ – це однорідний клас об'єктів, що визначає тип $t^{\cap}_{SqRtRh}$ і має наступну структуру

$$T^{\cap}_{SqRtRh} = \big(Core^3_1(T_{SqRtRh})\big) = (p_1(T_{SqRtRh}), p_2(T_{SqRtRh}), p_3(T_{SqRtRh}), f_1(T_{SqRtRh}))\,.$$

Неоднорідний перетин. Використовуючи Означення 2.5.4, обчислимо результат неоднорідного перетину типів t_{Sq}, t_{Rt} та t_{Rh}, тобто $t_{Sq} \Cap t_{Rt} \Cap t_{Rh} = T^{\Cap}_{SqRtRh}$, де клас $T^{\Cap}_{SqRtRh}$ – це неоднорідний багатоядерний клас об'єктів, що визначає типи $t_{sq} \subseteq t_{Sq}$, $t_{rt} \subseteq t_{Rt}$ та $t_{rh} \subseteq t_{Rh}$, і має наступну структуру

$$T^{\Cap}_{SqRtRh} = \big(Core^3_1(T_{SqRtRh}), Core^2_1(T_{SqRt}), Core^2_2(T_{SqRh}), Core^1_1(T_{Rt}), Core^1_2(T_{Rh})\big) =$$
$$= ((p_1(T_{SqRtRh}), p_2(T_{SqRtRh}), p_3(T_{SqRtRh}), f_1(T_{SqRtRh})), (p_1(T_{SqRt}), p_2(T_{SqRt}),$$
$$f_1(T_{SqRt})), (p_1(T_{SqRh})), (p_1(T_{Rt})), (p_1(T_{Rh}), p_2(T_{Rh}), f_1(T_{Rh})))\,.$$

Різниця. Використовуючи Означення 2.5.5, обчислимо різницю типів t_{Sq} та t_{Rh}, тобто $t_{Sq} \setminus t_{Rh} = T_{Sq\setminus Rh}$, де клас $T_{Sq\setminus Rh}$ – це однорідний клас об'єктів, що

визначає тип $t_{Sq\backslash Rh} \subseteq t_{Sq}$ і має наступну структуру

$$T_{Sq\backslash Rh} = \left(p_1\left(T_{Sq\backslash Rh}\right), p_2\left(T_{Sq\backslash Rh}\right), p_3\left(T_{Sq\backslash Rh}\right), f_1\left(T_{Sq\backslash Rh}\right)\right) =$$
$$= (p_5(t_{Sq}), p_6(t_{Sq}), p_7(t_{Sq}), f_2(t_{Sq})).$$

Симетрична різниця. Використовуючи Означення 2.5.6, обчислимо симетричну різницю типів t_{Sq} та t_{Rh}, тобто $t_{Sq} \div t_{Rh} = T_{Sq\div Rh}$, де клас $T_{Sq\div Rh}$ – це неоднорідний клас об'єктів, що визначає типи $t_{Sq\backslash Rh}$ та $t_{Rh\backslash Sq}$ і має наступну структуру

$$T_{S\div R} = (pr_1(t_{Sq\backslash Rh}), pr_2(t_{Rh\backslash Sq}) = ((p_5(t_{Sq}), p_6(t_{Sq}), p_7(t_{Sq}), f_1(t_{Sq})),$$
$$(p_5(t_{Rh}), p_6(t_{Rh}), p_7(t_{Rh}), p_8(t_{Rh}), f_1(t_{Rh}))).$$

Таким чином множина класів ООДМ *ConvexQuadrangles* розширюється за рахунок новоутворених класів об'єктів $T^{\cup}_{SqRtRh}$, $T^{\cap}_{SqRtRh}$, $T^{\Cap}_{SqRtRh}$, $T_{Sq\backslash Rh}$ та $T_{Sq\div Rh}$, тобто $C = \left\{T_{CQ}, T^{\cup}_{SqRtRh}, T^{\cap}_{SqRtRh}, T^{\Cap}_{SqRtRh}, T_{Sq\backslash Rh}, T_{Sq\div Rh}\right\}$. Застосовуючи універсальні експлуататори класів об'єктів до інших наборів типів опуклих чотирикутників можна утворювати нові класи об'єктів, і цим самим далі розширювати множину базових класів ООДМ.

3.6.2. Застосування експлуататорів об'єктів. Застосуємо почергово усі універсальні експлуататори об'єктів до об'єктів з множини O.

Об'єднання. Використовуючи Означення 2.5.8, обчислимо результат об'єднання об'єктів Sq_1, Rt_1 та Pr_2, тобто

$$Sq_1 \cup Rt_1 \cup Pr_2 = S_{SqRtpr}/T_{SqRtPr} = \{Sq_1, Rt_1, Pr_2\}/T^{\cup}_{SqRtPr},$$

де S_{SqRtPr} – це новоутворена множина об'єктів, а $T^{\cup}_{SqRtPr}$ – це неоднорідний багатоядерний клас об'єктів, що визначає типи t_{Sq}, t_{Rt} та t_{Pr} і має наступну структуру

$$T^{\cup}_{SqRtPr} = \big(Core^3_1(T_{SqRtPr}), Core^2_1(T_{SqRt}), Core^2_2(T_{RtPr}), Core^1_1(T_{Sq}), Core^1_2(T_{Pr}),$$
$$pr_1(t_{Sq}), pr_2(t_{Rt}), pr_3(t_{Rh})) = ((p_1(T_{SqRtPr}), p_2(T_{SqRtPr}), p_3(T_{SqRtPr}), f_1(T_{SqRtPr})),$$
$$(p_1(T_{SqRt}), p_2(T_{SqRt}), f_1(T_{SqRt})), (p_1(T_{RtPr})), (p_1(t_{Sq})), (p_1(t_{Pr}), p_2(t_{Pr}), f_1(t_{Pr})),$$

$$(p_1(t_{Sq})), (p_1(t_{Rt})), (p_1(t_{Pr}), p_2(t_{Pr})))) .$$

Однорідний перетин. Використовуючи Означення 2.5.10, обчислимо однорідний перетин об'єктів Sq_1 та Rt_1, тобто $Sq_1 \cap Rt_1 = T^{\cap}_{SqRt}$, де $T^{\cap}_{SqRt}$ – це однорідний клас об'єктів, що визначає тип $t_{Sq\cap Rt}$, і має наступну структуру

$$T^{\cap}_{SqRt} = \left(p_1\left(T^{\cap}_{SqRt}\right), p_2\left(T^{\cap}_{SqRt}\right), p_3\left(T^{\cap}_{SqRt}\right), p_4\left(T^{\cap}_{SqRt}\right), p_5\left(T^{\cap}_{SqRt}\right), f_1\left(T^{\cap}_{SqRt}\right)\right).$$

Неоднорідний перетин. Використовуючи Означення 2.5.11, обчислимо неоднорідний перетин об'єктів Sq_1, Rh_1 та Pr_2 тобто $Sq_1 \Cap Rh_1 \Cap Pr_2 = T^{\Cap}_{SqRhPr}$, де $T^{\Cap}_{SqRhPr}$ – це неоднорідний багатоядерний клас об'єктів, що визначає типи $t_{sq} \subseteq t_{Sq}$, $t_{rh} \subseteq t_{Rh}$ та $t_{pr} \subseteq t_{Pr}$ і має наступну структуру

$$T^{\Cap}_{SqRhPr} = \big(Core^3_1(T_{SqRhPr}), Core^2_1(T_{RhPr}), Core^2_2(T_{SqRh}), Core^1_1(T_{Sq}),$$
$$Core^1_2(T_{Pr}))\big) = ((p_1(T_{SqRhPr}), p_2(T_{SqRhPr}), p_3(T_{SqRhPr}), f_1(T_{SqRhPr})), (p_1(T_{RhPr}),$$
$$p_2(T_{RhPr}), f_1(T_{RhPr})), (p_1(T_{SqRh})), (p_1(T_{Sq}), p_2(T_{Sq}), f_1(T_{Sq})), (p_1(T_{Pr})))) .$$

Різниця. Використовуючи Означення 2.5.12, обчислимо результат різниці об'єктів Rh_1 та Pr_1, тобто $Rh_1 \setminus Pr_1 = T_{Rh\setminus Pr}$, де $T_{Rh\setminus Pr}$ – це однорідний клас об'єктів, що визначає тип $t_{Rh\setminus Pr}$, і має наступну структуру

$$T_{Rh\setminus Pr} = \left(p_1\left(T_{Rh\setminus Pr}\right), p_2\left(T_{Rh\setminus Pr}\right), p_3\left(T_{Rh\setminus Pr}\right)\right) = (p_4(t_{Rh}), p_7(t_{Rh}), p_8(t_{Rh})).$$

Симетрична різниця. Використовуючи Означення 2.5.13, обчислимо симетричну різницю об'єктів Rt_1 та Pr_1, тобто $Rt_1 \div Pr_1 = T_{Rt\div Pr}$, де $T_{Rt\div Pr}$ – це неоднорідний клас об'єктів, що визначає типи $t_{Rt\setminus Pr}$ та $t_{Pr\setminus Rt}$ і має наступну структуру

$$T_{Rt\div Pr} = (pr_1(t_{Rt\setminus Pr}), pr_2(t_{Pr\setminus Rt}) = ((p_4(T_{Rt}), p_6(T_{Rt}), p_7(T_{Rt}), f_2(T_{Rt})),$$
$$(p_5(T_{Pr}), p_6(T_{Pr}), p_7(T_{Pr}), p_8(T_{Pr}), f_2(T_{Pr}))),$$

Клонування. Використовуючи Означення 2.5.14, застосуємо операцію клонування до об'єктів Sq_2 та Tr_2 побудуємо по 2 їх клони, тобто

$$Clone_1(Sq_2) = Sq_3/t_{Sq},\ Clone_2(Sq_2) = Sq_4/t_{Sq},\ Clone_1(Tr_2) = Tr_3/t_{Tr},$$

$$Clone_2(Tr_2) = Tr_4/t_{Tr}.$$

Мультиоб'єднання. Використовуючи Означення 2.5.9, обчислимо результат мультиоб'єднання об'єктів Sq_1, Sq_2, Sq_3, Sq_4, Rt_1, Rt_2, Tr_1, Tr_2, Tr_3, Tr_4 тобто

$$\bigsqcup_{i=1}^{4} Sq_i/t_{Sq} \bigsqcup_{j=1}^{2} Rt_j/t_{Rt} \bigsqcup_{k=1}^{4} Tr_k/k_{Tr} = M_{SqRtTr}/T^{\sqcup}_{SqRtTr} =$$
$$= \{Sq_1, Sq_2, Sq_3, Sq_4, Rt_1, Rh_2, Tr_1, Tr_2, Tr_3, Tr_4\}/T^{\sqcup}_{SqRtTr} =$$
$$= \{(Sq_1, 1), (Sq_2, 3), (Rt_1, 1), (Rt_2, 1), (Tr_1, 1), (Tr_2, 3)\}/T^{\sqcup}_{SqRtTr},$$

