

Ю.В. Нікольський, В.В. Пасічник,  
Ю.М. Щербина

# СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Навчальний посібник



omputing

Міністерство освіти і науки, молоді та спорту України

Ю. В. Нікольський, В. В. Пасічник, Ю. М. Щербина

# СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

НАВЧАЛЬНИЙ ПОСІБНИК  
СЕРІЯ «КОМП'ЮТИНГ»

За науковою редакцією д.т.н., професора В.В. Пасічника

Видання друге, виправлене та доповнене

*Рекомендовано Міністерством освіти і науки України*

НБ ПНУС



783870

Видавництво «Магнолія – 2006»

Львів – 2013

ББК 32.813я7  
УДК 004.8 (075)  
Н 64

Відтворення цієї книги  
або будь-якої її частини заборонено  
без письмової згоди видавництва.  
Будь-які спроби порушення авторських прав  
переслідуватимуться у судовому порядку.

Гриф надано Міністерством освіти і науки України,  
лист № 1.4/18-Г-1825 від 26.10.2007 р.

#### Рецензенти:

**Соловійова Катерина Олександрівна** – завідувач кафедри соціальної інформатики Харківського національного університету радіоелектроніки, доктор технічних наук, професор;

**Ігачук Микола В'ячеславович** – доктор технічних наук, професор кафедри АСУ Національного університету «Харківський політехнічний інститут»;

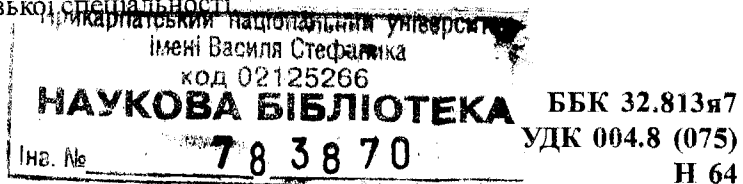
**Цегелик Григорій Григорович** – завідувач кафедри математичного моделювання соціально-економічних процесів Львівського національного університету імені Івана Франка, доктор фізико-математичних наук, професор.

Ю. В. Нікольський, В.В. Пасічник, Ю. М. Щербина  
Системи штучного інтелекту: навчальний посібник. – Львів: «Магнолія-2006», 2013.  
– 279 с.

ISBN 978-617-57-40-11-4

У навчальному посібнику викладено основні поняття та методи систем штучного інтелекту. Наведено методику побудови предметної області, описано процедури пошуку в просторі станів. Розглянуто теорію логічного виведення, а також реалізацію доведення теорем методом резолюцій. Систематизовано подано ідеї та алгоритми машинного навчання; зокрема, розглянуто дерева рішень та нейронні мережі. Окремий розділ присвячено міркуванням в умовах невизначеності.

Посібник адресовано студентам і аспірантам спеціальностей, пов'язаних із вивченням систем штучного інтелекту, він буде корисним і для професіоналів, які бажають вийти за межі вузької спеціальності.



ISBN 978-617-57-40-11-4

© Ю. В. Нікольський, В.В. Пасічник,  
Ю. М. Щербина, 2013  
© «Магнолія-2006», 2013

## ЗМІСТ

|  |   |
|--|---|
| ПЕРЕДМОВА НАУКОВОГО РЕДАКТОРА СЕРІЇ ПІДРУЧНИКІВ<br>ТА НАВЧАЛЬНИХ ПОСІБНИКІВ «КОМП'ЮТИНГ» ..... | 6 |
|--|---|

|                             |    |
|-----------------------------|----|
| ВСТУПНЕ СЛОВО АВТОРІВ ..... | 11 |
|-----------------------------|----|

|   |    |
|---|----|
| РОЗДІЛ 1.   |    |
| ПОДАННЯ ЗАДАЧ ТА ПОШУК РОЗВ'ЯЗКІВ .....                     | 14 |
| 1.1. Класифікація задач .....                               | 14 |
| 1.2. Основні поняття та означення .....                     | 15 |
| 1.3. Модель предметної області .....                        | 16 |
| 1.4. Процедура розв'язування задачі .....                   | 17 |
| 1.5. Приклади розв'язування задач .....                     | 18 |
| 1.6. Простір станів. Подання задачі в просторі станів ..... | 21 |
| 1.7. Метод пошуку вишир .....                               | 25 |
| 1.8. Метод пошуку вглиб .....                               | 27 |
| 1.9. Порівняння методів пошуку вишир і вглиб .....          | 29 |
| 1.10. Пошук вглиб з ітераційним заглибленням .....          | 29 |
| 1.11. Евристичні методи пошуку в просторі станів .....      | 30 |
| 1.12. Метод пошуку по першому найкращому збігу .....        | 32 |
| 1.13. Подання задач у вигляді графів AND/OR .....           | 35 |
| Задачі для самостійного розв'язування .....                 | 42 |
| Комп'ютерні проекти .....                                   | 42 |

|   |    |
|---|----|
| РОЗДІЛ 2.   |    |
| ЛОГІЧНЕ ВИВЕДЕННЯ .....   | 43 |
| 2.1. Короткі історичні відомості .....                            | 43 |
| 2.2. Логіка висловлювань .....                                    | 45 |
| 2.3. Закони логіки висловлювань .....                             | 51 |
| 2.4. Нормальні форми логіки висловлювань .....                    | 53 |
| 2.5. Логіка першого ступеня .....                                 | 54 |
| 2.6. Закони логіки першого ступеня .....                          | 57 |
| 2.7. Випереджена нормальна форма .....                            | 59 |
| 2.8. Логічне виведення в логіці висловлювань .....                | 60 |
| 2.9. Алгоритм Куайна .....  | 62 |
| 2.10. Алгоритм Девіса–Патнема .....                               | 66 |
| 2.11. Поняття формальної теорії. Числення висловлювань .....      | 70 |
| 2.12. Застосування правил виведення в численні висловлювань ..... | 73 |
| 2.13. Метод резолюцій у численні висловлювань .....               | 75 |



|  |     |
|--|-----|
| 2.14. Числення предикатів як формальна теорія. Логічне виведення в численні предикатів ..... | 79  |
| 2.15. Правила виведення в численні предикатів .....  | 81  |
| 2.16. Підстановка та уніфікація. Найзагальніший уніфікатор. Алгоритм уніфікації .....        | 83  |
| 2.17. Сколемівська нормальна форма .....   | 86  |
| 2.18. Метод резолюцій у численні предикатів .....  | 89  |
| 2.19. Хорнівські диз'юнкти та метод <i>SLD</i> -резолюцій .....                              | 95  |
| 2.20. Принцип логічного програмування .....  | 98  |
| <i>Задачі для самостійного розв'язування</i> .....   | 103 |
| <i>Комп'ютерні проекти</i> .....   | 107 |

## РОЗДІЛ 3.

## МАШИННЕ НАВЧАННЯ

|   |     |
|---|-----|
| 3.1. Інтуїтивне розуміння навчання .....                          | 108 |
| 3.2. Означення навчання .....                                     | 109 |
| 3.3. Програми, що навчаються .....                                | 110 |
| 3.4. Мотивація до навчання .....                                  | 113 |
| 3.5. Таксономія машинного навчання .....                          | 114 |
| 3.6. Споріднені галузі .....                                      | 116 |
| 3.7. Навчання як розділ штучного інтелекту .....                  | 117 |
| 3.8. Загальне формулювання задачі навчання за прецедентами .....  | 118 |
| 3.9. Основні поняття та означення .....                           | 119 |
| 3.10. Типологія задач навчання за прецедентами .....              | 119 |
| 3.11. Задачі з описом об'єктів на основі ознак .....              | 122 |
| 3.12. Приклади задач машинного навчання .....                     | 123 |
| 3.12.1. Задачі класифікації .....                                 | 123 |
| 3.12.2. Задача відновлення регресії .....                         | 127 |
| 3.12.3. Задачі прогнозування та прийняття рішень .....            | 127 |
| 3.12.4. Задачі кластеризації .....                                | 130 |
| 3.12.5. Задачі аналізу клієнтських середовищ .....                | 130 |
| 3.13. Навчання понять в штучному інтелекті .....                  | 131 |
| 3.13.1. Задача навчання понять – пошук у просторі гіпотез .....   | 132 |
| 3.13.2. Упорядкування гіпотез „від загальної до конкретної” ..... | 135 |
| 3.13.3. Алгоритм Find-S пошуку найконкретнішої гіпотези .....     | 136 |
| 3.13.4. Алгоритм „вилучення кандидата” .....                      | 139 |
| 3.14. Дерево рішень .....   | 153 |
| 3.14.1. Означення дерева рішень .....                             | 153 |
| 3.14.2. Алгоритми побудови дерева рішень .....                    | 155 |
| 3.15. Навчання на основі зв'язків .....                           | 168 |
| 3.15.1. Біологічні нейронні мережі .....                          | 169 |

|   |     |
|---|-----|
| 3.15.2. Модель штучного нейрона .....                             | 169 |
| 3.15.3. Подання нейромереж та їхньої архітектури .....            | 173 |
| 3.15.4. Сучасні архітектури нейромереж .....                      | 175 |
| 3.15.5. Навчання одношарових нейромереж прямого поширення .....   | 180 |
| 3.15.6. Навчання багатошарових нейромереж прямого поширення ..... | 186 |
| 3.16. Метод опорних векторів .....                                | 196 |
| 3.17. Мережі, що самоорганізуються .....                          | 205 |
| 3.17.1. Опис мереж, що самоорганізуються .....                    | 205 |
| 3.17.2. Міри відстані між векторами .....                         | 207 |
| 3.17.3. Проблема нормалізації векторів .....                      | 207 |
| 3.17.4. Міра організації мережі .....                             | 208 |
| 3.17.5. Механізм стомлення нейронів .....                         | 209 |
| 3.17.6. Методи навчання мереж, що самоорганізуються .....         | 210 |
| <i>Задачі для самостійного розв'язування</i> .....                | 213 |
| <i>Комп'ютерні проекти</i> .....                                  | 222 |

## РОЗДІЛ 4.

## МІРКУВАННЯ В УМОВАХ НЕВИЗНАЧЕНОСТІ

|  |     |
|--|-----|
| 4.1. Нечіткі множини .....   | 225 |
| 4.1.1. Означення нечіткої множини .....  | 225 |
| 4.1.2. Операції на нечітких множинах .....                                       | 231 |
| 4.1.3. Нечіткі числа .....   | 234 |
| 4.1.4. Нечіткі відношення та їхні властивості .....                              | 236 |
| 4.1.5. Трикутні норми .....  | 238 |
| 4.2. Міркування з використанням нечітких множин .....                            | 240 |
| 4.2.1. Логічне виведення з використанням апарату нечітких множин .....           | 240 |
| 4.2.2. Застосування теорії нечітких множин до побудови пристроїв керування ..... | 243 |
| 4.2.3. Приклад застосування нечітких множин до побудови пристрою керування ..... | 249 |
| 4.3. Стохастичний підхід до подання невизначеності .....                         | 255 |
| 4.3.1. Байєсівські міркування .....  | 255 |
| 4.3.2. Наївний байєсівський класифікатор .....                                   | 260 |
| 4.3.3. Байєсівські мережі довіри .....   | 262 |
| <i>Задачі для самостійного розв'язування</i> .....                               | 269 |
| <i>Комп'ютерні проекти</i> .....   | 270 |
| АЛФАВІТНИЙ ПОКАЖЧИК .....  | 271 |
| ЛІТЕРАТУРА .....   | 275 |



## Передмова наукового редактора серії підручників та навчальних посібників “КОМП’ЮТИНГ”

Шановний читачу!

Започатковуючи масштабний освітньо-науковий проект підготовки і видання серії сучасних підручників і навчальних посібників під загальною назвою “КОМП’ЮТИНГ” та із загальним методичним патронуванням його Інститутом інноваційних технологій та змісту освіти МОН України, мені як ініціатору та науковому керівнику неодноразово доводилось прискіпливо аналізувати загальну ситуацію в царині сучасного україномовного підручника комп’ютерно-інформатичного профілю. Загалом, позитивна тенденція останніх років ще не співмірна з надзвичайно динамічним розвитком як освітньо-наукової та виробничої сфери комп’ютингу, так і стрімким розширенням потенційної цільової читалької аудиторії цього профілю. Іншими словами, попередній аналіз засвідчує наявність значного соціального замовлення під реалізацію пропонованого вашій увазі проекту.

Ще одним фактором формування освітньо-наукової ініціативи, пропонованої групою відомих вітчизняних науковців-педагогів та практиків, які організовують наукові дослідження, готують фахівців та провадять бізнес в галузі комп’ютингу, постало завдання широкомасштабного включення Української вищої школи до загальноєвропейських і всесвітніх об’єднань, структур і асоціацій. Виконуючи функцію науково-технічного локомотиву суспільства, галузь комп’ютингу невідворотно зобов’язана зірвати роль активного творця загальної освітньо-наукової платформи, яка має бути методологічно-об’єднавчою та професійно-інтеграційною основою для багатьох сфер людської діяльності.

Третім суттєвим фактором, який спонукав започаткувати пропоновану серію підручників і посібників, є об’єктивно визріла ситуація, коли фахівцям та науковцям треба подати чіткий сигнал щодо науково-методологічного осмислення та викладення базових знань галузі комп’ютингу як освітньо-наукової, виробничо-економічної та сервісно-обслуговувальної сфери.

Читач, безсумнівно, зверне увагу на нашу послідовну промоцію нового терміну “КОМП’ЮТИНГ” (computing, англ.), який є вдатим та комплексно узагальнювальним для означення галузі знань, науки, виробництва, надання відповідних послуг та сервісів. Видається доречним подати ретроспективу як самого терміну комп’ютинг, так і широкої освітньої, наукової, бізнесової та виробничої сфери діяльності, що іменується комп’ютигом.

Уперше термін “комп’ютинг” уведений 1998 року Яном Фостером з Арагонської національної лабораторії чикагського університету та Карлом Кесельманом з Інституту інформатики штату каліфорнія (США) та запропонований для означення комплексної галузі знань, яка включає проектування та побудову апаратних і програмних систем для широкого кола застосувань: вивчення процесів, структур і керування інформацією різних видів; виконання наукових досліджень із застосування комп’ютерів та їх інтелектуальності; створення і використання комунікаційних та демонстраційних засобів, пошуку та збирання інформації для конкретної мети і т. ін.

У подальшому сфера використання терміну суттєво розширилась, зокрема, в освітньо-науковій царині його почали використовувати для означення відповідної галузі знань, для якої періодично (орієнтовно щодесять років) провідними університетами та професійними асоціаціями фахівців розробляються та імплементуються навчальні плани і програми, котрі в подальшому набувають статусу міжнародно визнаних освітньо-професійних стандартів. Зокрема, варто акцентувати увагу на версіях підсумкового документу “Computing CURRICULA” 2001 року. За окремими повідомленнями можна стверджувати, що черговий збірник стандартів “Computing CURRICULA” буде поданий професійному загалу до 2011 року. Перше організаційне засідання відповідних фахових робочих груп відбулось у Чикагському університеті влітку 2007 року.

Для формування цілісного однорідного подання суті “КОМП’ЮТИНГУ” ми базуємось на сучасних наукових уявленнях з максимально можливим строгим покомпонентним викладенням основних базових означень та понять, які склались історично і є загальноновизнаними в професійних колах. водночас для побудови цілісної зваженої картини ми використали певні узагальнення та загальносистемні класифікаційні підходи.

Безсумнівно, що базовим та фундаментальним поняттям було, є і залишається поняття ІНФОРМАТИКИ (informatique, франц.), як фундаментальної науки, котра вивчає найбільш загальні закони та закономірності процесів відбору, реєстрації, збереження, передавання, захисту, опрацювання та подання інформації. Як фундаментальна наука інформатика була подана в 70-х роках ХХ ст. При цьому хочу відразу ж застерегти від примітивного ототожнення, яке часто є наївно вживаним щодо еквівалентності понять “інформатика” (informatique, франц.) та “комп’ютерні науки” (computer science, англ.). Такі ототожнення, з певною мірою наближення, можливі щодо розширеного сучасного трактування інформатики як загалом прикладної науки про обчислення, збереження, опрацювання інформації та побудову прикладних інформаційних технологій і систем на їх базі. Таке трактування є характерним в ряді європейських країн. строге ж означення та подання предмету досліджень інформатики, а саме – інформації, має справу з фундаментальним не редукованим поняттям і фіксується у словниках як “informatio” (лат.) – відомості, повідомлення. Вивченням та всестороннім аналізом сутності інформації опікується наука, що називається “теорія інформації”. На нашу думку, основною принциповою відмінністю між інформатикою та комп’ютерними науками є те, що перша в своєму первинному поданні відноситься до категорії фундаментальних наук, як то фізика, математика, хімія і т. ін. У той же час комп’ютерні науки загалом за своєю сутнісною природою та всіма наявними ознаками належать до категорії прикладних наук, які базуються на фундаментальних законах та закономірностях інформаційних процесів, котрі вивчаються в рамках фундаментальної науки інформатики.

Особливо наголосимо на тому, що фундаментальна наука та її результати не призначені для безпосереднього промислового використання.

Для комп’ютерних наук характерною ознакою виділення їх у спектрі прикладних наук є об’єкт прикладення знань, умінь та навичок у контексті конкретного об’єкту – обчислювача (комп’ютера). Іншою відокремленою прикладною науковою галуззю, що базується на підвалинах інформатики, є розділ прикладних наук, основним об’єктом яких є сам процес обчислень. Це науки, які іменуються обчислювальними науками – “computationally science” (англ.). Традиційно сюди відносять обчислювальну та комп’ютерну математику.

Третьою прикладною науковою галуззю, яка ґрунтується на фундаментальних законах інформатики, є розділ прикладних наук, основним об'єктом яких є інформаційний ресурс (у сучасній літературі часто вживається поняття “*контент*” (content, англ.) у розумінні інформаційного наповнення). Ці прикладні науки одержали назву “*інформаційні науки*” (information science, англ.).

У галузі прикладних інформаційних наук базовий об'єкт досліджень, а саме інформаційний ресурс, подається, як правило, у формі даних та знань. За спрощеною формулою означатимемо дані як матеріалізовану інформацію, тобто інформацію, яку подано на матеріальних носіях, знання як суб'єктивізовану інформацію, тобто інформацію, яка природно належить суб'єкту, і в традиційному розумінні перебуває в людській пам'яті.

Узагальнюючи класифікаційно-ознакову схему, стверджуємо, що на базі фундаментальної науки ІНФОРМАТИКИ формуються три прикладні наукові галузі, а саме: *комп'ютерні науки, обчислювальні науки та інформаційні науки* з відповідними об'єктами досліджень у своїх сферах.

Ще раз підкреслимо, що результати фундаментальних наукових досліджень не призначені для безпосереднього промислового використання, у той же час результати прикладних наукових досліджень, як правило, призначені для створення та удосконалення нових технологій.

Гносеологічний аналіз подальшого формування інженерного рівня сфери КОМП'ЮТИНГУ невідворотно веде до структурного подання базових типів інженерій, які трактуються у класичному розумінні. ІНЖЕНЕРІЯ (майстерний – від лат. ingeniosus) – це наука про проектування та побудову (чит. створення) об'єктів певної природи. У цьому контексті природними для сфери “КОМП'ЮТИНГУ” є декілька видів інженерії. Мова йтиме про:

- КОМП'ЮТЕРНУ ІНЖЕНЕРІЮ (computer engineering, англ.), яка охоплює проблематику проектування та створення об'єктів комп'ютерної техніки;
- ПРОГРАМНУ ІНЖЕНЕРІЮ (software engineering, англ.), яка опікується проблематикою проектування та створення об'єктів, що іменуються програмними продуктами;
- ІНЖЕНЕРІЮ ДАНИХ ТА ЗНАНЬ (data & knowledge engineering, англ.), інженерія, яка опікується проектуванням та створенням інформаційних продуктів;
- інженерію, яка опікується проектуванням та створенням міжкомпонентних (інтерфейсних) взаємозв'язків та формуванням цілісних системних об'єктів, усе частіше іменують СИСТЕМНОЮ ІНЖЕНЕРІЄЮ (systems engineering, англ.).

У разі такого структурно-класифікаційного подання видів інженерій сфери комп'ютингу, зазначимо, що кожен з них у цьому трактуванні є “*відповідальним*” за певний тип забезпечення, а саме *апаратного* (hardware, англ.), *програмного* (software, англ.), *інформаційного* (dataware, англ.) та *міжкомпонентного* (middleware, англ.). Інформаційну технологію (ІТ) можна трактувати як певну точку в чотиривимірному просторі зазначених інженерій. При цьому необхідно обов'язково зважити на певну частку наближення та інтерпретації цього простору як дискретного та неметричного.

У зв'язку з поширеним різночитанням та трактуванням поняття інформаційної технології (ІТ), видається необхідним детальніше подати сутнісну структуру цього

терміну, використовуючи при цьому термінологічні статті популярного інформаційного ресурсу, яким є Wikipedia – [<http://www.wikipedia.org/>].

Технологія (від грецького techne – мистецтво, майстерність, вміння та грецького logos – знання) – сукупність методів та інструментів для досягнення бажаного результату, спосіб перетворення чогось заданого в необхідне. Технологія – це наукова дисципліна, в рамках якої розробляються та удосконалюються способи й інструменти виробництва.

У широкому розумінні – це знання, які можна використати для виробництва продуктів (товарів та послуг) з економічних ресурсів. У вузькому розумінні – технологія подається як спосіб перетворення речовини, енергії, інформації в процесі виготовлення продукції, обробки та переробки матеріалів, складання готових виробів, контроль якості та керування. Технологія включає в себе методи, прийоми, режими роботи, послідовість операцій та процедур. Вона тісно взаємопов'язана із засобами, що застосовуються, обладнанням, інструментами, використовуваними матеріалами. За методологією ООН технологія в чистому вигляді охоплює методи та техніку виробництва товарів і послуг (dissembled technology, англ.). *Втілена технологія* охоплює машини, обладнання, споруди, виробничі системи та продукцію з високими техніко-економічними параметрами (embodied technology, англ.). *Матеріальна технологія* (МТ) створює матеріальний продукт. *Інформаційна технологія* (ІТ) створює інформаційний продукт на основі інформаційних ресурсів.

Інформаційні технології (ІТ) використовують комп'ютерні та програмні засоби для реалізації процесів відбору, реєстрації, подання, збереження, опрацювання, захисту та передавання інформації – інформаційного ресурсу у формі даних та знань – з метою створення інформаційних продуктів.

Аналітична картина видаватиметься незавершеною, якщо не означити ще одну базову сутність сфери комп'ютингу, якою є *інформаційна система*. Не претендуючи на абсолютну точність пропонованого твердження, розглядатимемо інформаційну систему як множину координат у чотиривимірному просторі інженерій сфери комп'ютингу. Тобто інформаційну систему (ІС) подаємо як певний набір інформаційних технологій, що в комплексі зорієнтовані на досягнення певної системної мети, виконуючи задані функції та пропонуючи при цьому споживачам якісні інформаційні продукти та сервіси.

У свою чергу, для всіх штучних інформаційних систем притаманними є чотири життєвих фази їхнього формування та функціонування. Йдеться про фази системного аналізу, системного проектування, системної інтеграції та системного адміністрування, які генерують відповідні вимоги до професійної підготовки та практичної орієнтації фахівців у царині інформаційних систем. Ринок потребує системних аналітиків, системних проектувальників, системних інтеграторів та системних адміністраторів.

Комплексний виклад структурованого подання галузі “КОМП'ЮТИНГУ” дозволяє, загалом, чіткіше уявити проблематику та тематику підручників, котрі будуть виходити в світ у однойменній освітньо-науковій серії в 50-ти книгах. Для кращого розуміння в майбутньому ще раз наведемо означення сфери “КОМП'ЮТИНГУ” як галузі знань (науки, виробництва, бізнесу та надання послуг), предметом якої є комплексні дослідження, розроблення, впровадження та використання інформаційних систем, складовими елементами яких є інформаційні технології, що реалізовані на основі сучасних

інженерних досягнень комп'ютерної інженерії, інженерії програмного забезпечення, інженерії даних та знань, системної інженерії, котрі базуються на фундаментальних законах та закономірностях інформатики.

Автори підручників і навчальних посібників серії "КОМП'ЮТИНГ" пропонують значний перелік навчальних дисциплін, котрі, з одного боку, включаються до сфери комп'ютиingu за означенням, а, з іншого боку, їх предмет ще не знайшов якісного висвітлення у вітчизняній навчальній літературі для вищої школи. Перший крок ми робимо у 2008-2009 рр., виданням принаймні десяти книг серії з подальшим її п'ятикратним розширенням до 2011 року. Структурно серія подається узагальненими профілями як то:

- фундаментальні проблеми комп'ютиingu;
- комп'ютерні науки;
- комп'ютерна інженерія;
- програмна інженерія;
- інженерія даних та знань;
- системна інженерія;
- інформаційні технології та системи.

При цьому зауважу, що наведені укрупнені профілі серії підручників і навчальних посібників загалом співпадають з профілями бакалавратів, зафіксованих у підсумковому звіті "Computing CURRICULA" редакції 2006 року. Ми розуміємо, що чітка завершена будівля комп'ютиingu з'явиться лише в перспективі, а наша праця буде подаватись як активний труд будівничих з якнайшвидшого втілення в життя проекту цієї, без перебільшення, грандіозної будівлі сучасного інформаційного суспільства. Я запрошую потенційних авторів долучитись до цього освітньо-наукового проекту, а шановних читачів виступити в ролі творчих критиків та опонентів. Буду вдячний за ваші побажання, зауваження та пропозиції.

*З глибокою повагою, науковий редактор серії підручників і навчальних посібників "КОМП'ЮТИНГ", д.т.н., професор Володимир ПАСТУШНИК*

## Вступне слово авторів

Навчальний посібник "Системи штучного інтелекту" пропонуємо для студентів і викладачів вищих навчальних закладів III та IV рівнів акредитації, що готують фахівців галузей знань "Інформатика та обчислювальна техніка" (за базовими напрямками "Комп'ютерні науки", "Програмна інженерія", "Комп'ютерна інженерія") та "Системні науки і кібернетика" (за базовими напрямками "Інформатика", "Прикладна математика", "Системний аналіз"). Його зміст орієнтований на підготовку сучасного висококваліфікованого фахівця в галузі комп'ютерних наук.

У контексті підготовки магістрів за спеціальністю "Системи штучного інтелекту" автори дотримуються такого розуміння цієї галузі: *системи штучного інтелекту – галузь науки та техніки, в якій досліджуються, вивчаються, проектуються та створюються інформаційні, програмно-алгоритмічні й апаратні комплекси, результати дії яких аналогічні до результатів дії механізмів мислення та процесів комунікування людини, їх не можна відрізнити від рішень, які приймаються людиною-професіоналом, а також здійснюється природне комунікування фахівців у заданій предметній області.*

Нині з'явилося багато книг, які розкривають широке коло питань, об'єднаних назвою "Штучний інтелект". При всій широті питань, які висвітлено в них, залишається відкритою проблема систематичного викладення матеріалу в рамках навчальної дисципліни зі спорідненою назвою. Посібник відображає погляд авторів на структуру, зміст та спосіб викладення матеріалу.

Автори не мали на меті обговорювати особливості процесу міркування, притаманного людині, оскільки ці питання достатньо висвітлено в доступній літературі зі штучного інтелекту.

Своєю головною метою автори вбачали об'єднання на спільній ідейній основі технологій, опис яких розпорошений у великій кількості статей, монографій, підручників, наукових звітів. Запропонований підхід до відбору та подання матеріалу викликаний необхідністю вирішення практичних та освітніх задач. У сучасних комп'ютерних технологіях зазвичай застосовують такі математичні поняття та методи, які мало знайомі тим, хто не працює в галузі теоретичної інформатики. Тому автори поставили перед собою завдання написати книгу, яка б, з одного боку, розкривала сутність штучного інтелекту як дисципліни, яка має математичну основу, а з другого – мала спільну для всіх розділів структуру, якої вони й намагались дотримуватись під час викладення матеріалу.

Кожний розділ складається зі змістовної постановки проблеми, її математичної формалізації, побудови методів та алгоритмів її розв'язування, ілюстративних прикладів, завдань для самостійної роботи та комп'ютерних проектів, які можуть стати основою для студентських навчальних проектів та магістерської наукової роботи. Оскільки розв'язування задач штучного інтелекту потребує знання основ та володіння технікою із широкого кола математичних понять, зокрема, з дискретної математики та теорії ймовірностей, автори подали певний обсяг відповідного теоретичного матеріалу.

При відборі матеріалу для книги автори вважали доцільним обговорити декілька типів основних задач.



*Тип задач, яким присвячено перший розділ, – це задачі, що розв’язують шляхом пошуку в просторі станів для знаходження цільового розв’язку серед усіх можливих станів. Розглянуто моделі предметної області та способи організації пошукових процедур. Описано евристичні методи пошуку в просторі станів, а також декомпозицію задач на основі графів AND/OR.*

*У другому розділі розглянуто задачі, що вже стали класикою штучного інтелекту. Вони мають своєю основою класичну логіку та її фундаментальні результати. Розвиток цього напрямку співпав зі становленням обчислювальної техніки та викликаним нею розвитком обчислювальних методів математики, перетворив логіку в обчислювальну дисципліну. Теорія логічного виведення перетворилась на інструмент формальних логічних міркувань для доведення логічних теорем. У своїй основі в цьому типі задач використано дедуктивний підхід до побудови міркувань шляхом автоматичного доведення теорем. Тут позиція авторів полягає в тому, що розуміння основ теорії доведень – неодмінна передумова для логічної побудови передових комп’ютерних технологій. Тому для нас було важливою проблемою викладення відповідного матеріалу – для багатьох читачів запропонований математичний апарат може виявитись складним, незважаючи на наші спроби спростити його. Наскільки ми розуміємо, це неминуче, але на нашу думку включення певних доволі складних математичних понять виправдано практичною важливістю результатів. Цей підхід надав поштовх до створення спеціальних мов логічного програмування. У цьому розділі посібника докладно описано процес автоматичного доведення логічних теорем. Наведено велику кількість прикладів, для яких подано алгоритми розв’язування, зокрема, алгоритми Куайна, Девіса–Паттена, методу резолюцій та його модифікацій як у численні висловлювань, так і в численні предикатів, викладено принцип логічного програмування.*

*Третій розділ присвячено індуктивному підходу в штучному інтелекті; головна його мета – викладення в систематичній формі ідей, методів та алгоритмів машинного навчання (machine learning). Головна ідея побудови алгоритмів для розв’язування таких задач полягає в пошуку прихованих закономірностей у великих базах даних. Ці задачі традиційно об’єднані назвою “машинне навчання” і спрямовані на побудову комп’ютерних програм, які мають властивість “навчатись” у сенсі покращення якості отриманих розв’язків на основі накопичення досвіду. Останніми роками з’явилась нова дисципліна, які використовує ці методи – це інтелектуальний аналіз даних (Data Mining). Мета застосування зазначеної групи методів – побудова на основі даних, отриманих у результаті певних досліджень, наборів правил у формі логічних формул, або числових значень функцій, яким можна надавати змістовну інтерпретацію. У посібнику докладно обговорено низку таких задач та методів їхнього розв’язування. Їх об’єднано в змістовні групи за принципом формування обчислювальної моделі.*

*Розглянуто основні підходи до навчання – навчання з учителем (задачі класифікації) та навчання без учителя (задачі кластеризації). У рамках методів навчання з учителем розглянуто такі підходи: пошук загальних версій на основі навчальних даних із використанням алгоритму виключення кандидата; генерування правил за допомогою дерев рішень та відповідних алгоритмів; докладно розглянуто нейронні мережі, побудовані як багатошаровий перцептрон, і алгоритм зворотного поширення похибки для їхнього навчання.*

*Як дуже поширений метод навчання без учителя розглянуто мережі, що самоорганізуються (самоорганізовані карти Кохонена).*

*У четвертому розділі посібника розглянуто ще один підхід – подання знань та виведення в умовах невизначеності. Розглянуто формулювання задачі статистичного навчання, використання теореми Байєса для перевизначення апіорних імовірностей гіпотез і побудови на її основі байєсівських класифікаторів, зокрема, наївного байєсівського класифікатора. Для визначення причинно-наслідкових зв’язків описано застосування байєсівських мереж.*

*Значну увагу приділено проблемам та методам, які ґрунтуються на нечіткій (розмитій) логіці. Уведено поняття нечіткої множини, розглянуто операції над такими множинами. Описано процедуру нечіткого логічного виведення та застосування цієї теорії до побудови пристроїв керування.*

*Автори висловлюють щире подяку Олеся Васильовичу Годичу за люб’язно надані ним матеріали, які були використані під час роботи над підрозділом 3.15.*

Ю. В. Нікольський  
В. В. Пасічник  
Ю. М. Щербина

## РОЗДІЛ 1



# ПОДАННЯ ЗАДАЧ ТА ПОШУК РОЗВ'ЯЗКІВ

- ◆ Класифікація задач
- ◆ Основні поняття та означення
- ◆ Модель предметної області
- ◆ Процедура розв'язування задачі
- ◆ Приклади розв'язування задач
- ◆ Простір станів. Подання задачі в просторі станів
- ◆ Метод пошуку вшир
- ◆ Метод пошуку вглиб
- ◆ Порівняння методів пошуку вшир і вглиб
- ◆ Пошук вглиб з ітераційним заглибленням
- ◆ Евристичні методи пошуку в просторі станів
- ◆ Метод пошуку по першому найкращому збігу
- ◆ Подання задач у вигляді графів AND/OR

### 1.1. КЛАСИФІКАЦІЯ ЗАДАЧ

Усі задачі, які розв'язує людина, можна розділити на три головних класи.

1. Задачі, для яких відомі схеми розв'язування і які можна описати на формальній мові. Прикладом таких задач є всі задачі, пов'язані з обчисленнями (арифметика, алгебраїчні, диференціальні, інтегральні рівняння та системи тощо). Для таких задач завжди існує алгоритм розв'язування, який обов'язково приведе до шуканого результату. Сюди ж можна долучити задачі, для яких немає готових схем розв'язування, але вони можуть бути знайдені або побудовані, якщо залучити додаткові знання про дану предметну область (тобто виділену вузьку сферу діяльності).

Саме для задач першого класу найбільше були пристосовані перші комп'ютери фон-нейманівської архітектури. Загальну схему їхнього розв'язування в найпростішому вигляді зображено на рис. 1.1.

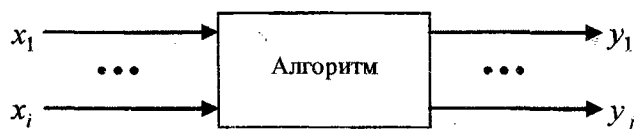


Рис. 1.1

На вході алгоритму – значення вхідних змінних, на виході – значення вихідних змінних. Усі обчислення в алгоритмі ґрунтуються на чотирьох арифметичних діях,

## Розділ 1. Подання задач та пошук розв'язків

якими б складними вони не були. Обчислювальний характер дій та чіткий алгоритм на багато років визначили шлях розвитку комп'ютерів і всіх систем автоматизації. Були створені відповідні мови – ФОРТРАН, АЛГОЛ, ПАСКАЛЬ та інші. Лише після довготривалих коливань задачі цього класу перестали відносити до класу інтелектуальних.

2. Іще один клас утворюють задачі, методологія розв'язування яких визначила цілу епоху в еволюції комп'ютерів. Це задачі інформаційного пошуку. На рис. 1.2 зображено загальну схему пошуку розв'язків для задач із цього класу.

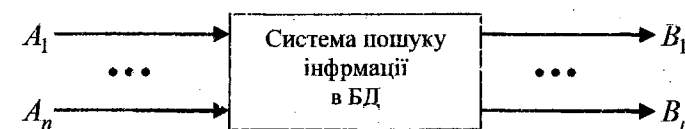


Рис. 1.2

На вході системи – запити абонентів  $A_1, A_2, \dots, A_n$ . Розв'язком задачі є деяка частина  $B_1, B_2, \dots, B_m$  бази даних, яка відповідає (є релевантною) запитам. При цьому інформація, яка видається як результат, не піддається яким-небудь змінам. Вона проходить лише операції сортування й деякі інші теоретико-множинні операції над символічною інформацією (тобто інформацією, поданою як у цифровій, так і в буквеній формі).

Перехід до символічних задач дав змогу вперше усвідомити можливості розділення інформації й механізмів її опрацювання, тобто відокремити самі бази даних від алгоритмів їхнього аналізу й одержання результатів. Це стало можливим завдяки розробці синтаксичних схем організації записів у базах даних і орієнтації алгоритмів тільки на синтаксичне опрацювання. Другою важливою перемогою цього напрямку стало створення достатньо універсальних схем керування базами даних із найширшою сферою практичного застосування.

3. Є задачі, які характеризуються відсутністю апріорі відомих схем розв'язування, навіть за умови залучення додаткових знань про предметну область. Тому для розв'язування таких задач використовують складні ієрархічні побудови й програми, які імітують механізми мислення людини. До цього класу належать, наприклад, задачі планування поведінки об'єкта в складних середовищах, задачі прийняття рішень у нестандартних (позаштатних) ситуаціях, задачі проектування й конструювання, ігри тощо. Цей клас задач вважають інтелектуальним, і саме для їхнього розв'язання розроблялася теорія систем штучного інтелекту. Головна задача програмування штучного інтелекту – сформулювати подання та структури керування, необхідні для розв'язування інтелектуальної задачі. Для розв'язування задач штучного інтелекту найчастіше використовують мови LISP і PROLOG.

### 1.2. ОСНОВНІ ПОНЯТТЯ ТА ОЗНАЧЕННЯ

Будь-яка наука відрізняється від інших предметом свого дослідження. Залежно від предмету формулюються цілі й задачі, а також методи та засоби їхнього розв'язування. Якщо говорити про таку науку, як штучний інтелект, то в самій її назві виділено об'єкт дослідження – інтелект, і основна її проблема – створення деякої подоби

„живого” інтелекту штучними засобами. Штучні засоби відомі були одразу – це комп’ютери. Сюди входять різні периферійні пристрої, за допомогою яких можуть створюватись штучні аналоги того, що може відноситись до класу інтелектуального. Але що зарахувати до класу інтелектуального, тобто які процеси мислення людини можна вважати інтелектуальними? Нині в клас інтелектуальних зараховують всі задачі, алгоритм розв’язування яких заздалегідь невідомий. Такими задачами є розпізнавання образів, машинний переклад, прийняття рішень у ситуаціях, які змінюються, тощо. Усе це стимулювало нові наукові дослідження, метою яких було визначення особливості інтелектуальної діяльності людини. Було показано, що в основі інтелектуальної поведінки людини є ряд метапроцедур, дія яких не залежить від конкретних задач і областей діяльності.

У першу чергу, це метапроцедури цілеспрямованого пошуку, подібні до пошуку в лабіринті можливостей, асоціативний пошук і асоціативне міркування тощо.

Важливим кроком у сучасному розвитку штучного інтелекту стало розуміння необхідності внутрішнього зображення проблемної ситуації. Це призвело до виділення метапроцедур, які оперують із сукупністю знань з предметної області, до якої належить задача. Це так звана модельна гіпотеза. Головні метапроцедури тут такі: подання (зображення) знань, їх поповнення й модифікація, міркування, пошук релевантної (відповідної) інформації в сукупності наявних знань тощо.

Усі перераховані метапроцедури в сукупності зі знаннями про предметну область зрештою утворили ядро штучного інтелекту. Означимо головні поняття, які будемо використовувати.

*Предметна область* (ПО) – виділена вузька сфера діяльності людини, яка відноситься до даної задачі.

*Модель предметної області* (МПО) – сукупність (система) знань, необхідних для автоматичного синтезу алгоритму розв’язування задачі в даній області.

*Штучний інтелект* (ШІ) (Artificial Intelligence – AI) – сукупність метапроцедур (подання знань, міркувань, пошуку релевантної інформації в середовищі наявних знань, їхнє поповнення, корекція тощо), які імітують діяльність людини.

*Система штучного інтелекту* (СШІ) – апаратний та інформаційно-програмний комплекс, дія якого аналогічна до дії механізмів мислення людини, і цю дію не можна відрізнити від рішень, які приймалися б людиною-експертом, тобто професіоналом у предметній області.

Із цих означень випливає, що будь-яку систему ШІ відрізняють такі властивості:

- ◆ наявність моделі предметної області;
- ◆ наявність моделей механізмів мислення, тобто метапроцедур, які працюють на системі знань, поданих моделлю предметної області (зокрема, процедур логічного виведення);
- ◆ наявність інтерфейсу на природній мові, який забезпечує взаємодію користувача з СШІ.

### 1.3. МОДЕЛЬ ПРЕДМЕТНОЇ ОБЛАСТІ

Усе, що нам потрібно знати для розв’язування задачі, утворює деяку предметну

область (ПО), яку ми формалізуємо у вигляді моделі предметної області [2].

*Модель предметної області* (МПО) – це четвірка  $(X, C, R, F)$ , де використано такі позначення.

$X = \{x_1, x_2, \dots, x_n\}$  – множина імен об’єктів (предметів, сутностей) зовнішнього світу.

$C = \{c_1, c_2, \dots, c_m\}$  – множина імен властивостей об’єктів, причому  $c_1 = \{c_{11}, c_{12}, \dots, c_{1r}\}$ ,  $c_2 = \{c_{21}, c_{22}, \dots, c_{2s}\}$ , ... – множини імен значень властивостей об’єктів. Якщо, наприклад,  $X$  – множина загальних властивостей (сутностей), що описують яблука як фрукт (загальний вигляд, ареал вирощування, час дозрівання, харчова цінність тощо), то множина  $C$  буде подавати сорти яблук, а множини  $c_i$  – відповідно властивості кожного сорту.

$R = \{R_1, R_2, \dots, R_n\}$  – множина імен предикатів, які описують можливі відношення між об’єктами ПО, яка моделюється. У нашому прикладі це, очевидно, перехід прилості від яблука до яблука, зіткнення при транспортуванні тощо.

$F = \{f_1, f_2, \dots, f_k\}$  – множина імен операторів (дій), які допускаються з цими об’єктами через зміну їхніх властивостей і відношень між ними. Стосовно прикладу з яблуками, то така множина дій може містити збір яблук, їхнє сортування тощо.