де M_{SqRtTr} – це новоутворена мультимножина об'єктів, а $T^{\sqcup}_{SqRtTr}$ – це неоднорідний багатоядерний клас об'єктів, що визначає типи t_{Sq}, t_{Rt} та t_{tr} і має наступну структуру

$$T^{\sqcup}_{SqRtTr} = \big(Core^3_1(T_{SqRtTr}), Core^2_1(T_{SqRt}), Core^1_1(T_{Sq}), Core^1_2(T_{Rt}), Core^1_3(T_{Tr}),$$
$$pr_1(t_{Sq}), pr_2(t_{Rt}), pr_3(t_{Tr})) = ((p_1(T_{SqRtTr}), p_2(T_{SqRtTr}), p_3(T_{SqRtTr}), f_1(T_{SqRtTr})),$$
$$(p_1(T_{SqRt}), p_2(T_{SqRt}), f_1(T_{SqRt})), (p_1(T_{Sq})), (p_1(T_{Rt})), (p_1(T_{Tr}), p_2(T_{Tr})),$$
$$(p_1(t_{Sq})), (p_1(t_{Rt})), (p_1(t_{Tr}), p_2(t_{Tr}))).$$

Таким чином множина класів ООДМ *ConvexQuadrangles* розширюється за рахунок новоутворених класів об'єктів $T^{\cup}_{SqRtPr}$, $T^{\cap}_{SqRt}$, $T^{\Cap}_{SqRhPr}$, $T_{Rh\setminus Pr}$, $T_{Rt\div Pr}$ та $T^{\sqcup}_{SqRtTr}$, тобто

$$C = \big\{T_{CQ}, T^{\cup}_{SqRtRh}, T^{\cap}_{SqRtRh}, T^{\Cap}_{SqRtRh}, T_{Sq\setminus Rh}, T_{Sq\div Rh}, T^{\cup}_{SqRtPr}, T^{\cap}_{SqRt}, T^{\Cap}_{SqRhPr},$$
$$T_{Rh\setminus Pr}, T_{Rt\div Pr}, T^{\sqcup}_{SqRtTr}\big\}.$$

Застосовуючи універсальні експлуататори об'єктів до інших наборів типів опуклих чотирикутників можна утворювати нові класи об'єктів, і цим самим далі розширювати множину базових класів ООДМ.

3.7. Застосування модифікаторів

Визначимо множину модифікаторів ООДМ *ConvexQuadrangles* за рахунок комбінованих модифікаторів об'єктів та класів об'єктів, які поєднують у собі прин-

ципи дії 4 базових модифікаторів (частковий, породжуючий, знищуючий та замінний).

3.7.1. Застосування модифікаторів класів об'єктів. Застосуємо до типу t_{Sq} універсальний експлуататор клонування, в результаті чого отримаємо його копію $Clone_1(t_{Sq1}) = t_{Sq1}/t_{Sq1}$. Тепер використовуючи Означення 2.5.41, визначимо для типу опуклих чотирикутників t_{Sq1} наступний комбінований модифікатор $M^{D_1,P_2,R_3,G_4}(t_{Sq1}) = M_4^G\left(M_3^R\left(M_2^P\left(M_1^D(t_{Sq1})\right)\right)\right)$, такий що

$$M_1^D(t_{Sq1}) = \left(m_1^D(p_5(t_{Sq1}))\right);\ M_2^P(t_{Sq1}) = \left(m_1^P(p_7(t_{Sq1})), m_2^P(p_8(t_{Sq1}))\right);$$
$$M_3^R(t_{Sq1}) = \left(m_1^R(p_4(t_{Sq1})), m_1^R(f_1(t_{Sq1})), m_2^R(f_2(t_{Sq1}))\right);$$
$$M_4^G(t_{Sq1}) = \left(m_1^G(p_8(t_{Sq1})), m_2^G(p_9(t_{Sq1}))\right);$$

де

$$m_1^D(p_5(t_{Sq1})) : (p_1(t_{Sq1}), \ldots, p_7(t_{Sq1})) \to (p_1(t_{Sq1}), \ldots, p_4(t_{Sq1}), p_6(t_{Sq1}), p_7(t_{Sq1}));$$
$$m_1^P(p_2(t_{Sq1})) : ((v_1(p_6(t_{Sq1})), v_2(p_6(t_{Sq1})), v_3(p_6(t_{Sq1})), v_4(p_6(t_{Sq1})))) \to$$
$$\to (80, 100, 80, 100);$$
$$m_2^P(p_4(t_{Sq1})) : ((v_1(p_7(t_{Sq1})), v_2(p_7(t_{Sq1})), v_3(p_7(t_{Sq1})), v_4(p_7(t_{Sq1})))) \to (2, 4, 2, 4);$$
$$m_1^R(p_7(t_{Sq1})) : vf_4(t_{Sq1}) \to vf_{4n}(t_{Sq1}) : p_{4n}(t_{Sq1}) \to \{0, 1\},$$
$$p_{4n}(t_{Sq1}) = ((v_1(p_7(t_{Sq1})) = v_3(p_7(t_{Sq1}))) \wedge (v_2(p_7(t_{Sq1})) = v_4(p_7(t_{Sq1}))));$$
$$m_1^R(f_1(t_{Sq1})) : f_1(t_{Sq1}) \to (2 \cdot (v_1(p_7(t_{Sq1})) + v_2(p_7(t_{Sq1}))), \text{см});$$
$$m_2^R(f_2(t_{Sq1})) : f_2(t_{Sq1}) \to \left(v_1(p_7(t_{Sq1})) \cdot v_2(p_7(t_{Sq1})) \cdot \sin(v_1(p_6(t_{Sq1}))), \text{см}^2\right);$$
$$m_1^G(p_8(t_{Sq1})) : (p_1(t_{Sq1}), \ldots, p_4(t_{Sq1}), p_6(t_{Sq1}), p_7(t_{Sq1})) \to (p_1(t_{Sq1}), \ldots, p_4(t_{Sq1}),$$
$$p_6(t_{Sq1}), p_7(t_{Sq1}), p_8(t_{Sq1})),\ p_8(t_{Sq1}) = ((0, r), (255, g), (0, b));$$
$$m_2^G(p_9(t_{Sq1})) : (p_1(t_{Sq1}), \ldots, p_4(t_{Sq1}), p_6(t_{Sq1}), p_7(t_{Sq1}), p_8(t_{Sq1})) \to$$
$$\to ((p_1(t_{Sq1}), \ldots, p_4(t_{Sq1}), p_6(t_{Sq1}), p_7(t_{Sq1}), p_8(t_{Sq1}), p_9(t_{Sq1}));$$
$$p_9(t_{Sq1}) = vf_{9n}(t_{Sq1}) : p_{9n}(t_{Sq1}) \to \{0, 1\};$$
$$p_{9n}(t_{Sq1}) = ((v_1(p_7(t_{Sq1})) = v_3(p_7(t_{Sq1}))) \wedge (v_2(p_7(t_{Sq1})) = v_4(p_7(t_{Sq1})))).$$

В результаті модифікації отримуємо тип опуклих чотирикутників t^M_{Sq1}, який має наступну структуру

$$t^M_{Sq1} = \left(p_1\left(t^M_{Sq1}\right) = (4, \text{сторони}),\ p_2\left(t^M_{Sq1}\right) = (4, \text{кути}),\ p_3\left(t^M_{Sq1}\right) = vf_3\left(t^M_{Sq1}\right) = 1,\right.$$
$$p_4\left(t^M_{Sq1}\right) = vf_{4n}\left(t^M_{Sq1}\right) = 1,\ p_6\left(t^M_{Sq1}\right) = ((80,^\circ), (100,^\circ), (80,^\circ), (100,^\circ)),$$
$$p_7\left(t^M_{Sq1}\right) = ((2, \text{см}), (4, \text{см}), (2, \text{см}), (4, \text{см})),\ p_8\left(t^M_{Sq1}\right) = ((0, r), (255, g), (0, b)),$$
$$p_9\left(t^M_{Sq1}\right) = vf_{8n}\left(t^M_{Sq1}\right) = 1,\ f_1\left(t^M_{Sq1}\right) = \left(2 \cdot \left(v_1\left(p_2\left(t^M_{Sq1}\right)\right) + v_2\left(p_2\left(t^M_{Sq1}\right)\right)\right), \text{см}\right),$$
$$\left.f_2\left(t^M_{Sq1}\right) = \left(v_1\left(p_2\left(t^M_{Sq1}\right)\right) \cdot v_2\left(p_2\left(t^M_{Sq1}\right)\right) \cdot \sin\left(v_1\left(p_4\left(t^M_{Sq1}\right)\right)\right), \text{см}^2\right)\right).$$

Аналізуючи результат застосування модифікатора $M^{D_1,P_2,R_3,G_4}(t_{Sq1})$ можна помітити, що клас об'єктів t_{Sq1}, що описував квадрат, був трансформований у клас зелених паралелограмів t^M_{Sq1} розмірів 2×4 см з кутами 80° та 100°. За рахунок модифікації клону класу t_{Sq}, між цим класом об'єктів та модифікацією його клону t^M_{Sq1} визначилося відношення модифікації (Означення 2.5.43), тобто $t^M_{Sq1} \xrightarrow{modification-of} t_{Sq}$. Дане відношення дозволяє будувати зв'язки між різними концептами і навіть цілими предметними областями.

Таким чином множина класів ООДМ *ConvexQuadrangles* розширюється за рахунок новоутвореного класу об'єктів t^M_{Sq1}, тобто

$$C = \left\{T_{CQ}, T^{\cup}_{SqRtRh}, T^{\cap}_{SqRtRh}, T^{\Cap}_{SqRtRh}, T_{Sq\setminus Rh}, T_{Sq\div Rh}, T^{\cup}_{SqRtPr}, T^{\cap}_{SqRt}, T^{\Cap}_{SqRhPr},\right.$$
$$\left.T_{Rh\setminus Pr}, T_{Rt\div Pr}, T^{\sqcup}_{SqRtTr}, t^M_{Sq1}\right\}.$$

3.7.2. Застосування модифікаторів об'єктів. Використовуючи Означення 2.5.26, визначимо для об'єкта Sq_3/t_{Sq} наступний комбінований модифікатор $M^{D_1,P_2,R_3,G_4}(Sq_3) = M^G_4\left(M^R_3\left(M^P_2\left(M^D_1(Sq_3)\right)\right)\right)$, такий що

$$M^D_1(Sq_3) = \left(m^D_1(p_5(Sq_3))\right);$$
$$M^P_2(Sq_3) = \left(m^P_1(p_1(Sq_3)), m^P_2(p_7(Sq_3)), m^P_3(p_2(S_3)), m^P_4(p_6(S_3))\right);$$
$$M^R_3(Sq_3) = \left(m^R_1(p_3(Sq_3)), m^R_2(p_4(Sq_3))\right);$$
$$M^G_4(Sq_3) = \left(m^G_1(p_8(Sq_3)), m^G_2(p_9(Sq_3))\right);$$