Уведені множини  $X, C, R, F$  дають змогу подати предметну область у вигляді деякого мовного еквівалента, ступінь адекватності якого до реального світу залежить від багатьох факторів. З іншого боку, стан ПО загалом не сталий, тому правильніше говорити про стан предметної області в даний момент. Означимо стан  $S_{ПО}(t)$  у момент часу  $t$  як трійку  $(X(t), C(t), R(t))$ . Ця трійка описує ситуацію, яка склалась у ПО в даний момент часу. Залежно від цієї ситуації в момент  $t \in (поч, кінь)$  людина (або робот) вибирає з множини  $F$  можливих операторів саме ті дії, які потрібні для розв’язування задачі. Якщо тепер позначити через  $\Phi$  відображення  $S_{ПО}(t)$  на множину  $F$ , то можна записати

$$\Phi: (X(t), C(t), R(t)) \rightarrow F.$$

Іншими словами, сама предметна область виступає тут як пасивний елемент дії (об’єкт перетворення), а людина або система, яка її замінює, виступає як перетворювач (суб’єкт дії). Отже, відображається зв’язок між мовним описом ПО, її станом (декларативна компонента), з одного боку, та діями, які виконуються в процесі розв’язування знання (процедурна компонента), з іншого.

Інв. №

7 8 3 8 7 0

### 1.4. ПРОЦЕДУРА РОЗВ’ЯЗУВАННЯ ЗАДАЧІ

Означимо (на інтуїтивному рівні) поняття „задача”, „розв’язок”, „алгоритм”.

Позначимо початковий стан ПО через  $S_{поч}$ . *Задача* полягає в тому, щоб перевести



предметну область зі стану  $S_{поч}$  у деякий заданий стан, який ми визначимо як цільовий ( $S_{ціль}$ ). Очевидно, виконати це можливо, лише застосовуючи допустимі в даній предметній області оператори з множини  $F = \{f_1, f_2, \dots, f_k\}$ . Які оператори  $f_i$  вибрати й у якій послідовності – невідомо. У цьому якраз і полягає розв'язування задачі. Отже схему розв'язування можна виразити так

$$S_{поч} \xrightarrow{F} S_{ціль}.$$

Нехай оператором  $f_1$  з множини  $G$  ми перевели  $S_{поч}$  в  $S_1$ , але цей стан виявився відмінним від  $S_{ціль}$ , тобто  $S_1 = f_1(S_{поч})$ ,  $S_1 \neq S_{ціль}$ . Для переводу  $S_1$  у стан  $S_2$  застосуємо оператор  $f_2 \in F$ :  $S_2 = f_2(S_1)$ . Якщо  $S_2$  знову не співпадає з  $S_{ціль}$ , то звернемось до оператора  $f_3$ , щоб одержати стан  $S_3$  і оцінити його, порівнявши з  $S_{ціль}$ . І так далі, доки не знайдеться такий оператор  $f_j$ , що  $S_j = f_j(S_{j-1})$ ;  $S_j = S_{ціль}$ . Описаний шлях пошуку розв'язку можна зобразити так:

$$S_{поч} \xrightarrow{f_1} S_1 \xrightarrow{f_2} S_2 \xrightarrow{f_3} \dots \xrightarrow{f_{j-1}} S_{j-1} \xrightarrow{f_j} S_{ціль}.$$

Це саме можна записати інакше:

$$S_{ціль} = f_j(f_{j-1}(\dots f_3(f_2(f_1(S_{поч}))))).$$

Послідовність операторів  $f_1, f_2, f_3, \dots, f_j$  подає алгоритм розв'язування задачі.

Зазначимо, що перехід  $S_{поч} \rightarrow S_{ціль}$  можливий не одним способом. У такому разі можна формулювати задачу про оптимізацію процесу розв'язування. Розглянемо приклади побудови предметної області та розв'язування задачі.

## 1.5. ПРИКЛАДИ РОЗВ'ЯЗУВАННЯ ЗАДАЧ

**Приклад 1.1.** Розглянемо класичну задачу про мавпу й банани [2]. У кімнаті, де знаходиться мавпа, є ящик і в'язка бананів, причому банани підвішені настільки високо, що мавпа до них не може дотягнутися з підлоги. Яку послідовність дій має виконати мавпа, щоб дістати ці банани? (Передбачається, що вона повинна здогадатись підтягнути ящик до бананів, стати на нього й дотягнутись до них).

Подамо модель предметної області.

$X$  – імена предметів і об'єктів МПО: *МАВПА* (далі  $M$ ), *ЯЩИК* ( $Я$ ), *БАНАНИ* ( $Б$ ).

$C$  – координати  $M, Я, Б$  у кімнаті – по горизонталі й вертикалі. Значення координат – довільна множина точок кімнати:  $a, b, c, \dots$  (припустимо, що ця множина скінченна).

$R$  – імена предикатів, які описують відношення між об'єктами: *ПОРУЧ*( $M, Я$ ), тобто мавпа знаходиться біля ящика; *НА*( $M, Я$ ) – мавпа на ящику, нарешті, *В*( $M, Б$ ) – банани в руках мавпи. Очевидно, що можливими є заперечення цих предикатів

*НЕ-ПОРУЧ*( $M, Я$ ), *НЕ-НА*( $M, Я$ ), *НЕ-В*( $M, Б$ ) або в інших позначеннях  $\neg$ *ПО-РУЧ*( $M, Я$ ),  $\neg$ *НА*( $M, Я$ ),  $\neg$ *В*( $M, Б$ ).

$F$  – множина імен операторів (дій), які може виконати мавпа:

*ПІДІЙТИ*( $M, Я$ ) =  $f_1$  – мавпі підійти до ящика;

*ПЕРЕСУНУТИ*( $M, Я$ ) =  $f_2$  – мавпі пересунути ящик;

*ПІДНЯТИСЬ*( $M, Я$ ) =  $f_3$  – мавпі піднятися на ящик;

*СХОПИТИ*( $M, Б$ ) =  $f_4$  – мавпі схопити банани;

*ПЕРЕЙТИ*( $M, d$ ) =  $f_5$  – мавпі перейти в точку  $d$  площини підлоги (далі таку точку називатимемо *двовимірною координатою*).

Усі дії мають як аргументи об'єкти МПО. Оператор  $f_5$  відображає можливі переміщення мавпи в кімнаті без ящика. Поки він нам не придасться, але надалі буде корисний.

Зараз необхідно побудувати відображення  $\varphi: S_{МПО}(t) \rightarrow F$ , тобто відображення множини станів предметної області на множину дій. З цією метою дамо опис початкового та цільового станів у МПО. Припустимо, що в  $S_{поч}$  двовимірною координатою  $M$  (мавпи) є точка  $a$ , двовимірною координатою  $Я$  (ящика) – точка  $b$ , двовимірною координатою  $Б$  (в'язки бананів) – точка  $c$ , яка розташована безпосередньо під бананами. Тоді початковий стан буде зручно записати у вигляді

$$S_{поч} = (a, b, c, \neg НА(M, Я), \neg В(M, Б)),$$

усі об'єкти в початкових позиціях, мавпа не на ящику, банани не в руках мавпи. Кожний з предикатів *НА* та *В* може приймати лише два значення Т (від True, що позначатимемо 1) і F (від False, що позначатимемо 0): мавпа або на ящику (1), або ні (0), в руках у неї або є банани (1), або ні (0). Скориставшись цим, ми можемо спростити запис станів:

$$S_{поч} = (a, b, c, 0, 0),$$

$$S_{ціль} = (c, c, c, 1, 1),$$

– усі об'єкти в одній точці підлоги кімнати, мавпа на ящику й вхопила в'язку бананів.

Отже, задача зводиться до того, щоб за допомогою введених вище операторів  $f_1, f_2, f_3, f_4$  зі стану  $S_{поч} = (a, b, c, 0, 0)$  перейти в стан  $S_{ціль} = (c, c, c, 1, 1)$ . Застосуємо до  $S_{поч}$  оператор  $f_1 = \text{ПІДІЙТИ}(M, Я)$ , тоді одержимо

$$S_{поч} = (a, b, c, 0, 0) \xrightarrow{f_1} (b, b, c, 0, 0) = S_1,$$

(мавпа підійшла до ящика і їхні координати співпали).

Далі застосуємо таку послідовність операторів:

$$f_2 = \text{ПЕРЕСУНУТИ}(M, Я)$$

$$S_1 = (b, b, c, 0, 0) \xrightarrow{f_2} (c, c, c, 0, 0) = S_2;$$

$$f_3 = \text{ПІДНЯТИСЬ}(M, Я)$$

$$S_2 = (c, c, c, 0, 0) \xrightarrow{f_3} (c, c, c, 1, 0) = S_3;$$

$$f_4 = \text{СХОПИТИ}(M, Б)$$

$$S_3 = (c, c, c, 1, 0) \xrightarrow{f_4} (c, c, c, 1, 1) = S_{ціль}.$$

Повністю процес можна описати формулою

$$S_{\text{ціль}} = f_4(f_3(f_2(f_1(a, b, c, 0, 0))))).$$

Очевидно, що цю модель можна як завгодно ускладнювати: ящиків може бути декілька, у кімнаті можуть бути інші предмети тощо, але зміст перетворень не зміниться.

#### Приклад 1.2. Наповнення цебра водою.

У початковий момент часу порожнє цебро стоїть поруч із раковиною, кран закритий. У цільовій ситуації необхідно, щоб цебро було повне, і стояло на підлозі біля раковини, а кран був закритий. Усі дії виконує робот. Потрібно побудувати програму його роботи.

Множина  $X = \{\text{РОБОТ} (P), \text{ЦЕБРО} (Ц), \text{КРАН} (К), \text{ПІДЛОГА} (П), \text{РАКОВИНА} (РК)\}$ .

Множина  $C = \{\text{стан РОБОТА}, \text{стан ЦЕБРА}, \text{стан КРАНА}\}$ ,

стан РОБОТА = {БІЛЯ КРАНА},

стан ЦЕБРА = {ПОРОЖНЄ, ПОВНЕ},

стан КРАНА = {ВІДКРИТИЙ, ЗАКРИТИЙ}.

Множина  $R$ :  $НА (Ц, П)$  – цебро на підлозі,

$В (Ц, РК)$  – цебро в раковині.

Якщо прийняти, що  $В(Ц, РК) = \neg НА (Ц, П)$ , то достатньо лише одного предиката  $НА (Ц, П)$ .

Множина  $F = \{f_1, f_2, f_3, f_4, f_5\}$ , де позначено:

$f_1$  – поставити цебро в раковину;

$f_2$  – відкрити кран;

$f_3$  – наповнити цебро;

$f_4$  – зняти цебро;

$f_5$  – закрити кран.

Оскільки робот увесь час перебуває в одній точці, його можна не розглядати. Тоді

$S_{\text{поч}} = (Ц(\text{ПОРОЖНЄ}), К(\text{ЗАКРИТИЙ}), НА(Ц, П)),$

$S_{\text{ціль}} = (Ц(\text{ПОВНЕ}), К(\text{ЗАКРИТИЙ}), НА(Ц, П)).$

Розглянемо стани й дії.

1.  $(Ц(\text{ПОРОЖНЄ}), К(\text{ЗАКРИТИЙ}), НА(Ц, П)) \xrightarrow{f_1} (Ц(\text{ПОРОЖНЄ}), К(\text{ЗАКРИТИЙ}), \neg НА(Ц, П)).$

2.  $(Ц(\text{ПОРОЖНЄ}), К(\text{ЗАКРИТИЙ}), \neg НА(Ц, П)) \xrightarrow{f_2} (Ц(\text{ПОРОЖНЄ}), К(\text{ВІДКРИТИЙ}), \neg НА(Ц, П)).$

3.  $(Ц(\text{ПОРОЖНЄ}), К(\text{ВІДКРИТИЙ}), \neg НА(Ц, П)) \xrightarrow{f_3} (Ц(\text{ПОВНЕ}), К(\text{ВІДКРИТИЙ}), \neg НА(Ц, П)).$

4.  $(Ц(\text{ПОВНЕ}), К(\text{ВІДКРИТИЙ}), \neg НА(Ц, П)) \xrightarrow{f_4} (Ц(\text{ПОВНЕ}), К(\text{ЗАКРИТИЙ}), \neg НА(Ц, П)).$

5.  $(Ц(\text{ПОВНЕ}), К(\text{ЗАКРИТИЙ}), \neg НА(Ц, П)) \xrightarrow{f_5} (Ц(\text{ПОВНЕ}), К(\text{ЗАКРИТИЙ}), НА(Ц, П)).$

Останній стан – цільовий.

Зазначимо, що відображення  $\Phi$  описує умови застосування операторів із множини  $F$ . Із прикладів можна дійти висновку, наскільки важливо правильно (вдало) вибрати

спосіб опису  $S_{\text{поч}}$  та  $S_{\text{ціль}}$ . Від цього залежить наочність подання й навіть швидкість пошуку розв'язку.

**Приклад 1.3.** Початковим станом є порожня гральна дошка  $3 \times 3$ . Цільовий стан – дошка, на якій в одному рядку, стовпчику або діагоналі міститься три  $\times$  (передбачається, що метою є перемога гравця  $\times$ ).

Станами в просторі є всі можливі конфігурації із хрестиків і нуликів, які можуть виникнути під час гри. Зазвичай, більшість із можливих  $3^9$  варіантів розміщення символів (пусто,  $\times$ ,  $0$ ) у дев'яти клітках взагалі ніколи не виникне в реальній грі. Елементи множини  $F$  тут визначаються допустимими ходами в грі.

## 1.6. ПРОСТІР СТАНІВ. ПОДАННЯ ЗАДАЧІ В ПРОСТОРІ СТАНІВ

У підрозділі 1.3 було означено поняття „стан МПО” і відображення  $\Phi$  між множиною станів і множиною тих дій, які переводять предметну область з одного стану в інший. Очевидно, що при цьому необхідно вказувати лише ті дії (оператори), котрі можуть змінити стан МПО. Наприклад, в задачі про мавпу та банани дію  $f_3$  (ПІДНЯТИСЬ) можна застосувати лише до стану  $(c, c, c, 0, 0)$ , а в результаті одержимо новий стан  $(c, c, c, 1, 0)$  як наслідок виконання цієї дії. Так само, використовуючи оператори, можна знайти й решту станів. Застосування операторів до початкового стану, а після цього і до всіх проміжних станів, породжує всю множину можливих станів предметної області – цю множину, подану (зображену) певним способом, називають простором станів. Для того, щоб сформулювати строге означення простору станів, наведемо деякі означення, пов'язані з графами.

◆ **Орієнтований граф** (у подальшому – **граф**) складається з множини **вершин**  $N_1, N_2, \dots, N_n$  і множини дуг, які з'єднують деякі пари вершин.

◆ **Дуга** – це впорядкована пара вершин, наприклад, дуга  $(N_j, N_k)$  з'єднує вершину  $N_j$  з вершиною  $N_k$ , але не вершину  $N_k$  з вершиною  $N_j$ , якщо пара  $(N_k, N_j)$  – не дуга. Це означає, що по дузі  $(N_j, N_k)$  можна рухатись лише від вершини  $N_j$  до вершини  $N_k$ .

◆ Якщо дуга з'єднує вершину  $N_j$  з вершиною  $N_k$ , то  $N_j$  називають **батьком**  $N_k$ , а  $N_k$  – **сином**  $N_j$ . Якщо граф має також дугу  $(N_k, N_j)$ , то  $N_k$  та  $N_j$  називають **вершинами-братями**.

◆ **Кореневий граф** має єдину вершину  $N_r$ , із якої існує шлях до будь-якої вершини графа. Цю вершину називають **коренем**. У корінь не входить жодна дуга, і, отже, він не має батька.

◆ **Кінцева вершина** – це вершина, яка не має синів.

◆ Упорядковану послідовність вершин  $N_{i_1}, N_{i_2}, \dots, N_{i_m}$ , у якій кожна пара  $(N_{i_j}, N_{i_{j+1}})$  являє собою дугу, називають **шляхом довжиною  $m-1$**  на графі.

◆ У кореновому графі вершину називають **предком** усіх вершин, які розташовані на кожному шляху після неї, і водночас **нащадком** усіх вершин, які розташовані на шляху до неї.

◆ Якщо шлях містить якусь вершину більше одного разу (тобто в наведеному вище означенні шляху деяка вершина повторюється), то говорять, що шлях містить **цикл**.

◆ Дві вершини називають **зв'язаними**, якщо існує шлях, який з'єднує ці вершини.

◆ **Дерево** – це граф, у якому існує єдиний шлях між будь-якими двома вершинами.

(Отже, шляхи в дереві не містять циклів.)

♦ У *кореновому дереві* ребра орієнтовані від кореня. Кожна вершина в кореновому дереві (крім кореня) має єдиного батька. Кінцеву вершину у кореновому дереві називають *листом*.

♦ *Рівень вершини* у кореновому дереві – це довжина шляху від кореня до цієї вершини; рівень кореня вважають рівним нулю.

Простір станів задачі задають за допомогою кореневого графа або кореневого дерева. У просторі станів вершини графа відповідають станам розв'язків часткових задач, а дуги – етапам розв'язування задачі. Один або декілька початкових станів, які відповідають початковій інформації сформульованої задачі, утворюють корінь графа. Граф також має одну або декілька цільових вершин, які відповідають розв'язку задачі.

У *позначеному графі* для кожної вершини задають один або декілька дескрипторів (міток), які дають змогу відрізнити одну вершину графа від іншої. На графі простору станів ці дескриптори позначають стани в процесі розв'язування задачі. Якщо дескриптори двох вершин співпадають, то ці вершини вважають однаковими (їх ототожнюють). Дуга між двома вершинами визначається мітками цих вершин.

Дуги графа також можуть бути позначеними. Мітки дуги використовують для того, щоб задати ім'я оператора, який зумовлює перехід у відповідний стан, або вагу дуги, як у задачі комівояжера (див. підрозділ 1.11).

Для кореневих дерев і графів відношення між вершинами описують поняттями предка, нащадка, батька, сина й вершин-братів. Ці поняття використовують у звичайному смислі наслідування: у разі проходження дуги предок передує потомку, а вершини-брати мають спільну вершину-батька.

Пошук у просторі станів характеризує розв'язування задачі як процес пошуку шляху розв'язку від початкового стану до цільового.

Означимо подання задачі в просторі станів формально.

*Простір станів* подається четвіркою  $(N, A, S, G)$  з такими компонентами:

- ♦  $N$  – множина вершин графа або станів у процесі розв'язування задачі;
- ♦  $A$  – множина дуг між вершинами, які відповідають крокам у процесі розв'язування задачі;
- ♦  $S$  – непорожня множина початкових станів задачі;
- ♦  $G$  – непорожня підмножина  $N$ , яка складається із цільових станів.

Стани з множини  $G$ , тобто цільові стани, описують одним із таких способів:

- 1) властивостями станів, які зустрічаються в процесі пошуку;
- 2) властивостями шляхів, які зустрічаються в процесі пошуку, наприклад, вартістю переміщення вздовж дуг шляху.

Ціль може описувати стан, наприклад, ситуацію, коли мавпа схопила банани (задача про мавпу й банани), або повне цебро з водою стоїть біля раковини, а кран закритий (задача про наповнення цебра водою). Інший приклад – стан, який описує виграну ситуацію в грі „хрестики-нулики”.

З іншого боку, ціль може описувати деякі властивості допустимих шляхів. У задачі комівояжера пошук закінчується в разі знаходження найкоротшого шляху через усі вершини графа без їх повторень і повернення в початкову вершину. У задачі граматичного аналізу пошук завершується знаходженням шляху успішного аналізу речення.

Отже, вершини графа простору станів відповідають станам задачі, а дуги – операторам. Дуги подають кроки процесу розв'язування, а шляхи – розв'язки на різних стадіях завершення. Шлях у просторі станів є метою пошуку. Він починається з початкового стану, і продовжується доти, доки не буде досягнуто умову мети. Породження нових станів уздовж шляху забезпечується такими операторами, як можливі дії мавпи (задача про мавпу й банани), цілеспрямовані дії робота (задача про наповнення цебра водою), допустимі ходи (гра „хрестики-нулики” або інша), правила виведення в логічній задачі або експертній системі.

Завдання алгоритму пошуку полягає в знаходженні шляху в просторі станів. Алгоритми пошуку повинні спрямовувати шляхи від початкової вершини до цільової, оскільки саме вони містять ланцюжки операторів, які призводять до розв'язку задачі.

Одна із загальних властивостей графа й одна з проблем, що виникають під час побудови алгоритму пошуку на графі, полягає в тому, що стани іноді можуть бути досягнуті різними шляхами. Тому важливо вибрати *оптимальний* шлях розв'язування даної задачі. Крім того, множина шляхів до цільового стану може призвести до утворення циклів. Тоді алгоритм ніколи не досягне мети. Зазначимо, що граф може не бути деревом, але, водночас, він може бути таким, що не має циклів. Такий граф називають *ациклічним* орієнтованим графом. Ациклічні графи часто зустрічаються в пошуку в просторі станів.

Якщо простір пошуку описується деревом або ациклічним графом, то проблема зациклення не виникає. Саме тому важливо відрізнити задачі пошуку на деревах і ациклічних графах від задач пошуку на графах з циклами. Алгоритми пошуку на довільних графах повинні виявляти та усувати цикли в шляхах. При цьому алгоритми пошуку на деревах і ациклічних графах виграють в ефективності за рахунок відсутності цих перевірок і витрат на них.

Задачі про мавпу й банани та про наповнення цебра водою можна використовувати для ілюстрації пошуку в просторі станів. Обидва ці приклади пояснюють смисл умови 1 в означенні простору станів. У задачі про комівояжера ціль описують умовою типу 2.

**Приклад 1.4.** Фрагмент простору станів для задачі про мавпу й банани (див. приклад 1.1) зображено на рис. 1.3. Цей граф, очевидно, є деревом.

Для повноти викладення тут використано оператор  $f_5$ , який відображає переміщення мавпи з точки в точку. Корінь дерева відповідає стану  $S_{\text{поч}} = (a, b, c, 0, 0)$ . Точки  $d, g, k, n, m$  – двовимірні координати можливої міграції мавпи по кімнаті. Таких точок, звичайно, багато, але ми вибрали декілька для прикладу. Вони не ведуть до розв'язку, але теоретично цілком допустимі. Шляху до цільової вершини  $S_{\text{ціль}} = (c, c, c, 1, 1)$  відповідає послідовність операторів  $f_4, f_3, f_2, f_1$ . Отже, алгоритм розв'язування можна зобразити формулою

$$S_{\text{ціль}} = f_4(f_3(f_2(f_1(a, b, c, 0, 0)))).$$



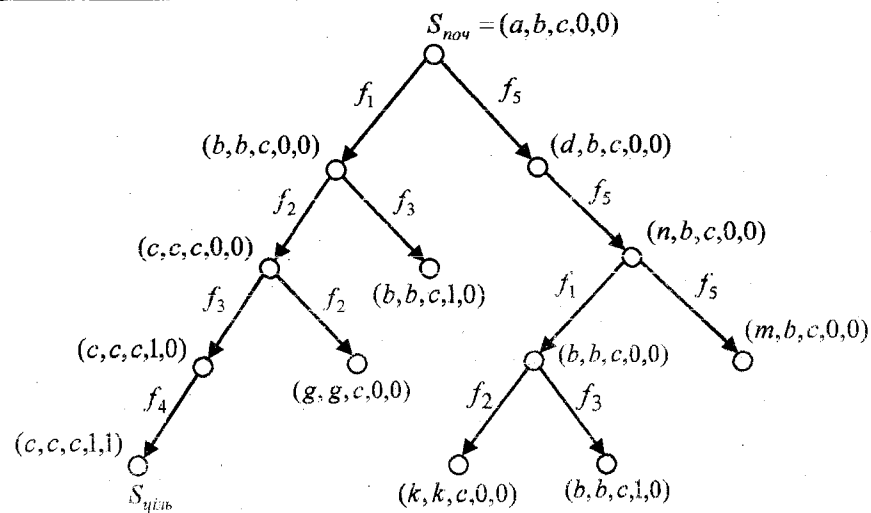


Рис. 1.3

**Приклад 1.5.** Простір станів гри хрестика-нулики подає граф, вершини якого відповідають різним варіантам заповнення гральної дошки, а дуги визначено допустимими ходами гри, які полягають у почерговому записуванні  $\times$  або  $0$  у порожню клітку. Корінь відповідає порожній дошці, що показує початок гри. Граф простору станів гри хрестика-нулики не є деревом, оскільки деякі стани третього та глибших рівнів можна досягти різними шляхами. Проте в цьому графі немає циклів, бо орієнтації дуг не дають змоги повернутись у вже пройдений стан. Звідси випливає, що граф простору станів гри хрестика-нулики є кореневим ациклічним графом. Отже, немає необхідності перевірки на цикл.

Як побудувати простір станів і знайти шлях до цільової вершини? Якщо ототожнити стан  $S_{пoc}$  з коренем графа, то, застосувавши до  $S_{пoc}$  якійсь оператор  $f_i \in F$  ми породжуємо новий стан  $S_p$ , утворюючи тим самим нову вершину графа. Ця вершина може бути проміжною або цільовою. Якщо вершина проміжна, то процес породження нових вершин за допомогою операторів із множини  $F$  буде продовжено, доки не буде знайдено цільову вершину. Процес застосування операторів з множини  $F$  до деякої вершини з метою одержання всіх її синів, називають *розгортанням вершини*. Загалом процедура побудови простору станів при цьому виглядає так.

1. До кореня  $S_{пoc}$  застосовують оператори  $f_i$  із множини  $F$  (їх може бути декілька). Одержані при цьому вершини утворюють перший рівень нових вершин.
2. Кожну з одержаних нових вершин перевіряють, чи не є вона цільовою. Якщо ні, то процес продовжують по відношенню до кожної з них. Утворюють другий рівень вершин. Якщо до якоїсь вершини жоден оператор із  $F$  незастосовний, то ця вершина стає термінальною (кінцевою). Як бачимо, на кожному кроці виконують дві операції: породження нової вершини та перевірку, чи не є нова вершина цільовою, тобто такою, що відповідає цільовому стану.
3. Коли цільову вершину знайдено, то в зворотному напрямку (від цілі до початку)

виділяють шлях розв'язку (для цього потрібно зберігати додаткову інформацію, див. 1. 7). Шлях можна відображати за допомогою операторів, якими позначено дуги.

Загалом кількість вершин може бути великою. Їхнє послідовне розгортання, аналіз і накопичення інформації для виділення шляху розв'язку ускладнюють задачу. Виникає проблема перебору вершин: у якому порядку вони будуть породжуватись і аналізуватись. Алгоритм пошуку має визначити порядок породження та дослідження вершини кореневого дерева або кореневого графа. У наступних двох підрозділах розглянуто два основних варіанти послідовності обходу вершин графа: пошук вглиб (depth first search) і пошук вшир (breadth first search).

## 1.7. МЕТОД ПОШУКУ ВШИР

Розглянемо граф на рис. 1.4. Стани в ньому позначено буквами,  $A, B, C, \dots$ , щоб на них можна було посилатись у подальших міркуваннях. Пошук вшир досліджує простір станів за рівнями, один за одним. І лише коли станів на даному рівні більше немає, алгоритм переходить до наступного рівня. Пошук вшир досліджує вершини графа на рис. 1.5 у такому порядку:  $A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U$ .

Пошук вшир виконується з використанням списків OPEN і CLOSED, які дають змогу відслідковувати просування в просторі станів. Список OPEN містить вершини, які підлягають розгортанню, а список CLOSED – вершини, які вже були розгорнуті.

### Алгоритм пошуку вшир у кореновому графі

#### Ініціалізація

Крок 1. OPEN :=  $S_{пoc}$

Крок 2. CLOSED :=  $\emptyset$

#### Ітерація

Крок 3. Вилучити крайню зліва вершину із OPEN, нехай це буде вершина  $X$

Крок 4. Якщо  $X = S_{циль}$ , то Вихід(УСПІХ); інакше перейти до кроку 5

Крок 5. Розгорнути вершину  $X$  (тобто згенерувати всіх її синів)

Крок 6. Помістити вершину  $X$  у список CLOSED

Крок 7. Виключити синів вершини  $X$ , які є в списку OPEN або CLOSED (це перевірка на цикл)

Крок 8. Решту синів помістити в правий кінець списку OPEN (отже, список OPEN – це черга)

#### Перевірка закінчення

Крок 9. Якщо OPEN  $\neq \emptyset$ , то перейти до кроку 3, інакше Вихід(НЕВДАЧА)

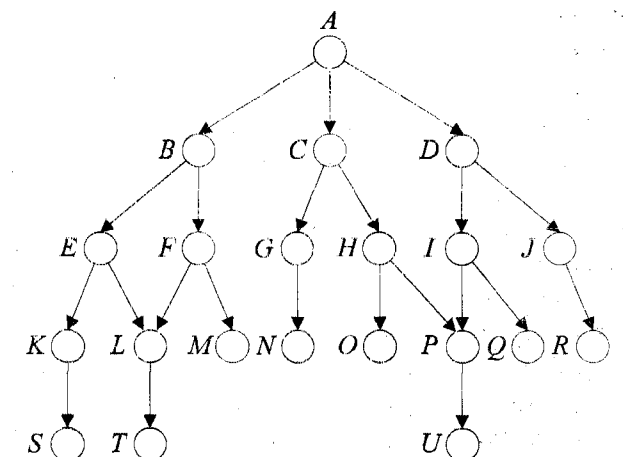


Рис. 1.4

Синівські вершини генерують за операторами з множини  $F$ , допустимими ходами гри, правилами виведення або іншими операціями переходу станів. На кожній ітерації генерують усі вершини, які є синами вершини  $X$  (розгортають  $X$ ). Саму вершину  $X$  записують у список CLOSED, а всіх її синів – у список OPEN, причому синів, які вже є в списках OPEN або CLOSED, повторно не записують. Вказівники дають змогу в разі успіху легко відновити шлях від початкової вершини до цільової. Зазначимо, що список OPEN працює як черга й опрацьовує дані в порядку їхнього надходження (або „першим надійшов – першим обслужений”). Це структура даних FIFO (first in – first out). Стани добавляють у список OPEN справа, а вилучають зліва. Отже, у пошуку беруть участь стани, які перебувають у списку OPEN найдовше – це й забезпечує пошук вшир. Якщо алгоритм завершається через невиконання умови  $OPEN \neq \emptyset$ , то можна зробити висновок, що весь граф досліджено, а бажану мету досягти не вдалося. Отже, пошук зазнав невдачі.

**Приклад 1.6.** Прослідкуємо роботу алгоритму пошуку вшир на графі, зображеному на рис. 1.4. Числа 2, 3, 4, ... означають номер ітерації, і нехай  $U$  – цільовий стан.

1.  $OPEN = [A]$ ;  $CLOSED = []$ .
2.  $OPEN = [B, C, D]$ ;  $CLOSED = [A]$ .
3.  $OPEN = [C, D, E, F]$ ;  $CLOSED = [B, A]$ .
4.  $OPEN = [D, E, F, G, H]$ ;  $CLOSED = [C, B, A]$ .
5.  $OPEN = [E, F, G, H, I, J]$ ;  $CLOSED = [D, C, B, A]$ .
6.  $OPEN = [F, G, H, I, J, K, L]$ ;  $CLOSED = [E, D, C, B, A]$ .
7.  $OPEN = [G, H, I, J, K, L, M]$  (оскільки  $L$  уже в OPEN);  $CLOSED = [F, E, D, C, B, A]$ .
8.  $OPEN = [H, I, J, K, L, M, N]$ ;  $CLOSED = [G, F, E, D, C, B, A]$ .

І так далі, доки не буде знайдено  $U$ , або  $OPEN = \emptyset$ .

На рис. 1.5 показано граф, зображений на рис. 1.4, після шістьох ітерацій пошуку вшир. Стани зі списків OPEN і CLOSED на рис. 1.5 виділено точковою й пунктирною лініями, відповідно. Зазначимо, що „прикордонні” стани пошуку на будь-якій стадії записують в OPEN, а вже розглянуті – у CLOSED.

Оскільки під час пошуку вшир вершини графа розглядають за рівнями, то спочатку досліджуються ті стани, шляхи до яких коротші. Отже, пошук вшир гарантує знаходження найкоротшого шляху від початкового стану до цільового. Більше того, оскільки скоріше досліджуються стани, знайдені вздовж найкоротшого шляху, у разі повторного проходження ці стани вже не беруться до уваги.

Іноді крім імен станів в OPEN і CLOSED необхідно зберігати додаткову інформацію. Якщо шлях є розв'язком, то він має повертатись алгоритмом. Це можна зробити, наприклад, накопиченням інформації про батька для кожного стану вздовж шляху. Стан у такому разі зберігають у вигляді пари (стан, батько).

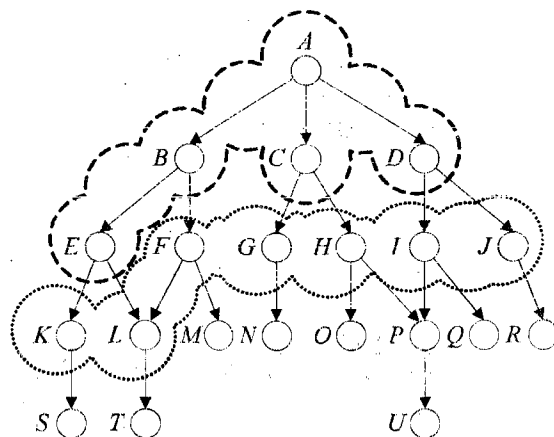


Рис. 1.5

**Приклад 1.7.** Для графа з прикладу 1.6 вміст списків OPEN і CLOSED на четвертій ітерації був би таким:

$OPEN = [(D, A), (E, B), (F, B), (G, C), (H, C)]$ ;  $CLOSED = [(C, A), (B, A), (A, nil)]$ .

Використовуючи цю інформацію, можна легко побудувати шлях  $A, B, F$ , який веде від  $A$  до  $F$ . Коли ціль знайдено, алгоритм може відновити розв'язок, відслідковуючи його у зворотньому напрямку від цілі до початкового стану за батьківськими вершинами. Зазначимо, що стан  $A$  має батька  $nil$  (нуль), тобто є початковим станом. Це слугує сигналом припинення відновлення шляху. Оскільки пошук ушир знаходить кожний стан уздовж найкоротшого шляху та зберігає першу версію кожного стану, то цей шлях від початку до цілі є найкоротшим.

## 1.8. МЕТОД ПОШУКУ ВГЛИБ

На відміну від методу перебору вшир, цей метод пропонує розгортати, передусім, ту вершину, яку було побудовано останньою. Першою вершиною, що розгортають, очевидно, є коренева вершина. Але процес спочатку завжди буде йти по самій лівій гілці вершин. Якщо утворений на поточний момент шлях виявився марним, тобто за заданої межі глибини цільову вершину не досягнуто, необхідно повернутись у вершину, яка передувала розгорнутій вершині, і спробувати ще раз застосувати до неї оператор розгортання. І так діяти доти, доки не буде одержано цільову вершину.

У цьому алгоритмі стани-сини додаються й вилучаються з лівого кінця списку OPEN, тобто список OPEN організовано як стек, або структура LIFO (last in – first out, що означає „останнім надійшов – першим обслужений”). У разі організації списку OPEN у вигляді стека перевага надається „наймолодшим” (нещодавно генерованим) станам, тобто здійснюється принцип пошуку вглиб.

### Алгоритм пошуку вглиб у кореновому графі

#### Ініціалізація

Крок 1.  $OPEN := S_{поч}$ .

Крок 2.  $CLOSED := \emptyset$ .

#### Ітерація

Крок 3. Вилучити крайню зліва вершину із OPEN, нехай це буде вершина  $X$ .

Крок 4. Якщо  $X = S_{ціль}$ , то Вихід(УСПІХ); інакше перейти до кроку 5.

Крок 5. Розгорнути вершину  $X$  (тобто згенерувати всіх її синів)

Крок 6. Помістити вершину  $X$  у список CLOSED.

Крок 7. Включити синів вершини  $X$ , які є в списку OPEN або CLOSED (це перевірка на цикл)

Крок 8. Решту синів помістити в лівий кінець списку OPEN (отже, список OPEN – це стек).

#### Перевірка закінчення

Крок 9. Якщо  $OPEN \neq \emptyset$ , то перейти до кроку 3, інакше Вихід(НЕВДАЧА).

**Приклад 1.8.** Прослідкуємо роботу алгоритму пошуку вглиб на графі, зображеному на рис. 1.4. Числа 2, 3, 4, ... означають номер ітерації, і нехай  $U$  – цільовий стан. Початковий вміст списків OPEN і CLOSED показано в рядку 1.

1. OPEN = [A]; CLOSED = [ ].
2. OPEN = [B, C, D]; CLOSED = [A].
3. OPEN = [E, F, C, D]; CLOSED = [B, A].
4. OPEN = [K, L, F, C, D]; CLOSED = [E, B, A].
5. OPEN = [S, L, F, C, D]; CLOSED = [K, E, B, A].
6. OPEN = [L, F, C, D]; CLOSED = [S, K, E, B, A].
7. OPEN = [T, F, C, D]; CLOSED = [L, S, K, E, B, A].
8. OPEN = [F, C, D]; CLOSED = [T, L, S, K, E, B, A].
9. OPEN = [M, C, D] (оскільки L уже в CLOSED); CLOSED = [F, T, L, S, K, E, B, A].
10. OPEN = [C, D]; CLOSED = [M, F, T, L, S, K, E, B, A].
11. OPEN = [G, H, D]; CLOSED = [C, M, F, T, L, S, K, E, B, A].

І так далі, доки не буде знайдено U, або OPEN = ∅.

Як і в разі пошуку вшир, у списку OPEN перелічені всі виявлені, але ще не оцінені стани (поточна „межа пошуку”), а в CLOSED записано вже розглянуті стани. На рис. 1.6 показано граф, зображений на рис. 1.4, після шістьох ітерацій пошуку вшир. Стани зі списків OPEN і CLOSED на рис. 1.6 виділено, як і раніше, точковою та штриховою лініями, відповідно. Як і в алгоритмі пошуку вшир, у даному алгоритмі можна зберігати для кожного стану запис про батька. Це дасть змогу алгоритму відновити шлях, який веде від початкового стану до цільового. Якщо під час роботи алгоритму позначати дуги, то повернення можна також здійснити за допомогою міток на дугах.

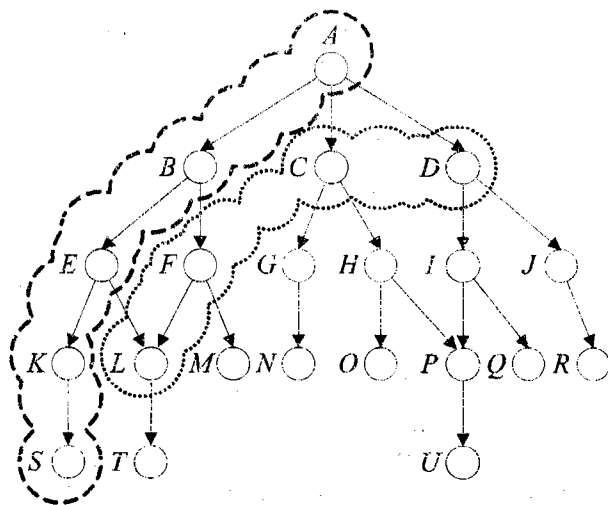


Рис. 1.6

На відміну від пошуку вшир, пошук углиб не гарантує знаходження оптимального шляху до стану, якщо він зустрівся вперше. Пізніше в процесі пошуку можуть бути знайдені різні шляхи до будь-якого стану. Якщо довжина шляху є важливою для розв'язку задачі, то в разі знаходження алгоритмом деякого стану повторно, необхідно зберегти коротший шлях. Це можна зробити, якщо зберігати для кожного стану трійку значень (стан, батько, довжина шляху). Під час генерування синівських вершин довжину шляху просто збільшують на одиницю й зберігають разом із синами. Якщо синівський стан досягнуто уздовж декількох шляхів, цю інформацію можна використати для збереження

кращого варіанту. Зазначимо, що збереження кращого варіанту стану під час звичайного пошуку вглиб зовсім не гарантує, що мету буде досягнуто саме уздовж найкоротшого шляху.

## 1.9. ПОРІВНЯННЯ МЕТОДІВ ПОШУКУ ВШИР І ВГЛИБ

Оскільки під час пошуку вшир перед переходом до рівня  $n+1$  завжди досліджуються всі вершини рівня  $n$ , пошук вшир завжди знаходить найкоротший шлях до цільової вершини. Якщо в задачі існує простий розв'язок, то його буде знайдено. Однак, у разі великого коефіцієнту розгалуження, якщо стани мають велику середню кількість синів, комбінаторний вибух може завадити алгоритму знайти розв'язки. Це відбувається тому, що всі перспективні й неперспективні вершини для кожного рівня містяться в списку OPEN. Для глибокого пошуку (коли цільова вершина розташована глибоко) або для простору станів з високим коефіцієнтом розгалуження цей пошук може бути досить громіздким.

Степінь використання простору в разі пошуку вшир вимірюється в термінах кількості станів у списку OPEN і експоненціальною функцією довжини шляху в будь-який момент часу. Якщо кожний стан породжує, у середньому,  $b$  синівських станів, то загальна кількість станів на даному рівні визначається множенням  $b$  на кількість станів попереднього рівня. Отже, на рівні  $n$  кількість станів складає  $b^n$ . Під час дослідження рівня  $n$  методом пошуку вшир усі стани містяться в списку OPEN, що може бути небажаним, якщо шлях до розв'язку досить довгий.

Пошук углиб швидко проникає вглиб простору. Якщо відомо, що шлях буде довгим, то такий пошук не буде витрачати час на знаходження великої кількості „поверхневих” станів у графі. З іншого боку, пошук вглиб може „загубитися в глибинах” графа, пропускаючи більш короткі шляхи до мети.

Міра використання простору станів у разі пошуку вглиб — це лінійна функція довжини шляху. На кожному рівні в OPEN зберігаються лише синівські вершини одного стану. Якщо в кожній вершині графа є в середньому  $b$  синів, і якщо пошук пресувається на  $n$  рівнів вглиб простору, то міра використання простору становить  $b \cdot n$ .

Так що ж краще, пошук углиб чи пошук вшир? На це запитання можна відповісти так. Необхідно дослідити простір станів і проконсультуватись з експертами в конкретній області. У шахах, наприклад, пошук вшир просто неможливий. У простіших іграх пошук вшир не тільки можливий, але може виявитись єдиним способом уникнути поразок або втрат у грі.

## 1.10. ПОШУК УГЛИБ З ІТЕРАЦІЙНИМ ЗАГЛИБЛЕННЯМ

Добрим розв'язком проблем пошуку вглиб є використання граничного значення глибини пошуку. *Гранична глибина* дає змогу обмежити пошук лише заданою кількістю рівнів. Це забезпечує щось на зразок „розгортання” області, де відбувається пошук, вшир під час пошуку вглиб.