де

$$m_1^D(p_5(Sq_3)) : (p_1(Sq_3), \ldots, p_7(Sq_3)) \to (p_1(Sq_3), \ldots, p_4(Sq_3)), p_6(Sq_3), p_7(Sq_3);$$

$$m_1^P(p_1(Sq_3)) : v_1(p_1(Sq_3)) \to (3);$$

$$m_2^P(p_7(Sq_3)) : ((v_1(p_7(Sq_3)), u_1(p_7(Sq_3))), (v_2(p_7(Sq_3)), u_2(p_7(Sq_3))), (v_3(p_7(Sq_3)),$$

$$u_3(p_7(Sq_3))), (v_4(p_7(Sq_3)), u_4(p_7(Sq_3)))) \to ((3, \text{см}), (3, \text{см}), (3, \text{см}));$$

$$m_3^P(p_2(Sq_3)) : v_1(p_2(Sq_3)) \to (3);$$

$$m_4^P(p_6(Sq_3)) : ((v_1(p_6(Sq_3)), u_1(p_6(Sq_3))), (v_2(p_6(Sq_3)), u_2(p_6(Sq_3))), (v_3(p_6(Sq_3)),$$

$$u_3(p_6(Sq_3))), (v_4(p_6(Sq_3)), u_4(p_6(Sq_3)))) \to ((60,^\circ), (60,^\circ), (60,^\circ));$$

$$m_1^R(p_4(Sq_3)) : vf_4(Sq_3) \to vf_{4n}(Sq_3) : p_{4n}(Sq_3) \to \{0, 1\},$$

$$p_{4n}(Sq_3) = (v_1(p_6(Sq_3)) + v_2(p_6(Sq_3)) + v_3(p_6(Sq_3)) = 180);$$

$$m_2^R(p_4(Sq_3)) : vf_4(Sq_3) \to vf_{4n}(Sq_3) : p_{4n} \to \{0, 1\},$$

$$p_{4n} = (v_1(p_7(Sq_3)) = v_2(p_7(Sq_3)) = v_3(p_7(Sq_3)));$$

$$m_1^G(p_8(Sq_3)) : (p_1(Sq_3), \ldots, p_4(Sq_3)), p_6(Sq_3), p_7(Sq_3) \to (p_1(Sq_3), \ldots, p_4(Sq_3)),$$

$$p_6(Sq_3), p_7(Sq_3), p_8(Sq_3)),\ p_8(Sq_3) = vf_{8n}(Sq_3) : p_{8n}(Sq_3) \to \{0, 1\},$$

$$p_{8n} = ((v_1(p_7(Sq_3)) + v_2(p_7(Sq_3)) > v_3(p_7(Sq_3))) \wedge (v_1(p_7(Sq_3)) +$$

$$+v_3(p_7(Sq_3)) > v_2(p_7(Sq_3))) \wedge (v_2(p_7(Sq_3)) + v_3(p_7(Sq_3)) > v_1(p_7(Sq_3))));$$

$$m_2^G(p_8(Sq_3)) : (p_1(Sq_3), \ldots, p_4(Sq_3)), p_6(Sq_3), p_7(Sq_3), p_8(Sq_3)) \to (p_1(Sq_3), \ldots,$$

$$p_4(Sq_3)), p_6(Sq_3), p_7(Sq_3), p_8(Sq_3), p_9(Sq_3)),\ p_9(Sq_3) = ((255, r), (0, g), (0, b)).$$

В результаті модифікації отримуємо об'єкт Sq_3^M, який має наступну структуру

$$Sq_3^M = \left(p_1\left(Sq_3^M\right) = (3, \text{сторони}),\ p_2\left(Sq_3^M\right) = (3, \text{кути}),\right.$$

$$p_3\left(Sq_3^M\right) = vf_{3n}\left(Sq_3^M\right) = 1,\ p_4\left(Sq_3^M\right) = vf_{4n}\left(Sq_3^M\right) = 1,$$

$$p_6\left(Sq_3^M\right) = ((3, \text{см}), (3, \text{см}), (3, \text{см})),\ p_7\left(Sq_3^M\right) = ((60,^\circ), (60,^\circ), (60,^\circ)),$$

$$\left.p_8\left(Sq_3^M\right) = vf_{7n}\left(Sq_3^M\right) = 1,\ p_9\left(Sq_3^M\right) = ((255, r), (0, g), (0, b))\right).$$

Аналізуючи результат застосування комбінованого модифікатора $M^{D_1,P_2,R_3,G_4}(Sq_3)$ можна помітити, що об'єкт Sq_3/t_{Sq}, який описував квадрат, був трансформований у червоний рівносторонній трикутник зі стороною 3 см, отже $Sq_3^M \notin t_{Sq}$. За

рахунок модифікації клону об'єкта Sq_3/T_{Sq}, між цим об'єктом та модифікацією його клону Sq_3^M визначилося відношення модифікації (Означення 2.5.42), тобто $Sq_3^M \xrightarrow{modification-of} Sq_3/T_{Sq}$. Таке відношення дозволяє будувати зв'язки між різними об'єктами і навіть цілими предметними областями. В даному випадку у результаті модифікації квадрату був встановлений зв'язок між класом квадратів та класом трикутників.

Тепер множина модифікаторів ООДМ $ConvexQuadrangles$ має наступний вигляд $M = \left\{M^{D_1,P_2,R_3,G_4}(t_{Sq1}), M^{D_1,P_2,R_3,G_4}(Sq_3)\right\}$.

3.8. Застосування генераторів

Визначимо множину генераторів ООДМ $ConvexQuadrangles$ на основі універсальних конструкторів множин та мультимножин об'єктів

$$G = \{USC1, USC2, USC3, UMC1, UMC2, UMC3, UMC4,$$
$$UMC5, UMC6, UMC7\}.$$

Таким чином використовуючи конструктори з множини G можна будувати довільні множини та мультимножини опуклих чотирикутників, а також класи цих множин та мультимножин об'єктів. При застосуванні універсальних експлуататорів об'єднання та мультиоб'єднання об'єктів, були продемонстровані універсальні конструкторів USC1 та UMC1.

3.9. Видобування знань

Розглянемо множину усіх базових типів опуклих чотирикутників в рамах ООДМ $ConvexQuadrangles$ $T = \{t_{Sq}, t_{Rt}, t_{Pr}, t_{Rh}, t_{Tr}\}$, очевидно що для усіх цих типів існують еквівалентні для них властивості та методи, а отже вони задовольняють умову Теореми 2.6.3, яка дозволяє розширити множину T новими класами об'єктів, які утворяться шляхом застосування до елементів множини T універсального експлуататора об'єднання класів об'єктів. Якщо застосувати до типів

з множини T універсальний експлуататор об'єднання класів, то згідно з Теоремою 2.6.3 утвориться $2^n - n - 1$ нових унікальних класів об'єктів, де $n = |T|$, тобто $2^5 - 5 - 1 = 32 - 5 - 1 = 26$. Таким чином множина T буде мати вигляд

$$T = \left\{t_{Sq}, t_{Rt}, t_{Pr}, t_{Rh}, t_{Tr}, t^{\cup}_{SqRt}, \ldots, t^{\cup}_{SqRtPr}, \ldots, t^{\cup}_{SqRtPrRh}, \ldots, t^{\cup}_{SqRtPrRhTr}\right\}.$$

Таким чином відповідно до Теореми 2.6.4 множина однорідних класів T, розширена відповідно до Теореми 2.6.3, разом з універсальним експлуататором об'єднання $\cup$, утворюють верхню обмежену класову напівґратку $JSL = (T, E = \{\cup, 1\})$, де клас $t^{\cup}_{SqRtPrRhTr} = t_{Sq} \cup \cdots \cup t_{Tr}$ – її найбільша верхня грань, тобто 1. У даному випадку клас $t^{\cup}_{SqRtPrRhTr}$ є нічим іншим як багатоядерним неоднорідним класом T_{CQ}, що дозволяє провести реструктуризацію бази знань, а саме, замість типів t_{Sq}, t_{Rt}, t_{Pr}, t_{Rh}, t_{Tr} зберігати клас T_{CQ}, оскільки його збереження потребує менше пам'яті, за рахунок ядер класу, що містять еквівалентні властивості та(або) методи для типів класу.

Розглянемо множину деяких базових типів опуклих чотирикутників в рамках ООДМ *ConvexQuadrangles* $T_1 = \{t_{Sq}, t_{Pr}, t_{Tr}\}$, очевидно що для усіх цих типів існують еквівалентні для них властивості та методи, а отже вони задовольняють умову Теореми 2.6.3, яка дозволяє розширити множину T_1 новими класами об'єктів, які утворяться шляхом застосування до елементів множини T_1 універсального експлуататора об'єднання класів об'єктів. Якщо застосувати до типів з множини T_1 універсальний експлуататор об'єднання класів, то згідно з Теоремою 2.6.3 утвориться $2^n - n - 1$ нових унікальних класів об'єктів, де $n = |T_1|$, тобто $2^3 - 5 - 1 = 8 - 3 - 1 = 4$. Таким чином множина T_1 буде мати вигляд

$$T_1 = \left\{t_{Sq}, t_{Pr}, t_{Tr}, t^{\cup}_{SqPr}, t^{\cup}_{SqTr}, t^{\cup}_{PrTr}, t^{\cup}_{SqPrTr}\right\}.$$

Відповідно до Теореми 2.6.4 множина однорідних класів T_1, розширена відповідно до Теореми 2.6.3, разом з універсальним експлуататором об'єднання $\cup$, утворюють верхню обмежену класову напівґратку $JSL = (T_1, E = \{\cup, 1\})$, де клас $t^{\cup}_{SqPrTr} = t_{Sq} \cup t_{Pr} \cup t_{Tr}$ – її найбільша верхня грань, тобто 1. Як і у попередньому випадку, це дозволяє провести реструктуризацію бази знань, а саме, замість типів t_{Sq}, t_{Pr}

та t_{Tr} зберігати клас $t^{\cup}_{SqPrTr}$, оскільки його збереження потребує менше пам'яті, за рахунок ядра класу, що містить еквівалентні властивості та(або) методи для типів класу.

Аналізуючи структуру типів з множини T_1, можна переконатися у тому що вони задовольняють умові Теореми 2.6.5, яка дозволяє розширити множину T_1 новими класами об'єктів, які утворяться шляхом застосування до елементів множини T_1 універсального експлуататора однорідного перетину класів об'єктів. Якщо застосувати до типів з множини T_1 універсальний експлуататор однорідного перетину класів, то згідно з Теоремою 2.6.5 утвориться $2^n - n - 1$ нових унікальних класів об'єктів, де $n = |T_1|$, тобто $2^3 - 3 - 1 = 8 - 3 - 1 = 4$. Таким чином множина T_1 буде мати вигляд

$$T_1 = \left\{t_{Sq}, t_{Pr}, t_{Tr}, t^{\cup}_{SqPr}, t^{\cup}_{SqTr}, t^{\cup}_{PrTr}, t^{\cup}_{SqPrTr}, t^{\cap}_{SqPr}, t^{\cap}_{SqTr}, t^{\cap}_{PrTr}, t^{\cap}_{SqPrTr}\right\}.$$

Таким чином відповідно до Теореми 2.6.6 підмножина однорідних класів $T_2 \in T_1$, де $T_2 = \left\{t_{Sq}, t_{Pr}, t_{Tr}, t^{\cap}_{SqPr}, t^{\cap}_{SqTr}, t^{\cap}_{PrTr}, t^{\cap}_{SqPrTr}\right\}$, розширена відповідно до Теореми 2.6.5, разом з універсальним експлуататором однорідного перетину $\cap$, утворюють нижню обмежену класову напівґратку $MSL = (T_2, E = \{\cap, 0\})$, де клас $t^{\cap}_{SqPrTr} = t_{Sq} \cap t_{Pr} \cap t_{Tr}$ – її найменша нижня грань, тобто 0.