Обмежений пошук углиб найбільш корисний, якщо відомо, що розв'язок знаходиться у межах деякої глибини, є обмеження в часі або простір станів надзвичайно великий (як у шахах). У таких задачах кількість станів, що розглядають, обмежують граничною глибиною пошуку. Така модифікація дає змогу виправити багато недоліків як пошуку вглиб, так і пошуку вшир. Ітераційним заглибленням називають пошук углиб, першу



ітерацію якого обмежено першим рівнем. Якщо ціль не знайдено, то виконують ще один крок з граничною глибиною, рівною двом. У процесі пошуку граничну глибину збільшують на одиницю на кожній ітерації. На кожній ітерації алгоритм виконує пошук углиб з урахуванням поточної граничної кількості рівнів. При цьому під час переходу від однієї ітерації до іншої інформація про простір станів не зберігається.

Оскільки алгоритм досліджує простір „за рівнями”, він може гарантувати знаходження найкоротшого шляху до цілі. Оскільки на кожній ітерації здійснюють пошук вглиб, міра використання простору на кожному рівні  $n$  становить  $b \cdot n$ , де  $b$  – середня кількість синів вершини.

На перший погляд здається, що ітераційне просування вглиб є менш ефективним за часом, ніж пошук вглиб або вшир. Проте, часова складність алгоритму з ітераційним заглибленням (час виконання алгоритму залежно від розмірності задачі) насправді має той самий порядок величини, що й кожний з цих алгоритмів –  $O(b^n)$ . Оскільки кількість вершин на даному рівні дерева зростає експоненціально зі збільшенням глибини, майже весь час витрачається на найглибшому рівні.

Можна показати, що всі розглянуті до цього стратегії пошуку (пошук углиб, ушир і з ітераційним заглибленням) мають експоненціальну складність. Це справджується загалом для всіх неінформованих алгоритмів пошуку. Для зменшення часової складності алгоритмів пошуку використовують направляючі евристички.

### 1.11. ЕВРИСТИЧНІ МЕТОДИ ПОШУКУ В ПРОСТОРІ СТАНІВ

Розглянуті вище методи повного перебору гарантують розв'язування задачі, якщо розв'язок існує. Проте експоненціальна складність цих методів призводить до того, що на практиці їх використовують для відносно невеликих за розмірністю графа станів задач. У реальних випадках частіше використовують додаткову інформацію, яка ґрунтується на попередньому досвіді або одержана на ґрунті теоретичних висновків.

Таку інформацію називають *евристичною*, а якщо її подати як правила – *евристичними правилами* або *евристичками*. Евристична інформація має цю спеціальний характер, і її можна застосовувати тільки для певного класу задач.

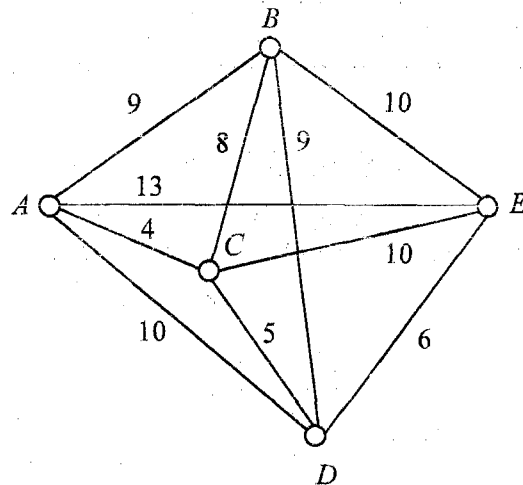


Рис. 1.7

### Розділ 1. Подання задач та пошук розв'язків

Іншими словами, евристична інформація перетворює повний перебір у цілеспрямований за певними правилами. Як приклад розглянемо задачу комівояжера.

Комівояжер повинен побудувати свій маршрут так, щоб побувати в кожному з  $n$  міст і повернутись у початкове місто. Потрібно, щоб цей маршрут був мінімальним за довжиною. Нехай  $n = 5$ , міста позначено через  $A, B, C, D, E$ . Довжини шляхів між містами наведено на рис. 1.7. Нехай початковим пунктом подорожі буде місто  $A$ , тому  $S_{\text{поч}} = A$ . Цільовий стан – також  $A$ . Оператор переходу – єдиний. Граф простору станів цієї задачі в разі використання методу пошуку вшир показано на рис. 1.8.

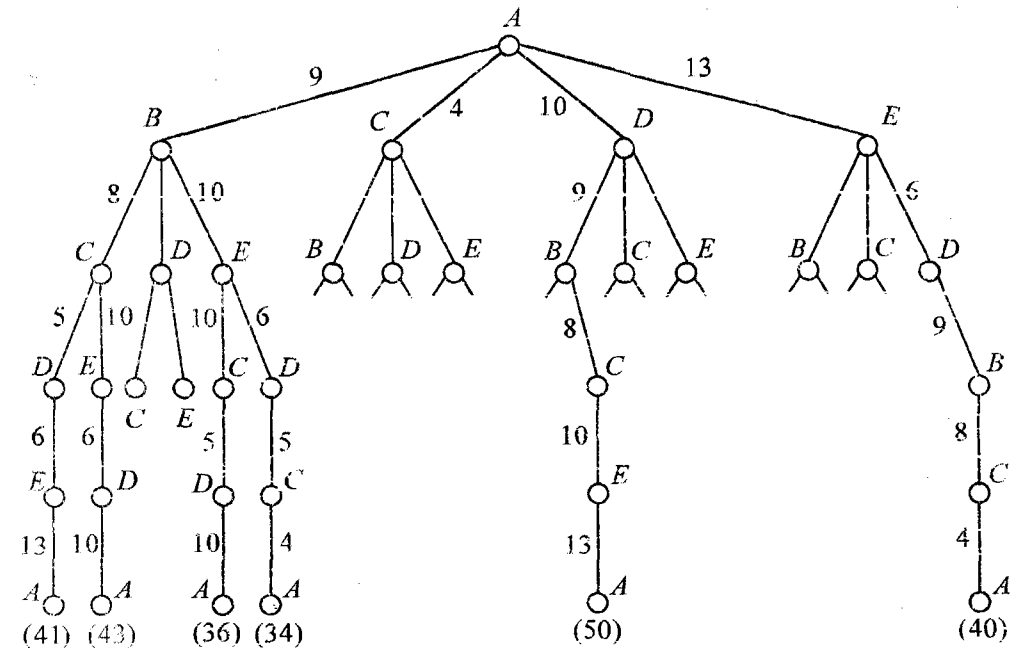


Рис. 1.8

Граф повного перебору, який містить усі можливі шляхи комівояжера, має 24 листки (кінцеві вершини). На рис. 1.8 наведено лише деякі з них. Очевидно, що серед шляхів буде і найкоротший, і найдовший. Довжина оптимального шляху становить 34 – це шлях  $A, B, E, D, C, A$ . Він єдиний, якщо не брати до уваги того, що його можна пройти у зворотному порядку:  $A, C, D, E, B, A$ . Доречи, якщо не брати до уваги „зворотні” шляхи, то варіантів буде всього 12.

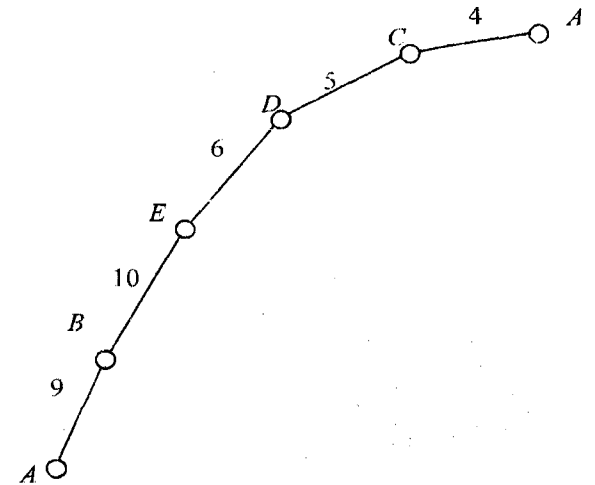


Рис. 1.9

Тепер побудуємо розв'язок методом пошуку вглиб з використанням, наприклад, такої евристики: на кожному кроці першою розгортають вершину, яка є найближчим сином із усіх можливих вершин (це так звана „стратегія найближчого сусіда”). Тоді граф пошуку буде таким, як показано на рис. 1.9. Шлях одержано оптимальний, але так буває далеко не завжди. Загалом евристичні методи не гарантують знаходження оптимальних розв'язків, але кращі з таких методів забезпечують добру близькість одержаних розв'язків до оптимальних. При цьому одержують велику економію у витратах на пошук точного розв'язку, що може бути дуже важливим чинником. Існує багато методів евристичних оцінних функцій [20].

## 1.12. МЕТОД ПОШУКУ ПО ПЕРШОМУ НАЙКРАЩОМУ ЗБІГУ

Найпростіший спосіб евристичного пошуку – це застосування процедури пошуку екстремуму. Стратегії, які ґрунтуються на пошуку екстремуму, оцінюють не тільки поточний стан пошуку, але і його синів. Для продовження пошуку вибирають найкращого сина; при цьому про його братів і батька просто забувають. Пошук припиняють, коли досягнуто стан, котрий ліпший ніж будь-який із його синів. Головна проблема стратегій пошуку екстремуму – це їх тенденція зупинятись у локальному максимумі. Інакше кажучи, як тільки вони досягають стану, який має кращу оцінку, ніж його сини, алгоритм завершується. Якщо цей стан не є розв'язком задачі, а лише локальним максимумом, то такий алгоритм не підходить для даної задачі. Це означає, що розв'язок може бути оптимальним на обмеженій множині, але через форму всього простору, можливо, ніколи не буде вибрано найліпший розв'язок.

Незважаючи на ці обмеження, алгоритм пошуку екстремуму може бути достатньо ефективним, якщо оцінна функція дає змогу уникнути локального максимуму й зациклювання алгоритму. Загалом евристичний пошук потребує гнучкого методу, передбаченого в алгоритмі пошуку по першому найкращому збігу (*best first search*, його ще називають „жадібним” алгоритмом), коли накопичення відповідної інформації дає змогу відновити алгоритм із точки локального максимуму [20, 32].

Подібно до алгоритмів пошуку вглиб і пошуку вшир, „жадібний” алгоритм пошуку використовує списки збережених станів: список OPEN відслідковує поточний стан пошуку, а в CLOSED записують уже перевірені стани. На кожному кроці алгоритм записує в список OPEN стан із урахуванням евристичної оцінки його „близькості до цілі”. Отже, на кожній ітерації розглядають найбільш перспективні стан із списку OPEN.

### Алгоритм пошуку по першому найкращому збігу у кореновому графі

#### Ініціалізація

- Крок 1.  $OPEN := S_{поч}$   
Крок 2.  $CLOSED := \emptyset$

#### Ітерація

- Крок 3. Вилучити крайню зліва вершину із OPEN, нехай це буде вершина X  
Крок 4. Якщо  $X = S_{ціль}$ , то Вихід(шлях від  $S_{поч}$  до X); інакше перейти до кроку 5  
Крок 5. Розгорнути вершину X (тобто згенерувати всіх її синів)  
Крок 6. Для кожного сина вершини X виконати залежно від випадку:  
1) син не міститься в OPEN або CLOSED: присвоїти сину евристичне значення та записати його в OPEN;

## Розділ 1. Подання задач та пошук розв'язків

- 2) син уже міститься в OPEN: якщо його було досягнуто коротшим шляхом, то присвоїти цьому стану в списку OPEN цей коротший шлях;  
3) син уже міститься в CLOSED: якщо його було досягнуто коротшим шляхом, то виключити цей стан зі списку CLOSED і записати його в OPEN

Крок 7. Помістити вершину X у список CLOSED

Крок 8. Переупорядкувати стани в списку OPEN відповідно до евристики (кращі зліва)

#### Перевірка закінчення

Крок 9. Якщо  $OPEN \neq \emptyset$ , то перейти до кроку 3, інакше Вихід(НЕВДАЧА).

На кожній ітерації алгоритм вилучає перший елемент зі списку OPEN. Досягнувши мети, алгоритм повертає (тобто видає) шлях, котрий веде до цілі. Значимо, що кожний стан зберігає попередню інформацію для визначення, чи було його раніше досягнуто коротшим шляхом. Це і дає змогу алгоритму повернути завершальний шлях до цілі. Збереження попередньої інформації здійснюють так, як це описано в підрозділі 1.7, тобто накопиченням інформації про батька для кожного стану вздовж шляху.

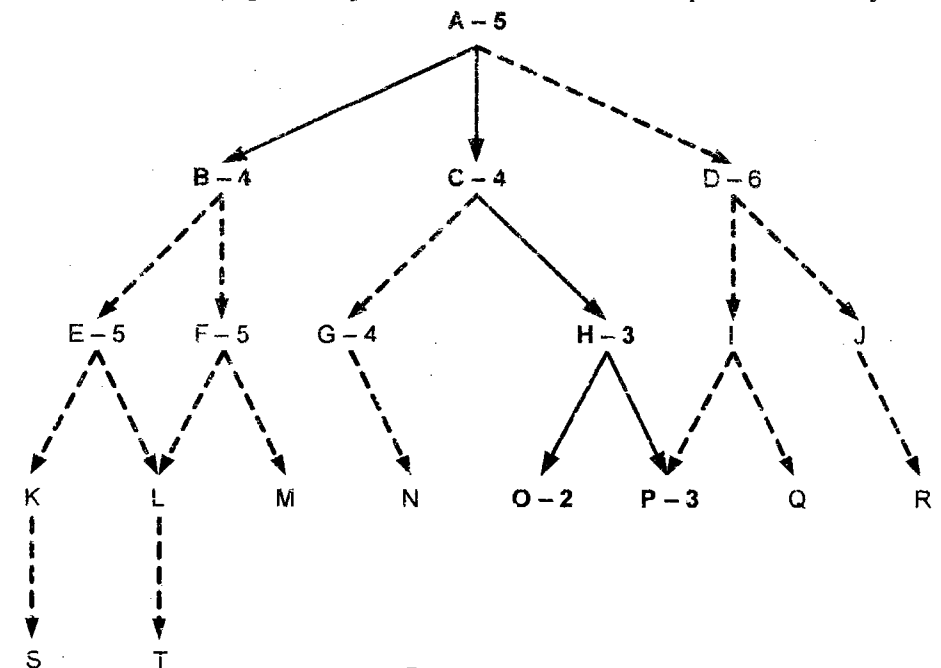


Рис. 1.10

Якщо перший елемент в OPEN – не  $S_{ціль}$ , то алгоритм використовує оператори з множини  $F$  для отримання всіх синів даного елемента. Якщо син уже наявний у списку OPEN або CLOSED, то алгоритм вибирає коротший з двох можливих шляхів досягнення цього стану. Оновлюючи історію предків вершин у списках OPEN і CLOSED, коли вони досягаються повторно, алгоритм з більшою ймовірністю знайде коротший шлях до мети  $S_{ціль}$ .

Алгоритм обчислює евристичну оцінку станів в OPEN і сортує список за цими евристичними значеннями. При цьому „кращі” стани поміщаються в початок списку.

Зазначимо, що через евристичну природу оцінювання наступний стан має перевірятись на кожному рівні простору станів. Посортований список OPEN часто називають *пріоритетною чергою*.

На рис. 1.10 зображено гіпотетичний простір із евристичними оцінками деяких станів. Стани, поруч із якими є евристична оцінка, були генеровані алгоритмом пошуку за першим найліпшим станом. Стани, за якими здійснювався евристичний пошук, позначено напівжирним шрифтом; зазначимо, що пошук не відбувається по всьому простору. Суть „жадібного” алгоритму пошуку полягає в тому, щоб підійти до цільового стану аналізом можливо меншої кількості станів; що більш обґрунтована евристика, то менше станів потрібно перевірити при пошуку цілі.

Шлях, за яким алгоритм пошуку по першому найкращому збігу знаходить цільовий стан, показано на рисунку напівжирними стрілками. Нехай  $P$  – цільовий стан (див. рис. 1.10). Оскільки  $P$  – ціль, то стани на шляху до  $P$  мають низькі евристичні значення (передбачається що оцінна функція  $h(N)$  оцінює довжину найкоротшого шляху від стану  $N$  до цільового стану). Евристика може помилятися: стан  $O$  має нижче евристичне значення ніж цільовий, і тому його досліджено раніше. На відміну від пошуку екстремуму, який не зберігає пріоритетну чергу для відбору „наступних” станів, даний алгоритм відновлюється після помилки і знаходить цільовий стан.

1. OPEN = [A-5]; CLOSED =  $\emptyset$ .
2. Розгортаємо A-5: OPEN = [B-4, C-4, D-6]; CLOSED = [A-5].
3. Розгортаємо B-4: OPEN = [C-4, E-5, F-5, D-6]; CLOSED = [B-4, A-5].
4. Розгортаємо C-4: OPEN = [H-3, G-4, E-5, F-5, D-6]; CLOSED = [C-4, B-4, A-5].
5. Розгортаємо H-3: OPEN = [O-2, F-5, G-4, E-5, F-5, D-6]; CLOSED = [H-3, C-4, B-4, A-5].
6. Розгортаємо O-2: OPEN = [P-3, G-4, E-5, F-5, D-6]; CLOSED = [O-2, H-3, C-4, B-4, A-5].
7. Розгортаємо P-3: розв'язок знайдено!

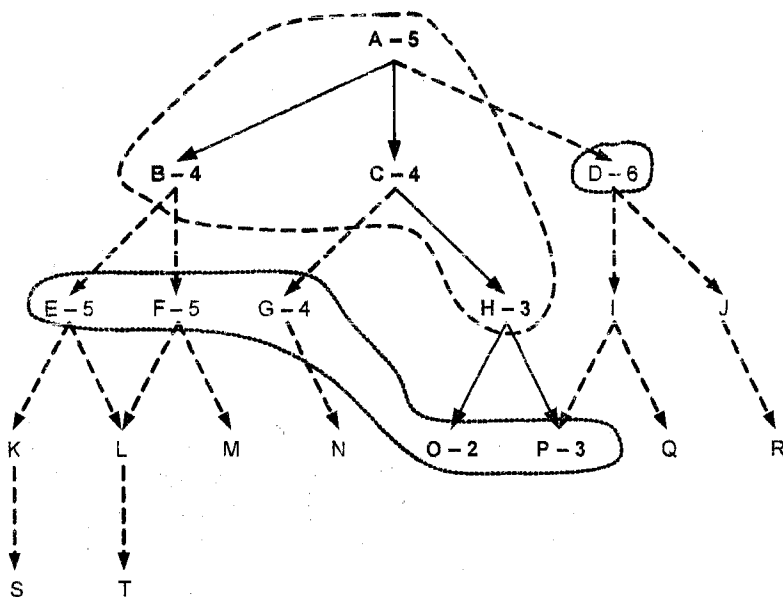


Рис. 1.11

На рис. 1.11 зображено простір після п'ятої ітерації. Стани зі списків OPEN і CLOSED виділено, відповідно, точковою та штриховою лініями. В OPEN зберігається поточна інформація, а в CLOSED – уже розглянуті стани. Зазначимо, що межа пошуку дуже викривлена, що відображає здатність наведеного алгоритму пристосовуватись.

„Жадібний” алгоритм пошуку завжди вибирає найбільш перспективний стан в OPEN для продовження. Проте, оскільки під час вибору стану використовується евристика, яка може виявитись помилковою, алгоритм не відмовляється від інших станів, і зберігає їх у списку OPEN. У даному випадку евристика спрямувала пошук шляхом, який виявився хибним, але алгоритм у результаті повернувся до деякого раніше генерованого „кращого” стану в OPEN і після цього продовжив пошук в іншій частині простору. У прикладі на рис. 1.10 після знаходження синів стану  $B$  здалося, що вони мають погані евристичні оцінки, і тому пошук змістився до стану  $C$ . Сини  $B$  збережені в OPEN на випадок, якщо алгоритму необхідно буде повернутись до них пізніше. В алгоритмі пошуку по першому найкращому збігу точно так, як в алгоритмі з підрозділу 1.7, список OPEN дає змогу повернутись назад шляхом, який не призвів до цілі.

Різні евристики та їх ефективність докладно розглянуто в [20, 32].

### 1.13. ПОДАННЯ ЗАДАЧ У ВИГЛЯДІ ГРАФІВ AND/OR

У попередніх підрозділах розглянуто способи розв'язування задач, поданих за допомогою простору станів. Розв'язування задач в такому разі зводиться до пошуку шляху в графі простору станів. Але для певних типів задач природнішим є інше подання, у вигляді графа AND/OR [4, 20]. Таке подання ґрунтується на декомпозиції задач на підзадачі. Декомпозиція на підзадачі може застосовуватись, якщо підзадачі взаємно незалежні і тому можуть розв'язуватись незалежно одна від одної.

Проілюструємо вищесказане на прикладі. Розглянемо задачу пошуку шляху між двома виділеними містами на дорожній карті, наведеній на рис. 1.12. Для спрощення довжини шляхів не розглядаємо. Цю задачу, безумовно, можна сформулювати як пошук шляху в просторі станів. Відповідний простір станів може виглядати повністю аналогічно цій карті: вершини в просторі станів відповідають містам, дуги – прямим з'єднанням між містами, а ваги дуг – відстаням між містами. Але можна розглядати й інше подання цієї задачі, яке ґрунтується на декомпозиції.

На карті, наведеній на рис. 1.12, є також річка. Припустимо, що її можна перетнути лише через два мости, один з яких знаходиться в місті  $f$ , а інший – у місті  $g$ . Очевидно, що необхідно включити в шлях один із мостів, тому шлях має пройти або через  $f$ , або через  $g$ . Отже, матимемо два описаних нижче варіанта.

Щоб знайти шлях від  $a$  до  $z$ , потрібно виконати одну з наступних дій.

1. Знайти шлях від  $a$  до  $z$  через  $f$ .
2. Знайти шлях від  $a$  до  $z$  через  $g$ .

Тепер кожен із цих двох варіантів задачі можна розкласти на підзадачі, як описано нижче.

1. Щоб знайти шлях від  $a$  до  $z$  через  $f$ , потрібно виконати такі дії.
  - 1.1. Знайти шлях від  $a$  до  $f$ .
  - 1.2. Знайти шлях від  $f$  до  $z$ .
2. Щоб знайти шлях від  $a$  до  $z$  через  $g$ , потрібно виконати такі дії.

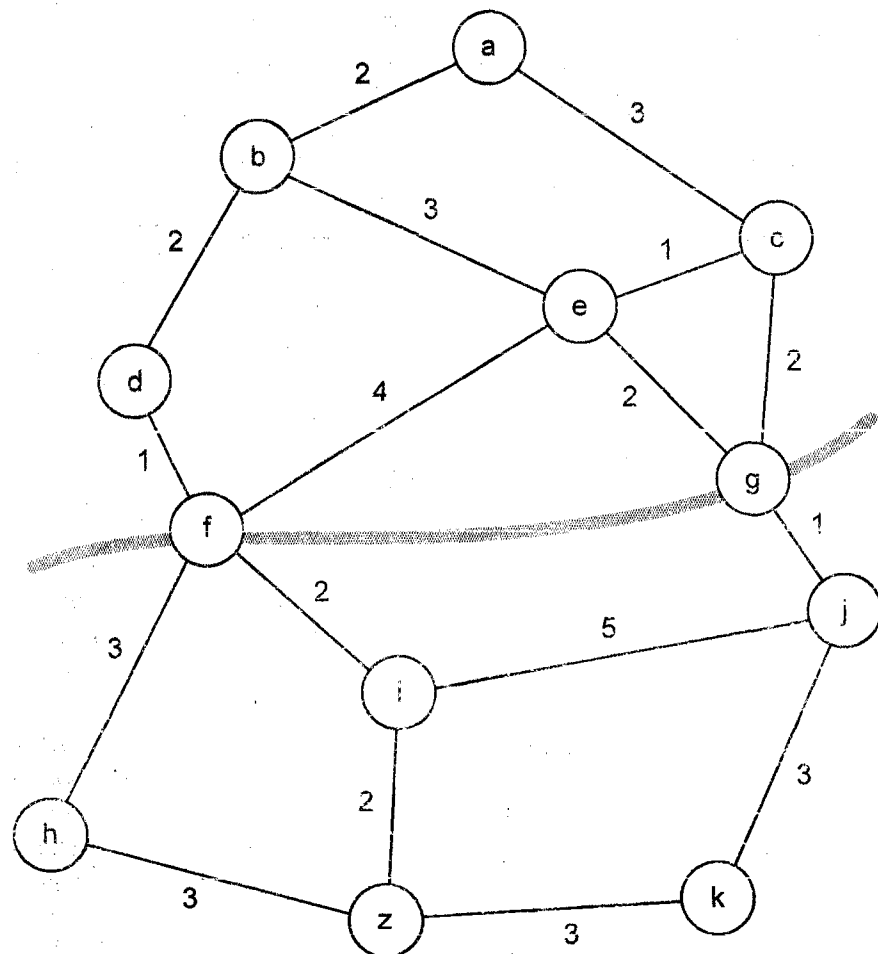
2.1. Знайти шлях від  $a$  до  $g$ .2.2. Знайти шлях від  $g$  до  $z$ .

Рис. 1.12

Отже, існує два варіанти розв'язування сформульованої задачі: прокласти шлях через  $f$  або прокласти його через  $g$ . Крім того, кожний із варіантів задачі можна розділити на дві підзадачі (відповідно, 1.1 і 1.2 або 2.1 і 2.2). Тут важливе те, що (в обох варіантах) кожен з підзадач можна розв'язати незалежно від іншої. Таку декомпозицію можна подати у вигляді графа AND/OR (рис. 1.13). Зверніть увагу на дуги, з'єднані кривою лінією, які позначають зв'язок AND між підзадачами. На рис. 1.13, зображено тільки верхню частину дерева AND/OR. Подальша декомпозиція підзадач може ґрунтуватись на розгляді інших проміжних міст.

Що являють собою цільові вершини в графі AND/OR? Цільові вершини відповідають підзадачам, котрі є *найпростішими*, або *тривіальними*. Так називають задачі, які можна розв'язати за один крок. У цьому прикладі такою підзадачею буде „пошук шляху від  $a$  до  $c$ ”, оскільки на дорожній карті є пряме з'єднання між містами  $a$  та  $c$ .

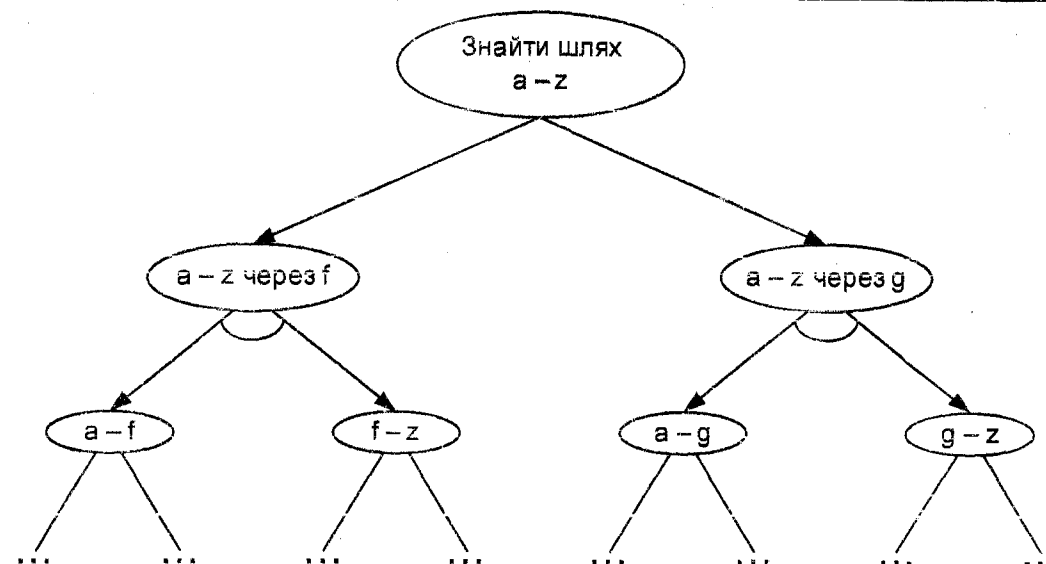


Рис. 1.13

У цьому прикладі введено важливі поняття. *Граф AND/OR* – це орієнтований граф, вершини якого відповідають задачам, а дуги позначають відношення між задачами. Причому, є два відношення між самими дугами. Цими відношеннями є AND і OR залежно від того, чи потрібно розв'язати тільки одну із задач – синів, чи одразу декілька (рис. 1.14). У принципі, із будь-якої вершини можуть виходити дуги, зв'язані відношенням AND, і дуги, зв'язані відношенням OR. Але ми будемо вважати, що кожна вершина має або тільки синів AND, або тільки синів OR. Річ у тім, що будь-який граф AND/OR можна зобразити в стандартній формі, уводячи за необхідністю допоміжні вершини OR. Тому вершину, із якої виходять тільки дуги AND, називають *вершиною AND*, а вершину, із якої виходять тільки дуги OR, називають *вершиною OR*.

а)

б)

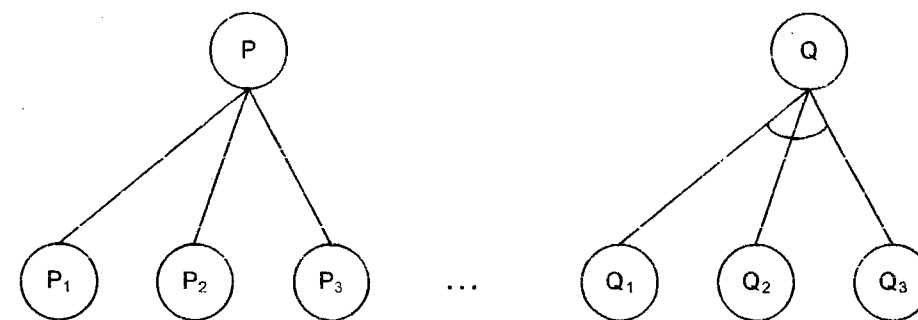


Рис. 1.14

На рис. 1.14 наведено два типи вершин у графі AND/OR:

а) щоб розв'язати задачу  $P$ , достатньо розв'язати будь-яку з підзадач  $P_1, P_2$  і т. д.;

б) щоб розв'язати задачу  $Q$ , необхідно розв'язати всі з підзадач  $Q_1, Q_2$  і т. д.

Якщо задачу подано у вигляді простору станів, то її розв'язок зводиться до пошуку

шляху в просторі станів. До чого зводиться розв'язок, якщо задачу подано у вигляді графа AND/OR? Очевидно, що будь-який розв'язок має містити всі підзадачі кожної вершини AND. Тому тепер розв'язок являє собою не шлях, а дерево. Це дерево розв'язку  $T$  означають за допомогою рекурсії.

- ◆ Задана проблема  $P$  є коренем дерева  $T$ .
- ◆ Якщо  $P$  – вершина OR, то в дереві  $T$  знаходиться один і тільки один із її синів (по графу AND/OR) разом із його власним деревом розв'язку.
- ◆ Якщо  $P$  – вершина AND, то в дереві  $T$  знаходиться всі її сини (по графу AND/OR) разом із їх деревами розв'язку.

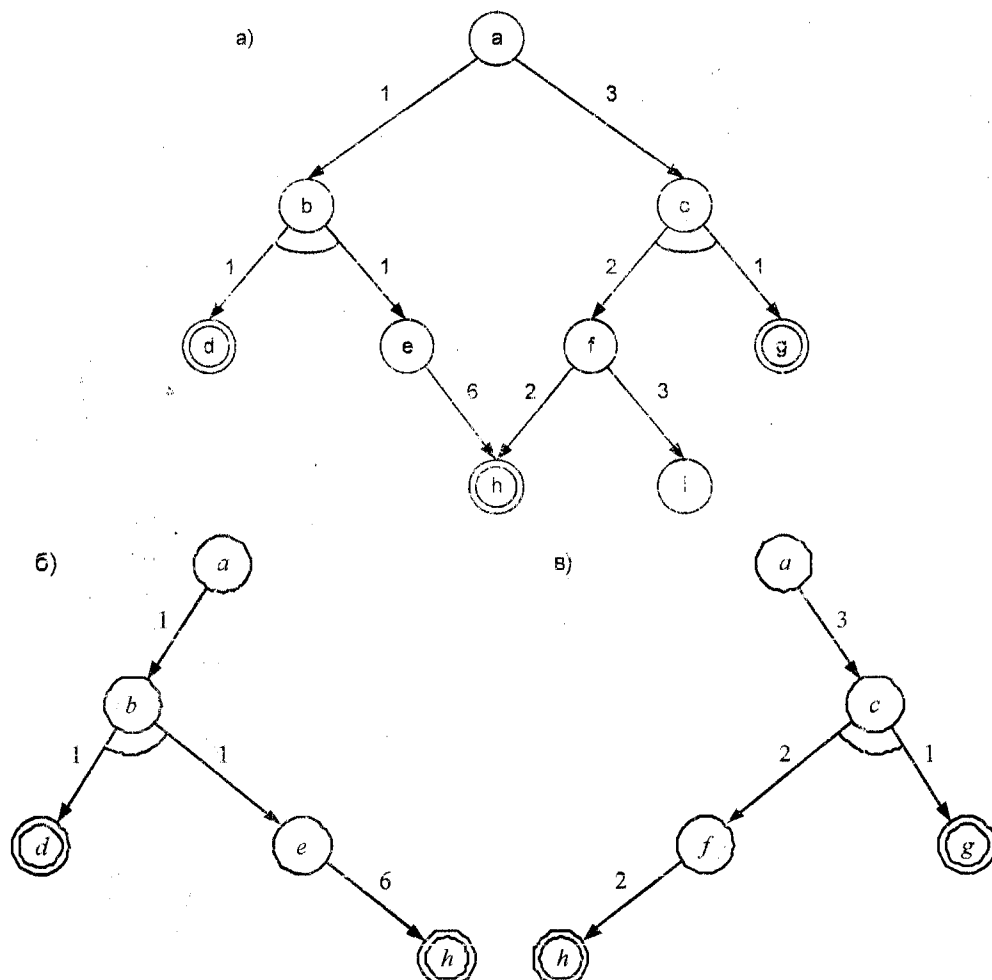


Рис. 1.15

Це означення проілюстровано на рис. 1.15. На рис. 1.15, а наведено граф AND/OR, у якому  $d$ ,  $g$ , та  $h$  – цільові вершини, вершина  $a$  – задача, яку треба розв'язати. На цьому рисунку наведено також ваги дуг. Ваги дають змогу сформулювати критерій оптимальності. Можна, наприклад, означити вагу дерева розв'язку як суму ваг всіх дуг у цьому дереві. На рис. 1.15, б та в наведено два дерева розв'язку, вага яких становить, відповідно, 9 і 8. Якщо нас цікавить мінімальна вага, то кращим є дерево

розв'язку, наведене на рис. 1.15, в.

Розглянемо механізм зведення задачі до підзадач. Подано алгоритм для побудови графа AND/OR із виділеним деревом розв'язку. Щоб описати цей метод подамо задачу  $Z$  у вигляді трійки  $Z = (S, F, G)$ , де  $S$  – множина початкових станів,  $F$  – множина операторів, які переводять предметну область із одного стану в інший,  $G$  – множина цільових станів. Кінцева мета зведення задачі до підзадач – отримання таких найпростіших (тривіальних) задач, розв'язки яких очевидні. Нагадаємо, що найпростішими вважають задачі, котрі можна розв'язати за один крок, тобто за одне застосування якогось оператора з множини  $F$ .

Отже, якщо задачу  $Z$  задано її початковим описом  $Z = (S, F, G)$ , то звести її до сукупності простіших задач можливо, якщо вдасться виділити основні проміжні стани  $S_1, S_2, S_3, \dots, S_n \in S$ . Ці стани відіграють роль нових (проміжних) цілей, і тому далі  $S_i$  ми позначатимемо як  $g_i$ . Кожному з цих станів можна зіставити свій опис у вигляді трійок:  $(S, F, g_1), (g_1, F, g_2), (g_2, F, g_3), \dots, (g_n, F, G)$ .

На цій ідеї побудовано механізм зведення задачі до підзадач.

1. Виділяємо принаймні один оператор  $f_i \in F$ , котрий обов'язково буде у розв'язувальній послідовності операторів. Оператори, які обов'язково використовуються, називають *ключовими*.

2. Для кожного (якщо їх декілька) з ключових операторів визначають проміжні стани, до яких ці оператори можна застосувати в умовах задачі  $Z$ . Нехай для оператора  $f_i$  це буде стан  $g_i$ . (Таких станів може бути багато, і тоді вони утворюють підмножину цільових станів  $Gf_i \subseteq G$ .) Тепер уже можна виділити підзадачу пошуку шляху від початку до стану  $g_i$  (або до підмножини станів  $Gf_i$ ). Отже, застосування оператора  $f_i$  привело до першої підзадачі з описом  $(S, F, g_i)$ .

3. Щойно такий опис знайдено, можна сформулювати другу підзадачу (вона буде найпростішою). Справді, оскільки стан  $g_i$  відповідає оператору  $f_i$ , то застосуємо цей оператор до  $g_i$  та отримаємо новий стан  $f_i(g_i)$ , який гарантовано наближає нас до мети, правда, лише на крок.

4. Від отриманого стану  $f_i(g_i)$  до кінцевої мети  $G$  може бути ще довгий шлях. Визначити його – третя підзадача. Отже, застосування вибраного оператора  $f_i$  до задачі з описом  $Z = (S, F, G)$  дає змогу виділити одразу три підзадачі:

$$(S, F, g_i), (g_i, f_i, f_i(g_i)), (f_i(g_i), F, G),$$

із яких принаймні одна – найпростіша.

Такому розбиттю відповідає граф, наведений на рис. 1.16.

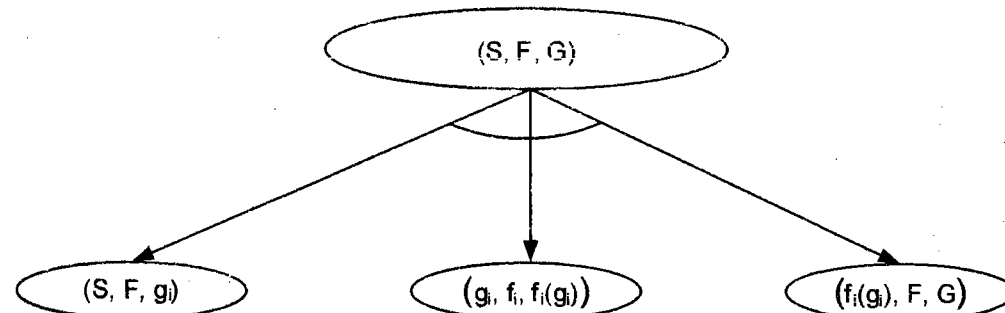


Рис. 1.16



Найпростіша задача типу  $(g_i, f_i, f_i(g_i))$  розв'язується завжди для будь-якої вибраної точки  $g_i$  простору станів, тому її можна не вказувати. Точка  $g_i$  – одна з можливих проміжних цілей,  $g_i \in Gf_i$ . Вибравши її, застосуємо до неї оператор  $f_i$ . Узагальнюючи сказане, приходимо до фрагменту, зображеному на рис. 1.17. Це граф розбиття на підзадачі для однієї точки.

Кожну з підзадач, які одержано при розбитті, якщо вона не найпростіша, можна знову розбити на підзадачі аналогічно (можливо, за допомогою іншого оператора).

Отже, для розбиття задачі на підзадачі й побудови відповідного графа AND/OR потрібні ключові оператори. Один із способів отримання операторів, які могли би бути ключовими, полягає в обчисленні відмінностей між станами на шляху від  $S_{поч} \in S$  до  $S_{ціль} \in G$ . Кожній можливій відмінності ставлять у відповідність оператор (або їх множину), який цю відмінність може усунути.

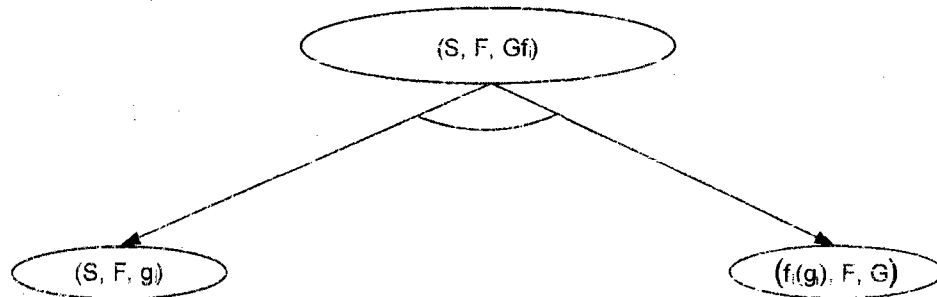


Рис. 1.17

Тепер проілюструємо метод розбиття задачі на підзадачі за допомогою графа AND/OR на прикладі з мавпою й бананами. Нехай початковий опис задачі задано так:

$$S_{поч} = (a, b, c, 0, 0),$$

$$S_{ціль} = (c, c, c, 1, 1),$$

$$F = (f_1, f_2, f_3, f_4).$$

При цьому зазначимо, що оператор

$f_1$  усуває різницю між станами  $(a, b, c, 0, 0)$  і  $(b, b, c, 0, 0)$ ,

$f_2$  – між  $(b, b, c, 0, 0)$  і  $(c, c, c, 0, 0)$ ,

$f_3$  – між  $(c, c, c, 0, 0)$  і  $(c, c, c, 1, 0)$ ,

$f_4$  – між  $(c, c, c, 1, 0)$  і  $(c, c, c, 1, 1)$ ,

Спробуємо далі виділити підзадачі, використовуючи, наприклад, оператор  $f_3$ . Йому відповідатимуть задачі з такими описами.

$Z1 = ((a, b, c, 0, 0), F, (c, c, c, 0, 0))$ , бо оператору  $f_3$  відповідає стан  $g_3 = (c, c, c, 0, 0)$ .

$Z2 = ((c, c, c, 0, 0), f_3, (c, c, c, 1, 0))$ . Тут  $(c, c, c, 1, 0) = f_3(c, c, c, 0, 0)$ , тобто являє собою результат застосування оператора  $f_3$  до стану  $(c, c, c, 0, 0)$ .

$Z3 = ((c, c, c, 1, 0), F, (c, c, c, 1, 1))$ .