Аналізуючи множину класів T_1, можна бачити що вона задовольняє умовам Тереми 2.6.7, яка дозволяє розширити множину T новими класами об'єктів, які утворяться шляхом застосування до елементів множини T універсальних експлуататорів об'єднання $\cup$ та однорідного перетину $\cap$ класів об'єктів. Справді, якщо застосувати до типів з множини T універсальні експлуататори об'єднання та однорідного перетину класів об'єктів, то згідно з Теоремою 2.6.7 утвориться $2^{n+1} - 2(n+1)$ нових унікальних класів об'єктів, де $n = |T|$, тобто $2^{3+1} - 2(3+1) = 2^4 - 8 = 16 - 8 = 8$.

Таким чином відповідно до Теореми 2.6.8 множина однорідних класів T, розширена відповідно до Теореми 2.6.7, разом з універсальними експлуататорами об'єднання $\cup$ та однорідного перетину $\cap$, утворюють повну обмежену дистрибутивну модулярну класову ґратку з умовою обриву строго зростаючих та спадаючих лан-

цюгів $L = (T_1, E = \{\cup, \cap, 1, 0\})$, де клас $t^{\cup}_{SqRtPrRhTr} = t_{Sq} \cup \cdots \cup t_{Tr}$ – її найбільша верхня грань, тобто 1, а клас $t^{\cap}_{SqPrTr} = t_{Sq} \cap t_{Pr} \cap t_{Tr}$ – її найменша нижня грань, тобто 0 (Рис. 3.4).

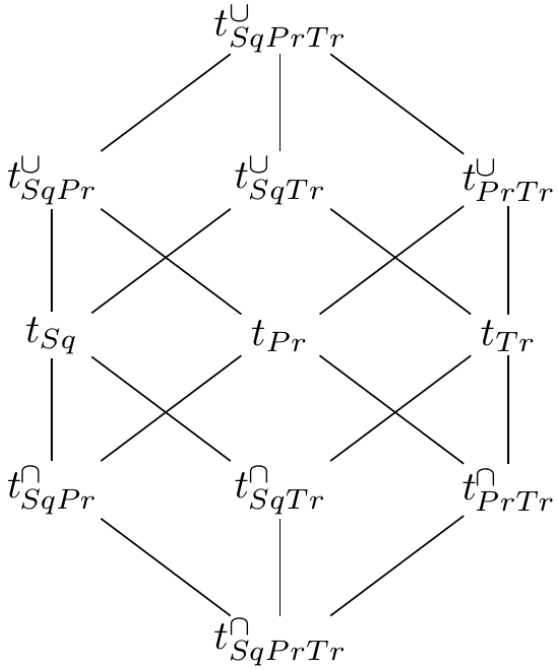


Рис. 3.4: Повна обмежена дистрибутивна модулярна класова ґратка з умовою обриву строго зростаючих та спадаючих ланцюгів

Аналізуючи побудовану класову ґратку, яка зображена на Рис. 3.4, можна бачити, що в процесі її побудови були утворені 8 нових класів об'єктів та 18 нових відношень між класами. Побудова найбільшої грані верхньої напівґратки $t^{\cup}_{SqPrTr}$ дозволяє провести ефективну реструктуризацію бази знань, при цьому за потреба, відповідно до Теореми 2.6.9, на основі класу $t^{\cup}_{SqPrTr}$ завжди можна відновити усі атоми напівґратки, тобто типи t_{Sq}, t_{Pr}, t_{Tr}. Такий підхід є доволі універсальним, оскільки його можна застосовувати до різної кількості класів, в залежності від необхідного масштабу.

3.10. Аналіз побудованої мережі

З метою подання знань про таку предметну область, як опуклі чотирикутники, в термінах об'єктно-орієнтованих динамічних мереж, була спроектована ООДМ

$ConvexQuadrangles = (O, C, R, E, M, G)$ яка має наступну структуру

$$
\begin{aligned}
O =& \{Sq_1, Sq_2, Sq_3, Sq_4, Rt_1, Rt_2, Rh_1, Rh_2, Tr_1, Tr_2, Tr_3, Tr_4, Pr_1, Pr_2\};\\
C =& \left\{T_{CQ}, T^{\cup}_{SqRtRh}, T^{\cap}_{SqRtRh}, T^{\Cap}_{SqRtRh}, T_{Sq\setminus Rh}, T_{Sq\div Rh}, T^{\cup}_{SqRtPr}, T^{\cap}_{SqRt}, T^{\Cap}_{SqRhPr},\right.\\
& \left. T_{Rh\setminus Pr}, T_{Rt\div Pr}, T^{\sqcup}_{SqRtTr}, t^{M}_{Sq1}\right\};\\
R =& \left\{Rh \xrightarrow{is-a} Pr,\ Rt \xrightarrow{is-a} Pr,\ Sq \xrightarrow{is-a} Rh,\ Sq \xrightarrow{is-a} Rt,\ Sq_1 \xrightarrow{instance-of} t_{Sq},\right.\\
& Sq_2 \xrightarrow{instance-of} t_{Sq},\ Rt_1 \xrightarrow{instance-of} t_{Rt},\ Rt_2 \xrightarrow{instance-of} t_{Rt},\\
& Rh_1 \xrightarrow{instance-of} t_{Rh},\ Rh_2 \xrightarrow{instance-of} t_{Rh},\ Tr_1 \xrightarrow{instance-of} t_{Tr},\\
& \left. Tr_2 \xrightarrow{instance-of} t_{Tr},\ Pr_1 \xrightarrow{instance-of} t_{Pr},\ Pr_2 \xrightarrow{instance-of} t_{Pr}\right\};\\
E =& \{\cup, \sqcup, \cap, \Cap, \setminus, \div, Clone_i\};\\
M =& \left\{M^{D_1,P_2,R_3,G_4}(t_{Sq1}), M^{D_1,P_2,R_3,G_4}(Sq_3)\right\};\\
G =& \{USC1, USC2, USC3, UMC1, UMC2, UMC3, UMC4, UMC5,\\
& UMC6, UMC7\}.
\end{aligned}
$$

Побудована об'єктно-орієнтована динамічна мережа на певному рівні деталізації описує опуклі чотирикутники за допомогою множини об'єктів O, множини класів об'єктів C та множин відношень R, що визначені на множинах O та C. Це дозволяє формально описати в рамках моделі інфраструктуру предметної області, а саме концепти, сутності та зв'язки між ними.

Множина експлуататорів E дозволяє виконувати над об'єктами та класами об'єктів певні операції, у результаті яких можна утворювати нові об'єкти, класи, множини та мультимножини об'єктів, і цим самим динамічно розширювати модель предметної області. Також використання універсальних експлуататорів дозволяє виконувати внутрішнє видобування знань у вигляді обмежених верхніх (нижніх) класових напівґраток або повних обмежених дистрибутивних модулярних класових ґраток з умовою обриву строго зростаючих та спадаючих ланцюгів.

Множина модифікаторів M дозволяє виконувати модифікації структури об'єктів та класів об'єктів, що дозволяє динамічно моделювати зміни об'єктів та класів об'єктів з часом та під дією сторонніх впливів. Поєднання використання універсального експлуататора клонування та модифікаторів об'єктів і класів об'єктів

різного типу можна будувати відношення модифікації, цим самим встановлюючи певні зв'язки між різними доменами та піддоменами. В даному випадку після клонування та модифікації об'єкту що описує квадрат був побудований новий об'єкт, який описує трикутник, таким чином відношення модифікації дозволило побудувати зв'язок між різними піддоменами домену опуклих многокутників.

Множина генераторів G дозволяє будувати множини та мультимножини об'єктів на основі множини O, що дає змогу проводити більш глибокий аналіз предметної області на рівні колекцій сутностей та концептів.

Очевидно що базові класи об'єктів, і як наслідок самі об'єкти, побудованої ООДМ можуть бути визначені на різному рівні деталізації, який залежить від якісного наповнення та розмірів специфікацій і сигнатур цих класів. В даному випадку ООДМ *ConvexQuadrangles* була побудована з метою демонстрації функціональних можливостей моделі подання знань, а не з метою вирішення конкретної проблеми в рамках цієї предметної області.

Структуру побудованої мережі можна зобразити графічно, побудувавши відповідний граф ООДМ, однак оскільки множини O та C містять багато об'єктів та класів об'єктів, для зручності побудуємо граф мережі на двох абстрактних рівнях – класів (Рис. 3.5) та об'єктів (Рис. 3.6).