Можна побачити, що підзадача  $Z1$  – не найпростіша, тобто вона допускає подальше розбиття, а підзадача  $Z3$  – найпростіша, якщо застосувати оператор  $f_4$ . Підзадача  $Z2$ , очевидно, також найпростіша.

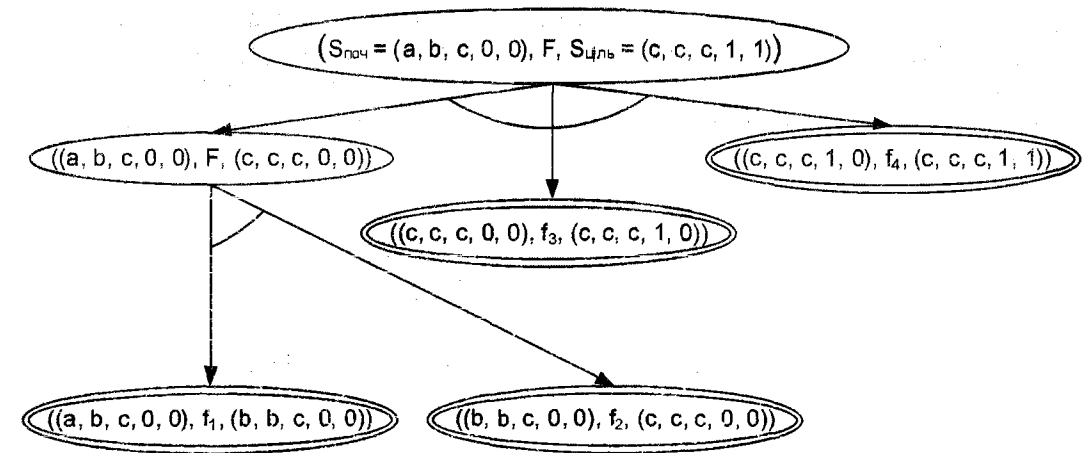


Рис. 1.18

Розбиваємо на підзадачі задачу  $Z1$ . Одну з відмінностей між початковим і кінцевим станами можна усунути застосуванням, наприклад, оператора  $f_2$ . Одержимо нові підзадачі з описами:

$$Z11 = ((a, b, c, 0, 0), F, (b, b, c, 0, 0));$$

$$Z12 = ((b, b, c, 0, 0), g_2, (c, c, c, 0, 0)).$$

Тут ми бачимо, що підзадача  $Z12$  – найпростіша, а відмінність у підзадачі  $Z11$  можна усунути, якщо використати оператор  $f_1 \in F$ , тобто вона теж найпростіша.

Повністю граф AND/OR наведено на рис. 1.18. З цього рисунка можна побачити, що всі підзадачі мають тип AND, а дерево розв'язку тут співпадає з усім графом.

Підсумовуючи сказане в цьому підрозділі можна зробити такі висновки.

◆ Подання задачі у вигляді графа AND/OR ґрунтується на принципі декомпозиції задачі на підзадачі.

◆ Вершини в графі AND/OR відповідають задачам, а дуги показують на зв'язки між задачами.

◆ Вершина, із якої виходять зв'язки OR, являє собою вершину OR. Для розв'язування задачі, яку позначено вершиною OR, достатньо розв'язати тільки одну із задач, якими позначено синів цієї вершини.

◆ Вершина, із якої виходять зв'язки AND, являє собою вершину AND. Для розв'язування задачі, яку позначено вершиною AND, необхідно розв'язати всі задачі, якими позначено синів цієї вершини.

◆ Для графа AND/OR конкретну задачу формулюють за допомогою таких двох понять:

1) початкова вершина;

2) цільова умова для розпізнавання цільових вершин.

◆ Цільові (або „кінцеві”) вершини відповідають тривіальним (або „найпростішим”) задачам.

◆ Будь-який розв'язок подають у вигляді дерева розв'язку, яке являє собою підграф графа AND/OR.

◆ Подання задачі у вигляді простору станів можна розглядати як частинний випадок подання у вигляді графа AND/OR, у якому всі вершини є вершинами OR.

◆ Щоб можна було скористатись перевагою подання задачі графом AND/OR, необхідно, щоб вершини, зв'язані відношеннями AND, були підзадачами, які можна розв'язати незалежно одну від одної.

◆ Для забезпечення можливості сформулювати критерій оптимізації можна призначити ваги дуг.



## ЗАДАЧІ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

1. Гамільтонів шлях – це шлях, який проходить через кожную вершину графа точно один раз. Які умови необхідні для існування такого шляху? Чи існує такий шлях на карті Кенігсберга (див. [25, стор. 108].

2. Людині треба переправитись через річку й перевезти із собою вовка, козу та капусту. На березі річки знаходиться човен, яким має керувати людина. Човен одночасно може перевозити не більше двох пасажирів (включаючи човняра). Якщо вовк залишиться на березі з козою, то він її їсть. Якщо коза залишиться на березі з капустою, то вона знищить капусту. Потрібно виробити послідовність переправи через річку так, щоб усі чотири пасажери були доставлені збереженими на інший берег річки. Подати цю задачу у вигляді графа. Нехай вершини відповідають розміщенню об'єктів у просторі. Наприклад, людина й коза знаходяться на західному березі річки, а вовк і капуста – на східному. Наведіть переваги стратегій пошуку вшир і вглиб для цього простору.

3. Наведіть приклад задачі комівояжера, у якій „стратегія найближчого сусіда” при знаходженні оптимального шляху приводить до невдачі. Запропонуйте іншу евристику для цієї задачі.

4. Побудуйте граф AND/OR для задачі з мавпою, починаючи з оператора  $f_4$ .

5. Побудуйте граф AND/OR для задачі комівояжера (див. підрозділ 1.11), почавши розбиття, наприклад, з вершини E.



## КОМП'ЮТЕРНІ ПРОЕКТИ

Скласти програми із зазначеними вхідними даними та результатами

1. Задано простір станів у вигляді орієнтованого ациклічного графа. Знайти цільову вершину пошуком ушир.

2. Задано простір станів у вигляді орієнтованого ациклічного графа. Знайти цільову вершину пошуком вглиб.

3. Задано простір станів у вигляді орієнтованого ациклічного графа. Знайти цільову вершину пошуком ушир з ітераційним заглибленням. Передбачити різні варіанти обмеження глибини пошуку.

4. Задано задачу комівояжера. Реалізувати евристичний пошук, використовуючи „стратегію найближчого сусіда”.

5. Задано задачу комівояжера. Реалізувати евристичний пошук, використовуючи евристику, відмінну від „стратегії найближчого сусіда”.

## РОЗДІЛ 2



## ЛОГІЧНЕ ВИВЕДЕННЯ

- ◆ *Короткі історичні відомості*
- ◆ *Логіка висловлювань*
- ◆ *Закони логіки висловлювань*
- ◆ *Нормальні форми логіки висловлювань*
- ◆ *Логіка першого ступеня*
- ◆ *Закони логіки першого ступеня*
- ◆ *Випереджена нормальна форма*
- ◆ *Логічне виведення в логіці висловлювань*
- ◆ *Алгоритм Куайна*
- ◆ *Алгоритм Девіса–Патнема*
- ◆ *Поняття формальної теорії. Числення висловлювань*
- ◆ *Застосування правил виведення в численні висловлювань*
- ◆ *Метод резолюцій у численні висловлювань*
- ◆ *Числення предикатів як формальна теорія. Логічне виведення в численні предикатів*
- ◆ *Правила виведення в численні предикатів*
- ◆ *Підстановка та уніфікація. Найзагальніший уніфікатор. Алгоритм уніфікації*
- ◆ *Сколемівська нормальна форма*
- ◆ *Метод резолюцій у численні предикатів*
- ◆ *Хорнівські диз'юнкти та метод SLD-резолюцій*
- ◆ *Принцип логічного програмування*

### 2.1. КОРОТКІ ІСТОРИЧНІ ВІДОМОСТІ

Логічне виведення – центральний принцип, який реалізовано в системах штучного інтелекту. Цей принцип впливає із загальнішого поняття формальної системи, формальної теорії, або числення та реалізовано відповідно до загальних властивостей такої системи. Теорію логічного виведення, яка являє собою розділ математичної логіки, цікавить саме процес знаходження розв'язку шляхом формального виведення. Істинність або фальшивість окремих тверджень або висновків мало цікавить логіка. Він бажає лише знати, чи є істинність висновків наслідком істинності гіпотез. Систематична формалізація та систематизація правильних способів міркувань – одна з головних задач логіки.

Спосіб побудови наукової теорії, у якому за її основу беруть певні аксіоми або

постулати, називають *аксіоматичним методом*. Аксіоматичний метод був закладений у роботах давньогрецьких геометрів. Блискучим зразком застосування цього методу аж до XIX ст. була геометрія Евкліда (біля 300 р. до н.е.). У неї вперше проявилась основна ідея аксіоматичного методу: отримання змісту геометрії дедуктивним методом шляхом доведення теорем, виходячи з невеликої кількості тверджень – аксіом, зміст яких був, як здавалось, очевидним. Неочевидною, як виявилось, була лише одна аксіома, виділена Евклідом, – п'ятий постулат про паралельні. У першій половині XIX ст. М. Лобачевський та Я. Бойяї (J. Bolyai) замінили цю аксіому та незалежно один від одного відкрили нову, неевклідову геометрію. Виявилось, що заміною лише однієї аксіоми та чисто формальними діями можна отримати іншу геометрію, яка за багатством не поступається евклідовій геометрії.

Одночасно з появою елементарної геометрії, спробами аксіоматизації арифметики та накопиченням досвіду побудови аксіоматичних теорій уточнювалось поняття формальної аксіоматичної системи, виникла теорія доведень. Нині час теорію доведень розглядають як розділ сучасної математичної логіки та передумову створення логічного програмування.

Подальший розвиток аксіоматичний метод отримав у роботах Д. Гілберта (D. Hilbert) та його школи у вигляді наукової програми з обґрунтування точним математичним способом несуперечливості математики. Програма передбачала уточнення змісту поняття доведення для того, щоб воно стало об'єктом математичної теорії логічного виведення. У рамках цього напрямку було уточнено поняття аксіоматичної теорії, виникло поняття формальної системи. У результаті стало можливим подати неформалізовані математичні теорії як точні математичні об'єкти та будувати теорію цих об'єктів – *метатеорію*.

Формальну теорію будують як точно визначений клас формул, у якому виділяють підклас теорем цієї формальної системи. Формули формальної системи безпосередньо не мають у собі жодного змісту, їх будують із довільних знаків або символів. Хоча формальну аксіоматизацію логічних міркувань у повному обсязі було сформульовано лише в працях Г. Фреге (G. Frege), Б. Рассела (B. Russell), А. Тьюрінга (A. Turing), А. Тарського (A. Tarski) та інших у XIX та XX ст., коріння цих праць можна прослідкувати аж до Аристотеля.

Відправною точкою для вивчення інтелекту вважають спадщину Аристотеля. У праці „Логіка” він обговорює свою науку пізнання, де основу знання вбачає у вивченні думки, та розглядає питання істинності міркувань на основі їхнього зв'язку з іншими істинними твердженнями. Прикладами його міркувань є логічні висновки – силісми – такого типу: якщо відомо, що всі люди смертні та Сократ – людина, то можна зробити висновок, що Сократ – смертний.

До створення формальної мови для опису мислення прагнув Дж. Буль. Найвідомішим його відкриттям стала математична формалізація законів логіки – основа сучасних комп'ютерних наук. Мета Дж. Буля за ідеєю близька до сучасного штучного інтелекту – дослідити фундаментальні закони мислення, описати їх у символічному вигляді, з „різних елементів істини” скласти припущення щодо природи та змісту людського розуму. Система Дж. Буля склала основу двійкової арифметики та показала, що надзвичайно проста формальна система може реалізувати силу логіки. Г. Фреге в

праці „Основи арифметики” (1884 р.) створив зрозумілу й точну мову для опису основ арифметики. Цю мову зараз називають численням предикатів.

Для фундаментальних принципів штучного інтелекту особливо важливими є праці Б. Рассела та А. Уайтхеда (A. Whitehead), оскільки ними була проголошена мета виведення всієї математики з набору аксіом шляхом формальних операцій. Це означає, що аксіоми та теореми треба розглядати виключно як набори символів, а доведення будувати лише на застосуванні строго визначених правил для маніпулювання рядками символів. У цьому випадку виключено використання інтуїції або змісту теорем як основи доведення. Кожний крок доведення впливає зі строгого застосування формальних правил до аксіом і вже виведених теорем, навіть тоді, коли в традиційних доведеннях теорем цей крок вважають очевидним. Зміст, який міститься в теоремах і аксіомах системи, має відношення лише до зовнішнього світу та абсолютно не залежить від логічного виведення. Такий повністю формальний підхід до математичних міркувань склав суттєву основу для його автоматизації в реальних обчислювальних системах. Логічний синтаксис та формальні правила виведення, розроблені Б. Расселом та А. Уайтхедом, складають основу систем автоматичного доведення теорем. Закладені ними принципи були основою досліджень Ж. Гербрана (J. Herbrand), Т. Сколема (T. Skolem) та К. Гьоделя (K. Gödel), які були здійснені біля 1930 р. і дозволили довести фундаментальні теореми числення предикатів. У кінці 1950-х років, ґрунтуючись на цих формальних правилах виведення, було запропоновано методи автоматичного доведення теорем, що є основним застосуванням логіки предикатів у комп'ютерних інформаційних технологіях (докладніше про це див. підрозділ 2.18). Ця проблема актуальна для розв'язування задач штучного інтелекту, задач, які виникають у базах даних і знань, для вирішення проблем подання знань і роботи з ними.

## 2.2. ЛОГІКА ВИСЛОВЛЮВАНЬ

*Висловлюванням* називають розповідне речення, про яке можна сказати, що воно або істинне, або фальшиве, але не одне й інше водночас. Розділ логіки, який вивчає висловлювання та їх властивості, називають *пропозиційною логікою* або *логікою висловлювань* [22, 57].

**Приклад 2.1.** Наведемо приклади речень.

1. Сніг білий.
2. Київ – столиця України.
3.  $x+1=3$ .
4. Котра година?
5. Читай уважно!

Два перших речення – висловлювання, решта три – ні. Третє речення набуває істинного або фальшивого значення залежно від значення змінної  $x$ , четверте та п'яте речення – не розповідні.

Значення „істина” або „фальш”, яких набуває висловлювання, називають його значенням *істинності*. Значення „істина” позначають буквою Т (від англ. „truth”), а „фальш” – буквою F (від „false”). Для позначення висловлювань використовують малі латинські букви з індексами чи без них. Символи, використовувані для позначення висловлювань, називають *атомарними формулами* чи *атомами*.

**Приклад 2.2.** Наведемо приклади висловлювань.

1.  $p$ : „Сніг білий”.

2.  $q$ : „Київ – столиця України”.

Тут символи  $p$ ,  $q$  – атомарні формули.

Багато речень утворюють об'єднанням одного або декількох висловлювань.

Отримане висловлювання називають *складним висловлюванням*. Побудову складних висловлювань уперше розглянуто 1854 р. у книзі англійського математика Джорджа Буля (George Boole) „The Laws of Truth”. Складне висловлювання утворюють з існуючих висловлювань застосуванням *логічних операцій* (використовують також термін *логічні зв'язки*). У логіці висловлювань використовують п'ять логічних операцій: *заперечення* (читають „не” та позначають знаком „ $\neg$ ”), *кон'юнкція* (читають „і” та позначають знаком „ $\wedge$ ”), *диз'юнкція* (читають „або” та позначають знаком „ $\vee$ ”), *імплікація* (читають “якщо..., то” та позначають знаком „ $\rightarrow$ ”), *еквівалентність* (читають „тоді й лише тоді” та позначають знаком „ $\leftrightarrow$ ”).

**Приклад 2.3.** Наведемо приклади складних висловлювань.

1. Сніг білий, і небо теж біле.

2. Якщо погода хороша, то ми їдемо відпочивати.

У наведених прикладах логічні операції – це „і” та „якщо..., то”.

**Приклад 2.4.** Розглянемо такі висловлювання:  $p$ : „Вологість велика”,  $q$ : „Температура висока”,  $r$ : „Ми почуваємо себе добре”. Тоді речення „Якщо вологість велика та температура висока, то ми не почуваємо себе добре” можна записати складним висловлюванням  $((p \wedge q) \rightarrow (\neg r))$ .

У логіці висловлювань атом  $p$  чи складне висловлювання називають *правильною побудованою формулою* або *формулою*. Вивчаючи формули, розглядають два аспекти – синтаксис і семантику.

*Синтаксис* – це сукупність правил, які дають змогу будувати формули та розпізнавати правильні формули серед послідовностей символів. *Формули* в логіці висловлювань визначають за такими правилами:

- ◆ атом – це формула;
- ◆ якщо  $p$  формула, то  $(\neg p)$  – також формула;
- ◆ якщо  $p$  та  $q$  – формули, то  $(p \wedge q)$ ,  $(p \vee q)$ ,  $(p \rightarrow q)$ ,  $(p \leftrightarrow q)$  – формули;
- ◆ формули можуть бути породжені тільки скінченною кількістю застосувань

указаних правил.

Формули, як і атоми, позначають малими латинськими буквами з індексами чи без них.

**Приклад 2.5.** Вирази  $(p \rightarrow)$ ,  $(p \wedge)$ ,  $(p \neg)$ ,  $(\vee q)$  – не формули.

Часто заперечення  $p$  позначають також  $\bar{p}$ . Такий спосіб запису заперечення не потребує дужок. Якщо не виникає непорозуміння, то зовнішні дужки у формулах можна випускати.

**Приклад 2.6.** Формули  $(p \wedge q)$ ,  $(p \rightarrow q)$  та  $((p \wedge q) \rightarrow (\neg r))$  можна записати відповідно у вигляді  $p \wedge q$ ,  $p \rightarrow q$ ,  $(p \wedge q) \rightarrow \bar{r}$ .

*Семантика* – це сукупність правил, за якими формулам надають значення істинності. Нехай  $p$  та  $q$  – формули. Тоді значення істинності формул  $\neg p$ ,  $p \wedge q$ ,  $p \vee q$ ,  $p \rightarrow q$  та  $p \leftrightarrow q$  так пов'язані зі значеннями істинності формул  $p$  та  $q$ .

1. Формула  $\neg p$  істинна, коли  $p$  фальшива, і фальшива, коли  $p$  істинна. Формулу  $\neg p$  читають „не  $p$ ” чи „це не так, що  $p$ ” та називають *запереченням* формули  $p$ .

2. Формула  $p \wedge q$  істинна, якщо  $p$  та  $q$  водночас істинні. У всіх інших випадках формула  $p \wedge q$  фальшива. Формулу  $p \wedge q$  читають „ $p$  і  $q$ ” й називають *кон'юнкцією* формул  $p$  та  $q$ .

3. Формула  $p \vee q$  фальшива, якщо  $p$  та  $q$  водночас фальшиві. У всіх інших випадках вона істинна. Формулу  $p \vee q$  читають „ $p$  або  $q$ ” й називають *диз'юнкцією* формул  $p$  та  $q$ .

4. Формула  $p \rightarrow q$  фальшива, якщо формула  $p$  істинна, а  $q$  – фальшива. У всіх інших випадках вона істинна. Формулу  $p \rightarrow q$  називають *імплікацією*;  $p$  називають *припущенням* (також *засновком*, *антецедентом*) імплікації, а  $q$  – *висновком* (консеквентом) імплікації. Оскільки імплікацію використовують у багатьох математичних міркуваннях, то існує багато термінологічних варіантів для формули  $p \rightarrow q$ . Ось деякі з них: „якщо  $p$ , то  $q$ ”, „з  $p$  випливає  $q$ ”, „ $p$  лише тоді, коли  $q$ ”, „ $p$  тільки, якщо  $q$ ”, „ $q$ , якщо  $p$ ”, „ $p$  достатнє для  $q$ ”, „ $q$  необхідне для  $p$ ”.

5. Формула  $p \leftrightarrow q$  істинна, якщо  $p$  та  $q$  мають однакові значення істинності. У всіх інших випадках формула  $p \leftrightarrow q$  фальшива. Формулу  $p \leftrightarrow q$  читають „ $p$  тоді й лише тоді, коли  $q$ ” чи „ $p$  еквівалентне  $q$ ” та називають *еквівалентністю* формул  $p$  та  $q$ .

Семантику логічних операцій зручно задавати з допомогою таблиць, які містять значення істинності формул залежно від значень істинності їх атомів. Такі таблиці називають *таблицями істинності*. Семантику введених операцій у формі таблиць істинності наведено в табл. 2.1.

Таблиця 2.1

| $p$ | $q$ | $\neg p$ | $p \wedge q$ | $p \vee q$ | $p \rightarrow q$ | $p \leftrightarrow q$ |
|-----|-----|----------|--------------|------------|-------------------|-----------------------|
| T   | T   | F        | T            | T          | T                 | T                     |
| T   | F   | F        | F            | T          | F                 | F                     |
| F   | T   | T        | F            | T          | T                 | F                     |
| F   | F   | T        | F            | F          | T                 | T                     |

**Приклад 2.7.** Знайдемо заперечення висловлювання „Сьогодні п'ятниця”. Воно має вигляд „Це не так, що сьогодні п'ятниця”. Це речення також можна сформулювати як „Сьогодні не п'ятниця” чи „П'ятниця не сьогодні”. Зазначимо, що речення, які пов'язані з часовою змінною, – не висловлювання доти, доки не визначено момент часу. Це стосується й змінних у реченнях, які характеризують місце чи особу. Ці речення – не висловлювання, якщо не зазначено відповідного місця чи конкретної особи.

**Приклад 2.8.** Знайдемо кон'юнкцію висловлювань  $p$  та  $q$ , де  $p$  – висловлювання „Сьогодні п'ятниця”, а  $q$  – „Сьогодні падає дощ”. Кон'юнкція цих висловлювань – „Сьогодні п'ятниця, і сьогодні падає дощ”. Воно істинне в дощову п'ятницю й фальшиве не в п'ятницю або в недощову п'ятницю.

**Приклад 2.9.** Що являє собою диз'юнкція висловлювань  $p$  та  $q$  з прикладу 2.8? Диз'юнкція висловлювань  $p$  та  $q$  – висловлювання „Сьогодні п'ятниця або сьогодні падає дощ”. Воно істинне в будь-яку п'ятницю чи в будь-який дощовий день (зокрема, „у дощову п'ятницю”) і фальшива тільки в недощові „не п'ятниця”.

Імплікацію як логічну операцію називають також *умовним реченням*. Щоб зрозуміти, чому імплікація набуває таких значень істинності, її слід сприймати як зв'язок обов'язкового й очікуваного. Наприклад, розглянемо звернення, адресоване студентам: „Якщо ви виконаєте всі завдання, то отримаєте відмінну оцінку”. Це означає, що в разі виконання студентами всіх завдань вони одержать відмінну оцінку. Якщо ж студенти не виконають усіх завдань, то вони можуть отримати оцінку „відмінно”, а можуть і не отримати її залежно від інших обставин. Однак якщо студенти зробили всі завдання, а викладач не поставив оцінку „відмінно”, то студенти відчуватимуть себе ображеними. Це відповідає ситуації, коли в імплікації  $p \rightarrow q$  припущення  $p$  „Ви виконаєте всі завдання” істинне, а її висновок  $q$  „Ви отримаєте відмінну оцінку” фальшивий.

Розуміння імплікації в логіці дещо відрізняється від її розуміння в природній мові. Наприклад „Якщо буде сонячно, то ми підемо на пляж” – умовне речення, уживане в звичайній мові. Воно залишається істинним до того моменту, коли настане сонячний день, а ми не підемо на пляж. За означенням імплікації умовне речення „Якщо сьогодні п'ятниця, то  $2+3=5$ ” істинне, бо висновок імплікації істинний. При цьому значення істинності припущення в імплікації тут не має відношення до висновку. Імплікація „Якщо сьогодні п'ятниця, то  $2+3=6$ ” істинна щодня, крім п'ятниці, хоча висловлювання  $2+3=6$  фальшиве. Останні дві імплікації ми не вживаємо в природній мові (хіба що як жарт), бо в кожному з відповідних умовних речень немає змістовного зв'язку між припущенням і висновком.

Конструкція „якщо  $p$ , то  $S$ ” у записі „if  $p$  then  $S$ ”, яку використовують в алгоритмічних мовах, відрізняється за змістом від імплікації в логіці. Тут  $p$  – висловлювання, а  $S$  – програмний сегмент, який складається з одного чи декількох операторів. Програмний сегмент  $S$  виконується, якщо висловлювання  $p$  істинне, і не виконується, якщо воно фальшиве.

Для знаходження значення істинності складного висловлювання потрібно надати значення істинності всім атомам, які містить відповідна формула. Набір значень істинності всіх атомів формули називають її *інтерпретацією*. Для обчислення значень істинності формули, яка зображає складне висловлювання, потрібно знаходити значення логічних операцій, визначених табл. 2.1. Послідовність обчислень задають парами дужок. Якщо формула має  $n$  атомів, то є  $2^n$  способів надати значення істинності її атомам, тобто така формула має  $2^n$  інтерпретацій, а всі її значення можна звести в таблицю істинності з  $2^n$  рядками.

Формула  $f$  істинна в якійсь інтерпретації, якщо  $f$  отримує значення Т в цій інтерпретації; у протилежному випадку  $f$  – *фальшива* в цій інтерпретації. Формулу  $f$  називають *виконаною*, якщо існує принаймні одна інтерпретація, у якій  $f$  істинна. У такому разі кажуть, що формула  $f$  *виконується* в цій інтерпретації.

Формулу  $f$  логіки висловлювань називають *загальнозначущою* чи *тавтологією*, якщо вона виконується в усіх інтерпретаціях. Якщо формула  $f$  – тавтологія, то використовують позначення  $\models f$ . Формулу, фальшиву в усіх інтерпретаціях, називають *заперечуваною*, *невиконаною* чи *суперечністю*. Оскільки кожна формула логіки висловлювань має скінченну кількість інтерпретацій, то завжди можна перевірити її загальнозначущість чи заперечуваність, знайшовши значення істинності в усіх можливих інтерпретаціях.

**Приклад 2.10.** У табл. 2.2 наведено значення істинності формули  $((p \rightarrow q) \wedge p) \rightarrow q$ , а в табл. 2.3 – значення істинності формули  $(p \rightarrow q) \wedge (p \wedge \bar{q})$ . Отже, формула  $((p \rightarrow q) \wedge p) \rightarrow q$  – загальнозначуща (тавтологія), а формула  $(p \rightarrow q) \wedge (p \wedge \bar{q})$  – заперечувана.

Таблиця 2.2

| $p$ | $q$ | $(p \rightarrow q)$ | $(p \rightarrow q) \wedge p$ | $((p \rightarrow q) \wedge p) \rightarrow q$ |
|-----|-----|---------------------|------------------------------|--|
| T   | T   | T                   | T                            | T  |
| T   | F   | F                   | F                            | T  |
| F   | T   | T                   | F                            | T  |
| F   | F   | T                   | F                            | T  |

Таблиця 2.3

| $p$ | $q$ | $p \rightarrow q$ | $\bar{q}$ | $p \wedge \bar{q}$ | $(p \rightarrow q) \wedge (p \wedge \bar{q})$ |
|-----|-----|-------------------|-----------|--------------------|---|
| T   | T   | T                 | F         | F                  | F   |
| T   | F   | F                 | T         | T                  | F   |
| F   | T   | T                 | F         | F                  | F   |
| F   | F   | T                 | T         | F                  | F   |

Якщо формула  $f$  містить  $n$  різних атомів, то вона має  $2^n$  інтерпретацій. Якщо  $p_1, p_2, \dots, p_n$  – атоми формули  $f$ , то зручно задавати інтерпретацію множиною  $I = \{m_1, m_2, \dots, m_n\}$ ,

$$\text{де } m_i = \begin{cases} p_i, & \text{якщо } p_i = T, \\ \bar{p}_i, & \text{якщо } p_i = F, \end{cases} (i = 1, 2, \dots, n).$$

**Приклад 2.11.** Множина  $\{p, \bar{q}, \bar{r}\}$  являє собою інтерпретацією, у якій атомам  $p$ ,  $q$  та  $r$  надано значення Т, F і F, відповідно.

Нехай  $P$  – множина всіх різних атомів формули  $f$ . Тоді  $P$  називають *базисом* формули  $f$ . Потужність базису називають *арністю* формули. *Модель* – це інтерпретація, у якій формула  $f$  набуває значення Т.

**Приклад 2.12.** Розглянемо формулу  $f = (p \wedge q) \rightarrow (p \sim \bar{r})$ , вона має  $2^3 = 8$  інтерпретацій. У табл. 2.4 дано значення істинності формули  $f$  у всіх її інтерпретаціях. Атоми в цій формулі –  $p$ ,  $q$  та  $r$ , тобто базис – множина  $P = \{p, q, r\}$ . Випишемо окремі інтерпретації формули  $f$ . Позначимо як  $I(N)$  інтерпретацію, яка відповідає рядку з номером  $N$ . Так  $I(0) = \{p, q, r\}$ ,  $I(1) = \{p, q, \bar{r}\}$ ,  $I(5) = \{\bar{p}, q, \bar{r}\}$ ,  $I(7) = \{\bar{p}, \bar{q}, \bar{r}\}$ .

Таблиця 2.4

| № інтерпретації | Інтерпретація                   | $p$ | $q$ | $r$ | $\bar{r}$ | $p \wedge q$ | $p \sim \bar{r}$ | $(p \wedge q) \rightarrow (p \sim \bar{r})$ |
|-----------------|---------------------------------|-----|-----|-----|-----------|--------------|------------------|---|
| 0               | $\{p, q, r\}$                   | T   | T   | T   | F         | T            | F                | F   |
| 1               | $\{p, q, \bar{r}\}$             | T   | T   | F   | T         | T            | T                | T   |
| 2               | $\{p, \bar{q}, r\}$             | T   | F   | T   | F         | F            | F                | T   |
| 3               | $\{p, \bar{q}, \bar{r}\}$       | T   | F   | F   | T         | F            | T                | T   |
| 4               | $\{\bar{p}, q, r\}$             | F   | T   | T   | F         | F            | T                | T   |
| 5               | $\{\bar{p}, q, \bar{r}\}$       | F   | T   | F   | T         | F            | F                | T   |
| 6               | $\{\bar{p}, \bar{q}, r\}$       | F   | F   | T   | F         | F            | T                | T   |
| 7               | $\{\bar{p}, \bar{q}, \bar{r}\}$ | F   | F   | F   | T         | F            | F                | T   |





Таблиця 2.7

|     | Назва закону                 | Формулювання закону  |
|-----|------------------------------|--|
| 1.  | Закони комутативності        | а) $p \vee q = q \vee p$<br>б) $p \wedge q = q \wedge p$   |
| 2.  | Закони асоціативності        | а) $(p \vee q) \vee r = p \vee (q \vee r)$<br>б) $(p \wedge q) \wedge r = p \wedge (q \wedge r)$                     |
| 3.  | Закони дистрибутивності      | а) $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$<br>б) $p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$ |
| 4.  | Закон суперечності           | $p \wedge \bar{p} = F$   |
| 5.  | Закон виключеного третього   | $p \vee \bar{p} = T$   |
| 6.  | Закон подвійного заперечення | $\bar{\bar{p}} = p$  |
| 7.  | Закони ідемпотентності       | а) $p \vee p = p$<br>б) $p \wedge p = p$   |
| 8.  | Закони де Моргана            | а) $\overline{p \vee q} = \bar{p} \wedge \bar{q}$<br>б) $\overline{p \wedge q} = \bar{p} \vee \bar{q}$               |
| 9.  | Закони поглинання            | а) $(p \vee q) \wedge p = p$<br>б) $(p \wedge q) \vee p = p$   |
| 10. | Закони тотожності            | а) $p \wedge T = p$<br>б) $p \vee F = p$   |
| 11. | Закони домінування           | а) $p \vee T = T$<br>б) $p \wedge F = F$   |

Два наступні правила дозволяють усувати логічні операції імплікації та еквівалентності з формул, тобто перетворювати їх у формули, які таких операцій не містять:  $p \rightarrow q = \bar{p} \vee q$  та  $p \sim q = (p \rightarrow q) \wedge (q \rightarrow p)$ . Ці правила можна також використовувати для введення імплікації та еквівалентності. Наведені логічні еквівалентності можна довести за допомогою таблиць істинності. Зокрема, у прикладі 2.15 доведено логічну еквівалентність  $p \rightarrow q = \bar{p} \vee q$ . Зазначимо, що приклад 2.16 показує некомутативність імплікації.

Застосовуючи закони логіки висловлювань, можна доводити логічні еквівалентності формул без використання таблиць, на основі тотожних перетворень. Покажемо на прикладах, як це робити.

**Приклад 2.17.** Застосувавши закони логіки висловлювань, доведемо логічну еквівалентність формул  $p \rightarrow (q \wedge r)$  і  $(p \rightarrow q) \wedge (p \rightarrow r)$ . Запишемо послідовність перетворень і назви використаних законів і правил:

$$\begin{aligned}
 p \rightarrow (q \wedge r) &= \bar{p} \vee (q \wedge r) && \text{(за правилом усунення імплікації)} \\
 &= (\bar{p} \vee q) \wedge (\bar{p} \vee r) && \text{(за законом дистрибутивності 3а)} \\
 &= (p \rightarrow q) \wedge (p \rightarrow r) && \text{(за правилом введення імплікації).}
 \end{aligned}$$

**Приклад 2.18.** За допомогою законів логіки висловлювань доведемо логічну еквівалентність формул  $p \rightarrow q$  та  $\bar{q} \rightarrow \bar{p}$ . Цю логічну еквівалентність називають правилом контрапозиції.

$$\begin{aligned}
 p \rightarrow q &= \bar{p} \vee q && \text{(за правилом усунення імплікації)} \\
 &= q \vee \bar{p} && \text{(за законом комутативності 1а)} \\
 &= \bar{\bar{q}} \vee \bar{p} && \text{(за законом подвійного заперечення б)} \\
 &= \bar{q} \rightarrow \bar{p} && \text{(за правилом введення імплікації).}
 \end{aligned}$$

## 2.4. НОРМАЛЬНІ ФОРМИ ЛОГІКИ ВИСЛОВЛЮВАНЬ

Говорять, що формулу  $f$  записано в *кон'юнктивній нормальній формі (КНФ)*, якщо вона має вигляд  $f = f_1 \wedge f_2 \wedge \dots \wedge f_n$  ( $n \geq 1$ ), і всі формули  $f_i$  ( $i=1, 2, \dots, n$ ) різні. Тут кожна з формул  $f_1, f_2, \dots, f_n$  – літерал або диз'юнкція різних літералів, причому контрарні пари не входять.

**Приклад 2.19.** Нехай  $p, q$  й  $r$  – атоми. Тоді  $f = (p \vee \bar{q} \vee \bar{r}) \wedge (\bar{p} \vee q)$  – формула, записана у КНФ. У ній  $f_1 = (p \vee \bar{q} \vee \bar{r})$  і  $f_2 = (\bar{p} \vee q)$ , тобто  $f_1$  – диз'юнкція літералів  $p, \bar{q}$  та  $\bar{r}$ , а  $f_2$  – диз'юнкція літералів  $\bar{p}$  та  $q$ .

Говорять, що формулу  $f$  записано в *диз'юнктивній нормальній формі (ДНФ)*, якщо  $f$  має вигляд  $f = f_1 \vee f_2 \vee \dots \vee f_n$  ( $n \geq 1$ ), і всі формули  $f_i$  ( $i=1, 2, \dots, n$ ) різні. Тут кожна з формул  $f_1, f_2, \dots, f_n$  – літерал або кон'юнкція різних літералів, причому контрарні пари не входять.

**Приклад 2.20.** Нехай  $p, q$  й  $r$  – атоми. Тоді  $f = (\bar{p} \wedge q) \vee (p \wedge \bar{q} \wedge \bar{r})$  – формула, записана у ДНФ. У ній  $f_1 = (\bar{p} \wedge q)$  і  $f_2 = (p \wedge \bar{q} \wedge \bar{r})$ ;  $f_1$  – кон'юнкція літералів  $\bar{p}$  та  $q$ , а  $f_2$  – кон'юнкція літералів  $p, \bar{q}$  й  $\bar{r}$ .

Довільну формулу можна перетворити в одну з нормальних форм, застосувавши закони логіки висловлювань. Для побудови нормальних форм необхідно виконати таку послідовність еквівалентних перетворень.

Крок 1. Застосувати правила  $f \rightarrow g = \bar{f} \vee g$  та  $f \sim g = (f \rightarrow g) \wedge (g \rightarrow f)$  (див. підрозділ 2.3) для усунення логічних операцій „ $\rightarrow$ ” та „ $\sim$ ”.

Крок 2. Застосувати закон подвійного заперечення та закони де Моргана для перенесення знака заперечення безпосередньо до атомів.

Крок 3. Застосувати відповідні закони дистрибутивності для побудови нормальної форми. Щоб побудувати КНФ, потрібно використати дистрибутивний закон для диз'юнкції щодо кон'юнкції (закон 3а з табл. 2.7). Для побудови ДНФ слід застосувати дистрибутивний закон для кон'юнкції щодо диз'юнкції (закон 3б з табл. 2.7).

**Приклад 2.21.** Побудуємо ДНФ формули  $((p \vee \bar{q}) \rightarrow r) \wedge (\bar{r} \rightarrow s)$ . Наведемо послідовність кроків та вкажемо застосовані закони логіки висловлювань.

$$\begin{aligned}
((p \vee \bar{q}) \rightarrow r) \wedge (\bar{r} \rightarrow s) &= ((\overline{p \vee \bar{q}}) \vee r) \wedge (\bar{r} \vee s) \quad (\text{усунення логічної операції „} \rightarrow \text{”}) \\
&= ((\bar{p} \wedge \bar{\bar{q}}) \vee r) \wedge (\bar{r} \vee s) \quad (\text{закон де Моргана 8a}) \\
&= ((\bar{p} \wedge q) \vee r) \wedge (\bar{r} \vee s) \quad (\text{закон подвійного заперечення 6}) \\
&= ((\bar{p} \wedge q) \wedge (r \vee s)) \vee (r \wedge (\bar{r} \vee s)) \quad (\text{закон дистрибутивності 3б}) \\
&= ((\bar{p} \wedge q \wedge r) \vee (\bar{p} \wedge q \wedge s)) \vee ((r \wedge r) \vee (r \wedge s)) \quad (\text{закон дистрибутивності 3б}) \\
&= (\bar{p} \wedge q \wedge r) \vee (\bar{p} \wedge q \wedge s) \vee (r \wedge r) \vee (r \wedge s) \quad (\text{закон асоціативності 2a}) \\
&= (\bar{p} \wedge q \wedge r) \vee (\bar{p} \wedge q \wedge s) \vee r \vee (r \wedge s) \quad (\text{закон ідемпотентності 7б}).
\end{aligned}$$

Ми одержали ДНФ. Зазначимо, що її можна спростити, якщо двічі використати закон поглинання 9б: диз'юнктивний член  $r$  поглинає члени  $(\bar{p} \wedge q \wedge r)$  та  $(r \wedge s)$ . Отже,  $(\bar{p} \wedge q \wedge s) \vee r$  – інша ДНФ заданої формули. Останні міркування свідчать, що ДНФ, загалом кажучи, не єдина.

**Приклад 2.22.** Побудуємо КНФ формули  $(p \wedge (q \rightarrow r)) \rightarrow s$ . Наведемо послідовність кроків та застосовані закони:

$$\begin{aligned}
(p \wedge (q \rightarrow r)) \rightarrow s &= (\overline{p \wedge (q \rightarrow r)}) \vee s \quad (\text{усунення логічної операції „} \rightarrow \text{”}) \\
&= (\bar{p} \vee \overline{(q \rightarrow r)}) \vee s \quad (\text{закон де Моргана 8б}) \\
&= \bar{p} \vee (\overline{q} \vee r) \vee s \quad (\text{закон асоціативності 2a}) \\
&= \bar{p} \vee (\overline{q \wedge \bar{r}}) \vee s \quad (\text{закон де Моргана 8a}) \\
&= \bar{p} \vee (q \wedge \bar{r}) \vee s \quad (\text{закон подвійного заперечення 6}) \\
&= \bar{p} \vee s \vee (q \wedge \bar{r}) \quad (\text{закон комутативності 1a}) \\
&= (\bar{p} \vee s) \vee (q \wedge \bar{r}) \quad (\text{закон асоціативності 2a}) \\
&= (\bar{p} \vee q \vee s) \wedge (\bar{p} \vee \bar{r} \vee s) \quad (\text{закон дистрибутивності 3a}).
\end{aligned}$$

Ми одержали шукану КНФ. Ця форма також не єдина, наприклад,  $(\bar{p} \vee q \vee r \vee s) \wedge (\bar{p} \vee q \vee \bar{r} \vee s) \wedge (\bar{p} \vee \bar{q} \vee \bar{r} \vee s)$  – інша КНФ заданої формули.

## 2.5. ЛОГІКА ПЕРШОГО СТУПЕНЯ

Як зазначено в підрозділі 2.2, існують речення, які не являють собою висловлювання та містять змінні. Було наведено приклад такого речення – „ $x+1=3$ ”. Речення зі змінними – це не висловлювання, але вони перетворюються на висловлювання, якщо надати змінним певних значень. Речення зі змінними дуже поширені. Вони містяться в математичних формулах і комп'ютерних програмах. Зокрема, у мовах програмування є оператори у вигляді: „Повторювати цикл доти, доки змінні  $x$  та  $y$  не стануть рівними, або припинити обчислення циклу після 100 повторень”. Позначивши як  $i$  лічильник повторень, умову закінчення циклу можна задати виразом „ $(x=y) \vee (i>100)$ ”. Тоді оператор циклу набирає вигляду: „Повторювати, якщо  $(\neg((x=y) \vee (i>100)))$ ”.

**Приклад 2.23.** Речення „ $x>3$ ”, „ $x=y+3$ ”, „ $x+y=z$ ” містять змінні. Вони не істинні й не фальшиві доти, доки змінним не буде надано якихось значень.

У наведеному прикладі речення „ $x>3$ ”, або, в іншому вигляді, „ $x$  більше 3”, складається з двох частин: першу, змінну  $x$ , називають *предметом*, а другу – „більше 3”, – яка показує властивість предмета, називають *предикатом*. Часто *предикатом* називають усе речення.

Розглянемо *логіку першого ступеня* (логіку предикатів), у якій до понять логіки висловлювань додано нові поняття [22, 29]. Для формулювання складних думок у логіці висловлювань використовують атоми як основні елементи формул. Атом розглядають як неподільне ціле – його структуру не аналізують.

Позаяк багато міркувань неможливо описати лише за допомогою висловлювань, уведемо поняття атома в логіці першого ступеня. Для запису атомів логіки першого ступеня використовують такі типи символів:

♦ *індивідні символи*, або *сталі* – це імена об'єктів, які починаються з великої букви, та сталі, наприклад, Іван, Марія, Дискретна\_математика, Т, F, 2, 5;

♦ *предметні символи*, *предметні змінні*, або просто *змінні* – імена, якими позначають змінні та записують малими буквами (можливо, з індексами), наприклад,  $x, y, z, v, w$ ;

♦ *предикатні символи* – імена, якими позначають предикати та які записують великими буквами (наприклад, P, Q, R) або змістовними словами, які записують великими буквами (наприклад, БІЛЬШЕ, ЛЮБИТЬ).

♦ *функціональні символи*. Для запису атомів логіки першого ступеня поруч з індивідними, предметними та предикатними символами використовують *функціональні символи* та *функції*. Нехай функція  $f: D^n \rightarrow D$ , де  $D$  – *предметна область*, використовують запис  $f(x_1, x_2, \dots, x_n)$ ,  $x_i \in D$ ,  $(i=1, 2, \dots, n)$ ,  $f$  називають *n-місним функціональним символом*. Функціональні символи позначають малими буквами або змістовними словами, записаними малими буквами  $f, g, h$ , *батько*, *плюс*. Індивідний символ, або сталу можна розглядати як функцію без аргументів.

**Приклад 2.24.** Функцію, яка ставить у відповідність людині на ім'я Іван її батька, позначимо як *батько* (Іван), де „*батько*” – одномісний функціональний символ. Предметна область – множина всіх людей, а *батько* (Іван) – людина із цієї області навіть тоді, коли його ім'я невідоме.

Уведемо означення *терма*:

- 1) стала є термом;
- 2) змінна є термом;
- 3) якщо  $f$  –  $n$ -місний функціональний символ та  $x_1, x_2, \dots, x_n$  – терми, то  $f(x_1, x_2, \dots, x_n)$  – терм.
- 4) жодних інших термів, крім породжених застосуванням пунктів 1–3, немає.

**Приклад 2.25.** Вираз „ $x+1$ ” запишемо у вигляді *плюс*( $x, 1$ ). Тут  $x$  та  $1$  – терми, „*плюс*” – двомісний функціональний символ. Тому *плюс*( $x, 1$ ) є термом; *плюс*(*плюс*( $x, 1$ ),  $x$ ) та *батько*(*батько*(Іван)) – також терми, з яких перший означає  $(x+1)+x$ , а другий – Іванового дідуся.

Якщо  $P$  –  $n$ -місний предикатний символ і  $x_1, x_2, \dots, x_n$  – терми, то  $P(x_1, x_2, \dots, x_n)$  – атом логіки першого ступеня.

**Приклад 2.26.** Вираз „ $x + y$ ” можна зобразити функцією *плюс*( $x, y$ ), а *батько*( $x$ )

означає „батько людини  $x$ ”. Речення „ $x + 1 > x$ ” записують за допомогою функцій та предикатних символів у вигляді *БІЛЬШЕ* (плюс( $x$ , 1),  $x$ ), речення „батько Івана любить Івана” як *ЛЮБИТЬ*(батько(Іван), Іван), а речення „Іван любить Марію” як *ЛЮБИТЬ* (Іван, Марія). Тут *БІЛЬШЕ* ( $x$ , 3), *ЛЮБИТЬ* (Іван, Марія), *БІЛЬШЕ* (плюс( $x$ , 1),  $x$ ) та *ЛЮБИТЬ* (батько (Іван), Іван) – атоми логіки першого ступеня, де *БІЛЬШЕ* та *ЛЮБИТЬ* – предикатні символи;  $x$  – змінна; 1, 3, Іван, Марія – індивідні символи (сталі); *батько* та *плюс* – функціональні символи.

Нехай  $P(x)$  – предикат,  $D$  – предметна область. Використовують два спеціальні символи  $\forall$  та  $\exists$ , які називають, відповідно, *кванторами загальності й існування*. Якщо  $x$  – предметна змінна, то вираз  $(\forall x)$  читають „для всіх  $x$ ”, „для кожного  $x$ ” або „для будь-якого  $x$ ”. Вираз  $(\forall x)P(x)$  означає, що „ $P(x)$  істинний для всіх значень  $x$  з предметної області  $D$ ”; його читають як „ $P(x)$  для всіх  $x$ ”.

Вираз  $(\exists x)$  читають „існує  $x$ ”, „для деяких  $x$ ” або „принаймні для одного  $x$ ”. Вираз  $(\exists x)P(x)$  означає, що „в області  $D$  існує таке  $x$ , що  $P(x)$  істинний”, або що „в області  $D$  існує принаймні одне  $x$  таке, що предикат  $P(x)$  істинний”, або що „предикат  $P(x)$  істинний для якогось  $x$  з області  $D$ ”. У подальшому дужки біля квантора будемо випускати, тобто замість  $(\forall x)$  та  $(\exists x)$  писатимемо відповідно  $\forall x$  та  $\exists x$ .

Перехід від  $P(x)$  до  $\forall xP(x)$  або  $\exists xP(x)$  називають *зв'язуванням* предметної змінної  $x$ , а саму змінну  $x$  – *зв'язаною*. Незв'язану змінну називають *вільною*. Говорять, що у виразах  $\forall xP(x)$  та  $\exists xP(x)$  предикат  $P(x)$  є в області дії відповідного квантора.

**Приклад 2.27.** У виразі  $\exists xP(x, y)$  змінна  $x$  зв'язана, а змінна  $y$  – *вільна*, бо предикат  $P(x, y)$  не знаходиться в області дії квантора зі змінною  $y$ .

*Правильно побудовані формули логіки першого ступеня, або формули логіки першого ступеня, визначають так:*

- ◆ атом – це формула;
- ◆ якщо  $H$  і  $G$  – формули, то  $(\neg H)$ ,  $(H \wedge G)$ ,  $(H \vee G)$ ,  $(H \rightarrow G)$  та  $(H \sim G)$  – формули;
- ◆ якщо  $H$  формула, а  $x$  – вільна змінна у формулі  $H$ , то  $\forall xH$  та  $\exists xH$  – формули;
- ◆ формули можна породити тільки скінченною кількістю застосувань попередніх трьох правил.

Зазначимо, що замість  $(\neg H)$  можна писати  $\bar{H}$ .

Інтерпретація формул у логіці висловлювань полягає в наданні атомам значень істинності. Це дає змогу знайти значення істинності цих формул. У логіці першого ступеня для знаходження значень істинності формул треба вказувати предметну область, значення сталих, предикатних і функціональних символів, які зустрічаються у формулі.

Отже, *інтерпретація* формули  $G$  логіки першого ступеня полягає у визначенні:

- ◆ непорожньої предметної області  $D$ ;
- ◆ значення сталої із предметної області  $D$ ;
- ◆ значення  $n$ -місного функціонального символу на множині  $D$ ;
- ◆ елемента множини  $\{T, F\}$  як значення  $n$ -місного предикатного символу.

У кожній інтерпретації формула отримує у предметній області  $D$  значення істинності за такими правилами.

- 1) кожній константі ставлять у відповідність значення з  $D$ .
- 2) якщо задані значення істинності формул  $P$  та  $Q$ , то значення істинності формул

$\bar{P}$ ,  $(P \wedge Q)$ ,  $(P \vee Q)$ ,  $(P \rightarrow Q)$  та  $(P \sim Q)$  отримують із таблиць істинності.

3) формула  $\forall xP$  отримує значення  $T$ , якщо  $P$  істинний для кожного  $x \in D$ , інакше вона отримує значення  $F$ .

4) формула  $\exists xP$  отримує значення  $T$ , якщо  $P$  істинний принаймні для одного  $x \in D$ , інакше вона отримує значення  $F$ .

Для заданої інтерпретації всяка формула без вільних змінних (або, інакше, *замкнена формула*) являє собою висловлювання, яке істинне чи фальшиве. Формула, яка містить вільні змінні, не може отримати значення істинності.

**Приклад 2.28.** Знайдемо значення істинності формул  $\forall xP(x)$  та  $\exists x\bar{P}(x)$  в інтерпретації, яка складається із предметної області  $D = \{1, 2\}$  та таких значень одномісного предикатного символу  $P$ :  $P(1) = T$ ,  $P(2) = F$ . Оскільки  $P(2) = F$ , то  $\forall xP(x) = F$ . Оскільки в цій інтерпретації  $P(2) = F$ , то  $\exists x\bar{P}(x) = T$ .

**Приклад 2.30.** Знайдемо значення істинності формули  $\forall x(P(x) \rightarrow Q(f(x), a))$  в інтерпретації, яка складається із предметної області  $D = \{1, 2\}$ , значення сталої  $a = 1$ , значень одномісного функціонального символу  $f$ :  $f(1) = 2$ ,  $f(2) = 1$ , та значень двомісних предикатних символів  $P$  та  $Q$ :  $P(1) = F$ ,  $P(2) = T$ ,  $Q(1, 1) = T$ ,  $Q(1, 2) = T$ ,  $Q(2, 1) = F$ ,  $Q(2, 2) = T$ .

Якщо  $x = 1$ , то

$$P(x) \rightarrow Q(f(x), a) = P(1) \rightarrow Q(f(1), a) = P(1) \rightarrow Q(2, 1) = F \rightarrow F = T.$$

Якщо  $x = 2$ , то  $P(x) \rightarrow Q(f(x), a) = P(2) \rightarrow Q(f(2), a) = P(2) \rightarrow Q(1, 1) = T \rightarrow T = T$ .

Оскільки  $P(x) \rightarrow Q(f(x), a)$  істинне для довільного елемента  $x$  з  $D$ , то формула  $\forall x(P(x) \rightarrow Q(f(x), a))$  істинна у вказаній інтерпретації.

## 2.6. ЗАКОНИ ЛОГІКИ ПЕРШОГО СТУПЕНЯ

Логічно еквівалентні формули логіки висловлювань залишаються такими й у логіці першого ступеня. Однак, у логіці першого ступеня є логічні еквівалентності (зако́ни), пов'язані зі специфікою визначення об'єктів логіки першого ступеня.

Дві формули логіки предикатів називають *логічно еквівалентними*, якщо вони набувають однакових значень істинності в довільних інтерпретаціях для будь-яких значень змінних із предметної області. Зокрема, якщо формули  $P$  та  $Q$  логічно еквівалентні, то формула  $P \sim Q$  – тавтологія, і навпаки. Логічну еквівалентність формул  $P$  та  $Q$  записують як  $P = Q$  (часто використовують знаки  $\Leftrightarrow$  та  $\equiv$ :  $P \Leftrightarrow Q$ ,  $P \equiv Q$ ). Проблема побудови законів логіки першого ступеня полягає в доведенні логічної еквівалентності формул  $P$  та  $Q$ .

Нижче наведено *основні закони логіки першого ступеня*. Зазначимо, що у формулах показано лише зв'язані змінні, а вільні змінні не показано.

1.  $\overline{\forall xP(x)} = \exists x\bar{P}(x)$ .
2.  $\overline{\exists xP(x)} = \forall x\bar{P}(x)$ .
3.  $\forall x(P(x) \wedge Q(x)) = \forall xP(x) \wedge \forall xQ(x)$ .
4.  $\exists x(P(x) \vee Q(x)) = \exists xP(x) \vee \exists xQ(x)$ .

5.  $\forall x(P(x) \wedge Q) = \forall xP(x) \wedge Q$ .
6.  $\forall x(P(x) \vee Q) = \forall xP(x) \vee Q$ .
7.  $\exists x(P(x) \wedge Q) = \exists xP(x) \wedge Q$ .
8.  $\exists x(P(x) \vee Q) = \exists xP(x) \vee Q$ .
9.  $\forall x\forall yP(x, y) = \forall y\forall xP(x, y)$ .
10.  $\exists x\exists yP(x, y) = \exists y\exists xP(x, y)$ .

Для доведення цих законів потрібні спеціальні методи. Проілюструємо це на прикладі доведення логічної еквівалентності  $\overline{\exists xP(x)} = \forall x\overline{P(x)}$ . Нехай для деякого предикатного символу  $P$  та предметної області  $D$  ліва частина логічної еквівалентності істинна. Тоді не існує такого  $a \in D$ , для якого  $P(a)$  істинне. Отже,  $P(a)$  фальшиве для довільного  $a$ , а  $\overline{P(a)}$  – істинне, тому істинна права частина еквівалентності. Якщо ліва частина логічної еквівалентності фальшива, то існує таке  $a \in D$ , для якого  $P(a)$  істинне, тобто права частина фальшива. Аналогічно доводять і закон  $\overline{\forall xP(x)} = \exists x\overline{P(x)}$ .

**Приклад 2.31.** Цей приклад ілюструє логічну еквівалентність  $\overline{\forall xP(x)} = \exists x\overline{P(x)}$ . Розглянемо заперечення речення „Кожний студент університету вивчає математичний аналіз”. Це речення записують з використанням квантора загальності як  $\forall xP(x)$ , де  $P(x)$  – речення „ $x$  вивчає математичний аналіз”. Запереченням заданого речення є „Це не так, що кожний студент університету вивчає математичний аналіз”, яке логічно еквівалентне реченню „Існує такий студент університету, який не вивчає математичного аналізу”.

**Приклад 2.32.** Розглянемо речення „В університеті є студент, який вивчає математичний аналіз”. Це речення можна записати як  $\exists xP(x)$ , де  $P(x)$  речення „ $x$  вивчає математичний аналіз”. Заперечення заданого речення – „Це не так, що в університеті є студент, який вивчає математичний аналіз” – логічно еквівалентне реченню „Кожний студент університету не вивчає математичний аналіз”. Останнє отримують застосуванням квантора загальності до заперечення заданого речення, тобто  $\forall x\overline{P(x)}$ , що ілюструє логічну еквівалентність  $\overline{\exists xP(x)} = \forall x\overline{P(x)}$ .

Доведемо закон  $\overline{\forall x(P(x) \wedge Q(x))} = \forall x\overline{P(x) \wedge Q(x)}$ . Нехай ліва частина істинна для деяких  $P$  та  $Q$ , тобто для довільного  $a \in D$  істинне  $P(a) \wedge Q(a)$ . Тому  $P(a)$  та  $Q(a)$  одночасно істинні для довільного  $a$ , тобто  $\forall xP(x) \wedge \forall xQ(x)$  істинне. Якщо ж ліва частина фальшива, то для деякого  $a \in D$  фальшиве принаймні одне з висловлювань  $P(a)$  або  $Q(a)$ . Це означає, що фальшиве принаймні одне з висловлювань  $\forall xP(x)$  або  $\forall xQ(x)$ , тобто фальшива й права частина. Аналогічно доводять логічну еквівалентність  $\overline{\exists x(P(x) \vee Q(x))} = \exists x\overline{P(x) \vee Q(x)}$ .

Зазначимо, що в законах 9 і 10 змінні в предикатах зв'язані однаковими кванторами, тому їх можна переставляти без порушення еквівалентності формул. Якщо ж квантори різні, подібна еквівалентність виконується не завжди, тобто загалом  $\forall x\exists yP(x, y) \neq \exists y\forall xP(x, y)$ . Наведемо приклад, який ілюструє це зауваження.

**Приклад 2.33.** Розглянемо двомісний предикат  $P(x, y)$ : „ $x \geq y$ ” на різних предметних областях. Формула  $\exists x\forall yP(x, y)$  означає, що в предметній області існує максимальний елемент. Ця формула істинна на предметній області, що являє собою будь-яку скінченну підмножину множини цілих чисел, але фальшива, наприклад, на множині  $\{1/2, 2/3, 3/4, \dots, n/(n+1), \dots\}$ . Висловлювання  $\forall y\exists xP(x, y)$  стверджує, що для довільного елемента  $y$  існує елемент  $x$ , не менший від  $y$ . Таке висловлювання істинне на довільній непорожній множині. Отже, переставлення кванторів існування та загальності може змінити зміст висловлювання та значення його істинності.

Якщо  $D = \{a_1, a_2, \dots, a_n\}$  – скінченна предметна область змінної  $x$  у предикаті  $P(x)$ , то можна скористатись логічними еквівалентностями  $\forall xP(x) = P(a_1) \wedge P(a_2) \wedge \dots \wedge P(a_n)$  та  $\exists xP(x) = P(a_1) \vee P(a_2) \vee \dots \vee P(a_n)$ . У такому разі заперечення квантифікованої формули дає той самий результат, що й застосування відповідного закону де Моргана. Це випливає з того, що

$$\overline{\forall xP(x)} = \overline{P(a_1) \wedge P(a_2) \wedge \dots \wedge P(a_n)} = \overline{P(a_1)} \vee \overline{P(a_2)} \vee \dots \vee \overline{P(a_n)},$$

остання формула логічно еквівалентна  $\exists x\overline{P(x)}$ .

Аналогічно,

$$\overline{\exists xP(x)} = \overline{P(a_1) \vee P(a_2) \vee \dots \vee P(a_n)} = \overline{P(a_1)} \wedge \overline{P(a_2)} \wedge \dots \wedge \overline{P(a_n)},$$

що логічно еквівалентно  $\forall x\overline{P(x)}$ .

## 2.7. ВИПЕРЕДЖЕНА НОРМАЛЬНА ФОРМА

Формулу логіки першого ступеня  $Q_1x_1Q_2x_2\dots Q_nx_nM$ , де  $Q_i x_i$  ( $i=1, 2, \dots, n$ ) – квантор загальності або існування,  $x_i$  та  $x_j$  різні для  $i \neq j$ , а  $M$  не містить кванторів, називають формулою у *випередженій нормальній формі*. (Сюди входить і випадок  $n=0$ , коли взагалі немає ніяких кванторів.) Вираз  $Q_1x_1\dots Q_nx_n$  називають *префіксом*, а  $M$  – *матрицею* формули у випередженій нормальній формі. Для будь-якої формули можна побудувати логічно еквівалентну формулу у випередженій нормальній формі.

**Приклад 2.34.** Надамо приклади формул у випередженій нормальній формі:

- 1)  $\forall x\forall y(P(x, y) \wedge Q(y))$ ;
- 2)  $\forall x\exists y(\overline{P(x)} \vee Q(y))$ ;
- 3)  $\forall x\forall y\exists z(Q(x, y) \wedge R(z))$ ;
- 4)  $\forall x\exists y\forall z((P(x, y) \vee Q(x, z)) \wedge \overline{R(y, z)})$ ;
- 5)  $\forall x\forall y\forall z\exists u(\overline{P(x, z)} \vee \overline{P(y, z)} \vee Q(x, y, u))$ .

**Алгоритм зведення довільної формули логіки першого ступеня до випередженої нормальної форми.**

Крок 1. Усунути з формули логічні операції „ $\sim$ ” та „ $\rightarrow$ ” застосуванням логічних еквівалентностей  $P \sim Q = (P \rightarrow Q) \wedge (Q \rightarrow P)$  та  $P \rightarrow Q = \overline{P} \vee Q$ .

Крок 2. Внести знак заперечення всередину формули безпосередньо до атома, для чого використати закони:

- ♦ подвійного заперечення  $\overline{\overline{P}} = P$ ;



♦ де Моргана  $\overline{P \vee Q} = \overline{P} \wedge \overline{Q}$  та  $\overline{P \wedge Q} = \overline{P} \vee \overline{Q}$ ;

♦  $\overline{\forall x P(x)} = \exists x \overline{P(x)}$  та  $\overline{\exists x P(x)} = \forall x \overline{P(x)}$ .

Перейменувати зв'язані змінні, якщо це потрібно,

Крок 3. Винести квантори у префікс, для чого скористатись законами 3–8 з підрозділу 2.6.

**Приклад 2.35.** Зведемо формулу  $\forall x P(x) \rightarrow \exists y Q(y)$  до випередженої нормальної форми.

$$\begin{aligned} \forall x P(x) \rightarrow \exists y Q(y) &= \overline{\forall x P(x)} \vee \exists y Q(y) \quad (\text{виключено логічну операцію „} \rightarrow \text{”}) \\ &= \exists x \overline{P(x)} \vee \exists y Q(y) \quad (\text{застосовано закон } \overline{\forall x P(x)} = \exists x \overline{P(x)}) \\ &= \exists x \exists y (\overline{P(x)} \vee Q(y)) \quad (\text{квантор існування винесено у префікс за законом 8 з підрозділу 2.6}). \end{aligned}$$

**Приклад 2.36.** Побудуємо випереджену нормальну форму для формули  $\forall x \forall y (\exists z (P(x, z) \wedge P(y, z)) \rightarrow \exists u Q(x, y, u))$ . Нижче подано процес побудови формули.

$$\begin{aligned} \forall x \forall y (\exists z (P(x, z) \wedge P(y, z)) \rightarrow \exists u Q(x, y, u)) &= \\ &= \forall x \forall y (\overline{\exists z (P(x, z) \wedge P(y, z))} \vee \exists u Q(x, y, u)) \quad (\text{виключено логічну операцію „} \rightarrow \text{”}) \\ &= \forall x \forall y (\overline{\exists z (\overline{P(x, z)} \wedge \overline{P(y, z)})} \vee \exists u Q(x, y, u)) \end{aligned}$$

(застосовано закон  $\overline{\exists x P(x)} = \forall x \overline{P(x)}$ )

$$= \forall x \forall y (\forall z (\overline{\overline{P(x, z)}} \vee \overline{\overline{P(y, z)}}) \vee \exists u Q(x, y, u)) \quad (\text{застосовано закон де Моргана})$$

$= \forall x \forall y \forall z \exists u (\overline{\overline{P(x, z)}} \vee \overline{\overline{P(y, z)}} \vee Q(x, y, u))$  (за законами 6 та 8 підрозділу 2.6 у префікс формули винесено  $\forall z$  та  $\exists u$ ).

## 2.8. ЛОГІЧНЕ ВИВЕДЕННЯ В ЛОГІЦІ ВИСЛОВЛЮВАНЬ

Говорять, що формула  $g$  – логічний наслідок формул  $f_1, f_2, \dots, f_n$ , або що  $g$  логічно випливає з  $f_1, f_2, \dots, f_n$ , якщо в кожній інтерпретації, у якій виконується формула  $f_1 \wedge f_2 \wedge \dots \wedge f_n$ , формула  $g$  також виконується. Формули  $f_1, f_2, \dots, f_n$  називають гіпотезами (аксіомами, постулатами чи засновками) формули  $g$ . Той факт, що формула  $g$  логічно випливає з  $f_1, f_2, \dots, f_n$ , позначають  $f_1, f_2, \dots, f_n \vdash g$ .

**Теорема 2.2.** Формула  $g$  – логічний наслідок формул  $f_1, f_2, \dots, f_n$  тоді й лише тоді, коли формула  $(f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g$  загальнозначуща.

**Доведення. Необхідність.** Нехай  $g$  – логічний наслідок формул  $f_1, f_2, \dots, f_n$  та  $I$  – довільна їх інтерпретація. Якщо формули  $f_1, f_2, \dots, f_n$  істинні в інтерпретації  $I$ , то за означенням логічного наслідку формула  $g$  також істинна в  $I$ . Звідси випливає, що формула  $(f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g$  істинна в інтерпретації  $I$ . З іншого боку, якщо не всі формули з  $f_1, f_2, \dots, f_n$  істинні в інтерпретації  $I$ , тобто принаймні одна з них фальшива в  $I$ , то й формула  $f_1 \wedge f_2 \wedge \dots \wedge f_n$  фальшива в інтерпретації  $I$ , а формула  $(f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g$  –

істинна в  $I$ . Отже, формула  $(f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g$  істинна в довільній інтерпретації. Це й означає, що вона загальнозначуща,  $\vdash (f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g$ .

**Достатність.** Припустимо, що формула  $(f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g$  – загальнозначуща. Тоді, якщо формула  $f_1 \wedge f_2 \wedge \dots \wedge f_n$  істинна в якійсь інтерпретації, то й формула  $g$  має бути істинною в цій інтерпретації. Тобто,  $g$  – логічний наслідок формул  $f_1, f_2, \dots, f_n$ . Теорему доведено.

Якщо  $g$  – логічний наслідок формул  $f_1, f_2, \dots, f_n$ , то формулу  $(f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g$  називають логічною теоремою, а  $g$  – її висновком. У такому разі говорять, що формулу  $g$  можна вивести з формул  $f_1, f_2, \dots, f_n$ , і  $g$  – вивідна формула. Вираз  $f_1, f_2, \dots, f_n \vdash g$  називають правилом виведення. Тут гіпотези записано зліва від знаку  $\vdash$ , а висновок – справа; сам знак  $\vdash$  має зміст „отже”.

**Теорема 2.3 (принцип прямої дедукції).** Формула  $g$  – логічний наслідок формул  $f_1, f_2, \dots, f_n$  тоді й лише тоді, коли формула  $f_1 \wedge f_2 \wedge \dots \wedge f_n \wedge \overline{g}$  – суперечність.

**Доведення.** За теоремою 2.2 формула  $g$  – логічний наслідок формул  $f_1, f_2, \dots, f_n$  тоді й лише тоді, коли формула  $(f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g$  загальнозначуща. Отже,  $g$  – логічний наслідок формул  $f_1, f_2, \dots, f_n$  тоді й лише тоді, коли формула  $(f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g$  – суперечність.

Далі,

$$\begin{aligned} \overline{(f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g} &= \overline{(f_1 \wedge f_2 \wedge \dots \wedge f_n) \vee \overline{g}} = \\ &= \overline{(f_1 \wedge f_2 \wedge \dots \wedge f_n)} \wedge \overline{\overline{g}} = f_1 \wedge f_2 \wedge \dots \wedge f_n \wedge \overline{g}. \end{aligned}$$

**Приклад 2.37.** Розглянемо формули  $f_1 = (p \rightarrow q)$ ,  $f_2 = \overline{q}$ ,  $g = \overline{p}$ . Покажемо, що формула  $g$  – логічний наслідок формул  $f_1$  та  $f_2$ .

**Спосіб 1.** Скористаємося таблицями істинності, щоб показати, що формула  $g$  виконується в кожній інтерпретації, у якій виконується формула  $(p \rightarrow q) \wedge \overline{q}$ . З табл. 2.8 видно, що є лише одна інтерпретація, у якій  $(p \rightarrow q) \wedge \overline{q}$  виконується, а саме  $p=F, q=F$ ; у цій інтерпретації формула  $\overline{p}$  також виконується. Отже, за означенням формула  $\overline{p}$  є логічним наслідком формул  $p \rightarrow q$  та  $\overline{q}$ .

Таблиця 2.8

| $p$ | $q$ | $p \rightarrow q$ | $\overline{q}$ | $(p \rightarrow q) \wedge \overline{q}$ | $\overline{p}$ |
|-----|-----|-------------------|----------------|---|----------------|
| T   | T   | T                 | F              | F                                       | F              |
| T   | F   | F                 | T              | F                                       | F              |
| F   | T   | T                 | F              | F                                       | T              |
| F   | F   | T                 | T              | T                                       | T              |

**Спосіб 2.** Скористаємося теоремою 2.2. Покажемо, що формула  $(f_1 \wedge f_2) \rightarrow g$  загальнозначуща. Для цього побудуємо табл. 2.9 для формули  $(f_1 \wedge f_2) \rightarrow g = ((p \rightarrow q) \wedge \overline{q}) \rightarrow \overline{p}$ .

Таблиця 2.9

| $p$ | $q$ | $((p \rightarrow q) \wedge \bar{q}) \rightarrow \bar{p}$ |
|-----|-----|--|
| T   | T   | T  |
| T   | F   | T  |
| F   | T   | T  |
| F   | F   | T  |

Оскільки формула  $((p \rightarrow q) \wedge \bar{q}) \rightarrow \bar{p}$  загальнозначуща, то формула  $\bar{p}$  – логічний наслідок формул  $p \rightarrow q$  та  $\bar{q}$ .

Таблиця 2.10

| $p$ | $q$ | $p \rightarrow q$ | $\bar{q}$ | $(p \rightarrow q) \wedge \bar{q} \wedge p$ |
|-----|-----|-------------------|-----------|---|
| T   | T   | T                 | F         | F   |
| T   | F   | F                 | T         | F   |
| F   | T   | T                 | F         | F   |
| F   | F   | T                 | T         | F   |

Спосіб 3. Скористаємось теоремою 2.3 та покажемо, що формула

$$f_1 \wedge f_2 \wedge \bar{g} = (p \rightarrow q) \wedge \bar{q} \wedge \bar{p} = (p \rightarrow q) \wedge \bar{q} \wedge p$$

заперечувана. Побудуємо таблицю істинності для цієї формули. З табл. 2.10 можна побачити, що формула  $(p \rightarrow q) \wedge \bar{q} \wedge p$  фальшива в кожній інтерпретації.

Тепер відповідно до теореми 2.3 можна дійти висновку, що формула  $\bar{p}$  логічно випливає з формул  $p \rightarrow q$  та  $\bar{q}$ .

## 2.9 АЛГОРИТМ КУАЙНА.

Доведення того, що формула  $g$  є логічним наслідком формул  $f_1, f_2, \dots, f_n$ , полягає в перевірці загальнозначущості формули  $(f_1 \wedge f_2 \wedge \dots \wedge f_n) \rightarrow g$  обчисленням її в усіх повних інтерпретаціях. Для формули, базис якої складається з  $n$  атомів, тривіальний алгоритм вимагає розгляду  $2^n$  повних інтерпретацій, що неефективно. Обчислення формули у повній інтерпретації відповідає знаходженню значень істинності цієї формули в листку семантичного дерева, побудованого для її базису.

Алгоритм Куайна є вдосконаленням тривіального алгоритму, який обговорювали в підрозділі 2.2. Якщо формула набуває істинне або фальшиве значення у внутрішній вершині  $N$  семантичного дерева, яка відповідає частковій інтерпретації  $I(N)$  цієї формули, то вона також приймає це значення у всіх вершинах піддерева, для якого вершина  $N$  є коренем. У такому разі продовжувати обчислення формули в її часткових інтерпретаціях, які відповідають таким вершинам, недоцільно. Це дозволяє не обчислювати формули в усіх повних інтерпретаціях, і в такий спосіб зменшити загальний обсяг обчислень.

**Приклад 2.38.** Задано формулу  $f = ((p \wedge q) \rightarrow r) \wedge (p \rightarrow q) \rightarrow (p \rightarrow r)$ . Покажемо, що перевірку її загальнозначущості можна виконати обчисленням лише в часткових інтерпретаціях. Це дасть змогу зменшити загальну кількість обчислень. Базис

формули  $f$  – множина  $P = \{p, q, r\}$ , а семантичне дерево, яке використаємо для обчислення часткових інтерпретацій, наведено на рис. 2.2.

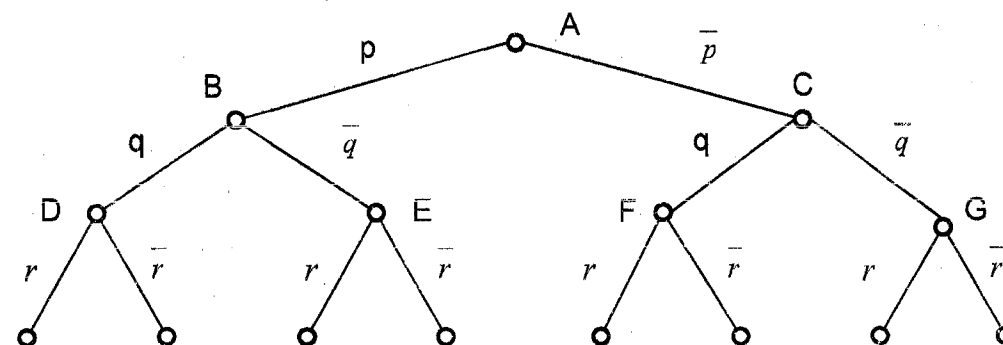


Рис. 2.2

Розглянемо таку послідовність часткових інтерпретацій.

1.  $I(B) = \{p\}$  або  $p = T$ . У такій інтерпретації формула  $f$  набуде вигляду  $((T \wedge q) \rightarrow r) \wedge (T \rightarrow q) \rightarrow (T \rightarrow r) = ((q \rightarrow r) \wedge q) \rightarrow r$ . Отримана формула може набувати різні значення істинності, отже, її обчислення треба продовжити.

2.  $I(D) = \{p, q\}$  або  $p = T$  та  $q = T$ . Формула  $f$  в інтерпретації  $I(D)$  має істинне значення:

$$(((T \rightarrow r) \wedge T) \rightarrow r) = (r \wedge T) \rightarrow r = r \rightarrow r = T.$$

Продовжувати її обчислення у вершинах піддерева з коренем у вершині  $D$  немає потреби, бо формула  $f$  у цих вершинах теж істинна.

3.  $I(E) = \{p, \bar{q}\}$  або  $p = T$  та  $q = F$ . Формула  $f$  в інтерпретації  $I(E)$  набуває істинне значення:

$$((q \rightarrow r) \wedge q) \rightarrow r = ((F \rightarrow r) \wedge F) \rightarrow r = (T \wedge F) \rightarrow r = F \rightarrow r = T.$$

Тому її обчислення у вершинах піддерева з коренем у вершині  $E$  не виконуємо.

4.  $I(C) = \{\bar{p}\}$  або  $p = F$ . Формула  $f$  набуває істинне значення:

$$(((F \wedge q) \rightarrow r) \wedge (F \rightarrow q)) \rightarrow (F \rightarrow r) = ((F \rightarrow r) \wedge (F \rightarrow q)) \rightarrow (F \rightarrow r) = (T \wedge T) \rightarrow T = T.$$

Формулу  $f$  обчислено лише в чотирьох часткових інтерпретаціях  $I(B)$ ,  $I(D)$ ,  $I(E)$ ,  $I(C)$ ; вона істинна у вершинах  $D$ ,  $E$  та  $C$  семантичного дерева. Тому вона істинна в усіх вершинах піддерев із коренями у зазначених вершинах. Отже, доведено загальнозначущість формули  $f$ .

Алгоритм Куайна також можна використати для перевірки невиконаності формули.

**Диз'юнкт** (або **елементарна диз'юнкція**) – це літерал або диз'юнкція різних літералів, яка не містить контрарних пар. **Рангом диз'юнкта** називають кількість літералів у ньому. Диз'юнкт із рангом 0 називають порожнім і позначають як  $\square$ . Надання літералу в диз'юнкті значення  $F$  називають **викреслюванням** цього літерала. Порожній диз'юнкт одержують викреслюванням усіх його літералів. Якщо формулу записано в кон'юнктивній нормальній формі  $f = d_1 \wedge d_2 \wedge \dots \wedge d_m$ , то кожна з формул  $d_1, d_2, \dots, d_m$  – диз'юнкт, а саму формулу  $f$  ще зображають як множину її диз'юнктів  $S = \{d_1, d_2, \dots, d_m\}$ . У літературі зі штучного інтелекту диз'юнкт називають також **клаузою**.

Розглянемо модифікований алгоритм Куайна, у якому формула  $f$  записана в кон'юнктивній нормальній формі як множина диз'юнктів  $S$ . У цьому разі проблема

доведення невиконанності значно спрощується, бо перевірку невиконанності здійснюють для кожного диз'юнкту окремо. Множина  $S$  невиконанна, якщо невиконанна формула, яка їй відповідає, та виконанна, якщо ця формула виконанна. Доведення невиконанності множини  $S$  називають її *спростуванням*.

Нехай кон'юнктивну нормальну форму задано множиною  $S$  її диз'юнктів,  $P$  – базис формули, а  $p$  ( $p \in P$ ) – атом цього базису. Розіб'ємо множину  $S$  на підмножини:  $S_{\bar{p}}$  – підмножина диз'юнктів із літералом  $\bar{p}$ ,  $S_p$  – підмножина диз'юнктів із літералом  $p$ , та  $S'' = S \setminus (S_p \cup S_{\bar{p}})$  – підмножина, яка містить решту диз'юнктів. Уведемо позначення:

- $S'_p$  – множина, отримана з  $S_p$  викреслюванням літерала  $p$  з її диз'юнктів;
- $S'_{\bar{p}}$  – множина, отримана з  $S_{\bar{p}}$  викреслюванням літерала  $\bar{p}$  з її диз'юнктів;
- $S_1 = S'_p \cup S''$ ;
- $S_2 = S'_{\bar{p}} \cup S''$ .

### Модифікований алгоритм Куайна

Крок 1. Задану множину диз'юнктів  $S$  та її базис  $P$  поміщаємо до стека як пару  $(P, S)$ .

Крок 2. Вибираємо з верхівки стека пару  $(P, S)$ . З базису  $P$  вибираємо довільний елемент  $p$  та вилучаємо його з базису.

Крок 3. Будуємо множини  $S_p$  та  $S_{\bar{p}}$ . Якщо множини  $S_p$  або  $S_{\bar{p}}$  порожні, то множина  $S$  – виконанна. Кінець.

Крок 4. Будуємо множини  $S'_p$ ,  $S'_{\bar{p}}$ ,  $S_1$  та  $S_2$ .

Крок 5. Якщо множина  $S_1$  не містить порожнього диз'юнкту, то до стека поміщаємо пару  $(P, S_1)$ . Якщо множина  $S_2$  не містить порожнього диз'юнкту, то до стека поміщаємо пару  $(P, S_2)$ .

Крок 6. Якщо стек порожній, то задана множина невиконанна. Кінець. Інакше повертаємося до кроку 2.

Множина  $S$  невиконанна тоді й лише тоді, коли невиконанні множини  $S_1 = S'_p \cup S''$  та  $S_2 = S'_{\bar{p}} \cup S''$ . Модифікований алгоритм Куайна рекурсивний, тобто проблема перевірки невиконанності множини  $S$ , яка містить літерали  $p$  та  $\bar{p}$ , зводиться до вирішення проблеми невиконанності множин  $S_1$  та  $S_2$ , які не містять цих літералів. Доведення невиконанності множини  $S$  еквівалентне доведенню невиконанності множини  $S_1 = S'_p \cup S''$  для  $p=F$  та множини  $S_2 = S'_{\bar{p}} \cup S''$  для  $p=T$ .

Виконання модифікованого алгоритму Куайна ілюструють побудовою семантичного дерева, або *дерева виведення*, у якого вершини позначено так:

♦ вершина з міткою  $S'_p \cup S''$  – це кінцева вершина ребра, позначеного літералом  $p$ ;

♦ вершина з міткою  $S'_{\bar{p}} \cup S''$  – це кінцева вершина ребра, позначеного літералом  $\bar{p}$ .

Проілюструємо роботу модифікованого алгоритму Куайна на прикладі.

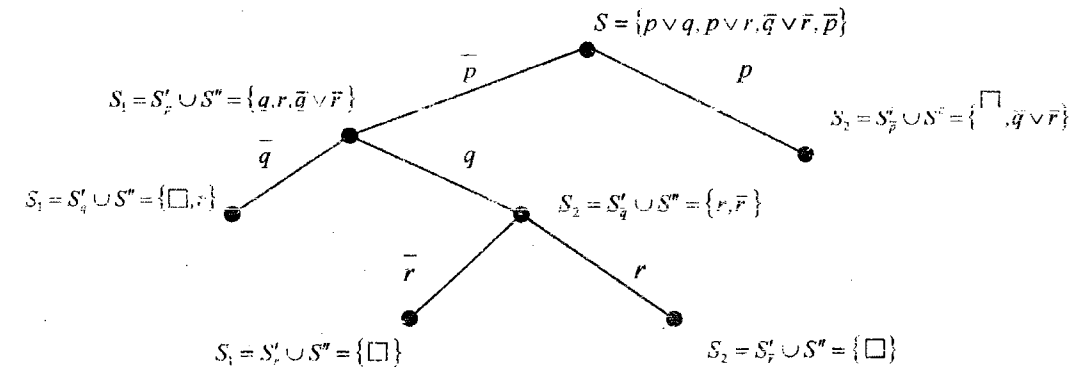


Рис. 2.3

**Приклад 2.39.** Доведемо вивідність  $p$  з формул  $p \vee q, p \vee r, \bar{q} \vee \bar{r}$ , що згідно з принципом прямої дедукції полягає в перевірці невиконанності множини диз'юнктів  $S = \{p \vee q, p \vee r, \bar{q} \vee \bar{r}, \bar{p}\}$ . Базис цієї множини –  $P = \{p, q, r\}$ . Спростування множини  $S$  виконаємо за модифікованим алгоритмом Куайна, семантичне дерево (дерево виведення) зображено на рис. 2.3. Вершини дерева позначено множинами, утвореними в процесі спростування.

Опишемо всі кроки виконання алгоритму.

Крок 1. Помістимо пару  $(P, S)$  до стека.

Крок 2. Вибираємо з верхівки стека пару  $(P, S)$ , а з  $P$  – атом  $p$ , який виключаємо із множини  $P$ . Отримаємо  $P = \{q, r\}$  та  $S$  – задану множину.

Крок 3. Будуємо множини  $S_p = \{p \vee q, p \vee r\}$ ,  $S_{\bar{p}} = \{\bar{p}\}$ . Множини  $S_p$  та  $S_{\bar{p}}$  непорожні.

Крок 4. Будуємо множини  $S'' = \{\bar{q} \vee \bar{r}\}$ ,  $S'_p = \{q, r\}$ ,  $S'_{\bar{p}} = \{\square\}$ ,

$S_1 = S'_p \cup S'' = \{q, r, \bar{q} \vee \bar{r}\}$ ,  $S_2 = S'_{\bar{p}} \cup S'' = \{\square, \bar{q} \vee \bar{r}\}$ . Очевидно, що  $S_2$  – невиконанна, бо містить порожній диз'юнкт.

Крок 5. Множина  $S_1$  не містить порожнього диз'юнкта. Поміщаємо пару  $(P, S_1)$  у стек.

Крок 6. Оскільки стек не порожній, то повертаємось до кроку 2.

Крок 2. Вибираємо з верхівки стека пару  $(P, S)$ , а з  $P$  – атом  $q$ , який виключаємо із множини  $P$ . Отримаємо  $P = \{r\}$  та  $S = S_1$ .

Крок 3. Будуємо множини  $S_q = \{q\}$ ,  $S_{\bar{q}} = \{\bar{q} \vee \bar{r}\}$ . Множини  $S_q$  та  $S_{\bar{q}}$  непорожні.

Крок 4. Побудуємо множини  $S'' = \{r\}$ ,  $S'_q = \{\square\}$ ,  $S'_{\bar{q}} = \{\bar{r}\}$ ,  $S_1 = S'_q \cup S'' = \{\square, r\}$ ,  $S_2 = S'_{\bar{q}} \cup S'' = \{r, \bar{r}\}$ .