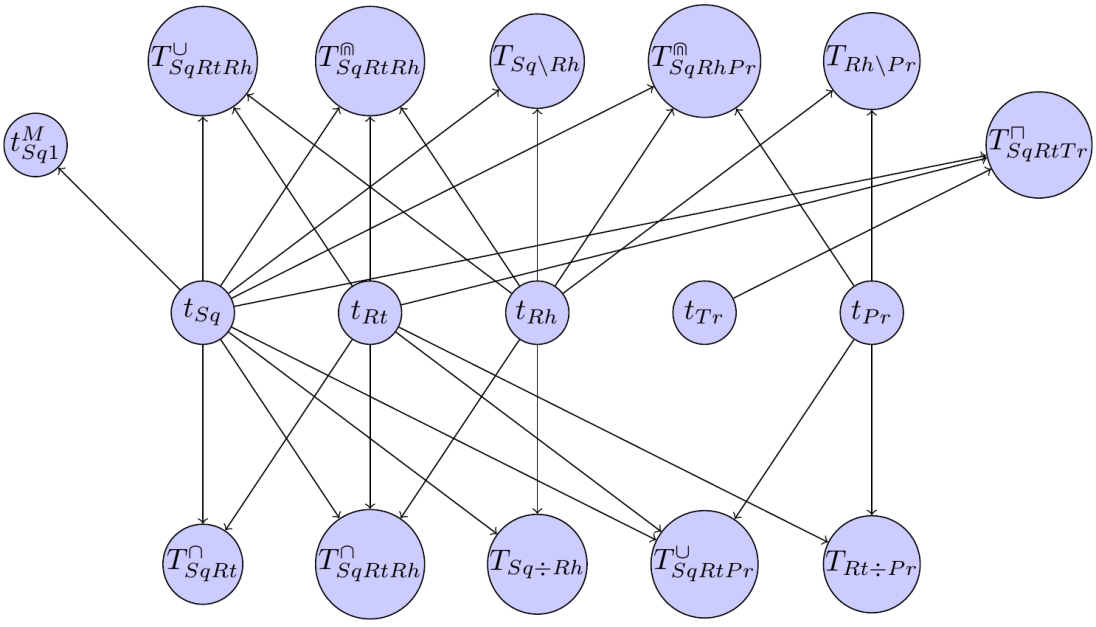


Рис. 3.5: Граф ООДМ *ConvexQuadrangles*: рівень класів об'єктів

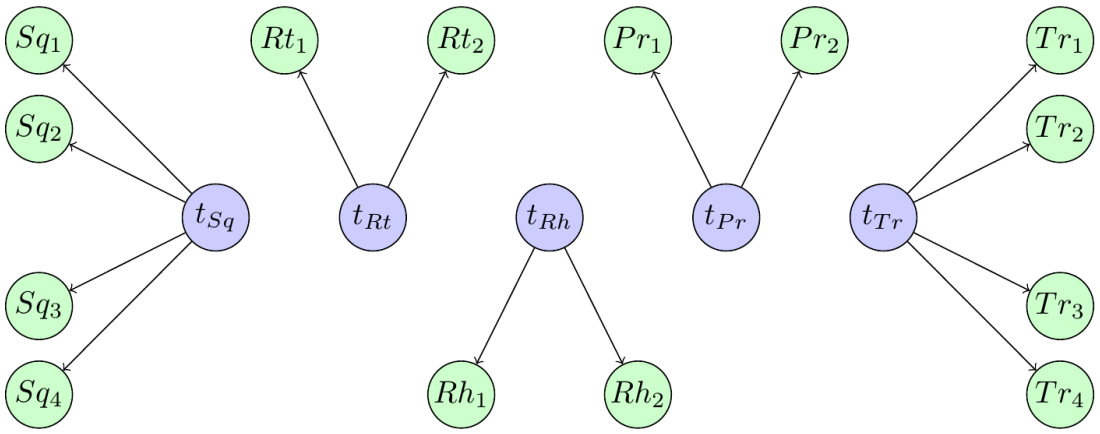


Рис. 3.6: Граф ООДМ *ConvexQuadrangles*: рівень об'єктів

Аналізуючи Рис. 3.5 та Рис. 3.6, можна бачити структуру усієї мережі, за виключенням множини відношень R. Відношення між об'єктами та класами об'єктів ООДМ не були зображені оскільки це б значно ускладнило аналіз структури побудованої мережі.

Висновки до розділу 3

В рамках даного розділу була спроектована та побудова об'єктно-орієнтована динамічна мережа *ConvexQuadrangles*, яка формалізує в термінах ООДМ деяку частину такої предметної області як опуклі чотирикутники. В рамках цього процесу були визначені множини об'єктів, класів об'єктів та відношень між ними, за допомогою яких були формалізовані декларативні знання про дану предметну область. Після чого були визначені множини експлуататорів, модифікаторів та генераторів, за допомогою яких були формалізовані процедурні знання. Після опису базових класів були розглянуті різні варіанти побудови концептуальної (класової) ієрархій в рамках ООДМ.

За допомогою експлуататорів об'єктів та класів об'єктів були побудовані нові об'єкти, класи, множини та мультимножини об'єктів, які розширили базовий набір об'єктів та класів.

За допомогою модифікаторів була трансформована структура деяких об'єктів та класів об'єктів. За допомогою модифікацій клонів об'єктів було побудовано відношення модифікації між класом квадратів та класом трикутників, що є певного

роду зв'язком між різними піддоменами домені опуклих чотирикутників.

За допомогою універсальних експлуататорів об'єднання та однорідного перетину були видобуті нові неявні знання у вигляді обмеженої верхньої класової напівґратки та повної обмеженої дистрибутивної модулярної класової ґратки з умовою обриву строго зростаючих та спадаючих ланцюгів, що дозволило провести ефективну реструктуризацію частини бази знань, за рахунок збереження найбільшої грані верхньої обмеженої класової напівґратки, замість атомів напівґратки.

Після побудови ООДМ був побудований та проаналізований граф її структури, який складається з двох абстрактних рівнів – об'єктів та класів ООДМ.

ВИСНОВКИ

Основним результатом дисертаційної роботи є розробка та дослідження властивостей такої об'єктно-орієнтованої моделі подання знань, як "Об'єктно-орієнтовані динамічні мережі" (ООДМ), що надає розширені можливості для проектування та розробки інтелектуальних програмних систем на основі знань. В ході виконання дисертаційного дослідження були отримані наступні результати:

1. Введені поняття одноядерних та багатоядерних неоднорідних класів об'єктів, які дозволяють визначати в рамках одного класу декілька різних (або подібних) типів об'єктів та подавати їх більш ефективно у порівнянні з однорідними класами, які використовуються в об'єктно-орієнтованому програмуванні та системах фреймів. Ефективність досягається шляхом побудови ядра (ядер) та проекцій класу, що дозволяє уникати дублювання властивостей, що, в свою чергу, зменшує розміри програмного коду та дозволяє компактніше зберігати інформацію в базах даних.
2. Експериментально підтверджено, що використання одноядерних неоднорідних класів об'єктів для збереження інформації у об'єктно-орієнтованих реляційних базах даних, у порівнянні з використанням однорідних класів об'єктів та однорідних класів об'єктів, зв'язаних одиничним успадкуванням, зменшує розміри таких БД і їх експортованих *.sql-файлів та пришвидшує виконання SQL-запитів.
3. Експериментально підтверджено, що використання багатоядерних неоднорідних класів об'єктів для збереження інформації у об'єктно-орієнтованих реляційних базах даних, у порівнянні з використанням однорідних класів об'єктів, однорідних класів об'єктів, зв'язаних одиничним успадкуванням, та одноядерних неоднорідних класів об'єктів, зменшує розміри таких БД і їх експортованих *.sql-файлів та пришвидшує виконання SQL-запитів.
4. Визначено відношення еквівалентності, агрегації, узагальнення та спеціа-

лізації для однорідних, одноядерних та багатоядерних неоднорідних класів об'єктів.

5. Розроблено механізми P, SgFSt, SgFW, SgPSt, SgPW, MFSt, MFW, MPSt, MPW успадкування та розширену і узагальнену класифікацію механізмів успадкування.
6. Запропоновано підходи до вирішення проблем надлишковості, винятків, неоднозначності та несумісності, що виникають під час проектування та побудови концептуальних (класових) ієрархій, що базуються на використанні концепції багатоядерних неоднорідних класів об'єктів та SgPSt, SgPW, MPSt, MPW механізмів успадкування.
7. Введено поняття експлуататорів об'єктів та класів об'єктів. Визначено універсальні експлуататори об'єднання, однорідного перетину, неоднорідного перетину, мультиоб'єднання, різниці, симетричної різниці та клонування, що дозволяють будувати нові об'єкти, класи, множини та мультимножини об'єктів на основі базових класів, що є важливим кроком у розв'язку задачі генерації нових класів об'єктів під час виконання програм (Runtime Class Generation), що в свою чергу дозволить програмним системам самостійно модифікувати власний код та бази знань, зокрема шляхом додавання нових об'єктів та класів об'єктів.
8. Введено поняття модифікаторів об'єктів та класів об'єктів. Визначено поняття повних, часткових, породжуючих, знищуючих, замінних і комбінованих модифікаторів, які дозволяють модифікувати структуру об'єктів та класів об'єктів, що дає змогу динамічно моделювати зміни сутностей об'єктів з часом і тим самим оновлювати (актуалізовувати) ООДМ. Визначені відношення модифікації об'єктів та класів об'єктів.
9. Введено поняття множини та мультимножини об'єктів, класів множин та мультимножин об'єктів, а також генераторів (конструкторів) множин та мультимножин об'єктів. Побудовано 3 універсальні конструктори множин та 7 універсальних конструкторів мультимножин об'єктів, а також CP, RCL, PS, D2 конструктори мультимножин об'єктів, які дозволяють буду-

вати такі структури знань, як множини об'єктів, мультимножини об'єктів, класи множин та мультимножин об'єктів.

10. Розроблено точні методи обчислення кратностей елементів та потужностей мультимножин об'єктів згенерованих за допомогою CP, RCL, PS, D2 конструкторів.
11. Доведено скінченність розширень підмножини однорідних класів множини базових класів ООДМ за допомогою універсальних експлуататорів об'єднання та однорідного перетину.
12. Доведено що підмножина однорідних класів множини базових класів ООДМ, розширена за допомогою експлуататора об'єднання (однорідного перетину), разом з цим експлуататором утворює верхню (нижню) обмежену класову напівґратку, що є одним із підходів до вирішення задачі видобування неявних (нових) знань на основі базових знань. Запропонований підхід дозволяє проводити ефективну реструктуризацію баз даних та знань, оскільки побудова верхніх обмежених класових напівґраток дозволяє зберігати одиниці цих напівґраток, які є неоднорідними класами об'єктів, замість множини базових однорідних класів об'єктів (атомів напівґратки).
13. Доведено що на основі найбільших верхніх граней верхніх обмежених класових напівґраток, побудованих шляхом розширення підмножини однорідних класів множини базових класів ООДМ за допомогою універсального експлуататора об'єднання, можна відновити усі елементи цих напівґраток.
14. Доведено що підмножина однорідних класів множини базових класів ООДМ, розширена за допомогою експлуататорів об'єднання та однорідного перетину, разом з цими експлуататорами утворює повну обмежену дистрибутивну модулярну класову ґратку з умовами обриву строго зростаючих та строго спадаючих ланцюгів.
15. Описано використання запропонованої моделі подання знань для представлення такого розділу геометрії, як опуклі чотирикутники.

Розроблена модель подання знань якісно доповнює парадигми об'єктно-

орієнтованого подання знань та програмування, зокрема за рахунок одноядерних та багатоядерних неоднорідних класів об'єктів, нових механізмів успадкування та узагальненої класифікації механізмів успадкування, експлуататорів та модифікаторів об'єктів і класів об'єктів, конструкторів множин та мультимножин об'єктів.

Запропонована модель подання знань у подальшому може бути інтегрована в об'єктно-орієнтовану парадигму програмування, що дозволить об'єднати рівень абстракції моделі подання знань та рівень абстракції мови програмування, обраної для її реалізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Д. О. Терлецький, "Множини як сукупність сутностей-об'єктів," *Збірник наукових праць "Комп'ютерна математика"*, № 2, с. 64–71, 2013.

[2] ——, "Конструктори множин та мультимножин об'єктів," *Проблеми програмування*, том 16, №. 1, с. 18–30, 2014.

[3] D. O. Terletskyi, "Universal and determined constructors of multisets of objects," *Information Theories & Applications*, vol. 21, no. 4, pp. 339–361, 2014.

[4] ——, "Inheritance in object-oriented knowledge representation," in *Information and Software Technologies: Proceedings of 21st International Conference, ICIST 2015, Druskininkai, Lithuania, October 15-16, 2015*, ser. Communication in Computer and Information Science, G. Dregvaite and R. Domaševičius, Eds. Springer, 2015, vol. 538, pp. 293–305.

[5] ——, "Object-oriented knowledge representation and data storage using inhomogeneous classes," in *Information and Software Technologies: Proceedings of 23rd International Conference, ICIST 2017, Druskininkai, Lithuania, October 12-14, 2017*, ser. Communication in Computer and Information Science, R. Domaševičius and V. Mikašytė, Eds. Springer, 2017, vol. 756, pp. 48–61.

[6] ——, "Process of the homogeneous multisets creation," in *Proceedings of the 11th International Scientific Conference of Students and Young Scientists "Shevchenkivska vesna 2013", Section Cybernetics, Kyiv, March 18-22*, 2013, pp. 37–39.

[7] Д. О. Терлецький, "Дослідження процесу утворення однорідних множин," *Матеріали IV Всеукраїнської науково-практичної конференції "Інформатика та системні науки", м. Полтава, 21-23 березня*, 2013, с. 278–281.

[8] ——, "Дослідження процесу утворення та класифікації неоднорідних множин," *Матеріали IX Міжнародної наукової конференції "Інтелектуальні системи прийняття рішень і проблеми обчислювального інтелекту", м. Євпа-*

торія, 20-24 травня, 2013, с. 503–505.

[9] ——, "Операції над сутностями в інтелектуальному об'єктному середовищі," *Праці X Міжнародної науково-практичної конференції "Теоретичні і прикладні аспекти побудови програмних систем", м. Ялта, 26 травня - 2 червня*, 2013, с. 181–186.

[10] D. O. Terletskyi, "The system of set theory for operating with essences in the objects intellectual environment," in *Proceedings of the 6th International Conference "Advanced Computer Systems and Networks: Design and Application", Lviv, September 16-18*, 2013, с. 226–229.

[11] Д. О. Терлецький, "Декомпозиційна генерація мультимножин об'єктів," *Тези доповідей XI Міжнародної науково-практичної конференції "Математичне і програмне забезпечення інтелектуальних систем", м. Дніпропетровськ 20-22 листопада*, 2013, с. 238–239.

[12] D. O. Terletskyi, "Finitely-generated algebras and constructors of multisets of objects," in *Proceedings of the 3rd International scientific conference of students and young scientists "Theoretical and Applied Aspects of Cybernetics", Kyiv, November 25-29*, 2013, pp. 48–63.

[13] Д. О. Терлецький, "Представлення знань за допомогою об'єктно-орієнтованих динамічних мереж," *Матеріали XIV Міжнародної наукової конференції ім. проф. Т.А. Таран "Інтелектуальний аналіз інформації", м. Київ, 14-16 травня*, 2014, с. 207–213.

[14] ——, "Комбіновані модифікатори класів та об'єктів," *Матеріали X Міжнародної наукової конференції "Інтелектуальні системи прийняття рішень і проблеми обчислювального інтелекту", м. Херсон, 28-31 травня*, 2014, с. 337–339.

[15] ——, "Модифікатори нечітких об'єктів та класів," *Матеріали III Міжнародної науково-практичної конференції "Обчислювальний інтелект (результати, проблеми, перспективи)", м. Черкаси, 12-15 травня*, 2015, с. 111–112.

[16] ——, "Комбіновані модифікатори нечітких об'єктів та класів," *Матеріали XIII Всеукраїнської науково-практичної конференції студентів, аспірантів*

та молодих вчених "Теоретичні і прикладні проблеми фізики, математики та інформатики", м. Київ, 21-23 травня, том 2, 2015, с. 89–92.

[17] D. O. Terletskyi, "Exploiters-based knowledge extraction in object-oriented knowledge representation," in *Proceedings of the 24th International Workshop "Concurrency, Specification & Programming", Rzeszow, Poland, September 28-30*, Z. Suraj and L. Czaja, Eds. Rzeszow University, 2015, vol. 2, pp. 211–221.

[18] Д. О. Терлецький, "Узагальнення концепції неоднорідних класів об'єктів," *Матеріали IV Міжнародної науково-практичної конференції "Обчислювальний інтелект (результати, проблеми, перспективи)", м. Київ, 16-18 травня*, 2017, с. 148–149.

[19] ——, "Узагальнення концепції неоднорідних класів нечітких об'єктів," *Матеріали XIII Міжнародної наукової конференції "Інтелектуальні системи прийняття рішень і проблеми обчислювального інтелекту", с. Залізний Порт, 22-26 травня*, 2017, с. 308–310.

[20] D. O. Terletskyi, "Object-oriented knowledge extraction using universal exploiters," in *Proceedings of 12th International Scientific and Technical Conference "Computer Science and Information Technologies" (CSIT'2017), Ukraine, Lviv, September 5-8*, 2017, pp. 257–266.

[21] D. O. Terletskyi and O. I. Provotar, "Mathematical foundations for designing and development of intelligent systems of information analysis," *Problems in Programming*, vol. 16, no. 2-3, pp. 233–241, 2014.

[22] ——, "Object-oriented dynamic networks," in *Computational Models for Business and Engineering Domains*, 1st ed., ser. International Book Series Information Science & Computing, G. Setlak and K. Markov, Eds. ITHEA, 2014, vol. 30, pp. 123–136.

[23] Д. А. Терлецкий and А. И. Провотар, "Нечеткие объектно-ориентированные динамические сети. I," *Кибернетика и системный анализ*, том 51, №. 1, с. 34–40, 2015.

[24] ——, "Нечеткие объектно-ориентированные динамические сети. II," *Кибернетика и системный анализ*, том 52, № 1, с. 38–45, 2016.

[25] Д. О. Терлецький and О. І. Провотар, "Дослідження процесу утворення неоднорідних мультимножин та їх класифікація," *Матеріали XI Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених "Теоретичні і прикладні проблеми фізики, математики та інформатики", м. Київ, 18-19 квітня*, 2013, с. 76–77.

[26] О. І. Провотар and Д. О. Терлецький, "Конструктивна модифікація семантичних мереж," *Збірник тез доповідей VI Наукової конференції магістрантів та аспірантів "Прикладна математика та комп'ютинг", м. Київ, 16-18 квітня*, 2014, с. 171–176.

[27] Д. О. Терлецький and О. І. Провотар, "Класові та об'єктні модифікатори," *Матеріали XII Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених "Теоретичні і прикладні проблеми фізики, математики та інформатики", м. Київ, 24-25 квітня*, 2014, с. 85–87.

[28] T. J. M. Bench-Capon, *Knowledge Representation: An Approach to Artificial Intelligence*, ser. A.P.I.C. Studies in Data Processing. Academic Press, 1990, vol. 32.

[29] C. Ramirez and B. Valdes, "A general knowledge representationmodel of concepts," in *Advances in Knowledge Representation*, C. R. Gutiérrez, Ed. InTech, 2012, pp. 43–76.

[30] B. Nebel, *Reasoning and Revision in Hybrid Representation Systems*. Springer-Verlag, 1990.

[31] A. Das, "Knowledge representation," in *Encyclopedia of Information Systems*, H. Bidgoli, Ed. Academic Press, 2002, vol. 3, pp. 33–41.

[32] D. Graham, "Knowledge representation," in *Knowledge-Based Image Processing Systems*, ser. Applied Computing, D. Graham and A. Barrett, Eds. Springer-Verlag, 1997, pp. 41–55.

[33] G. Jakus, V. Milutinović, S. Omerović, and S. Tomažič, *Concepts, Ontologies, and Knowledge Representation*, ser. SpringerBriefs in Computer Science. Springer, 2013.

[34] A. Barr and E. A. Feigenbaum, Eds., *The Handbook of Artificial Intelligence*.

William Kaufmannn, Inc., 1981, vol. 1, ch. Representation of Knowledge, pp. 143–222.

[35] J. C. Miles and C. J. Moore, *Practical Knowledge-Based Systems in Conceptual Design*. Springer-Verlag, 1994.

[36] J. H. Heinrichs, L. J. Hudspeth, and J. S. Lim, "Knowledge management," in *Encyclopedia of Information Systems*, H. Bidgoli, Ed. Academic Press, 2002, vol. 3, pp. 13–31.

[37] M. Minsky, "A framework for representing knowledge," AI Laboratory, Massachusetts Institute of Technology, Technical Report 306, 1974.

[38] R. J. Brachman and H. J. Levesque, *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, 2004.

[39] B. Coppin, *Artificial intelligence illuminated*. Jones and Bartlett Publishers, Inc., 2004.

[40] M. Negnevitsky, *Artificial Intelligence: A Guide to Intelligent Systems*, 2nd ed. Addison-Wesley, 2004.

[41] E. C. Way, *Knowledge Representation and Metaphor*. Springer Science + Business Media, B.V., 1991.

[42] P. H. Winston, *Artificial Intelligence*, 3rd ed. Addison-Wesley, 1992.

[43] G. F. Luger, *Artificial Intelligence Structures and Strategies for Complex Problem Solving*, 6th ed. Addison-Wesley, 2008.

[44] S. L. Kendal and M. Creen, *An Introduction to Knowledge Engineering*. Springer-Verlag, 2007.

[45] P. D. Karp, "The design space of frame knowledge representation systems," Artificial Intelligence Center, Computing and Engineering Sciences Division, Technical Report 520, 1992.

[46] D. S. Touretzky, *The Mathematics of Inheritance Systems*. Morgan Kaufmann Publishers, 1986.

[47] R. Al-Asady, *Inheritance Theory: An Artificial Intelligence Approach*. Ablex Publishing Corporation, 1995.

[48] F. Puppe, *Systematic Introduction to Expert Systems: Knowledge Representations and Problem-Solving Methods*. Springer-Verlag, 1993.

[49] D. Partridge, "Representation of knowledge," in *Artificial Intelligence*, 2nd ed., ser. Handbook of Perception and Cognition, M. A. Boden, Ed. Academic Press, Inc., 1996, pp. 55–85.

[50] R. J. Brachman, ""I lied about the trees" or, defaults and definitions in knowledge representation," *AI Magazine*, vol. 6, no. 3, pp. 80–93, 1985.

[51] R. C. Schank and R. P. Abelson, "Scripts, plans, and knowledge," in *Proceedings of the 4th international joint conference on Artificial intelligence (IJCAI'75), Tbilisi, Georgia*, vol. 1, 1975, pp. 151–157.

[52] R. C. Schank and Y. A. Project, "SAM – a story understander," Yale University, Department of Computer Science, Technical Report 43, 1975.

[53] R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*. Lawrence Erlbaum Associates, Inc. Publishers, 1977.

[54] R. E. Cullingford, "Script application: Computer understanding of newspaper stories," Yale University, Department of Computer Science, Technical Report 116, 1978.

[55] W. Lehnert, "What makes sam run? script based techniques for question answering," in *Proceedings of the Workshop on Theoretical Issues in Natural Language Processing, Cambridge, MA, USA*, 1975, pp. 16–21.

[56] ——, "Human and computational question answering," *Cognitive Science*, vol. 1, no. 1, pp. 47–73, 1977.

[57] R. C. Schank, *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, 1982.

[58] ——, *Dynamic Memory Revisited*, 2nd ed. Cambridge University Press, 1999.

[59] S. J. Anderson and M. A. Conway, "Representations of autobiographical memories," in *Cognitive Models of Memory*, M. A. Conway, Ed. The MIT Press, 1997, pp. 217–246.

[60] G. H. Bower, J. B. Black, and T. J. Turner, "Scripts in memory for text," *Cognitive Psychology*, vol. 11, no. 2, pp. 177–220, 1979.

[61] R. Miikkulainen, *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*. The MIT Press, 1993.

[62] R. P. Abelson, "Concepts for representing mundane reality in plans," in *Representation and Understanding: Studies in Cognitive Science*, D. G. Bobrow and A. Collins, Eds. Academic Press, Inc., 1975, pp. 273–309.