Крок 5. Множина  $S_2$  не містить порожнього диз'юнкта. Поміщаємо пару  $(P, S_2)$  у стек.

Крок 6. Оскільки стек не порожній, то повертаємось до кроку 2.

Крок 2. Вибираємо з верхівки стека пару  $(P, S)$ , а з  $P$  – атом  $r$ , який виключаємо із множини  $P$ . Отримаємо  $P = \emptyset$ .

Крок 3. Будуємо множини  $S_r = \{\bar{r}\}$ ,  $S_{\bar{r}} = \{r\}$ . Множини  $S_r$  та  $S_{\bar{r}}$  непорожні.

Крок 4. Будуємо множини  $S'' = \emptyset$ ,  $S'_r = \{\square\}$ ,  $S'_{\bar{r}} = \{\square\}$ ,  $S_1 = S'_r \cup S'' = \{\square\}$ ,  $S_2 = S'_{\bar{r}} \cup S'' = \{\square\}$ .

Крок 5. Множини  $S_1$  та  $S_2$  містять порожній диз'юнкт. До стека нічого не поміщаємо.

Крок 6. Оскільки стек порожній, та задана множина невиконання.

Розв'язування прикладу завершено.

Модифікований алгоритм Куайна формалізує процедуру обчислення формули логіки висловлювань шляхом виконання формальних дій по викреслюванню літералів і переписуванню рядків символів з метою отримання рядка необхідного вигляду. За виглядом цього рядка можна зробити висновок про виконаність формули. В алгоритмі Куайна жодним чином не обумовлено послідовність вибору вершин семантичного дерева з метою скорочення кількості обчислень формули в її часткових інтерпретаціях для внутрішніх вершин цього дерева.

## 2.10. АЛГОРИТМ ДЕВІСА–ПАТНЕМА

Алгоритм Девіса–Патнема (M. Davis, H. Putnam) є модифікацією алгоритму Куайна, у якій доведення виконують побудовою відповідного семантичного дерева з вибором атомів базису в певній послідовності. Такий вибір елементів виконують тоді, коли в множині диз'юнктів  $S$  є диз'юнкт із рангом 1.

Нехай, для прикладу, в  $S$  є диз'юнкт  $p$  із рангом 1. Тоді множина  $S'_p \cup S''$  містить порожній диз'юнкт, тобто ця множина невиконання. Тепер невиконаність множини  $S$  можна звести до перевірки невиконаності множини  $S'_p \cup S''$ .

Розглянемо приклад, який показує, що в разі наявності диз'юнктів із рангом 1 процес доведення спрощується.

**Приклад 2.40.** Доведемо  $h, h \rightarrow (p \vee q), p \rightarrow c, q \rightarrow c \vdash c$ , або невиконаність множини  $S = \{h, \bar{h} \vee p \vee q, \bar{p} \vee c, \bar{q} \vee c, \bar{c}\}$ . Розберемо процедуру спростування за алгоритмом Куайна.

1. Утворимо базис  $P = \{h, c, p, q\}$ . Виберемо з базису атом  $h$  та побудуємо множини  $S_h = \{h\}$ ,  $S_{\bar{h}} = \{\bar{h} \vee p \vee q\}$ ,  $S'' = \{\bar{p} \vee c, \bar{q} \vee c, \bar{c}\}$ ,  $S'_h = \{\square\}$ ,  $S'_h = \{p \vee q\}$ ,  $S_1 = S'_h \cup S'' = \{p \vee q, \bar{p} \vee c, \bar{q} \vee c, \bar{c}\}$ . Множина  $S_2 = S'_h \cup S'' = \{\square, \bar{p} \vee c, \bar{q} \vee c, \bar{c}\}$  невиконання. Продовжуємо доведення невиконаності для множини  $S = S_1 = S'_h \cup S''$ .

2. Вибираємо з базису атом  $c$  та будуємо множини  $S_c = \{c\}$ ,  $S_{\bar{c}} = \{\bar{c}\}$ ,  $S'' = \{p \vee q\}$ ,  $S'_c = \{\bar{p}, \bar{q}\}$ ,  $S'_c = \{\square\}$ ,  $S'_c \cup S'' = \{p \vee q, \bar{p}, \bar{q}\}$ . Множина  $S_1 = S'_c \cup S'' = \{\square, p \vee q\}$  невиконання. Продовжимо доведення для множини  $S = S_2 = S'_c \cup S'' = \{\bar{p}, \bar{q}, p \vee q\}$ .

3. Вибираємо з базису атом  $p$  та утворимо множини  $S_p = \{p \vee q\}$ ,  $S_{\bar{p}} = \{\bar{p}\}$ ,  $S'' = \{\bar{q}\}$ ,  $S'_p = \{q\}$ ,  $S'_p = \{\square\}$ ,  $S_1 = S'_p \cup S'' = \{q, \bar{q}\}$ ,  $S_2 = S'_p \cup S'' = \{\square\}$ . Множина  $S_2$  невиконання.

4. Вибираємо останній атом із базису –  $q$ . Отримуємо множини  $S_q = \{q\}$ ,  $S_{\bar{q}} = \{\bar{q}\}$ ,  $S'' = \emptyset$ ,  $S'_q = \{\square\}$ ,  $S'_{\bar{q}} = \{\square\}$ . Оскільки  $S_1$  та  $S_2$  містять порожній диз'юнкт, то задана множина  $S$  невиконання.

**Приклад 2.41.** Перепишемо послідовність кроків розв'язування прикладу 2.40, виписуючи результати виконання кожного кроку в окремий рядок. У множинах напівжирним шрифтом виділено диз'юнкти рангу 1, які вибираємо на кожному кроці. Справа зазначено, які множини одержано як результат виконання відповідного кроку.

1)  $\{h, \bar{h} \vee p \vee q, \bar{p} \vee c, \bar{q} \vee c, \bar{c}\}$  – множина  $S$ ;

2)  $\{p \vee q, \bar{p} \vee c, \bar{q} \vee c, \bar{c}\}$  – множина  $S'_h \cup S''$ ;

3)  $\{p \vee q, \bar{p}, \bar{q}\}$  – множина  $S'_c \cup S''$ ;

4)  $\{\bar{q}, q\}$  – множина  $S'_p \cup S''$ ;

5)  $\{\square\}$ .

Можна зауважити, що на кожному кроці алгоритму, записаного в такій формі, літерал, який є диз'юнктом рангу 1, на наступному кроці зникає з множини, а з решти диз'юнктів пропадають його заперечення.

Уже в модифікованому методі Куайна перевірка формули на загальнозначущість не вимагала знаходження її значення істинності. Цю перевірку було замінено послідовними перетвореннями логічних формул як ланцюжків символів. Такі

перетворення полягають у викреслюванні в формулах літералів при побудові семантичного дерева. Висновок про властивості формули здійснюється за її виглядом, а не за значенням.

Алгоритм Девіса–Патнема являє собою вдосконалення модифікованого алгоритму Куайна. Надамо додаткові пояснення щодо термінології, яку будемо використовувати. Особливістю методу Девіса–Патнема є те, що множину диз'юнктів  $S$  не розбивають на дві підмножини. У той самий час щодо множини  $S$  зроблено припущення, що вона містить диз'юнкти рангу 1, тобто відповідний диз'юнкт – літерал. Тоді решта диз'юнктів можуть містити лише заперечення цього літерала. Якщо надати цьому літералу значення  $T$ , то його заперечення набуде в диз'юнкті значення  $F$ . Отже, виглядає так, що відповідний літерал та його заперечення *викреслено* з елементів множини  $S$ . Аналогічно, якщо літералу в складі диз'юнкта надати значення  $F$ , то в подальшому в формулі його не записують, а виглядає це так, що його просто викреслили. Якщо ж літералу у диз'юнкті надати значення  $T$ , то і весь диз'юнкт набуває цього значення. Це означає, що в множині  $S$  цей диз'юнкт можна не враховувати, тобто його можна *вилучити* із запису множини  $S$ . Отже, літерал у диз'юнкті *викреслюємо*, а весь диз'юнкт – *вилучаємо*.

Нехай  $S$  – множина диз'юнктів, яку треба спростувати. Алгоритм Девіса–Патнема відрізняється від алгоритму Куайна перетвореннями множин на кроках 2–5. Ці перетворення здійснюють із дотриманням таких правил.

**1. Правило тавтології.** З  $S$  вилучають усі диз'юнкти, які є тавтологіями, тобто містять контрарну пару літералів.

**2. Правило диз'юнктів із рангом 1.** Якщо множина  $S$  має диз'юнкт  $p$  із рангом 1, то виконують такі дії.

♦ Вилучають із  $S$  цей диз'юнкт та всі диз'юнкти, які містять літерал  $p$ . У результаті одержують множину  $S'$ . Якщо  $S'$  порожня, то  $S$  – виконання.

♦ Викреслюють із диз'юнктів множини  $S'$  всі літерали  $\bar{p}$  та одержують множину  $S''$ . Множина  $S$  невиконання тоді й лише тоді, коли невиконання  $S''$ . Значимо, що коли  $\bar{p}$  – диз'юнкт із рангом 1, то його викреслювання утворює порожній диз'юнкт.

**3. Правило „чистих” літералів.** Літерал  $p$  в диз'юнкті множини  $S$  називають „чистим”, якщо в жодному з диз'юнктів множини  $S$  немає літерала  $\bar{p}$ . Якщо літерал  $p$  „чистий” в  $S$ , то вилученням усіх диз'юнктів, які містять  $p$ , утворюють множину  $S'$ . Множина  $S$  невиконання тоді й лише тоді, коли невиконання множини  $S'$ .

**4. Правило розщеплення.** Якщо множину  $S$  можна записати у вигляді  $\{A_1 \vee p, \dots, A_m \vee p, B_1 \vee \bar{p}, \dots, B_n \vee \bar{p}, R\}$ , а диз'юнкти  $A_i$  ( $i=1, 2, \dots, m$ ),  $B_j$  ( $j=1, 2, \dots, n$ ),  $R$  не містять літералів  $p$  та  $\bar{p}$ , то будують множини  $S_1 = \{A_1, A_2, \dots, A_m, R\}$  та  $S_2 = \{B_1, B_2, \dots, B_n, R\}$ . Множина  $S$  невиконання тоді й лише тоді, коли множини  $S_1$  та  $S_2$  невиконання.

У правилі диз'юнктів із рангом 1 вилучення диз'юнкту пов'язане із застосування закону поглинання для формул логіки висловлювань.

**Приклад 2.42.** Доведемо невиконанність множини  $S = \{p \vee q \vee \bar{r}, p \vee \bar{q}, \bar{p}, r, u\}$ . Наведемо послідовність кроків доведення невиконанності із застосуванням алгоритму Девіса–Патнема.

$$1) \{p \vee q \vee \bar{r}, p \vee \bar{q}, \bar{p}, r, u\}.$$

Для  $\bar{p}$  застосуємо правило диз'юнктів із рангом 1. Для цього з диз'юнктів  $p \vee q \vee \bar{r}$  та  $p \vee \bar{q}$  викреслюємо літерал  $p$ , а літерал  $\bar{p}$  вилучаємо з множини.

$$2) \{q \vee \bar{r}, \bar{q}, r, u\}.$$

Для  $\bar{q}$  застосуємо правило диз'юнктів із рангом 1. Для цього з диз'юнкту  $q \vee \bar{r}$  викреслюємо літерал  $q$ , а літерал  $\bar{q}$  вилучаємо з множини.

$$3) \{\bar{r}, r, u\}.$$

Аналогічно до двох попередніх кроків застосуємо правило диз'юнктів із рангом 1 для  $\bar{r}$ . Вилучаємо  $\bar{r}$ . Після викреслювання  $r$  одержуємо порожній диз'юнкт.

$$4) \{\square, u\}.$$

Оскільки одержана множина містить порожній диз'юнкт, то множина  $S$  – невиконання.

**Приклад 2.43.** Доведемо, що множина  $S = \{p \vee q, \bar{q}, \bar{p} \vee q \vee \bar{r}\}$  – виконання. Для доведення використаємо правило диз'юнктів із рангом 1.

$$1) \{p \vee q, \bar{q}, \bar{p} \vee q \vee \bar{r}\}.$$

$$2) \{p, \bar{p} \vee \bar{r}\}.$$

$$3) \{\bar{r}\}.$$

З останньої множини можна лише вилучити диз'юнкт із рангом 1 та одержати порожню множину. Тут немає диз'юнктів, у яких можна викреслити літерали.

$$4) \emptyset.$$

Отримання порожньої множини свідчить, що множина  $S$  – виконання.

**Приклад 2.44.** Доведемо, що множина  $S = \{p \vee \bar{q}, \bar{p} \vee q, q \vee \bar{r}, \bar{q} \vee \bar{r}\}$  – виконання. Наведемо доведення із застосуванням правил алгоритму Девіса–Патнема.

$$1) \{p \vee \bar{q}, \bar{p} \vee q, q \vee \bar{r}, \bar{q} \vee \bar{r}\}.$$

Оскільки диз'юнктів із рангом 1 у множині немає, то застосуємо правило 4 – розщеплення множини диз'юнктів для пари  $\{p, \bar{p}\}$ .

$$2) \{\bar{q}, q \vee \bar{r}, \bar{q} \vee \bar{r}\} \cup \{q, q \vee \bar{r}, \bar{q} \vee \bar{r}\}.$$

До кожної множини застосуємо правило диз'юнктів із рангом 1. У першій множині вилучимо  $\bar{q}$  та  $\bar{q} \vee \bar{r}$  і викреслимо  $q$ ; одержимо  $\bar{r}$ . У другій множині вилучимо  $q$  та  $q \vee \bar{r}$  і викреслимо  $\bar{q}$ ; одержимо  $\bar{r}$ .

$$3) \{\bar{r}\} \cup \{\bar{r}\}.$$

В одержаних множинах можна вилучити диз'юнкти із рангом 1 та одержати порожні множини.

$$4) \emptyset \cup \emptyset.$$

Отже,  $S$  – виконання множини.

**Приклад 2.45.** Доведемо, що множина  $S = \{p \vee q, p \vee \bar{q}, r \vee q, r \vee \bar{q}\}$  – виконання. Наведемо доведення із застосуванням правил алгоритму Девіса–Патнема.

$$1) \{p \vee q, p \vee \bar{q}, r \vee q, r \vee \bar{q}\}.$$

Оскільки літерал  $p$  „чистий”, то вилучаємо диз'юнкти, у яких він міститься.



2)  $\{r \vee q, r \vee \bar{q}\}$ .

Літерал  $r$  „чистий”. Після вилучення диз'юнктивів із цим літералом одержують порожню множину.

3)  $\emptyset$ .

Одержано порожню множину, отже, задана множина  $S$  – виконання.

## 2.11. ПОНЯТТЯ ФОРМАЛЬНОЇ ТЕОРІЇ. ЧИСЛЕННЯ ВИСЛОВЛЮВАНЬ.

Розглянемо основні принципи побудови числення висловлювань і числення предикатів, які відносять до логіко-математичних формальних теорій (формальних, або *дедуктивних систем*). Ці формальні теорії моделюють процес міркувань людини, процес логічного виведення, який здійснює людина. Основна їхня ідея полягає в тому, що існує формальна мова, на якій можна описати мінімально необхідний і достатній набір висловлювань, який не вимагає доведень, та набір правил, за допомогою яких будують висновки.

Основна мета створення цих теорій – узагальнення та ревізія накопиченого математичного досвіду. Починаючи з евклідової геометрії й арифметики та закінчуючи сучасними математичними дисциплінами, побудова кожної із цих теорій, ґрунтується, з одного боку, на наборі означень та аксіом, які є абстрактним узагальненням людського досвіду, а з іншого боку – на узагальненні попереднього математичного досвіду.

Окрім того, виникла можливість узагальнювати поняття логічних числень (*числення висловлювань та числення предикатів*) на основі цих принципів. За допомогою набору аксіом, правил виведення та визначення вивідності можна описати не тільки множину виразів, які інтерпретують як висловлювання, але й об'єкти довільного типу.

Прикладом узагальнення логічних числень, який знайшов своє застосування при описі мов програмування за допомогою формальних граматик, є канонічне числення Е. Поста. Його ідея полягає в тому, що є деякий алфавіт символів, аксіоми та правила виведення, які дозволяють замінити (переписувати) одні ланцюжки символів на інші.

Формальну теорію задають четвіркою  $M=(T, P, A, R)$ , де  $T$  – множина базових елементів,  $P$  – множина синтаксичних правил,  $A$  – множина аксіом,  $R$  – множина правил виведення.

Множина базових елементів  $T$  утворює *алфавіт* формальної теорії. З елементів алфавіту будують усі інші складові формальної теорії. На алфавіт можна не накладати жодних обмежень (він може бути скінченним або нескінченним), але повинен існувати алгоритм, який дає змогу перевіряти належність якогось елемента множині  $T$ .

Множина синтаксичних правил  $P$  дозволяє будувати сукупності елементів алфавіту, які називають *формулами*. Тут також вимагають існування алгоритму, який дає змогу за скінченну кількість кроків перевірити, чи є побудована формула синтаксично правильною. Серед усіх можливих формул виділяють підмножину формул  $A$ , які називають *аксіомами*.

*Правилом виведення*  $R_i$  ( $i=1,2,\dots,n$ ) називають деяке відношення на множині формул. Правила утворюють *множину правил виведення*  $R$ . Якщо формули  $F_1, F_2, \dots, F_m, G$  перебувають у відношенні  $R_i$ , то формулу  $G$  називають *безпосереднім наслідком* формул  $F_1, F_2, \dots, F_m$  за правилом  $R_i$ .

Застосуванням правил виведення до множини аксіом  $A$  отримують нові формули, до яких можна також застосувати правила виведення. У такий спосіб виводять нові формули.

*Виведенням* формули  $B$  з формул  $\Gamma=\{A_1, A_2, \dots, A_n\}$  називають таку послідовність формул  $F_1, F_2, \dots, F_m$ , що  $F_m=B$ , а будь-яка формула  $F_i$  ( $i=1, 2, \dots, m$ ) є або аксіомою формальної теорії, або формулою з множини  $\Gamma$ , або безпосереднім наслідком формул  $F_1, F_2, \dots, F_{i-1}$ , отриманих за правилами виведення. Формулу  $B$  називають *вивідною* з формул  $A_1, A_2, \dots, A_n$ , а формули з множини  $\Gamma$  називають *гіпотезами*. Перехід від  $F_{i-1}$  до  $F_i$  називають  $i$ -м кроком виведення. Той факт, що  $B$  виводиться з  $\Gamma$ , записують  $\Gamma \vdash B$ , а виведення  $B$  з  $A_1, A_2, \dots, A_n$  – у вигляді  $A_1, A_2, \dots, A_n \vdash B$ .

*Доведенням* формули  $B$  називають виведення цієї формули з порожньої множини формул, тобто виведення, у якому гіпотезами є лише аксіоми, що записують  $\emptyset \vdash B$  або  $\vdash B$ . Формулу  $B$ , для якої існує доведення, називають *довідною*, або *теоремою*. Очевидно, що приєднання формул до гіпотез не порушує вивідності. Тому, якщо  $\vdash B$ , то  $A \vdash B$  та, якщо  $A_1, A_2, \dots, A_n \vdash B$ , то  $A_1, A_2, \dots, A_n, A_{n+1} \vdash B$  для довільних  $A$  та  $A_{n+1}$ .

Найважливішими формальними теоріями є числення висловлювань та *числення першого ступеня* (числення предикатів).

Числення висловлювань визначають так.

1. Алфавіт  $T$  числення висловлювань складається зі *змінних висловлювань* (*пропозиційних букв*)  $a, b, \dots$ , знаків логічних зв'язок  $\vee, \wedge, \rightarrow, \neg$  та дужок  $(, )$ .

2. Множину формул  $P$  числення висловлювань визначають рекурсивно:

♦ змінні висловлювання є формулами;

♦ якщо  $p$  та  $q$  формули, то  $(p \vee q)$ ,  $(p \wedge q)$ ,  $(p \rightarrow q)$  та  $(\neg p)$  також формули. Замість  $(\neg p)$  часто пишуть  $\bar{p}$ .

3. Множина аксіом  $A$  числення висловлювань. Існує декілька систем (множин) аксіом. Ці системи аксіом еквівалентні в тому сенсі, що вони породжують одну й ту ж множину формул. Наведемо дві системи аксіом. Першою з них є така система аксіом:

1.1.  $A \rightarrow (B \rightarrow A)$ ;

1.2.  $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$ ;

1.3.  $(A \vee B) \rightarrow A$ ;

1.4.  $(A \vee B) \rightarrow B$ ;

1.5.  $(A \rightarrow (B \rightarrow (A \wedge B)))$ ;

1.6.  $A \rightarrow (A \vee B)$ ;

1.7.  $B \rightarrow (A \vee B)$ ;

1.8.  $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow ((A \vee B) \rightarrow C))$ ;

1.9.  $(\overline{A \rightarrow B}) \rightarrow ((A \rightarrow \bar{B}) \rightarrow \bar{A})$ ;

1.10.  $\bar{\bar{A}} \rightarrow A$ .

Друга система аксіом є такою:

2.1.  $A \rightarrow (B \rightarrow A)$ ;

2.2.  $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$ ;

2.3.  $(\bar{A} \rightarrow \bar{B}) \rightarrow ((\bar{A} \rightarrow B) \rightarrow A)$ .

4. Множину правил виведення  $R$  числення висловлювань задають двома правилами:

♦ *правило підстановки*, яке має вигляд  $F(A) \vdash F(B)$ . Зміст цього правила полягає в тому, що якщо  $F$  є вивідною формулою, яка містить букву  $A$ , то заміною її на довільну

формулу  $B$  буде отримана також вивідна формула.

♦ **правило висновку** (*modus ponens*), яке записують  $A, A \rightarrow B \vdash B$ , має такий зміст: якщо формули  $A$  та  $A \rightarrow B$  вивідні, то вивідна й формула  $B$ .

Якщо доведене виведення  $B$  з  $A_1, A_2, \dots, A_n$ , то  $A_1, A_2, \dots, A_n \vdash B$  можна розглядати як додаткове правило виведення. Це правило можна приєднати до наявних і використовувати його в подальшому.

Отже, крім основних правил виведення (підстановки та *modus ponens*) можна використовувати інші правила побудови вивідних формул. Тобто, будь-яке виведення вигляду  $\Gamma \vdash B$ , де  $\Gamma$  – множина формул, а  $B$  – формула, можна розглядати як правило виведення, та приєднати до вже наявних правил.

Формули числення висловлювань можна інтерпретувати як формули логіки висловлювань. Для цього атоми логіки висловлень можна трактувати як змінні числення висловлювань, які приймають значення Т та Ф. Логічні зв'язки числення висловлювань визначимо так само, як і в логіці висловлювань. Тоді будь-яка формула для довільних значень змінних буде приймати значення Т або Ф, яке обчислюватиметься за правилами логіки висловлювань.

Надамо формулювання деяких теорем, які будуть покладені в основу побудови алгоритмів автоматичного доведення теорем у штучному інтелекті.

**Теорема 2.4 (дедукції).** Якщо  $\Gamma, U \vdash B$ , то  $\Gamma \vdash U \rightarrow B$ . Тут  $\Gamma$  – множина формул,  $U$  – формула.

Зазначимо, що в разі  $\Gamma = \emptyset$  теорема дедукції набуває такого вигляду: якщо  $U \vdash B$ , то  $\vdash U \rightarrow B$ .

**Приклад 2.46.** Поширеним методом доведення є метод *доведення від протилежного*, який формулюють так: припускають, що  $A$  правильне та доводять, що з  $A$  можна вивести як  $B$ , так і  $\bar{B}$ . Оскільки це неможливо, то воно означає, що  $A$  неправильне, тобто правильне  $\bar{A}$ .

Цей метод формулюють так: якщо  $\Gamma, A \vdash B$  та  $\Gamma, A \vdash \bar{B}$ , то  $\Gamma \vdash \bar{A}$ . Обґрунтуємо цей метод у численні висловлювань.

З теореми дедукції випливає, що, якщо  $\Gamma, A \vdash B$ , то  $\Gamma \vdash A \rightarrow B$ . Аналогічно, якщо  $\Gamma, A \vdash \bar{B}$ , то  $\Gamma \vdash A \rightarrow \bar{B}$ . Із отриманих імплікацій та аксіоми  $(A \rightarrow B) \rightarrow ((A \rightarrow \bar{B}) \rightarrow \bar{A})$  (аксіома 1.9), застосовуючи правило *modus ponens*, одержимо:

$$1) \Gamma, A \rightarrow B, (A \rightarrow B) \rightarrow ((A \rightarrow \bar{B}) \rightarrow \bar{A}) \vdash (A \rightarrow \bar{B}) \rightarrow \bar{A};$$

$$2) \Gamma, A \rightarrow \bar{B}, (A \rightarrow \bar{B}) \rightarrow \bar{A} \vdash \bar{A};$$

$$3) \Gamma \vdash \bar{A}.$$

Обґрунтований метод ще називають правилом *уведення заперечення*.

**Приклад 2.47.** Покажемо, що формула  $(A \rightarrow B) \rightarrow (A \rightarrow A)$  вивідна в численні висловлювань. Для доведення підставимо у формулу  $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$  (аксіома 1.2)  $A$  замість  $C$  та отримаємо  $(A \rightarrow (B \rightarrow A)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow A))$ . Оскільки  $(A \rightarrow (B \rightarrow A))$  є аксіомою 1.1, то за правилом *modus ponens* маємо

$$A \rightarrow (B \rightarrow A), (A \rightarrow (B \rightarrow A)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow A)) \vdash (A \rightarrow B) \rightarrow (A \rightarrow A),$$

що доводить вивідність заданої формули.

**Теорема 2.5.** Будь-яка теорема числення висловлювань є тавтологією.

Отже, будь-яка вивідна формула числення висловлювань є тавтологією. Обернене запитання полягає в наступному: чи будь-яка тавтологія числення висловлювань є вивідною формулою в численні висловлювань? Іншими словами, чи достатньо аксіом та правил виведення для того, щоб вивести довільну формулу, яка є тавтологією. Відповідь на це питання дає така теорема.

**Теорема 2.6.** Будь-яка загальнозначуща формула є теоремою числення висловлювань.

Отже, в численні висловлювань вивідна будь-яка загальнозначуща формула, а числення висловлювань розв'язує задачу породження загальнозначущих висловлювань. *Перевірка загальнозначущості формули логіки висловлювань еквівалентна доведенню відповідної логічної теореми числення висловлювань.*

Необхідність обґрунтування математики та конкретні задачі з цієї області привели до виникнення ідей метаматематичного характеру, пов'язаних із довільною аксіоматичною (формальною) теорією. До цих ідей відносяться поняття та проблеми несуперечливості, повноти та незалежності системи аксіом, які лежать в основі теорії.

Логічне числення називають *несуперечливим*, якщо в ньому не є вивідними жодні дві формули, з яких одна є запереченням іншої.

**Теорема 2.7.** Числення висловлювань несуперечливе.

Якщо є алгоритм, який дає змогу виявити, що існує виведення заданої формули в певній формальній теорії, то таку формальну теорію називають *розв'язною*.

**Теорема 2.8.** Числення висловлювань є розв'язною теорією.

## 2.12. ЗАСТОСУВАННЯ ПРАВИЛ ВИВЕДЕННЯ В ЧИСЛЕННІ ВИСЛОВЛЮВАНЬ

Правила виведення обґрунтовують кроки доведення логічних теорем, яке полягає у перевірці того, що висновок являє собою логічний наслідок множини гіпотез.

Деякі важливі правила виведення та відповідні їм тавтології наведено в табл. 2.11. Для прикладу, правило виведення *modus ponens* має вигляд  $p, p \rightarrow q \vdash q$ . Згідно з теоремою 2.2, воно ґрунтується на тавтології  $(p \wedge (p \rightarrow q)) \rightarrow q$ .

Таблиця 2.11

| Правило виведення   | Тавтологія   | Назва правила виведення |
|---|--|-------------------------|
| $p \vdash p \vee q$                                       | $p \rightarrow (p \vee q)$   | Уведення диз'юнкції     |
| $p \wedge q \vdash p$                                     | $(p \wedge q) \rightarrow p$   | Виключення кон'юнкції   |
| $p, q \vdash p \wedge q$                                  | $((p) \wedge (q)) \rightarrow (p \wedge q)$                                  | Уведення кон'юнкції     |
| $p, p \rightarrow q \vdash q$                             | $(p \wedge (p \rightarrow q)) \rightarrow q$                                 | Modus ponens            |
| $\bar{q}, p \rightarrow q \vdash \bar{p}$                 | $(\bar{q} \wedge (p \rightarrow q)) \rightarrow \bar{p}$                     | Modus tollens           |
| $p \rightarrow q, q \rightarrow r \vdash p \rightarrow r$ | $((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$ | Гіпотетичний силізм     |
| $p \vee q, \bar{p} \vdash q$                              | $((p \vee q) \wedge \bar{p}) \rightarrow q$                                  | Диз'юнктивний силізм    |
| $p \vee q, \bar{p} \vee r \vdash q \vee r$                | $((p \vee q) \wedge (\bar{p} \vee r)) \rightarrow (q \vee r)$                | Резолюція               |

**Приклад 2.48.** Припустимо, що імплікація „Якщо падає сніг, то ми катаємося на лижах” та її гіпотеза „Падає сніг” істинні. Тоді за правилом *modus ponens* висновок імплікації „Ми катаємося на лижах” також істинний.

**Приклад 2.49.** Нехай істинна імплікація: „Якщо  $n > 3$ , то  $n^2 > 9$ ”. Отже, якщо  $n > 3$ , то за правилом *modus ponens* висновок  $n^2 > 9$  правильний для цього  $n$ .

Далі наведено декілька прикладів міркувань із використанням інших правил виведення з табл. 2.11.

**Приклад 2.50.** З’ясуємо, яке правило виведення використано в такому міркуванні: „Похолоднішало. Отже, похолоднішало або почав падати дощ.”

Нехай  $p$  – висловлювання „Похолоднішало”, а  $q$  – висловлювання „Почав падати дощ”. Тоді це міркування можна записати як  $p \vdash p \vee q$ , тобто наведене міркування використовує *правило введення диз’юнкції*.

**Приклад 2.51.** З’ясуємо, яке правило виведення використано в такому міркуванні: „Похолоднішало та почав падати дощ. Отже, похолоднішало.”

Нехай  $p$  – висловлювання „Похолоднішало”, а  $q$  – висловлювання „Почав падати дощ”. Тоді це міркування можна записати як  $p \wedge q \vdash p$ . Це міркування використовує *правило виключення кон’юнкції*.

**Приклад 2.52.** З’ясуємо, яке правило виведення використано в такому міркуванні. „Якщо сьогодні падатиме дощ, то сьогодні ми не поїдемо на пікнік. Якщо ми не поїдемо на пікнік сьогодні, то поїдемо на пікнік завтра. Отже, якщо сьогодні падатиме дощ, то ми поїдемо на пікнік завтра.”

Нехай  $p$  – висловлювання „Сьогодні падатиме дощ”,  $q$  – висловлювання „Сьогодні ми не поїдемо на пікнік”, а  $r$  – висловлювання „Ми поїдемо на пікнік завтра”. Наведене міркування можна записати так:  $p \rightarrow q$ ,  $q \rightarrow r \vdash p \rightarrow r$ . Отже, у міркуванні використано *правило гіпотетичного силісмізму*.

Якщо твердження містить багато гіпотез, то потрібно застосувати декілька правил виведення для того, щоб довести істинність висновку. Проілюструємо це такими прикладами.

**Приклад 2.53.** Покажемо, що з гіпотез „Сьогодні не сонячний день і холодніше, ніж учора”, „Ми підемо купатися лише якщо сьогодні сонячний день”, „Якщо ми не підемо купатися, то поїдемо плавати на човні” та „Якщо ми поїдемо плавати на човні, то повернемося пізно ввечері” впливає висновок „Ми повернемося пізно ввечері”.

Нехай  $p$  – висловлювання „Сьогодні сонячний день”,  $q$  – „Сьогодні холодніше, ніж учора”,  $r$  – „Ми підемо купатися”,  $s$  – „Ми поїдемо плавати на човні”,  $t$  – „Ми повернемося пізно ввечері”. Тут гіпотези –  $\bar{p} \wedge q$ ,  $r \rightarrow p$ ,  $\bar{r} \rightarrow s$  і  $s \rightarrow t$ , а висновок –  $t$ .

Нижче наведено послідовність кроків отримання висновку із заданої множини гіпотез і зазначено застосовані правила виведення.

1.  $\bar{p} \wedge q$  – гіпотеза.
2.  $\bar{p}$  – правило виключення кон’юнкції до 1.
3.  $r \rightarrow p$  – гіпотеза.
4.  $\bar{r}$  – *modus tollens* до 2 та 3.

5.  $\bar{r} \rightarrow s$  – гіпотеза.

6.  $s$  – *modus ponens* до 4 та 5.

7.  $s \rightarrow t$  – гіпотеза.

8.  $t$  – *modus ponens* до 6 та 7.

Висновок отримано.

**Приклад 2.54.** Покажемо, що з гіпотез „Якщо ти надішлеш мені повідомлення електронною поштою, то я закінчу писати програму”, „Якщо ти не надішлеш мені повідомлення електронною поштою, то я рано піду спати”, та „Якщо я рано піду спати, то прокинуся бадьорим” впливає висновок „Якщо я не закінчу писати програму, то я прокинуся бадьорим”.

Уведемо позначення:  $p$  – „Ти надішлеш мені повідомлення електронною поштою”,  $q$  – „Я закінчу писати програму”,  $r$  – „Я рано піду спати”,  $s$  – „Я прокинуся бадьорим”. Гіпотези можна записати у вигляді  $p \rightarrow q$ ,  $\bar{p} \rightarrow r$ ,  $r \rightarrow s$ . Потрібно обґрунтувати висновок  $\bar{q} \rightarrow s$ .

Нижче наведено послідовність кроків для отримання висновку із заданої множини гіпотез. Застосовані правила виведення записано справа.

1.  $p \rightarrow q$  – гіпотеза.
2.  $\bar{q} \rightarrow \bar{p}$  – контрапозиція (див. приклад 2.18).
3.  $\bar{p} \rightarrow r$  – гіпотеза.
4.  $\bar{q} \rightarrow r$  – гіпотетичний силісмізм до 2 та 3.
5.  $r \rightarrow s$  – гіпотеза.
6.  $\bar{q} \rightarrow s$  – гіпотетичний силісмізм до 4 та 5.

Висновок отримано.

## 2.13. МЕТОД РЕЗОЛЮЦІЙ У ЧИСЛЕННІ ВИСЛОВЛЮВАНЬ

З матеріалу підрозділів 2.11 та 2.12 випливає, що доведення теорем у численні висловлювань може викликати певні труднощі. Жоден із запропонованих методів не може бути використаний для автоматичного доведення теорем. Цю проблему вирішив Дж. Робінсон (G. Robinson), який 1965 року для автоматичного доведення логічних теорем запропонував метод резолюцій, в основу якого покладено правило резолюції. *Правило резолюції* записують у вигляді  $p \vee q$ ,  $\bar{p} \vee r \vdash q \vee r$ .

За принципом прямої дедукції (теорема 2.3) формулу  $g$  можна вивести з формул  $f_1, f_2, \dots, f_n$  тоді й лише тоді, коли  $f_1 \wedge f_2 \wedge \dots \wedge f_n \wedge \bar{g}$  є суперечністю. Це можливо тоді, коли або одна з формул  $f_1, f_2, \dots, f_n$  є фальшивою, або  $g$  – істинною. Крім того, якщо формули  $f_1, f_2, \dots, f_n$  є диз’юнктами, то один з них може бути диз’юнктом із рангом 0, який позначають  $\square$  та називають порожнім диз’юнктом.

Якщо формула  $f = f_1 \wedge f_2 \wedge \dots \wedge f_n \wedge \bar{g}$  записана у кон’юнктивній нормальній формі  $f = d_1 \wedge d_2 \wedge \dots \wedge d_m$ , то  $d_1, d_2, \dots, d_m$  – її диз’юнкти, які можна записати як множину  $S = \{d_1, d_2, \dots, d_m\}$ .

Нехай диз'юнкції  $d_1$  та  $d_2$  такі, що  $d_1$  містить літерал  $l_1$ , контрарний літералу  $l_2$  з  $d_2$ . Викреслимо  $l_1$  з  $d_1$  та  $l_2$  з  $d_2$  й побудуємо диз'юнкції з решти літералів цих диз'юнкцій. Отриманий диз'юнкції називають *резольвентою*.

**Приклад 2.55.** Побудуємо резольвенту диз'юнкцій  $d_1 = p \vee r$  та  $d_2 = \bar{p} \vee q$ . Ці диз'юнкції містять контрарну пару літералів  $p$  та  $\bar{p}$ , які викреслимо з  $d_1$  та  $d_2$  відповідно. Утворимо диз'юнкцію літералів, що залишилися, яка є резольвентою  $r \vee q$ .

**Приклад 2.56.** Побудуємо резольвенту диз'юнкцій  $d_1 = \bar{p} \vee q \vee r$ ,  $d_2 = \bar{q} \vee s$ . Резольвента цих диз'юнкцій —  $\bar{p} \vee r \vee s$ .

**Приклад 2.57.** Розглянемо диз'юнкції  $d_1 = \bar{p} \vee r$  та  $d_2 = \bar{p} \vee q$ . Оскільки контрарної пари літералів у диз'юнкціях  $d_1$  та  $d_2$  немає, то резольвенти  $d_1$  та  $d_2$  не існують.

**Приклад 2.58.** Розглянемо множину диз'юнкцій  $S = \{d_1, d_2, d_3\}$ , де  $d_1 = p$ ,  $d_2 = \bar{p} \vee q$ ,  $d_3 = q$ . Очевидно, що  $d_1, d_2 \vdash d_3$ .

**Теорема 2.9.** Резольвента  $d$  диз'юнкцій  $d_1$  і  $d_2$  є логічним наслідком  $d_1$  і  $d_2$ .

**Доведення.** Нехай  $d_1 = l \vee d'_1$ ,  $d_2 = \bar{l} \vee d'_2$ , де  $d'_1$  і  $d'_2$  — диз'юнкції літералів, відмінних від  $l$  та  $\bar{l}$ . Нехай  $d_1$  та  $d_2$  істинні в інтерпретації  $I$ . Покажемо, що резольвента  $d$  диз'юнкцій  $d_1$  і  $d_2$  також істинна в цій інтерпретації. Відзначимо, що або  $l$ , або  $\bar{l}$  фальшиві в інтерпретації  $I$ . Нехай літерал  $l$  — фальшивий в  $I$ . Тоді  $d_1$  не може бути диз'юнкцією рангу 1, бо  $d_1$  був би фальшивим в  $I$ . Це означає, що  $d'_1$  істинна в  $I$  та резольвента  $d = d'_1 \vee d'_2$  істинна в  $I$ . Аналогічно покажемо, що, коли  $\bar{l}$  фальшивий в  $I$ , то  $d'_2$  є істинним в  $I$ . Звідси отримуємо, що диз'юнкція  $d = d'_1 \vee d'_2$  істинна в інтерпретації  $I$ .

**Приклад 2.59.** Нехай  $f_1 = p \vee r$ ,  $f_2 = q \vee \bar{r}$ ,  $g = p \vee q$ . Довести, що  $f_1, f_2 \vdash g$ . Формула  $g$  отримана вилученням контрарної пари літералів  $\{r, \bar{r}\}$  із диз'юнкцій формул  $f_1$  та  $f_2$ . Формула  $g$  є резольвентою формул  $f_1$  та  $f_2$ , тому  $f_1, f_2 \vdash g$ .

Виведення формули  $d$  з диз'юнкцій множини  $S$  за методом резолюцій полягає в побудові такої скінченної послідовності диз'юнкцій  $d'_1, d'_2, \dots, d'_k$ , що для кожного з них, або  $d'_i \in S$  ( $i=2, \dots, k$ ), або  $d'_i$  є резольвентою диз'юнкцій, які передують  $d'_i$ , причому  $d = d'_k$ .

Диз'юнкція  $d$  можна *вивести* з  $S$ , якщо існує виведення  $d$  з  $S$ .

Основну ідею методу резолюцій формулюють так: перевірити, чи містить множина диз'юнкцій  $S$  порожній диз'юнкції  $\square$ . Якщо  $S$  містить  $\square$ , то  $S$  — невиконання. Якщо  $S$  не містить  $\square$ , то перевіряють, чи можна вивести  $\square$  з  $S$ . Виведення  $\square$  з  $S$  називають доведенням невиконання  $S$  або *спростуванням*  $S$ .

## Алгоритм методу резолюцій

Задано множину гіпотез  $f_1, f_2, \dots, f_n$  та висновок  $g$ . Алгоритм дає змогу визначити, чи являє собою формула  $g$  логічний наслідок множини гіпотез.

**Крок 1.** Побудувати кон'юнкцію множини гіпотез  $f_1, f_2, \dots, f_n$  та заперечення висновку  $\bar{g}$  у вигляді  $f_1 \wedge f_2 \wedge \dots \wedge f_n \wedge \bar{g}$ . Звести отриману формулу до КНФ і записати множину її диз'юнкцій  $S$ .

**Крок 2.** Записати кожний диз'юнкції множини  $S$  в окремому рядку.

**Крок 3.** Вибрати два диз'юнкції, які містять контрарну пару літералів, і побудувати їх резольвенту. Записати одержану резольвенту в новому рядку, якщо в попередніх рядках ще немає такого диз'юнкції.

**Крок 4.** Крок 3 виконувати до отримання диз'юнкції з рангом 0. Одержання диз'юнкції з рангом 0 свідчить про те, що формулу  $g$  можна вивести з формул  $f_1, f_2, \dots, f_n$ . Якщо неможливо отримати резольвенту, відмінну від елементів множини  $S$  та вже побудованих резольвент, то множина  $S$  неспростовна. Кінець.

**Приклад 2.60.** За алгоритмом резолюцій покажемо невиконання множини  $S = \{\bar{p} \vee q, \bar{q}, p\}$ . Випишемо кожний диз'юнкції в окремому рядку та перенумеруємо їх:

- (1)  $\bar{p} \vee q$
- (2)  $\bar{q}$
- (3)  $p$
- (4)  $\bar{p}$

Резольвенту (4) одержано з диз'юнкцій (1) та (2).

(5)  $\square$

Резольвента (5) є порожнім диз'юнкції, його отримано з диз'юнкції (3) та резольвенти (4). Виведення порожнього диз'юнкції із множини  $S$  спростовує цю множину.

**Приклад 2.61.** Методом резолюцій доведемо, що  $p \vee q, p \rightarrow r, q \rightarrow s \vdash r \vee s$ . За принципом прямої дедукції побудуємо формулу, невиконання якої треба довести. Вона має вигляд  $(p \vee q) \wedge (p \rightarrow r) \wedge (q \rightarrow s) \wedge \overline{(r \vee s)}$ . Застосуємо алгоритм резолюцій. Запишемо гіпотези у формі диз'юнкцій і випишемо кожний з них в окремому рядку:

- (1)  $p \vee q$
- (2)  $\bar{p} \vee r$
- (3)  $\bar{q} \vee s$

Оскільки заперечення висновку  $\overline{r \vee s} = \bar{r} \wedge \bar{s}$  утворює два диз'юнкції  $\bar{r}$  та  $\bar{s}$  із рангом 1, то їх теж випишемо в окремі рядки:

- (4)  $\bar{r}$
- (5)  $\bar{s}$

Послідовною побудовою всіх можливих резольвент за методом резолюцій виводимо порожній диз'юнкції. Біля кожної резольвенти випишемо номери диз'юнкцій, з яких цю резольвенту отримано.

- (6)  $\bar{p}$  (2),(4)  
 (7)  $q$  (6),(1)  
 (8)  $\bar{q}$  (3),(5)  
 (9)  $\square$  (7),(8).

На рис. 2.4 у вигляді дерева зображений процес спрощування множини диз'юнктив. Це дерево називають *деревом спрощування*; його листки позначають диз'юнктами – елементами множини, яка підлягає спрощуванню, а внутрішні вершини – резольвентами, які отримують під час виведення за методом резолюцій. Якщо множину можна спростувати, то корінь дерева спрощування відповідає диз'юнкту із рангом 0.

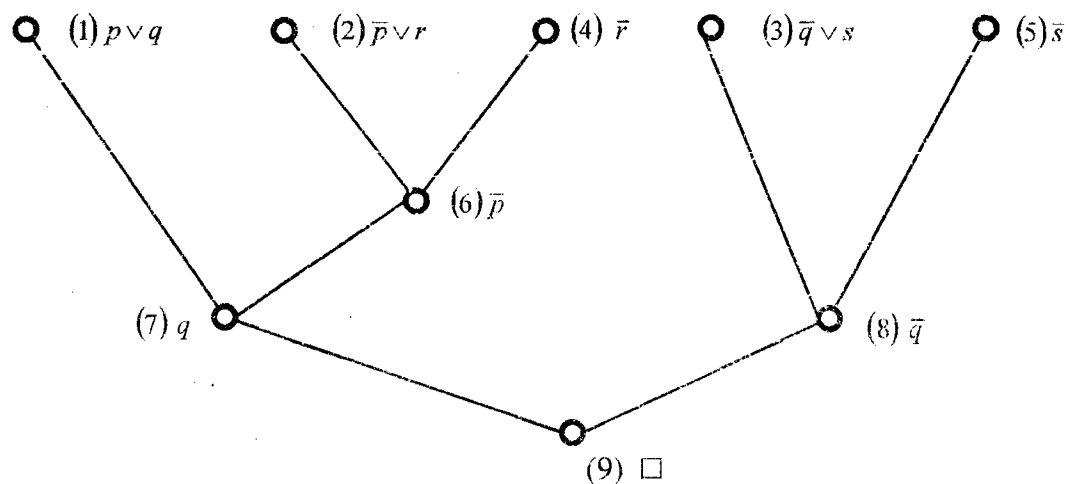


Рис. 2.4

Наведемо інший спосіб запису диз'юнктив та кроків методу резолюцій. Кожний диз'юнкт будемо зображати множиною його літералів. Наприклад, диз'юнкт  $p \vee \bar{q} \vee s$  зобразимо множиною  $\{p, \bar{q}, s\}$ . Порожню множину літералів, або порожній диз'юнкт зобразимо знаком порожньої множини. Для застосування алгоритму резолюцій послідовно вибираємо пари диз'юнктив, що містять контрарні пари літералів. Наприклад, резольвентою диз'юнктив  $d_1 = \{p, q\}$  та  $d_2 = \{\bar{p}, r\}$  є диз'юнкт  $d = d_1 * d_2 = \{q, r\}$ . Побудову резольвент виконуємо для заданої множини диз'юнктив і для їхніх резольвент.

Проілюструємо запропонований спосіб запису диз'юнктив і резольвент на прикладі спрощування множини диз'юнктив.

**Приклад 2.62.** Доведемо  $\bar{p} \rightarrow r, p \rightarrow s, r \rightarrow q, s \rightarrow \bar{t}, t \vdash q$ . За методом прямої дедукції побудуємо формулу, невиконаність якої треба перевірити. Ця формула має вигляд

$$(\bar{p} \rightarrow r) \wedge (p \rightarrow s) \wedge (r \rightarrow q) \wedge (s \rightarrow \bar{t}) \wedge t \wedge \bar{q},$$

а її кон'юнктивна нормальна форма –

$$(p \vee r) \wedge (\bar{p} \vee s) \wedge (r \vee q) \wedge (s \vee \bar{t}) \wedge t \wedge \bar{q}.$$

Випишемо диз'юнкти  $d_1 = \{p, r\}$ ,  $d_2 = \{\bar{p}, s\}$ ,  $d_3 = \{r, q\}$ ,  $d_4 = \{s, \bar{t}\}$ ,  $d_5 = \{t\}$ ,  $d_6 = \{\bar{q}\}$  та резольвенти в послідовності їхньої побудови:

$$\begin{aligned} d_7 &= d_1 * d_2 = \{r, s\}, & d_{12} &= d_4 * d_5 = \{\bar{s}\}, & d_{17} &= d_5 * d_{14} = \{q\}, \\ d_8 &= d_1 * d_3 = \{p, q\}, & d_{13} &= d_4 * d_7 = \{r, \bar{t}\}, & d_{18} &= d_6 * d_8 = \{p\}, \\ d_9 &= d_2 * d_4 = \{\bar{p}, \bar{t}\}, & d_{14} &= d_4 * d_{10} = \{q, \bar{t}\}, & d_{19} &= d_6 * d_{10} = \{s\}, \\ d_{10} &= d_2 * d_3 = \{q, s\}, & d_{15} &= d_5 * d_9 = \{\bar{p}\}, & d_{20} &= d_6 * d_{14} = \{\bar{t}\}, \\ d_{11} &= d_3 * d_6 = \{\bar{r}\}, & d_{16} &= d_5 * d_{13} = \{r\}, & d_{21} &= d_6 * d_{17} = \emptyset. \end{aligned}$$

Оскільки одержано порожню множину, то факт виведення висновку доведено.

**Приклад 2.63.** Наведемо ще один розв'язок попередньої задачі 2.36, який відрізняється за послідовністю вибору диз'юнктив для побудови резольвент. Запишемо відповідне спрощування в скороченій формі  $((((d_3 * d_6) * d_1) * d_2) * d_4) * d_5 = \emptyset$ .

Наведене доведення цікаве не лише тим, що воно коротке. Можна зауважити, що його починають із використанням заперечення висновку, а для отримання наступної резольвенти використовують побудовану резольвенту. На жаль, так діяти можна не завжди. Зокрема, для диз'юнктив  $d_1 = \{p, q\}$ ,  $d_2 = \{p, \bar{q}\}$ ,  $d_3 = \{\bar{p}, q\}$ ,  $d_4 = \{\bar{p}, \bar{q}\}$ , послідовність отримання порожнього диз'юнкту виглядає так:  $d_5 = d_3 * d_1 = \{p\}$ ,  $d_6 = d_5 * d_4 = \{\bar{p}\}$ ,  $d_7 = d_6 * d_2 = \emptyset$ .

У разі пошуку спрощування за методом резолюцій виникає два запитання, відповіди на які вимагають досвіду та інтуїції.

1. Оскільки у загальному випадку є багато способів побудови послідовності резольвент, то як будувати найкоротші такі послідовності?
2. Чи зроблено достатньо спроб для висновку, що спрощування множини  $S$  не існує?

## 2.14. ЧИСЛЕННЯ ПРЕДИКАТИВ ЯК ФОРМАЛЬНА ТЕОРІЯ. ЛОГІЧНЕ ВИВЕДЕННЯ В ЧИСЛЕННІ ПРЕДИКАТИВ

Числення предикатів як формальну систему означають так.

1. *Алфавіт  $T$  числення предикатів* містить предметні змінні, предметні сталі, предикатні символи, функціональні символи, знаки логічних зв'язок  $\vee, \wedge, \rightarrow, \neg$  та дужки  $(, )$ .

2. *Множину формул  $P$  числення предикатів* означають у два етапи.

2.1. *Терми:*

- ♦ предметні змінні та сталі є термами;
- ♦ якщо  $f$  – функціональний символ, а  $t_1, t_2, \dots, t_n$  – терми, то  $f(t_1, t_2, \dots, t_n)$  – терм.

2.2. *Формули:*

- ♦ якщо  $P$  – предикатний символ,  $t_1, t_2, \dots, t_n$  – терми, то  $P(t_1, t_2, \dots, t_n)$  – формула. Усі входження предметних змінних у формулу вигляду називають вільними;



♦ якщо  $P$  та  $Q$  – формули, то  $(P \vee Q)$ ,  $(P \wedge Q)$ ,  $(P \rightarrow Q)$  та  $(\neg P)$  також формули. Усі входження змінних, які вільні в  $P$  та  $Q$ , вільні в усіх зазначених формулах;

♦ якщо  $P(x)$  формула, яка містить вільну змінну  $x$ , то  $\forall xP(x)$  та  $\exists xP(x)$  – формули. У цих формулах усі входження змінної  $x$  називають зв'язаними, входження всіх інших змінних називають вільними.

3. Аксиоми числення предикатів поділяють на аксиоми числення висловлювань і предикатні аксиоми:

♦ аксиоми числення висловлювань – це будь-яка із систем аксіом числення висловлювань;

♦ предикатні аксиоми:

A1.  $\forall xP(x) \rightarrow P(y)$ ;

A2.  $P(y) \rightarrow \exists xP(x)$ .

У предикатних аксіомах  $P(x)$  – будь-яка формула з вільними входженнями  $x$ , причому жодна змінна  $x$  не знаходиться в області дії квантора по  $y$ . Формулу  $P(y)$  отримують з  $P(x)$  заміною на  $y$  всіх вільних входжень  $x$ .

4. Правила виведення:

♦ правило висновку (modus ponens) – те ж, що й у численні висловлювань;

♦ правило узагальнення  $P \rightarrow Q(x) \vdash P \rightarrow \forall xQ(x)$ ;

♦ правило введення квантора існування  $Q(x) \rightarrow P \vdash \exists xQ(x) \rightarrow P$ .

У цих правилах  $Q(x)$  містить вільні входження змінної  $x$ , а  $P$  – не містить.

Наведемо приклади, які ілюструють особливості вживання вільних і зв'язаних змінних у разі застосування аксіом і правил числення предикатів.

**Приклад 2.64.** Для пояснення суттєвості вимог до входження змінних  $x$  та  $y$ , розглянемо формулу  $\exists yP(x, y)$ , де вільна змінна  $x$  перебуває в області дії квантора існування. Підстановка цієї формули в аксіому A1 дає формулу  $\forall x\exists yP(x, y) \rightarrow \exists yP(y, y)$ . Розглянемо зміст цієї формули на множині натуральних чисел. Нехай предикат  $P(x, y)$  означає „ $x$  більше  $y$ ”. Тоді отримана формула означає „Якщо для довільного  $x$  існує  $y$ , яке більше від  $x$ , то існує таке  $y$ , яке більше самого себе”. У цій імплікації припущення є істинним, а висновок – фальшивим, тобто все висловлювання фальшиве.

**Приклад 2.65.** Порушення вимоги входження вільних та зв'язаних змінних може привести до неправильних висновків з істинних висловлювань. Нехай предикат  $P(x)$  означає „ $x$  має дільник 6”,  $Q(x)$  – „ $x$  має дільник 3”. Очевидно, що формула  $P(x) \rightarrow Q(x)$  істинна для довільного натурального  $x$ . Однак, застосуванням правила узагальнення отримаємо формулу  $P(x) \rightarrow \forall xQ(x)$ , яка не завжди істинна на множині натуральних чисел. Якщо тепер застосувати до заданої формули правило введення квантора існування, то отримаємо формулу  $\exists xP(x) \rightarrow Q(x)$ , яка коректна. Якщо тепер до останньої формули застосувати правило узагальнення, то отримане висловлювання  $\exists xP(x) \rightarrow \forall xQ(x)$  фальшиве на множині натуральних чисел.

Зауважимо, що існують інші системи аксіом і правил виведення.

Наведемо приклади виведення в численні предикатів.

**Приклад 2.66.** Нехай  $F(x)$  – вивідна формула числення предикатів, що містить вільну змінну  $x$ , яка не перебуває в області дії квантора по змінній  $y$ . Тоді формула  $F(y)$

теж вивідна, тобто,  $F(x) \vdash F(y)$ . Це правило називають правилом *перейменування вільних змінних*.

1)  $F(x)$  – вивідна формула за умовою.

2)  $F(x) \rightarrow (G \rightarrow F(x))$  – аксіома 2.1 з підрозділу 2.11; тут  $G$  – будь-яка вивідна формула, яка не містить вільної змінної  $x$ .

3)  $G \rightarrow F(x)$  – правило modus ponens до 1 та 2.

4)  $G \rightarrow \forall xF(x)$  – правило узагальнення до 3.

5)  $\forall xF(x)$  – правило modus ponens до 4.

6)  $F(y)$  – правило modus ponens до предикатної аксіоми A1 та 5.

**Приклад 2.67.** Нехай формула  $\forall xF(x)$  вивідна в численні предикатів і  $F(x)$  не містить вільної змінної  $y$  та містить вільну змінну  $x$ , яка не входить в область дії кванторів по  $y$ . Тоді вивідною є й формула  $\forall yF(y)$ , тобто,  $\forall xF(x) \vdash \forall yF(y)$ . Це правило називають правилом *перейменування зв'язаних змінних*.

1)  $\forall xF(x)$  – вивідна формула за умовою.

2)  $\forall xF(x) \rightarrow F(y)$  – предикатна аксіома A1.

3)  $\forall xF(x) \rightarrow \forall yF(y)$  – правило узагальнення до 2.

4)  $\forall yF(y)$  – правило modus ponens до 1 та 3.

Наступні дві теореми дають змогу зв'язати загальнозначущість у логіці першого ступеня та вивідність у численні предикатів.

**Теорема 2.10.** Будь-яка вивідна формула числення предикатів є загальнозначущою формулою логіки першого ступеня.

**Теорема 2.11.** Будь-яка загальнозначуща формула логіки першого ступеня є вивідною в численні предикатів.

**Теорема 2.12.** Нехай  $P(A)$  – формула, яка містить формулу  $A$ , а  $P(B)$  – формула, отримана з  $P(A)$  заміною  $A$  на  $B$ . Тоді, якщо  $\vdash A \sim B$ , то  $\vdash P(A) \sim P(B)$ .

Остання теорема дає правило числення предикатів, завдяки якому можна отримувати вивідні формули без побудови їхніх безпосередніх виведень.

## 2.15. ПРАВИЛА ВИВЕДЕННЯ В ЧИСЛЕННІ ПРЕДИКАТИВ

Розглянемо деякі важливі правила виведення для формул, що містять квантори.

*Універсальна конкретизація* – це правило виведення того, що  $P(c)$  істинне для довільного елемента  $c$  із предметної області в припущенні, що формула  $\forall xP(x)$  істинна. Наприклад, універсальну конкретизацію можна використати тоді, коли із твердження „Всі люди смертні” потрібно дійти висновку „Сократ смертний”. Тут Сократ – елемент предметної області, яка складається з усіх людей.

*Універсальне узагальнення* – це правило виведення, яке стверджує, що  $\forall xP(x)$  істинне, якщо істинне  $P(c)$  для довільного  $c$  із предметної області. Це правило використовують, коли на підставі істинності  $P(c)$  для кожного елемента  $c$  із предметної області стверджують, що істинне  $\forall xP(x)$ .

*Екзистенційна конкретизація* – це правило, яке дозволяє зробити висновок про те, що на підставі істинності  $\exists xP(x)$  можна стверджувати, що в предметній області є елемент  $c$ , для якого  $P(c)$  істинне. Як правило, про елемент  $c$  відомо тільки те, що він існує, а його значення може бути невідомим.

*Екзистенційне узагальнення* – це правило виведення, яке використовують для того, щоб на підставі істинності  $P(c)$  на деякому елементі  $c$  із предметної області зробити висновок, що  $\exists xP(x)$  істинне. Уведені правила виведення зібрано в табл. 2.12.

Таблиця 2.12

| №  | Правило виведення           | Назва                       |
|----|-----------------------------|-----------------------------|
| 1. | $\forall xP(x) \vdash P(c)$ | Універсальна конкретизація  |
| 2. | $P(c) \vdash \forall xP(x)$ | Універсальне узагальнення   |
| 3. | $\exists xP(x) \vdash P(c)$ | Екзистенційна конкретизація |
| 4. | $P(c) \vdash \exists xP(x)$ | Екзистенційне узагальнення  |

У правилах 1 і 2 елемент  $c$  предметної області – довільний, а в правилах 3 та 4 в предметній області повинен існувати принаймні один такий елемент.

**Приклад 2.68.** Покажемо, що гіпотези „Кожний, хто вчиться на комп'ютерних науках, слухає курс дискретної математики” та „Марія вчиться на комп'ютерних науках” дозволяє сформулювати висновок, що „Марія слухає курс дискретної математики”.

Нехай предикат  $D(x)$  означає „ $x$  вчиться на комп'ютерних науках”,  $C(x)$ : „ $x$  слухає курс дискретної математики”. Тоді гіпотезами є формули  $\forall x(D(x) \rightarrow C(x))$  та  $D(\text{Марія})$ , а висновком –  $C(\text{Марія})$ . Для доведення висновку із заданої множини гіпотез потрібно виконати такі кроки.

- 1)  $\forall x(D(x) \rightarrow C(x))$  – гіпотеза.
- 2)  $D(\text{Марія}) \rightarrow C(\text{Марія})$  – універсальна конкретизація до 1.
- 3)  $D(\text{Марія})$  – гіпотеза.
- 4)  $C(\text{Марія})$  – modus ponens до 2 та 3.

**Приклад 2.69.** Покажемо, що на основі гіпотез „У групі є студент, який не читав підручника” та „Всі студенти групи склали іспит” можна одержати висновок „Дехто з тих, хто склав іспит, не читав підручника”.

Нехай  $C(x)$  означає „ $x$  вчиться в групі”,  $B(x)$  – „ $x$  читав підручник” та  $P(x)$  – „ $x$  склав іспит”. Гіпотезами є  $\exists x(C(x) \wedge \bar{B}(x))$  та  $\forall x(C(x) \rightarrow P(x))$ , а висновком –  $\exists x(P(x) \wedge \bar{B}(x))$ . Виконаємо таку послідовність кроків обґрунтування.

- 1)  $\exists x(C(x) \wedge \bar{B}(x))$  – гіпотеза.
- 2)  $C(a) \wedge \bar{B}(a)$  – екзистенційна конкретизація до 1.
- 3)  $C(a)$  – виключення кон'юнкції до 2.
- 4)  $\forall x(C(x) \rightarrow P(x))$  – гіпотеза.
- 5)  $C(a) \rightarrow P(a)$  – універсальна конкретизація до 4.
- 6)  $P(a)$  – modus ponens до 3 та 5.

- 7)  $\bar{B}(a)$  – виключення кон'юнкції до 2.
- 8)  $P(a) \wedge \bar{B}(a)$  – введення кон'юнкції до 6 та 7.
- 9)  $\exists x(P(x) \wedge \bar{B}(x))$  – екзистенційне узагальнення до 8.

## 2.16. ПІДСТАНОВКА ТА УНІФІКАЦІЯ. НАЙЗАГАЛЬНІШИЙ УНІФІКАТОР. АЛГОРИТМ УНІФІКАЦІЇ

*Підстановкою* називають скінченну множину пар вигляду  $\sigma = \{t_1/x_1, t_2/x_2, \dots, t_n/x_n\}$ , де кожне  $x_i$  ( $i=1, 2, \dots, n$ ) є змінною,  $t_i$  – термом, відмінним від  $x_i$ , та всі  $x_i$  – різні.

**Приклад 2.70.** Множини  $\{f(z)/x, y/z\}$  та  $\{a/x, g(y)/y, f(g(b))/z\}$  являють собою підстановки.

Нехай  $\sigma = \{t_1/x_1, t_2/x_2, \dots, t_n/x_n\}$  – підстановка, а  $E$  – формула. Тоді формулу  $E\sigma$ , яку одержують заміною в  $E$  всіх входжень змінної  $x_i$  на терм  $t_i$  ( $i=1, 2, \dots, n$ ), називають *прикладом* формули  $E$ . Якщо  $E\sigma$  не містить змінних, то  $\sigma$  називають *основною підстановкою*, а приклад  $E\sigma$  – *основним прикладом*.

**Приклад 2.71.** Для формули  $E = P(x, y, z)$  та підстановки  $\sigma = \{a/x, f(b)/y, c/z\}$ , формула  $E = P(a, f(b), c)$  є прикладом (основним).

**Приклад 2.72.** Нехай формула  $B = B(x, y)$  має зміст „ $x$  брат  $y$ ”. Підстановкою  $\sigma = \{\text{Богдан}/x, \text{Степан}/y\}$  отримаємо висловлювання, яке є основним прикладом  $B\sigma = B(\text{Богдан}, \text{Степан})$  заданої формули. Підстановкою  $\theta = \{\text{Іван}/x\}$  отримуємо формулу  $B\theta = B(\text{Іван}, y)$ , яка не є основним прикладом, бо містить змінну  $y$ .

**Приклад 2.73.** Прикладом формули  $E = G(f(x), g(f(z)), y)$ , отриманої за допомогою підстановки  $\sigma = \{f(x)/x, g(x, z)/y\}$ , є формула  $E\sigma = G(f(f(x)), g(f(z)), g(x, z))$ .

*Підстановку можна застосовувати лише до вільних змінних у формулі.*

**Приклад 2.74.** Побудуємо довільний приклад формули

$$\forall x(P(x, y, z) \vee Q(a, x)) \rightarrow \exists uR(b, f(u), v),$$

де  $y, z, v$  – вільні змінні;  $a, b$  – сталі, а  $f$  – функціональний символ. Приклад можна отримати підстановкою  $\{t_1/y, t_2/z, t_3/v\}$ . Обов'язковою умовою є те, що терми  $t_1, t_2, t_3$  не повинні містити входжень змінних  $x$  та  $u$ , які зв'язані кванторами в заданій формулі. У результаті отримаємо такий приклад

$$\forall x(P(x, t_1, t_2) \vee Q(a, x)) \rightarrow \exists uR(b, f(u), t_3).$$

*Означимо процедурним способом композицію підстановок.* Нехай  $\theta = \{t_1/x_1, t_2/x_2, \dots, t_n/x_n\}$  і  $\lambda = \{u_1/y_1, u_2/y_2, \dots, u_m/y_m\}$  – підстановки. Тут  $t_j, u_i$  – терми,  $x_j, y_i$  – змінні;  $i=1, \dots, n, j=1, \dots, m$ . Композицію підстановок  $\theta$  і  $\lambda$  називають підстановкою  $\lambda \circ \theta$ , яку одержують так.

1. Застосовують підстановку  $\lambda$  до кожного терму з  $\theta$  і результат об'єднують з  $\lambda$ :

$$\{t_1\lambda/x_1, t_2\lambda/x_2, \dots, t_n\lambda/x_n, u_1/y_1, u_2/y_2, \dots, u_m/y_m\}.$$

2. Із цієї множини вилучають всі елементи  $t_k\lambda/x_k$ , для яких  $x_k = t_k\lambda$ , а також всі елементи  $u_l/y_l$ , для яких  $y_l \in \{x_1, \dots, x_n\}$ .

**Приклад 2.75.** Побудуємо композицію підстановок  $\theta = \{f(y)/x, z/y\}$  і  $\lambda = \{a/x, b/y, y/z\}$ . Застосуємо  $\lambda$  до кожного терму з  $\theta$ :  $\{f(y)\lambda/x, z\lambda/y\} = \{f(b)/x, y/y\}$ . Тут застосування  $\lambda$  до  $f(y)$  дає  $f(b)$ , бо за підстановкою  $\lambda$  змінна  $y$  замінюється на терм  $b$ , а застосування  $\lambda$  до  $z$  дає  $y$ , бо за підстановкою  $\lambda$  змінна  $z$  замінюється на терм  $y$ . Далі об'єднуємо  $\{f(b)/x, y/y\}$  з  $\lambda$ , що дасть  $\{f(b)/x, y/y, a/x, b/y, y/z\}$ . Тепер вилучаємо із цієї множини  $y/y$ , бо  $z\lambda = y$ , а також  $a/x$  та  $b/y$ , бо змінні  $x$  та  $y$  є в  $\theta$  після слеша. Остаточоно  $\lambda \circ \theta = \{f(b)/x, y/z\}$ .

Нехай  $E$  – формула. Із означення композиції випливає, що  $(E\theta)\lambda = E(\lambda \circ \theta)$ . Застосуємо до формули  $E(x, y, z)$  підстановку  $\theta$ ; одержимо  $E(x, y, z)\theta = E(f(y), z, z)$ . Тепер застосуємо підстановку  $\lambda$  і одержимо  $(E(x, y, z)\theta)\lambda = (E(f(y), z, z))\lambda = E(f(b), y, y)$ . Цей же результат одержимо, застосувавши до формули  $E(x, y, z)$  підстановку  $\lambda \circ \theta = \{f(b)/x, y/z\}$ .

Значимо, що  $(\lambda \circ \theta) \circ \sigma = \lambda \circ (\theta \circ \sigma)$ , але в загальному випадку  $\lambda \circ \theta \neq \theta \circ \lambda$ .

**Приклад 2.76.** Нехай формула  $E = E(x, y)$  має зміст „ $x$  батько  $y$ ”. Підстановкою  $\theta = \{\text{Іван}/x\}$  отримаємо приклад  $E\theta = E(\text{Іван}, y)$ . Ще однією підстановкою  $\lambda = \{\text{Богдан}/y\}$ , тобто, побудовою композиції підстановок  $\sigma = \lambda \circ \theta$ , отримаємо  $E\sigma = E(\text{Іван}, \text{Богдан})$ , що є основним прикладом формули  $E(x, y)$ .

*Означимо поняття уніфікатора.* Підстановку  $\theta$  називають уніфікатором множини  $\{E_1, E_2, \dots, E_k\}$ , якщо  $E = E_1\theta = E_2\theta = \dots = E_k\theta$ , де  $E$  – спільний приклад елементів заданої множини. Множина  $\{E_1, E_2, \dots, E_k\}$  уніфікується, якщо для неї існує уніфікатор. Будь-який уніфікатор визначає спільний приклад, і, навпаки, будь-який спільний приклад визначає уніфікатор.

**Приклад 2.77.** Перевіримо, чи можна уніфікувати множину  $\{P(a, y), P(x, f(b))\}$ . Ця множина уніфікується, оскільки підстановка  $\theta = \{a/x, f(b)/y\}$  є її уніфікатором. Справді,  $P(a, y)\theta = P(x, f(b))\theta = P(a, f(b))$ .

**Приклад 2.78.** Задано формули

$$E_1 = \text{КНИГА}(x, y, \text{видавництво}(x, 1985)),$$

$$E_2 = \text{КНИГА}(u, v, \text{видавництво}(d, 1985))$$

та підстановку  $\sigma = \{\text{Каменяр}/x, w/y, \text{Каменяр}/u, w/v, \text{Каменяр}/d\}$ . Переконаємось, що  $\sigma$  – уніфікатор  $E_1$  та  $E_2$ , і побудуємо приклади цих формул.

Формула

$$E_1\sigma = E_2\sigma = \text{КНИГА}(\text{Каменяр}, w, \text{видавництво}(\text{Каменяр}, 1985))$$

– спільний приклад формул  $E_1$  та  $E_2$ . Знайдемо інші уніфікатори.

Уніфікатор

$$\sigma_1 = \{\text{Каменяр}/x, \text{Франко}/y, \text{Каменяр}/u, \text{Франко}/v, \text{Каменяр}/d\}$$

дає такий спільний приклад заданих формул:

$$E_1\sigma_1 = E_2\sigma_1 = \text{КНИГА}(\text{Каменяр}, \text{Франко}, \text{видавництво}(\text{Каменяр}, 1985)),$$

а уніфікатор

$$\sigma_2 = \{\text{Каменяр}/x, z/y, \text{Каменяр}/u, z/v, \text{Каменяр}/d\}$$

дає інший спільний приклад заданих формул

$$E_1\sigma_2 = E_2\sigma_2 = \text{КНИГА}(\text{Каменяр}, z, \text{видавництво}(\text{Каменяр}, 1985)).$$

Отже, множина може мати декілька уніфікаторів.

Уніфікатор  $\sigma$  множини формул  $\{E_1, E_2, \dots, E_k\}$  називають найзагальнішим уніфікатором, якщо для кожного уніфікатора  $\theta$  цієї множини існує підстановка  $\lambda$  така, що  $\theta = \lambda \circ \sigma$ .

Уведемо поняття множини неузгодженості непорожньої множини формул  $W$ . Для цього формули, елементи множини  $W$ , розглядають як рядки символів. Зліва направо проглядають попарно ці рядки та знаходять першу позицію, у якій хоча б одна пара рядків містить різні символи термів. У множину неузгодженості  $D$  виписують саме ті терми, які відрізняються.

**Приклад 2.79.** Розглянемо множину формул  $W = \{P(a), P(x)\}$ . Їхня неузгодженість полягає в тому, що для одного предикатного символу  $P$  формула  $P(a)$  містить сталу  $a$ , а формула  $P(x)$  – змінну  $x$ . Тому  $D = \{a, x\}$  – множина неузгодженості цієї пари формул. Оскільки  $x$  – змінна, а стала  $a$  – терм, то підстановка  $\{a/x\}$  усуває неузгодженість.

**Приклад 2.80.** Задано множину  $W = \{P(x, f(y, x)), P(x, a), P(x, g(h(k(x))))\}$ . Побудуємо її множину неузгодженості. Формули в множині  $W$  мають однакові перші чотири символи  $P(x, (кома), i$  відрізняються між собою, починаючи з  $n$ -'ятого символу. Отже,  $D = \{f(y, x), a, g(h(k(x)))\}$ .

Розглянемо алгоритм знаходження найзагальнішого уніфікатора скінченної множини атомів логіки першого ступеня. Якщо вказану множину неможливо уніфікувати, то алгоритм виявляє цей факт.

### Алгоритм уніфікації

- Крок 1. Покласти  $k = 0$ . Увести множину  $W_k = W$  та порожній уніфікатор  $\sigma_k = \varepsilon$
- Крок 2. Якщо множина  $W_k$  одноелементна, то уніфікатор  $\sigma_k$  – найзагальніший уніфікатор для  $W$ . Кінець.
- Крок 3. Якщо множина  $D_k$  містить таку змінну  $x_k$  та терм  $t_k$ , що змінної  $x_k$  немає в термі  $t_k$ , то перейти до кроку 4. Інакше, множину  $W$  неможливо уніфікувати. Кінець.
- Крок 4. Побудувати  $\sigma_{k+1} = \{t_k/x_k\} \circ \sigma_k$  та  $W_{k+1} = W_k \setminus \{t_k/x_k\}$  (Зауважимо, що  $W_{k+1} = W_0 \circ \sigma_{k+1}$ ).
- Крок 5. Виконати  $k := k+1$  та перейти до кроку 2.

**Приклад 2.81.** Знайдемо найзагальніший уніфікатор множини формул

$$W = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}.$$

Опишемо роботу алгоритму уніфікації по кроках.

Крок 1. Покладемо  $k = 0$ ,  $\sigma_0 = \varepsilon$ ,  $W_0 = W$ .

Крок 2. Оскільки множина  $W_0$  не одноелементна, то уніфікатор  $\sigma_0$  не є найзагальнішим уніфікатором множини  $W_0$ , а множина неузгодженості має вигляд  $D_0 = \{a, z\}$ .

Крок 3. Множина  $D_0$  містить змінну  $x_0 = z$ , якої немає в термі  $t_0 = a$ .

Крок 4. Побудуємо уніфікатор  $\sigma_1 = \{t_0/x_0\} \circ \sigma_0 = \{t_0/x_0\} \circ \varepsilon = \{t_0/x_0\} = \{a/z\}$  та множину  $W_1 = W_0 \setminus \{t_0/x_0\} =$

$$= \{P(a, x, f(g(y))), P(z, f(z), f(u))\} \setminus \{a/z\} = \{P(a, x, f(g(y))), P(a, f(a), f(u))\}.$$

Крок 5. Покладемо  $k = 1$  та повернемося до кроку 2 алгоритму.

Крок 2. Оскільки множина  $W_1$  не одноелементна, то уніфікатор  $\sigma_1$  не є найзагальнішим уніфікатором множини  $W_1$ , а множина неузгодженості має вигляд  $D_1 = \{x, f(a)\}$

Крок 3. Множина  $D_1$  містить змінну  $x_1 = x$ , якої немає в термі  $t_1 = f(a)$ .

Крок 4. Побудуємо уніфікатор

$$\sigma_2 = \{t_1/x_1\} \circ \sigma_1 = \{f(a)/x\} \circ \{a/z\} = \{a/z, f(a)/x\}$$

та множину

$$W_2 = W_1\{t_1/x_1\} =$$

$$= \{P(a, x, f(g(y))), P(a, f(a), f(u))\} \{f(a)/x\} = \{P(a, f(a), f(g(y))), P(a, f(a), f(n))\},$$

застосувавши цей уніфікатор.

Крок 5. Покладемо  $k = 2$  та повернемося до кроку 2 алгоритму.

Крок 2. Оскільки множина  $W_2$  не одноелементна, то уніфікатор  $\sigma_2$  не є найзагальнішим уніфікатором множини  $W_2$ , а множина неузгодженості набуває вигляду  $D_2 = \{g(y), u\}$ .

Крок 3. Множина  $D_2$  містить змінну  $x_2 = u$ , якої немає в термі  $t_2 = g(y)$ .

Крок 4. Побудуємо підстановку

$$\sigma_3 = \{t_2/x_2\} \circ \sigma_2 = \{g(y)/u\} \circ \{a/z, f(a)/x\} = \{a/z, f(a)/x, g(y)/u\},$$

і уніфікуємо множину

$$W_3 = W_2\{t_2/x_2\} = \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\} \{g(y)/u\} = \\ = \{P(a, f(a), f(g(y))), P(a, f(a), f(g(y)))\} = \{P(a, f(a), f(g(y)))\}.$$

Крок 5. Покладемо  $k = 3$  і повернемося до кроку 2 алгоритму.

Крок 2. Оскільки множина  $W_3$  одноелементна, то уніфікатор  $\sigma_3 = \{a/z, f(a)/x, g(y)/u\}$  – найзагальніший уніфікатор множини  $W$ .

**Приклад 2.82.** Визначимо, чи можна уніфікувати множину  $W = \{Q(f(a), g(x)), Q(y, y)\}$ .

Опишемо роботу алгоритму уніфікації по кроках.

Крок 1. Покладемо  $k = 0$ ,  $\sigma_0 = \varepsilon$ ,  $W_0 = W$ .

Крок 2. Оскільки множина  $W_0$  не одноелементна, то уніфікатор  $\sigma_0$  не є найзагальнішим уніфікатором множини  $W_0$ , а множина неузгодженості набуває вигляду  $D_0 = \{f(a), y\}$ .

Крок 3. Множина  $D_0$  містить змінну  $x_0 = y$ , якої немає в термі  $t_0 = f(a)$ .

Крок 4. Побудуємо підстановку

$$\sigma_1 = \{t_0/x_0\} \circ \sigma_0 = \{f(a)/y\} \circ \varepsilon = \{f(a)/y\}$$

та множину

$$W_1 = W_0\{t_0/x_0\} = \{Q(f(a), g(x)), Q(y, y)\} \{f(a)/y\} = \{Q(f(a), g(x)), Q(f(a), f(a))\}.$$

Крок 5. Покладемо  $k = 1$  та повернемося до кроку 2 алгоритму.

Крок 2. Оскільки множина  $W_1$  не одноелементна, то уніфікатор  $\sigma_1$  не є найзагальнішим уніфікатором множини  $W_1$ , а множина неузгодженості набуває вигляду  $D_1 = \{g(x), f(a)\}$ .

Крок 3. У множині  $D_1$  відсутні елементи, які можна використати для побудови підстановки. Отже, множину  $W$  неможливо уніфікувати.

## 2.17. СКОЛЕМІВСЬКА НОРМАЛЬНА ФОРМА

Метод резолюцій застосовують для перевірки невиконанності формул логіки першого ступеня, зведених до стандартної форми, яку називають *нормальною формою* *Сколема*, або *сколемівською нормальною формою*. Для того, щоб побудувати

сколемівську нормальну форму, спочатку потрібно звести формулу логіки першого ступеня до випередженої нормальної форми  $Q_1x_1Q_2x_2\ldots Q_nx_nM$ , у якій матриця  $M$  (див. підрозділ 2.7) не містить жодних кванторів, а в префіксі  $Q_1x_1Q_2x_2\ldots Q_nx_n$  кожне  $Q_ix_i$  ( $i=1, 2, \ldots, n$ ) є або  $\exists x_i$ , або  $\forall x_i$ .

Тепер необхідно вилучити квантори існування та ввести *сколемівські сталі* й *сколемівські функції*. Процес вилучення кванторів існування виконують у два етапи.

**Етап 1.** Нехай  $Q_1x_1Q_2x_2\ldots Q_nx_nM$  – випереджена нормальна форма формули, а  $Q_i$  – квантор існування у її префіксі  $Q_1x_1Q_2x_2\ldots Q_nx_n$ ,  $1 \leq i \leq n$ . Якщо в префіксі зліва від  $Q_i$  не стоїть жодного квантора загальності, то сталою  $c$ , відмінною від інших сталих в  $M$ , замінимо всі входження змінної  $x_i$  у матрицю  $M$  і вилучимо  $Q_ix_i$  із префікса.

**Етап 2.** Якщо  $Q_{s_1}, \ldots, Q_{s_m}$  – квантори загальності, які розташовані зліва від  $Q_i$ ,

$1 \leq s_1 \leq s_2 \leq \ldots \leq s_m < i$ , то вводим нову  $m$ -місну функцію  $f(x_{s_1}, \ldots, x_{s_m})$ , відмінну від усіх інших функцій у матриці  $M$ . Замінимо цією функцією всі входження змінних  $x_i$  у  $M$  та вилучимо  $Q_ix_i$  із префікса. Проглядаючи префікс зліва направо, вилучимо з нього квантори існування. Отримана формула є сколемівською нормальною формою.

Сталі та функції, які використано для заміни змінних, зв'язаних кванторами існування, є сколемівськими сталими та функціями. Ці сталі та функції не мають жодного змістовного значення.

**Приклад 2.83.** Побудуємо сколемівську нормальну форму формули

$$\exists x \forall y \forall z \exists u \forall v \exists w P(x, y, z, u, v, w).$$

У цій формулі  $P(x, y, z, u, v, w)$  – матриця, а  $\exists x \forall y \forall z \exists u \forall v \exists w$  – префікс. Матриця  $P(x, y, z, u, v, w)$  не містить жодних кванторів, тобто задана формула записана у випередженій нормальній формі. Виконаємо такі дії.

1. Переглядаємо послідовно зліва направо квантори у формулі. Оскільки зліва від  $\exists x$  немає жодного квантора загальності, замінимо змінну  $x$  на сколемівську сталу  $a$  та вилучимо  $x$  із заданої формули. Отримаємо  $\forall y \forall z \exists u \forall v \exists w P(a, y, z, u, v, w)$ .

2. Зліва від  $\exists u$  стоять  $\forall y$  та  $\forall z$ . Тому замінимо змінну  $u$  на двомісну сколемівську функцію  $f(y, z)$ , вилучимо  $\exists u$  із формули та отримаємо  $\forall y \forall z \forall v \exists w P(a, y, z, f(y, z), v, w)$ .

3. Зліва від  $\exists w$  стоять  $\forall y$ ,  $\forall z$  та  $\forall v$ . Замінимо змінну  $w$  на тримісну сколемівську функцію  $g(y, z, v)$ , вилучимо  $\exists w$  із формули та остаточно отримаємо сколемівську нормальну форму  $\forall y \forall z \forall v P(a, y, z, f(y, z), v, g(y, z, v))$  заданої формули.

Якщо матрицю сколемівської нормальної форми записано в кон'юнктивній нормальній формі, то таку сколемівську форму називають *клаузальною* нормальною формою.

**Приклад 2.84.** Побудуємо клаузальну нормальну форму формули

$$\forall x \exists y \exists z ((\bar{P}(x, y) \wedge G(x, z)) \vee R(x, y, z)).$$

Цю формулу записано у випередженій нормальній формі із префіксом  $\forall x \exists y \exists z$  та матрицею  $((\bar{P}(x, y) \wedge G(x, z)) \vee R(x, y, z))$ .

Спочатку запишемо матрицю в кон'юнктивній нормальній формі  $\forall x \exists y \exists z ((\bar{P}(x, y) \vee R(x, y, z)) \wedge (G(x, z) \vee R(x, y, z)))$ . Оскільки перед  $\exists y$  та  $\exists z$  розміщений  $\forall x$ , то змінні  $y$  та  $z$  замінимо одномісними сколемівськими функціями  $f(x)$  та  $g(x)$ , відповідно. Клаузальна нормальна форма заданої формули набуває вигляду

$$\forall x(\bar{P}((x, f(x)) \vee R(x, f(x), g(x))) \wedge (G(x, g(x)) \vee R(x, f(x), g(x)))).$$

У побудованій формі диз'юнктами є  $\bar{P}(x, f(x)) \vee R(x, f(x), g(x))$  та  $G(x, g(x)) \vee R(x, f(x), g(x))$ , а кожна змінна зв'язана квантором загальності. Цю нормальну форму можна також записати у вигляді множини диз'юнктивів

$$S = \{ \bar{P}(x, f(x)) \vee R(x, f(x), g(x)), G(x, g(x)) \vee R(x, f(x), g(x)) \}.$$

**Теорема 2.13.** Нехай  $S$  – сколемівська нормальна форма формули  $G$ . Формула  $G$  невиконання тоді й лише тоді, коли невиконання формула  $S$ .