[63] B. Charlin, H. P. A. Boshuizen, E. J. Custers, and P. J. Feltovich, "Scripts and clinical reasoning," *Medical Education*, vol. 41, no. 12, pp. 1178–1184, 2007.

[64] S. Lubarsky, V. Dory, M.-C. Audétat, E. Custers, and B. Charlin, "Using script theory to cultivate illness script formation and clinical reasoning in health professions education," *Canadian Medical Education Journal*, vol. 6, no. 2, pp. 61–70, 2015.

[65] E. J. Custers, "Thirty years of illness scripts: Theoretical origins and practical applications," *Medical Teacher*, vol. 37, no. 5, pp. 1–6, 2014.

[66] F. Fischer, I. Kollar, K. Stegmann, and C. Wecker, "Toward a script theory of guidance in computer-supported collaborative learning," *Educational Psychologist*, vol. 48, no. 1, pp. 56–66, 2013.

[67] L. Bozinoff, "A script theoretic approach to informatin processing: an energy conservation application," *Advances in Consumer Research*, vol. 9, no. 1, pp. 481–486, 1982.

[68] I. D. Craig, *Object-Oriented Programming Languages: Interpretation*, ser. Undergraduate Topics in Computer Science. Springer, 2007.

[69] D. L. Schmoldt and H. M. Rauscher, *Building Knowledge-Based Systems for Natural Resource Management*. Chapman & Hall, 1996.

[70] M. Abadi and L. Cardelli, *A Theory of Objects*, ser. Monographs in Computer Science. Springer Science+Business Media, LLC, 1996.

[71] G. Blaschek, *Object-Oriented Programming with Prototypes*. Springer-Verlag, 1994.

[72] K. B. Bruce, *Foundations of Object-Oriented Languages: Types and Semantics*. The MIT Press, 2002.

[73] I. Graham, *Object-Oriented Methods*, 1st ed. Addison-Wesley, 1991.

[74] G. Booch, R. A. Maksimchuk, M. W. Engle, B. J. Young, J. Conallen, and K. A. Houston, *Object-Oriented Analysis and Design with Applications*, 3rd ed. Addison-Wesley Professional, 2007.

[75] B. Meyer, *Object-Oriented Software Construction*, 2nd ed. Prentice Hall, 1997.

[76] M. Mezini, *Variational Object-Oriented Programming Beyond Classes and Inheritance*, ser. The Springer International Series in Engineering and Computer Science. Springer Science+Business Media, LLC, 1998.

[77] M. Weisfeld, *The Object-Oriented Thought Process*, 4th ed., ser. Developer's Library. Addison-Wesley Professional, 2013.

[78] R. A. Akerkar and P. S. Sajja, *Knowledge-Based Systems*. Jones and Bartlett Publishers, LLC, 2010.

[79] D. Graham, "Introduction to knowledge-based systems," in *Knowledge-Based Image Processing Systems*, ser. Applied Computing, D. Graham and A. Barrett, Eds. Springer-Verlag, 1997, pp. 3–13.

[80] N. Bassiliades and I. Vlahavas, "Active knowledge-based systems," in *Knowledge-Based Systems: Techniques and Applications*, C. T. Leondes, Ed. Academic Press, 2000, vol. 1, pp. 1–36.

[81] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, *Recommender Systems: An Introduction*. Cambridge University Press, 2011.

[82] K. M. Hangos, R. Lakner, and M. Gerzson, *Intelligent Control Systems: An Introduction with Examples*. Kluwer Academic Publishers, 2001.

[83] C. K. Mohan, *Frontiers of Expert Systems: Reasoning with Limited Knowledge*. Springer Science+Business Media, LLC, 2000.

[84] B. Deltor, *Towards Knowledge Portals: From Human Issues to Intelligent Agents*, ser. Information Science and Knowledge Management. Springer Science+Business Media, LLC, 2004, vol. 5.

[85] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about Knowledge*. The MIT Press, 2004.

[86] S. Philippi, "Model driven generation and testing of object-relational mappings," *Journal of Systems and Software*, vol. 77, no. 2, pp. 193–207, 2005.

[87] A. Torres, R. Galante, M. S. Pimenta, and A. J. B. Martins, "Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design," *Information and Software Technology*, vol. 82, pp. 1–18, 2017.

[88] S. W. Ambler, *Agile Database Techniques: Effective Strategies for the Agile Software Developer*, 1st ed. Wiley, 2003.

[89] C. Nock, *Data Access Patterns: Database Interactions in Object-Oriented Applications*, ser. The Software Patterns Series. Addison-Wesley Professional, 2003.

[90] S. Roman, *Lattices and Ordered Sets*, 2nd ed. Springer, 2008.

[91] І. В. Сергієнко, С. Л. Кривий, and О. І. Провотар, *Алгебраїчні аспекти інформаційних технологій: частина 1*. Київ: Наукова думка, 2011.

[92] А. Г. Курош, *Лекции по общей алгебре*, 2nd ed. Москва: Наука, 1973.

[93] D. A. Simovici and C. Djeraba, *Mathematical Tools for Data Mining: Set Theory, Partial Orders, Combinatorics*, 2nd ed., ser. Advanced Information and Knowledge Processing. Springer, 2014.

[94] R. Sarnath and D. Brachma, *Object-Oriented Analysis, Design and Implementation: An Integrated Approach*, 2nd ed., ser. Undergraduate Topics in Computer Science. Springer, 2015.

[95] B. Stroustrup, *The C++ Programming Language*, 4th ed. Addison-Wesley Professional, 2013.

[96] C. J. Date, *Type Inheritance and Relational Theory: Subtypes, Supertypes, and Substitutability*. O'Reilly Media, Inc., 2016.

[97] I. Graham and P. L. Jones, "A theory of fuzzy frames: Part-1," *Bulletin for studies and exchanges on fuzziness and its applications*, no. 31, pp. 109–132, 1987.

[98] ——, "A theory of fuzzy frames: Part-2," *Bulletin for studies and exchanges on fuzziness and its applications*, no. 32, pp. 120–135, 1987.

[99] F. Smith, D. Grossman, G. Morrisett, L. Hornof, and T. Jim, "Compiling for run-time code generation (extended version)," Cornell University, Technical Report TR2000-1824, 2000.

[100] S. Kamin, "Routine run-time code generation," *ACM SIGPLAN Notices*, vol. 38, no. 12, pp. 44–56, 2003.

[101] L. Hornof and T. Jim, "Certifying compilation and run-time code generation," *Higher-Order and Symbolic Computation*, vol. 12, no. 4, pp. 337–375, 1999.

[102] O. Beckmann, A. Houghton, M. Mellor, and P. H. J. Kelly, "Runtime code generation in C++ as a foundation for domain-specific optimisation," in *Domain-Specific Program Generation: International Seminar, Dagstuhl Castle, Germany, March 23-28, 2003. Revised Papers*, ser. Lecture Notes in Computer Science, C. Lengauer, D. Batory, C. Consel, and M. Odersky, Eds. Springer, 2004, vol. 3016, pp. 291–306.

[103] N. Fujinami, "Automatic run-time code generation in C++," in *Scientific Computing in Object-Oriented Parallel Environments First International Conference, ISCOPE 97 Marina del Rey, California, USA December 8-11, 1997 Proceedings*, ser. Lecture Notes in Computer Science, Y. Ishikawa, R. R. Oldehoeft, J. V. W. Reynders, and M. Tholburn, Eds. Springer, 1997, vol. 1343, pp. 9–16.

[104] J. Unbehauen, S. Hellmann, S. Auer, and C. Stadler, "Knowledge extraction from structured sources," in *Search Computing: Broadening Web Search*, ser. Lecture Notes in Computer Science, S. Ceri and M. Brambilla, Eds. Springer, 2012, vol. 7538, pp. 34–52.

[105] N. Takhirov, "Extracting knowledge for cultural heritage knowledge base population," Ph.D. dissertation, Department of Computer and Information Science, Faculty of Information Technology, Mathematics and Electrical Engineering, Norwegian University of Science and Technology, Trondheim, Norway, 2013.

[106] J. Polpinij, "Ontology-based knowledge discovery from unstructured and semi-structured text," Ph.D. dissertation, School of Computer Science and Software

Engineering, University of Wollongong, Wollongong, NSW, Australia, 2014.

[107] The TaskGraph meta-programming library. [Online]. Available: https://www.doc.ic.ac.uk/~phjk/Software/TGL/

[108] Code generation library CGLib. [Online]. Available: https://github.com/cglib/cglib

[109] Code document object model. [Online]. Available: http://msdn.microsoft.com/en-us/library/650ax5cx.aspx

[110] Г. Кантор, *Труды по теории множеств*. Москва: Наука, 1985.

[111] B. Russell, "Mathematical logic as based on the theory of types," *American Journal of Mathematics*, vol. 30, no. 3, pp. 222–262, 1908.

[112] В. Хао and Р. Мак-Нортон, *Аксиоматические системы теории множеств*. Москва: Издательство иностранной литературы, 1963.

[113] A. A. Fraenkel, Y. Bar-Hillel, and A. Levy, *Foundations of Set Theory*, 2nd ed., ser. Studies in Logic and the Foundations of Mathematics. Elsevier, 1973, vol. 67.

[114] R. Goldblatt, *Topoi the Categorial Analysis of Logic*, reviced ed. NewYork: Dover Publications, Inc., 2006.

[115] П. Вопенка, *Математика в альтернативной теории множеств*. Москва: Мир, 1983.

[116] ——, *Альтернативная теория множеств: Новый взгляд на бесконечность*. Новосибирск: Издательство института математики, 2004.

[117] R. R. Stoll, *Set Theory And Logic*. NewYork: Dover Publications, Inc., 1979.

[118] N. G. de Bruijn, "Denumerations of rooted trees and multisets," *Discrete Applied Mathematics*, vol. 6, no. 1, pp. 25–33, 1983.

[119] W. D. Blizard, "Multiset theory," *Notre Dame Journal of Formal Logic*, vol. 30, no. 1, pp. 36–66, 1989.

[120] S. P. Jenaa, S. K. Ghosha, and B. K.Tripathy, "On the theory of bags and lists," *Information Sciences*, vol. 132, no. 1-4, pp. 241–254, 2001.

[121] А. Б. Петровский, *Пространства множеств и мультимножеств*. Москва: Едиториал УРСС, 2003.

[122] R. R. Yager, "On the theory of bags," *International Journal of General Systems*, vol. 13, no. 1, pp. 23–37, 2007.

[123] С. Л. Кривий, *Дискретна математика*, 2nd ed. Чернівці: Букрек, 2017.

[124] K. S. Leung and M. H. Wong, "Fuzzy concepts in an object oriented expert system shell," *International Journal of Intelligent Systems*, vol. 7, no. 2, pp. 171–192, 1992.

[125] J. Lee, N.-L. Xue, K.-H. Hsu, and S. J. Yang, "Modeling imprecise requirements with fuzzy objects," *Information Sciences*, vol. 118, no. 1–4, pp. 101–119, 1999.

[126] T. D. Ndousse, "Intelligent systems modeling with reusable fuzzy objects," *International Journal of Intelligent Systems*, vol. 12, no. 2, pp. 137–152, 1997.

[127] Z. M. Ma, W. J. Zhang, and W. Y. Ma, "Extending object-oriented databases for fuzzy information modeling," *Information Systems*, vol. 29, no. 5, pp. 421–435, 2004.

[128] F. Berzal, N. Marín, O. Pons, and M. A. Vila, "Managing fuzziness on conventional object-oriented platforms," *International Journal of Intelligent Systems*, vol. 22, no. 7, pp. 781–803, 2007.

[129] N. Marín, O. Pons, and M. A. Vila, "Fuzzy types: A new concept of type for managing vague structures," *International Journal of Intelligent Systems*, vol. 15, no. 11, pp. 1061–1085, 2000.

[130] S. Hellmann, J. Unbehauen, A. Zaveri, J. Lehmann, S. Auer, S. Tramp, H. Williams, O. Erling, T. Thibodeau, K. Idehen, A. Blumauer, and H. Nagy, "Lod2 deliverable 3.1.1: Report on knowledge extraction from structured sources," Technical Report 257943, 2010.

[131] J. M. Tirado, O. Serban, Q. Guo, and E. Yoneki, "Web data knowledge extraction," University of Cambridge, Computer Laboratory, Technical Report UCAM-CL-TR-881, 2016.

[132] H. Paredes-Frigolett and L. F. A. M. Gomes, "A novel method for rule extraction in a knowledge-based innovation tutoring system," *Knowledge-Based Systems*, vol. 92, pp. 183–199, 2016.

[133] C. Wang, "Knowledge extraction and retrieval for domain-specific documants," Ph.D. dissertation, University of California, Santa Cruz, USA, 2015.

[134] S. Metzger, "User-centric knowledge extraction and maintenance," Ph.D. dissertation, Faculty of Natural Sciences and Technologies, Saarland University, Saarbrücken, Germany, 2014.

[135] Z. Ma, F. Zhang, L. Yan, and J. Cheng, *Fuzzy Knowledge Management for the Semantic Web*, ser. Studies in Fuzziness and Soft Computing. Springer, 2014, vol. 306.

[136] L. Yan, Z. Ma, and F. Zhang, *Fuzzy XML Data Management*, ser. Studies in Fuzziness and Soft Computing. Springer, 2014, vol. 311.

[137] G. Birkhoff, *Lattice Theory*, 3rd ed. American Mathematical Society Colloquium Publications, 1967.

[138] S. Burris and H. P. Sankappanavar, *A Course in Universal Algebra*, ser. Graduate Texts in Mathematics. Springer, 1981, vol. 78.

[139] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, 2nd ed. Cambridge University Press, 2002.

[140] R. Freese, J. Jezek, and J. B. Nation, *Free Lattices*, ser. Mathematical Surveys and Monographs. Volume 42. American Mathematical Society, 1995.

Додаток А

Експериментальні дані

Таблиця А.1

Розміри БД розгорнутих на сервері (Mb)

Кількість об'єктів	**ОК**	**ОК+ОУ**	**НОК**	**БНОК**
12000	4.828125	6.750000	5.078125	1.906250
24000	7.968750	11.921875	8.359375	5.953125
36000	11.109375	18.218750	8.703125	6.515625
48000	15.296875	23.234375	11.984375	7.828125
60000	18.437500	31.062500	15.296875	8.437500
72000	21.531250	33.500000	17.484375	10.765625
84000	26.765625	39.500000	21.953125	12.546875
96000	29.859375	48.812500	22.140625	12.859375
108000	33.000000	53.093750	25.421875	15.328125
120000	40.140625	65.406250	35.703125	26.796875
132000	40.140625	65.406250	35.703125	26.796875
144000	45.140625	70.406250	38.703125	27.796875
156000	49.140625	77.406250	41.703125	29.796875
168000	49.140625	77.406250	41.703125	29.796875
180000	56.140625	85.421875	46.703125	30.796875
192000	61.171875	91.421875	50.703125	33.796875
204000	61.171875	91.421875	50.703125	33.796875
216000	61.171875	99.421875	50.703125	33.796875
228000	61.171875	101.421875	57.703125	35.796875
240000	75.234375	107.421875	57.703125	35.796875

Таблиця А.2

Розміри експортованих *.sql-файлів БД (Mb)

Кількість об'єктів	**ОК**	**ОК+ОУ**	**НОК**	**БНОК**
12000	3.5189104	3.8692379	2.4798737	0.8951645
24000	7.0324945	7.7591295	4.9537010	1.7825556
36000	10.5518064	11.6663580	7.4334583	2.6758118
48000	14.0768347	15.5907383	9.9187317	3.5746469
60000	17.6018629	19.5147543	12.4042053	4.4734821
72000	21.1268911	23.4391985	14.8894768	5.3726444
84000	24.6519194	27.3635788	17.3749523	6.2714796
96000	28.1767168	31.2878027	19.8602238	7.1705780
108000	31.7017450	35.2274446	22.3454952	8.0694132
120000	35.2267733	39.1747131	24.8311729	8.9684381
132000	38.7518015	43.1220455	27.3164444	9.8674107
144000	42.2765989	47.0692110	29.8017159	10.7662458
156000	45.8016272	51.0164280	32.2871914	11.6653442
168000	49.3268862	54.9637604	34.7726650	12.5643692
180000	52.8519144	58.9112368	37.2579365	13.4632044
192000	56.3767118	62.8584538	39.7432098	14.3621769
204000	59.9017401	66.8056812	42.2286835	15.2610121
216000	63.4267683	70.7528858	44.7139549	16.1599846
228000	66.9517965	74.7002182	47.1996326	17.0591354
240000	70.4765940	78.6474352	49.6847019	17.9581079

Таблиця А.3

Коефіцієнти ефективності використання концепцій НОК та БНОК у порівнянні з ОК, ОК+ОУ для збереження інформації у реляційних базах даних: БД розгорнуті на сервері

Кількість об'єктів	**НОК/ОК**	**НОК/ОК+ОУ**	**БНОК/ОК**	**БНОК/ОК+ОУ**	**БНОК/НОК**
12000	−0.0518	0.2477	0.6052	0.7176	0.6246
24000	−0.0490	0.2988	0.2529	0.5007	0.2879
36000	0.2166	0.5223	0.4135	0.6424	0.2513
48000	0.2165	0.4842	0.4883	0.6631	0.3468
60000	0.1703	0.5075	0.5424	0.7284	0.4484
72000	0.1879	0.4781	0.5000	0.6786	0.3843
84000	0.1798	0.4442	0.5312	0.6824	0.4285
96000	0.2585	0.5464	0.5693	0.7366	0.4192
108000	0.2296	0.5212	0.5355	0.7113	0.3971
120000	0.1105	0.4541	0.3324	0.5903	0.2495
132000	0.1105	0.4541	0.3324	0.5903	0.2495
144000	0.1426	0.4503	0.3842	0.6052	0.2818
156000	0.1514	0.4612	0.3936	0.6151	0.2855
168000	0.1514	0.4612	0.3936	0.6151	0.2855
180000	0.1681	0.4533	0.4514	0.6395	0.3406
192000	0.1711	0.4454	0.4475	0.6303	0.3334
204000	0.1711	0.4454	0.4475	0.6303	0.3334
216000	0.1711	0.4900	0.4475	0.6601	0.3334
228000	0.0567	0.4311	0.4148	0.6471	0.3796
240000	0.2330	0.4628	0.5242	0.6668	0.3796
Середнє	**0.1498**	**0.4529**	**0.4504**	**0.6475**	**0.3519**
Відсотки	**15%**	**45.3%**	**45%**	**64.8%**	**35.2%**

Таблиця А.4

Коефіцієнти ефективності використання концепцій НОК та БНОК у порівнянні з ОК, ОК+ОУ для збереження інформації у реляційних базах даних: експортовані *.sql-файли БД

Кількість об'єктів	**НОК/ОК**	**НОК/ОК+ОУ**	**БНОК/ОК**	**БНОК/ОК+ОУ**	**БНОК/НОК**
12000	0.2953	0.3591	0.7456	0.7686	0.6390
24000	0.2956	0.3616	0.7465	0.7703	0.6402
36000	0.2955	0.3628	0.7464	0.7706	0.6400
48000	0.2954	0.3638	0.7461	0.7707	0.6396
60000	0.2953	0.3644	0.7459	0.7708	0.6394
72000	0.2952	0.3648	0.7457	0.7708	0.6392
84000	0.2952	0.3650	0.7456	0.7708	0.6391
96000	0.2952	0.3652	0.7455	0.7708	0.6389
108000	0.2951	0.3657	0.7455	0.7709	0.6389
120000	0.2951	0.3661	0.7454	0.7711	0.6388
132000	0.2951	0.3665	0.7454	0.7712	0.6388
144000	0.2951	0.3669	0.7453	0.7713	0.6387
156000	0.2951	0.3671	0.7453	0.7713	0.6387
168000	0.2951	0.3674	0.7453	0.7714	0.6387
180000	0.2951	0.3676	0.7453	0.7715	0.6386
192000	0.2950	0.3677	0.7452	0.7715	0.6386
204000	0.2950	0.3679	0.7452	0.7716	0.6386
216000	0.2950	0.3680	0.7452	0.7716	0.6386
228000	0.2950	0.3681	0.7452	0.7716	0.6386
240000	0.2950	0.3683	0.7452	0.7717	0.6386
Середнє	**0.2952**	**0.3657**	**0.7455**	**0.7710**	**0.6389**
Відсотки	**29.5%**	**36.6%**	**74.6%**	**77%**	**64%**

Додаток Б

Допоміжні таблиці

Таблиця Б.1

Структура неоднорідного класу T_{CQ}

Ядро / Проекція	**Властивість / Метод**	**Спільні для типів**
$Core_1^5(T_{CQ})$	$p_1(T_{CQ})$, $p_2(T_{CQ})$, $p_3(T_{CQ})$, $f_1(T_{CQ})$	t_{Sq}, t_{Rt}, t_{Pr}, t_{Rh}, t_{Tr}
$Core_1^2(T_{RhPr})$	$p_1(T_{RhPr})$, $p_2(T_{RhPr})$, $f_1(T_{RhPr})$	t_{Rh}, t_{Pr}
$Core_2^2(T_{SqRt})$	$p_1(T_{SqRt})$, $p_2(T_{SqRt})$, $f_1(T_{SqRt})$	t_{Sq}, t_{Rt}
$Core_3^2(T_{SqRh})$	$p_1(T_{SqRh})$	t_{Sq}, t_{Rh}
$Core_4^2(T_{RtPr})$	$p_1(T_{RtPr})$	t_{Rt}, t_{Pr}
$Core_1^1(T_{Tr})$	$p_1(T_{Tr})$, $f_1(T_{Tr})$	t_{Tr}
$pr_1(t_{Sq})$	$p_1(t_{Sq})$	t_{Sq}
$pr_1(t_{Rt})$	$p_1(t_{Rt})$	t_{Rt}
$pr_1(t_{Rh})$	$p_1(t_{Rh})$, $p_2(t_{Rh})$	t_{Rh}
$pr_1(t_{Tr})$	$p_1(t_{Tr})$, $p_2(t_{Tr})$	t_{Tr}
$pr_1(t_{Pr})$	$p_1(t_{Pr})$, $p_2(t_{Pr})$	t_{Pr}