**Доведення.** Нехай формулу  $G$  записано у випередженій нормальній формі  $G = Q_1 x_1 Q_2 x_2 \dots Q_n x_n M(x_1, x_2, \dots, x_n)$  та  $x_r$  – перша змінна в префіксі в разі перегляду його зліва направо, яка зв'язана квантором існування. Тому

$G_1 = \forall x_1 \dots \forall x_{r-1} Q_{r+1} x_{r+1} \dots Q_n x_n M(x_1, \dots, x_{r-1}, f(x_1, \dots, x_{r-1}), x_{r+1}, \dots, x_n)$ , де  $f(x_1, \dots, x_{r-1})$  – сколемівська функція, яка відповідає змінній  $x_r$ ,  $1 \leq r \leq n$ .

Покажемо, що  $G$  невиконання тоді й лише тоді, коли невиконання формула  $G_1$ . Припустимо, що формула  $G$  невиконання. Якщо  $G_1$  виконання, то існує така інтерпретація  $I$ , у якій  $G_1$  істинна. Тому для всіх  $x_1, x_2, \dots, x_{r-1}$  існує принаймні один елемент, а саме  $f(x_1, x_2, \dots, x_{r-1})$ , для якого  $Q_{r+1} x_{r+1} \dots Q_n x_n M(x_1, \dots, x_{r-1}, f(x_1, \dots, x_{r-1}), x_{r+1}, \dots, x_n)$  істинна в  $I$ . Таким чином,  $G$  істинне в  $I$ , а це суперечить припущенню, що  $G$  невиконання. Отже,  $G_1$  має бути невиконанням в інтерпретації  $I$ .

Тепер припустимо, що  $G_1$  невиконання. Якщо  $G$  виконання, то існує така інтерпретація  $I$ , що  $G$  істинна в  $I$ . Тобто, для всіх  $x_1, x_2, \dots, x_n$  існує такий елемент  $x_r$ , що формула

$$Q_{r+1} x_{r+1} \dots Q_n x_n M(x_1, \dots, x_{r-1}, x_r, x_{r+1}, \dots, x_n)$$

істинна в  $I$ . Нехай  $I'$  є розширенням інтерпретації  $I$  функцією  $f(x_1, x_2, \dots, x_{r-1})$  такою, що  $f(x_1, x_2, \dots, x_{r-1}) = x_r$  для всіх  $x_1, x_2, \dots, x_{r-1}$  із предметної області  $D$ . Зрозуміло, що для всіх  $x_1, x_2, \dots, x_{r-1}$  формула

$$Q_{r+1} x_{r+1} \dots Q_n x_n M(x_1, \dots, x_{r-1}, f(x_1, \dots, x_{r-1}), x_{r+1}, \dots, x_n)$$

істинна в  $I'$ , тобто  $G_1$  істинна в  $I'$ , що суперечить припущенню про невиконання формули  $G_1$ . Отже, формула  $G$  має бути невиконанням.

Припустимо тепер, що формула  $G$  має  $m$  кванторів існування. Нехай  $G_0 = G$ , а  $G_k$ , ( $k=1, 2, \dots, m$ ), отримують з  $G_{k-1}$  заміною першого квантора існування в  $G_{k-1}$  сколемівською функцією. Очевидно, що  $S = G_m$ . Використовуючи ті самі міркування, що й наведено вище, можна показати, що  $G_{k-1}$  невиконання тоді й лише тоді, коли невиконання  $G_k$ . Звідси доходимо висновку, що  $G$  невиконання тоді й лише тоді, коли  $S$  невиконання.

Нехай  $S$  – сколемівська нормальна форма формули  $G$ . Якщо  $G$  невиконання, то за доведеною теоремою  $G$  еквівалентна  $S$ , або  $G=S$ . Якщо ж  $G$  виконання, то вона, взагалі кажучи, не еквівалентна  $S$ . Проілюструємо це на прикладі.

**Приклад 2.85.** Нехай  $G = \exists x P(x)$ ,  $S = P(a)$ , тобто,  $S$  – сколемівська нормальна форма формули  $G$ . Визначимо інтерпретацію  $I$  на предметній області  $D = \{1, 2\}$  так:  $a = 1$ ,  $P(1) = F$ ,  $P(2) = T$ . Тоді  $G$  істинна в  $I$ , а  $S$  фальшива в  $I$ , тобто  $G$  та  $S$  не еквівалентні.

## 2.18. МЕТОД РЕЗОЛЮЦІЙ У ЧИСЛЕННІ ПРЕДИКАТИВ

Проблема доведення теорем у логіці першого ступеня (логіці предикатів) істотно відрізняється від аналогічної проблеми в логіці висловлювань. Це пояснюється тим, що в логіці висловлювань існує лише скінченна кількість інтерпретацій формули, тоді як у логіці предикатів інтерпретацій може існувати нескінченно багато. А. Чорч (A. Church) та А. Тьюрінг незалежно один від одного довели, що не існує алгоритму, який перевіряє тотожну істинність формул логіки першого ступеня. Але, незважаючи на це, існують алгоритми пошуку доведення теорем, які можуть підтверджувати, що формула загальнозначуща, коли вона справді є такою. Зазначимо, що коли формула не є загальнозначущою, то ці алгоритми, взагалі кажучи, свою роботу можуть і не завершити. У зв'язку з результатами Чорча та Тьюрінга це найліпше, на що можна сподіватися. Найбільш відомий з таких алгоритмів – метод резолюцій.

Якщо два або більше атомів з однаковими предикатними символами у диз'юнкті  $d$  мають найзагальніший уніфікатор  $\sigma$ , то  $d\sigma$  називають *склейкою*  $d$ . Якщо  $d\sigma$  – диз'юнкт рангу 1, то склейку називають *одиночною*.

**Приклад 2.86.** Задано диз'юнкт  $d = P(x) \vee P(f(y)) \vee \bar{Q}(x)$ . Атоми із предикатним символом  $P$  мають найзагальніший уніфікатор  $\sigma = \{f(y)/x\}$ , тобто,  $d\sigma = P(f(y)) \vee \bar{Q}(f(y))$  – склейка диз'юнкта  $d$ .

Нехай  $l_1$  та  $l_2$  – атоми логіки першого ступеня в диз'юнктах  $d_1$  та  $d_2$ , відповідно, та існує підстановка  $\sigma$  така, що  $l_1\sigma$  та  $l_2\sigma$  утворюють контрарну пару. Якщо диз'юнкти  $d_1$  та  $d_2$  зображені множинами літералів, то диз'юнкт  $\{d_1\sigma \vee \sigma\} \cup \{d_2\sigma \wedge \bar{\sigma}\}$  називають *бінарною резольвентою*  $d_1$  та  $d_2$ , а літерали  $l_1$  та  $l_2$  – *відрізними*.

**Приклад 2.87.** Задано диз'юнкти  $d_1 = P(x) \vee Q(x)$  та  $d_2 = \bar{P}(a) \vee R(y)$ . Задамо ці диз'юнкти множинами атомів  $d_1 = \{P(x), Q(x)\}$ ,  $d_2 = \{\bar{P}(a), R(y)\}$  та виберемо літерали  $l_1 = P(x)$  та  $l_2 = \bar{P}(a)$ . Застосуємо підстановку  $\sigma = \{a/x\}$  та отримаємо такі формули  $d_1\sigma = \{P(a), Q(a)\}$ ,  $d_2\sigma = \{\bar{P}(a), R(y)\}$ ,  $l_1\sigma = P(a)$ ,  $l_2\sigma = \bar{P}(a)$ ,  $d_1\sigma \vee l_1\sigma = \{Q(a)\}$ ,  $d_2\sigma \vee l_2\sigma = \{R(y)\}$ . Формули  $l_1\sigma$  та  $l_2\sigma$  утворили контрарну пару. Тоді бінарна резольвента має вигляд  $\{d_1\sigma \vee \sigma\} \cup \{d_2\sigma \wedge \bar{\sigma}\} = \{Q(a), R(y)\}$ , або в іншому записі  $Q(a) \vee R(y)$ . Тут літерали  $l_1$  та  $l_2$  – відрізні.

*Резольвентою диз'юнктив  $d_1$  та  $d_2$  є одна з формул:*

- бінарна резольвента  $d_1$  та  $d_2$ ;
- бінарна резольвента  $d_1$  та склейки  $d_2$ ;
- бінарна резольвента  $d_2$  та склейки  $d_1$ ;
- бінарна резольвента склейки  $d_1$  та склейки  $d_2$ .

**Приклад 2.88.** Нехай

$$d_1 = P(x) \vee P(f(y)) \vee R(g(y)) \text{ та } d_2 = \bar{P}(f(g(a))) \vee Q(f(b)).$$

Склейкою  $d_1$  є диз'юнкт  $d'_1 = P(f(y)) \vee R(g(y))$ , а бінарною резольвентою диз'юнктивів

$$d'_1 \text{ та } d_2 \text{ – диз'юнкт } R(g(g(a))) \vee Q(f(b)).$$

У разі обґрунтування міркувань у численні предикатів за методом резолюцій розглядають два типи задач.

1. Перевірити, чи можна спростувати множину диз'юнктивів.
2. Знайти елемент предметної області, на якому спростована множина диз'юнктивів.

Наведемо приклади міркувань та їхнього обґрунтування із застосуванням методу резолюцій у поєднанні з уніфікацією відповідних формул. Спочатку розглянемо задачу першого типу.

**Приклад 2.89.** Розглянемо класичний приклад міркувань – силізізм Аристотеля: “Кожна людина смертна. Оскільки Сократ людина, то він смертний”. Ці міркування інтуїтивно коректні. Уведемо такі позначення:

- $p$ : „Кожна людина смертна”;  
 $q$ : „Сократ – людина”;  
 $r$ : „Сократ смертний”.

У таких позначеннях  $r$  не буде логічним наслідком  $p$  та  $q$  у численні висловлювань, оскільки зміст атомів  $p$ ,  $q$  та  $r$  не аналізують і для побудови логічного висновку не використовують.

Проілюструємо доведення силізізму методом резолюцій у численні предикатів. Для цього введемо предикати:

- $H(x)$ : „ $x$  – людина”  
 $M(x)$ : „ $x$  – смертна”.

Тепер речення „Кожна людина смертна” задамо формулою  $\forall x(H(x) \rightarrow M(x))$ , речення „Сократ – людина” – формулою  $H(\text{Сократ})$ , „Сократ смертний” –  $M(\text{Сократ})$ , а все речення запишемо формулою  $\forall x(H(x) \rightarrow M(x)) \wedge H(\text{Сократ}) \rightarrow M(\text{Сократ})$ .

Для скорочення запису формул позначатимемо  $C$  замість „Сократ”. Згідно з принципом прямої дедукції (теорема 2.3) побудуємо формулу

$$\forall x(H(x) \rightarrow M(x)) \wedge H(C) \wedge \bar{M}(C),$$

невиконанність якої доведемо за методом резолюцій.

Виключивши імплікацію одержимо формулу

$$\forall x((\bar{H}(x) \vee M(x)) \wedge H(C) \wedge (C))$$

– клаузайну форму попередньої формули. Випускаючи квантор загальності та зайві дужки, одержимо кон'юнктивну нормальну форму матриці

$$(\bar{H}(x) \vee M(x)) \wedge H(C) \wedge \bar{M}(C).$$

Застосуємо алгоритм резолюцій для спростування такої множини диз'юнктивів:

- (1)  $\bar{H}(x) \vee M(x)$ ;
- (2)  $H(C)$ ;
- (3)  $\bar{M}(C)$ .

Застосуванням методу резолюцій разом із підстановкою  $\{C/x\}$  отримуємо резольвенти. Біля кожної резольвенти виписано номери диз'юнктивів, які використані для їхньої побудови.

- (4)  $M(C)$  (1), (2);
- (5)  $\square$  (3), (4).

Оскільки отримано порожній диз'юнктив, то силізізм доведено.

**Приклад 2.90.** Задано формули:

- $f_1$ :  $\exists x(C(x) \wedge Q(x))$ ;  
 $f_2$ :  $\forall x(C(x) \rightarrow (W(x) \wedge R(x)))$ ;  
 $g$ :  $\exists x(Q(x) \wedge R(x))$ .

Довести, що формулу  $g$  можна вивести з формул  $f_1$  та  $f_2$ .

Сформулюємо цю задачу як задачу перевірки невиконанності формули  $f_1 \wedge f_2 \wedge \bar{g}$ , записаної згідно з принципом прямої дедукції. Перейменуємо змінні у формулах, оскільки однакові змінні можуть мати різний зміст. У результаті отримаємо формулу, невиконанність якої доведемо методом резолюцій:

$$\exists y(C(y) \wedge Q(y)) \wedge \forall x(C(x) \rightarrow (W(x) \wedge R(x))) \wedge (\neg(\exists z(Q(z) \wedge R(z)))).$$

Зведемо формулу до клаузальної форми.

1)  $\exists y(C(y) \wedge Q(y)) \wedge \forall x(C(x) \rightarrow (W(x) \wedge R(x))) \wedge (\neg(\exists z(Q(z) \wedge R(z))))$  – задана формула;

2)  $\exists y(C(y) \wedge Q(y)) \wedge \forall x(\bar{C}(x) \vee (W(x) \wedge R(x))) \wedge (\neg(\exists z(Q(z) \wedge R(z))))$  – вилучено імплікацію;

3)  $\exists y(C(y) \wedge Q(y)) \wedge \forall x(\bar{C}(x) \vee (W(x) \wedge R(x))) \wedge (\forall z(\neg(Q(z) \wedge R(z))))$  – закон 2 логіки першого ступеня (див. підрозділ 2.6);

4)  $\exists y(C(y) \wedge Q(y)) \wedge \forall x(\bar{C}(x) \vee (W(x) \wedge R(x))) \wedge (\forall z(\bar{Q}(z) \vee \bar{R}(z)))$  – закон де Моргана;

5)  $\exists y \forall x \forall z((C(y) \wedge Q(y)) \wedge (\bar{C}(x) \vee (W(x) \wedge R(x))) \wedge (\bar{Q}(z) \vee \bar{R}(z)))$  – випереджена форма;

6)  $\forall x \forall z((C(a) \wedge Q(a)) \wedge (\bar{C}(x) \vee (W(x) \wedge R(x))) \wedge (\bar{Q}(z) \vee \bar{R}(z)))$  – сколемівська форма;

7)  $\forall x \forall z(C(a) \wedge Q(a) \wedge (\bar{C}(x) \vee W(x)) \wedge (\bar{C}(x) \vee R(x)) \wedge (\bar{Q}(z) \vee \bar{R}(z)))$  – клаузальна форма.

Вилучимо з останньої формули квантори загальності та перепишемо її у вигляді множини диз'юнктивів:

- (1)  $\bar{C}(x) \vee W(x)$ ;
- (2)  $\bar{C}(x) \vee R(x)$ ;
- (3)  $\bar{Q}(z) \vee \bar{R}(z)$ ;
- (4)  $C(a)$ ;
- (5)  $Q(a)$ .

Методом резолюцій доведемо невиконанність цієї множини диз'юнктивів у разі уніфікації її з допомогою відповідних підстановок:

- (6)  $R(a)$  (2), (4),  $\{a/x\}$ ;
- (7)  $\bar{R}(a)$  (3), (5),  $\{a/z\}$ ;
- (8)  $\square$  (6), (7).

Отже, доведено, що  $g$  є логічним наслідком  $f_1$  та  $f_2$ .

**Приклад 2.91.** Задано гіпотези „Деякі студенти поважають усіх викладачів”, „Жоден студент не поважає неуків”. Покажемо, що речення „Жоден із викладачів не є неуком” логічно випливає з наведених гіпотез.



Уведемо предикати:

$C(x)$ : „ $x$  – студент”;

$P(x)$ : „ $x$  – викладач”;

$H(x)$ : „ $x$  – неук”;

$L(x, y)$ : „ $x$  поважає  $y$ ”.

З допомогою введених предикатів запишемо гіпотези  $f_1, f_2$  та висновок  $g$ :

$f_1$ :  $\exists x(C(x) \wedge \forall y(P(y) \rightarrow L(x, y)))$ ;

$f_2$ :  $\forall z(C(z) \rightarrow \forall v(H(v) \rightarrow \bar{L}(z, v)))$ ;

$g$ :  $\forall w(P(w) \rightarrow \bar{H}(w))$ .

Запишемо цю задачу згідно з принципом прямої дедукції як задачу перевірки невиконаності формули  $f_1 \wedge f_2 \wedge \bar{g}$ . Побудуємо її клаузальну форму:

1)  $\exists x(C(x) \wedge \forall y(P(y) \rightarrow L(x, y))) \wedge \forall z(C(z) \rightarrow \forall v(H(v) \rightarrow \bar{L}(z, v))) \wedge (\neg(\forall w(P(w) \rightarrow \bar{H}(w))))$ ;

2)  $\forall y(C(a) \wedge (\bar{P}(y) \vee L(a, y))) \wedge \forall z \forall v(\bar{C}(z) \vee \bar{H}(v) \vee \bar{L}(z, v)) \wedge \exists w(P(w) \wedge H(w))$ ;

3)  $\exists w \forall y \forall z \forall v(P(w) \wedge H(w) \wedge C(a) \wedge (\bar{P}(y) \vee L(a, y)) \wedge (\bar{C}(z) \vee \bar{H}(v) \vee \bar{L}(z, v)))$ ;

4)  $\forall y \forall z \forall v(P(b) \wedge H(b) \wedge C(a) \wedge (\bar{P}(y) \vee L(a, y)) \wedge (\bar{C}(z) \vee \bar{H}(v) \vee \bar{L}(z, v)))$ .

Вилучимо квантори загальності та перепишемо останню формулу як множину диз'юнктив:

(1)  $\bar{P}(y) \vee L(a, y)$ ;

(2)  $\bar{C}(z) \vee \bar{H}(v) \vee \bar{L}(z, v)$ ;

(3)  $C(a)$ ;

(4)  $P(b)$ ;

(5)  $H(b)$ .

Для доведення невиконаності застосуємо метод резолюцій у поєднанні з уніфікацією:

(6)  $\bar{H}(v) \vee \bar{L}(a, v)$  (2), (3),  $\{a/z\}$ ;

(7)  $L(a, b)$  (1), (4),  $\{b/y\}$ ;

(8)  $\bar{H}(b)$  (6), (7),  $\{b/v\}$ ;

(9)  $\square$  (5), (8).

Отже, доведено, що  $g$  є логічним наслідком  $f_1$  та  $f_2$ .

**Приклад 2.92.** Доведемо рівність внутрішніх кутів  $abd$  та  $bdc$ , утворених діагоналлю  $db$  трапеції  $abcd$ , наведеної на рис. 2.5.

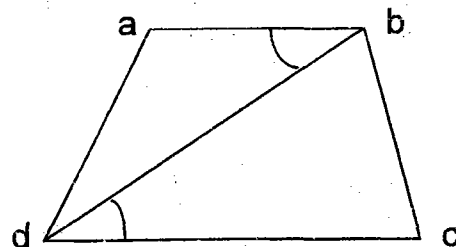


Рис. 2.5

Позначимо через  $T(x, y, u, v)$  довільну трапецію  $xuyv$ , зображену на рис. 2.6. Предикат  $P(x, y, u, v)$  означає, що основа  $xu$  трапеції  $T(x, y, u, v)$  паралельна до основи  $uv$ , а предикат  $E(x, y, z, u, v, w)$  – рівність довільних кутів  $xuz$  та  $uvw$ .

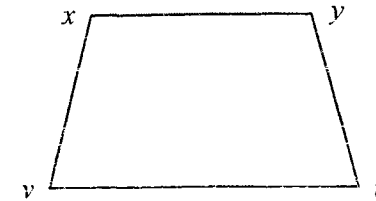


Рис. 2.6

Уведемо позначення:

$f_1$ :  $\forall x \forall y \forall u \forall v(T(x, y, u, v) \rightarrow P(x, y, u, v))$  – означення трапеції;

$f_2$ :  $\forall x \forall y \forall u \forall v(P(x, y, u, v) \rightarrow E(x, y, v, u, v, y))$  – внутрішні кути при паралельних лініях рівні;

$f_3$ :  $T(a, d, c, d)$  – задана трапеція  $abcd$ ;

$g$ :  $E(a, b, d, c, d, b)$  – рівність кутів  $abd$  та  $cdb$ .

Тут  $f_1, f_2, f_3$  – гіпотези, а  $g$  – висновок. Тепер твердження про рівність кутів трапеції, яке треба довести, згідно з принципом прямої дедукції можна записати формулою

$$f_1 \wedge f_2 \wedge f_3 \wedge \bar{g},$$

або

$$\forall x \forall y \forall u \forall v((T(x, y, u, v) \rightarrow P(x, y, u, v)) \wedge \forall x \forall y \forall u \forall v(P(x, y, u, v) \rightarrow E(x, y, v, u, v, y)) \wedge T(a, d, c, d) \wedge \bar{E}(a, b, d, c, d, b)).$$

Виконаємо перетворення останньої формули та отримаємо її клаузальну форму:

$$\forall x \forall y \forall u \forall v(((\bar{T}(x, y, u, v) \vee P(x, y, u, v)) \wedge ((\bar{P}(x, y, u, v) \vee E(x, y, v, u, v, y)) \wedge T(a, d, c, d) \wedge \bar{E}(a, b, d, c, d, b))).$$

Останню формулу перепишемо як множину диз'юнктив

$$S = \{\bar{T}(x, y, u, v) \vee P(x, y, u, v), \bar{P}(x, y, u, v) \vee E(x, y, v, u, v, y), T(a, b, c, d), \bar{E}(a, b, d, c, d, b)\}.$$

Спростуємо множину  $S$  за методом резолюцій. Для цього випишемо диз'юнкти:

(1)  $\bar{T}(x, y, u, v) \vee P(x, y, u, v)$ ;

(2)  $\bar{P}(x, y, u, v) \vee E(x, y, v, u, v, y)$ ;

(3)  $T(a, d, c, d)$ ;

(4)  $\bar{E}(a, b, d, c, d, b)$ .

У наведеному нижче процесі побудови послідовності резольвент застосовано підстановку  $\{a/x, b/y, c/u, d/v\}$ .

(5)  $\bar{P}(a, b, c, d)$  (2), (4);

(6)  $\bar{T}(a, b, c, d)$  (1), (5);

(7)  $\square$  (3), (6).

Оскільки отримано порожній диз'юнкт, то множину  $S$  спростовано.

**Приклад 2.93.** Сформулюємо гіпотези: „Той, хто вміє обчислювати інтеграли – математик”; „Діти – не математики”; „Деякі діти мають математичні здібності”. Покажемо, що речення „Дехто з тих, хто має математичні здібності, не обчислює інтегралів” є логічним наслідком сформульованих гіпотез.

Уведемо предикати:

$N(x)$ : „ $x$  вміє обчислювати інтеграли”;

$M(x)$ : „ $x$  – математик”;

$D(x)$ : „ $x$  – дитина”;

$C(x)$ : „ $x$  має математичні здібності”.

Запишемо гіпотези та висновок у вигляді таких формул:

$f_1: \exists z(D(z) \wedge C(z))$ ;

$f_2: \forall x(N(x) \rightarrow M(x))$ ;

$f_3: \forall y(D(y) \rightarrow \bar{M}(y))$ ;

$g: \exists u(C(u) \wedge \bar{N}(u))$ .

Доведемо, що формула  $g$  є логічним наслідком формул  $f_1 - f_3$ . Згідно з принципом прямої дедукції побудуємо формулу  $f_1 \wedge f_2 \wedge f_3 \wedge \bar{g}$ , невиконаність якої доведемо методом резолюцій. Отже, треба довести, що формула

$\exists z(D(z) \wedge C(z)) \wedge \forall x(N(x) \rightarrow M(x)) \wedge \forall y(D(y) \rightarrow \bar{M}(y)) \wedge (\neg(\exists u(C(u) \wedge \bar{N}(u))))$

невиконання.

Побудуємо клаузальну форму цієї формули:

1)  $\exists z(D(z) \wedge C(z)) \wedge \forall x(\bar{N}(x) \vee M(x)) \wedge \forall y(\bar{D}(y) \vee \bar{M}(y)) \wedge (\neg(\exists u(C(u) \wedge \bar{N}(u))))$ ;

2)  $\exists z \forall x \forall y \forall u((D(z) \wedge C(z)) \wedge (\bar{N}(x) \vee M(x)) \wedge (\bar{D}(y) \vee \bar{M}(y)) \wedge (\neg(C(u) \wedge \bar{N}(u))))$ ;

3)  $\forall x \forall y \forall u(D(x) \wedge C(x) \wedge (\bar{N}(x) \vee M(x)) \wedge (\bar{D}(y) \vee \bar{M}(y)) \wedge (\bar{C}(u) \vee N(u)))$ .

Тепер вилучимо квантори загальності та випишемо множину диз'юнктив:

(1)  $\bar{N}(x) \vee M(x)$ ;

(2)  $\bar{D}(y) \vee \bar{M}(y)$ ;

(3)  $\bar{C}(u) \vee N(u)$ ;

(4)  $D(a)$ ;

(5)  $C(a)$ .

Застосуємо правило резолюцій у поєднанні з уніфікацією:

(6)  $N(a)$  (3), (5),  $\{a/u\}$ ;

(7)  $M(a)$  (1), (6),  $\{a/x\}$ ;

(8)  $\bar{D}(a)$  (2), (7),  $\{a/y\}$ ;

(9)  $\square$  (4), (8).

Наступний приклад ілюструє задачі другого типу.

**Приклад 2.94.** Нехай задана предметна область задачі  $D = \{\text{Іван, Степан, Петро, Марія}\}$  та факти про певну родину „Іван батько Степана”, „Іван батько Петра” та „Петро батько Марії”. Отримаємо відповідь на питання „Чи має Степан брата й, якщо так, то як його звати?”.

Уведемо такі предикати:

$F(x, y)$ : „ $x$  батько  $y$ ”;

$M(x)$ : „ $x$  – чоловік”;

$S(x, y)$ : „ $x$  брат або сестра  $y$ ”;

$B(x, y)$ : „ $x$  брат  $y$ ”.

Факти про цю родину запишемо з допомогою введеного поняття „батько”:

$F(\text{Іван}, \text{Степан})$ ;

$F(\text{Іван}, \text{Петро})$ ;

$F(\text{Петро}, \text{Марія})$ .

Окрім того, опишемо відношення, які існують між членами родини.

1. Кожний батько є чоловіком:  $\forall x \forall y((F(x, y) \rightarrow M(x)))$ .

2. Діти, які мають спільного батька, один одному є або братом або сестрою:  
 $\forall z \forall t \forall v((F(z, t) \wedge F(z, v)) \rightarrow S(t, v))$ .

3. Дитина чоловічої статі певного батька іншій дитині цього ж батька є братом:  
 $\forall q \forall u((S(q, u) \wedge M(q)) \rightarrow B(q, u))$ .

Питання, на яке потрібно отримати відповідь, має вигляд  $\exists w B(w, \text{Степан})$ .

Запишемо сформульовану задачу згідно з принципом прямої дедукції та підготуємо її до застосування методу резолюцій:

$\forall x \forall y((F(x, y) \rightarrow M(x)) \wedge \forall z \forall t \forall v((F(z, t) \wedge F(z, v)) \rightarrow S(t, v)) \wedge \forall q \forall u((S(q, u) \wedge M(q)) \rightarrow B(q, u)) \wedge F(\text{Іван}, \text{Степан}) \wedge F(\text{Іван}, \text{Петро}) \wedge F(\text{Петро}, \text{Марія}) \wedge (\neg(\exists w B(w, \text{Степан}))))$ .

З останньої формули отримуємо таку множину диз'юнктив:

(1)  $\bar{F}(x, y) \vee M(x)$ ;

(2)  $\bar{F}(z, t) \vee \bar{F}(z, v) \vee S(t, v)$ ;

(3)  $\bar{S}(q, u) \vee \bar{M}(q) \vee B(q, u)$ ;

(4)  $F(\text{Іван}, \text{Степан})$ ;

(5)  $F(\text{Іван}, \text{Петро})$ ;

(6)  $F(\text{Петро}, \text{Марія})$ ;

(7)  $\bar{B}(w, \text{Степан})$ .

У кожному з рядків, які записані нижче, зазначено номери диз'юнктив, які використані для побудови резольвент, і підстановки, які необхідно виконати.

(8)  $\bar{F}(\text{Іван}, v) \vee S(\text{Петро}, v)$  (2), (5),  $\{\text{Іван}/x, \text{Петро}/y, \text{Іван}/z, \text{Петро}/t\}$ ,

(9)  $S(\text{Петро}, \text{Степан})$  (4), (8),  $\{\text{Степан}/v\}$ ,

(10)  $M(\text{Петро})$  (6), (1),  $\{\text{Петро}/x, \text{Марія}/y\}$ ,

(11)  $\bar{S}(\text{Петро}, u) \vee B(\text{Петро}, u)$  (10), (3),  $\{\text{Петро}/q\}$ ,

(12)  $B(\text{Петро}, \text{Степан})$  (9), (11),  $\{\text{Степан}/u\}$ ,

(13)  $\square$  (12), (7),  $\{\text{Петро}/w\}$ .

Остання підстановка надає змінній  $w$  значення „Петро”, що являє собою відповідь на сформульоване вище питання: брат Степана – Петро.

## 2.19. ХОРНІВСЬКІ ДИЗ'ЮНКТИ ТА МЕТОД SLD-РЕЗОЛЮЦІЙ

У деяких часткових випадках ефективність використання методу резолюцій можна підвищити. Розглянемо один із таких випадків, який ґрунтується на використанні диз'юнктив спеціального вигляду.

Диз'юнкт називають *хорнівським диз'юнктом*, якщо він містить не більше одного позитивного літерала. У літературі зі штучного інтелекту ще використовують термін *хорнівське речення*. Хорнівський диз'юнкт називають *точним*, якщо він містить позитивний літерал, інакше його називають *цільовим*.

**Приклад 2.95.**  $\bar{p} \vee \bar{q} \vee \bar{r}$  – цільовий хорнівський диз'юнкт,  $\bar{p} \vee \bar{q} \vee \bar{r} \vee s$  – точний хорнівський диз'юнкт.

**Приклад 2.96.** Задано множину диз'юнктів

$$S = \{ p \vee \bar{r} \vee \bar{t}, q, r, t \vee \bar{p} \vee \bar{r}, t \vee \bar{q}, \bar{p} \vee \bar{q} \vee \bar{r} \}.$$

Усі диз'юнкти в цій множині – хорнівські. Покажемо, що ця множина невиконання, тобто застосування методу резолюцій породжує порожній диз'юнкт. Спростування множини хорнівських диз'юнктів методом резолюцій наведено нижче.

- (1)  $p \vee \bar{r} \vee \bar{t}$ ;
- (2)  $q$ ;
- (3)  $r$ ;
- (4)  $t \vee \bar{p} \vee \bar{r}$ ;
- (5)  $t \vee \bar{q}$ ;
- (6)  $\bar{p} \vee \bar{q} \vee \bar{r}$ ;
- (7)  $t$  (2), (5);
- (8)  $p \vee \bar{t}$  (1), (3);
- (9)  $\bar{p} \vee \bar{r}$  (2), (6);
- (10)  $\bar{p}$  (3), (9);
- (11)  $p$  (7), (8);
- (12)  $\square$  (10), (11).

Можна побачити, що пара хорнівських диз'юнктів, один з яких цільовий, а другий – точний, утворює резольвенту, яка є теж цільовим хорнівським диз'юнктом.

Сформулюємо алгоритм методу резолюцій для випадку хорнівських диз'юнктів (алгоритм *SLD-резолюцій*).

Нехай множина  $S = \{d_1, d_2, \dots, d_m, g_1, g_2, \dots, g_l\}$  містить точні  $d_1, d_2, \dots, d_m$  та цільові  $g_1, g_2, \dots, g_l$  хорнівські диз'юнкти. Алгоритм *SLD-резолюцій* для хорнівських диз'юнктів полягає в розбитті множини всіх хорнівських диз'юнктів  $S$  на дві підмножини, одна з яких містить лише цільові, а друга – лише точні хорнівські диз'юнкти. Резольвента, отримана вилученням позитивного літерала з точного хорнівського диз'юнкта, – це цільовий диз'юнкт, який додають до множини цільових диз'юнктів.

Якщо точний хорнівський диз'юнкт  $d_i$  має ранг 1 та є літералом  $p$ , а хорнівський диз'юнкт  $d_j$  містить літерал  $\bar{p}$ , то резольвента хорнівських диз'юнктів  $d_i$  та  $d_j$  – це хорнівський диз'юнкт  $d_j$  з видаленням із нього літералом  $\bar{p}$ . Якщо множина  $S$  невиконання, то її завжди можна спростувати за будь-якої стратегії вибору хорнівських диз'юнктів для побудови резольвент.

Для спрощення запису хорнівські диз'юнкти доцільно зображати впорядкованими

множинами літералів, а порожній хорнівський диз'юнкт – знаком порожньої множини.

### Алгоритм *SLD-резолюції*

Задано множину хорнівських диз'юнктів  $S$ . Потрібно спростувати цю множину, або визначити, що її спростувати неможливо.

Крок 1. Розбити множину  $S$  на підмножини  $S_1 = \{d_1, d_2, \dots, d_m\}$  точних і  $S_2 = \{g_1, g_2, \dots, g_l\}$  цільових диз'юнктів. Однаково впорядкувати елементи базису  $P = \{p_1, p_2, \dots, p_n\}$  та літерали в диз'юнктах. Помістити всі цільові диз'юнкти до стека.

Крок 2. Покласти  $k = 1$ . Якщо стек порожній, то – кінець: множину  $S$  неможна спростувати. Інакше вибрати цільовий хорнівський диз'юнкт  $g$  із верхівки стека.

Крок 3. Вибрати елемент  $p_k$  із базису, а із множини  $S_1$  – точний хорнівський диз'юнкт  $d_i$ , який містить позитивний літерал  $p_k$ .

Крок 4. Побудувати резольвенту  $r = g * d_i$ . Якщо резольвента  $r$  є порожньою множиною, то множина  $S$  спростована. Кінець. Якщо виконано критерій закінчення алгоритму, то спростувати множину  $S$  не вдалось. Кінець.

Крок 5. Якщо резольвенту  $r$  одержано вперше, то впорядкувати її елементи та додати до стека.

Крок 6. Повторити кроки 4-5 для кожного точного хорнівського диз'юнкта з  $S_1$ , який містить позитивний літерал  $p_k$ .

Крок 7. Виконати  $k := k + 1$ . Якщо  $k \leq n$ , то повернутись до кроку 3, інакше перейти до кроку 2.

Важливо врахувати, що в разі реалізації алгоритму *SLD-резолюцій* усі літерали в хорнівських диз'юнктах і резольвентах упорядковують так само, як і елементи базису  $P = \{p_1, p_2, \dots, p_n\}$ . Оскільки наперед невідомо, чи можна спростувати задану множину хорнівських диз'юнктів, потрібно виробити критерій закінчення алгоритму. За такий критерій можна взяти визначену наперед кількість резольвент, після обчислення яких вважати, що спростувати множину хорнівських диз'юнктів неможливо. Іншим критерієм можна вважати неможливість побудови резольвенти, відмінної від уже побудованих та від елементів множини  $S_2$ .

**Приклад 2.97.** Розглянемо множину хорнівських диз'юнктів із прикладу 2.96:

$$S = \{p \vee \bar{r} \vee \bar{t}, q, r, t \vee \bar{p} \vee \bar{r}, t \vee \bar{q}, \vee \bar{q} \vee \bar{r}\}.$$

Послідовність кроків спрощування множини  $S$  методом *SLD-резолюцій* наведено нижче. Усі резольвенти, які записано в рядках 7-10, – це цільові хорнівські диз'юнкти. Їх одержано з двох хорнівських диз'юнктів, один із яких завжди цільовий.

- (1)  $p \vee \bar{r} \vee \bar{t}$ ;
- (2)  $q$ ;
- (3)  $r$ ;
- (4)  $t \vee \bar{p} \vee \bar{r}$ ;
- (5)  $t \vee \bar{q}$ ;
- (6)  $\bar{p} \vee \bar{q} \vee \bar{r}$ ;
- (7)  $\bar{r} \vee \bar{t} \vee \bar{q}$  (1), (6);
- (8)  $\bar{t} \vee \bar{q}$  (3), (7);

- (9)  $\bar{q}$  (5), (8);  
 (10)  $\square$  (2), (9).

Застосуємо запропонований алгоритм для розв'язування такого прикладу.

**Приклад 2.98.** Спростуємо множину хорнівських диз'юнктивів  $S = \{\{q\}, \{p, \bar{q}, \bar{s}\}, \{\bar{p}, q, \bar{r}\}, \{\bar{q}, \bar{r}, s\}, \{\bar{q}, r\}, \{\bar{p}, \bar{q}\}\}$  за алгоритмом SLD-резолюцій. Усі диз'юнкти записані множинами літералів. Упорядкуємо базис цього прикладу  $P = \{p, q, r, s\}$  та літерали в хорнівських диз'юнктах. Розбиттям множини  $S$  отримаємо множину точних хорнівських диз'юнктивів

$$S_1 = \{\{q\}, \{p, \bar{q}, \bar{s}\}, \{\bar{p}, q, \bar{r}\}, \{\bar{q}, \bar{r}, s\}, \{\bar{q}, r\}\}$$

та множину цільових хорнівських диз'юнктивів  $S_2 = \{\{\bar{p}, \bar{q}\}\}$ .

Уведемо позначення  $d_1 = \{q\}$ ,  $d_2 = \{p, \bar{q}, \bar{s}\}$ ,  $d_3 = \{\bar{p}, q, \bar{r}\}$ ,  $d_4 = \{\bar{q}, \bar{r}, s\}$ ,  $d_5 = \{\bar{q}, r\}$ ,  $g_1 = \{\bar{p}, \bar{q}\}$ . Нижче наведено перебіг алгоритму з урахуванням цих позначень.

$$\begin{aligned} g_2 &= g_1 * d_2 = \{\bar{q}, \bar{s}\}, & g_6 &= g_2 * d_3 = \{\bar{p}, \bar{r}, \bar{s}\}, & g_{10} &= g_7 * d_1 = \{\bar{r}\}, \\ g_3 &= g_1 * d_1 = \{\bar{p}\}, & g_7 &= g_2 * d_2 = \{\bar{q}, \bar{r}\}, & g_{11} &= g_7 * d_5 = \{\bar{q}\}, \\ g_4 &= g_1 * d_3 = \{\bar{p}, \bar{r}\}, & g_8 &= g_4 * d_2 = \{\bar{q}, \bar{r}, \bar{s}\}, & g_{12} &= g_8 * d_1 = \{\bar{r}, \bar{s}\}, \\ g_5 &= g_2 * d_1 = \{\bar{s}\}, & g_9 &= g_6 * d_5 = \{\bar{p}, \bar{q}, \bar{r}\}, & g_{13} &= g_{11} * d_1 = \emptyset. \end{aligned}$$

Одержання порожньої множини засвідчує успішне спростування множини  $S$ .

## 2.20. ПРИНЦИП ЛОГІЧНОГО ПРОГРАМУВАННЯ

Точний хорнівський диз'юнкт  $\bar{p}_1 \vee \bar{p}_2 \vee \dots \vee \bar{p}_n \vee g$  можна перетворити до вигляду

$$(\neg(\bar{p}_1 \vee \bar{p}_2 \vee \dots \vee \bar{p}_n) \rightarrow g) = (p_1 \wedge p_2 \wedge \dots \wedge p_n) \rightarrow g.$$

Цей запис має зміст „якщо  $A$ , то  $B$ ”, де  $A$  – кон'юнкція позитивних літералів  $p, p_1 \wedge \dots \wedge p_n$ , а  $B = g$ . У такому вигляді хорнівський диз'юнкт називають *правилом*, де кожний позитивний літерал  $p_i$  ( $i=1, 2, \dots, n$ ) називають *гіпотезою*, а  $g$  – *висновком*. Прийнято інший спосіб запису такого правила у вигляді  $(B: -A)$ , у якому  $B$  називають *заголовком*, а  $A$  – *тілом* правила. Хорнівський диз'юнкт  $\bar{p}_1 \vee \bar{p}_2 \vee \dots \vee \bar{p}_n \vee g$  записують у вигляді правила  $(g: -p_1, p_1, \dots, p_n)$ .

**Приклад 2.99.** Запишемо точний хорнівський диз'юнкт  $\bar{p} \vee \bar{q} \vee \bar{r} \vee s$  еквівалентними формулами

$$\bar{p} \vee \bar{q} \vee \bar{r} \vee s = (\neg(\bar{p} \vee \bar{q} \vee \bar{r})) \rightarrow s = (p \wedge q \wedge r) \rightarrow s.$$

У правилі  $(p \wedge q \wedge r) \rightarrow s$  гіпотезами є літерали  $p, q$  та  $r$ , а висновком –  $s$ . Тому  $s$  є заголовком, а  $p \wedge q \wedge r$  – тілом правила  $(s: -p, q, r)$ .

Точні хорнівські диз'юнкти записують у вигляді правил, що містять заголовок та тіло. Правило відповідає формулі логіки висловлювань у кон'юнктивній нормальній формі або формулі логіки першого ступеня, записаній у клаузальній формі. Цільовий хорнівський диз'юнкт записують правилом, у якого є тіло й немає заголовку. Розглянемо приклад такого правила, яке запишемо як імплікацію вигляду

$$\bar{p} \vee \bar{q} \vee \bar{r} = \bar{p} \vee \bar{q} \vee \bar{r} \vee F = (\neg(\bar{p} \vee \bar{q} \vee \bar{r})) \rightarrow F = (p \wedge q \wedge r) \rightarrow F.$$

Тут заголовком правила є логічна стала  $F$ , замість якої будемо писати знак „?”. Таке правило називають *питанням*. Літерали в тілі питання називають *цільми*.

Точний хорнівський диз'юнкт із рангом 1, який складається з одного позитивного літерала, зображають правилом, яке не має тіла, а його заголовком є цей літерал. Таке правило називають *фактом*, і воно є істинним висловлюванням.

Множину правил (правила, факти та питання) називають *логічною програмою*, а її виконанням – спростування множини відповідних хорнівських диз'юнктивів. Множину правил логічної програми називають *базою знань*, множину фактів – *базою даних* програми, а тіло питання – *цілью програми*. У логічних програмах прийнято писати з малої букви всі імена, крім імен змінних.

**Приклад 2.100.** Розглянемо множину диз'юнкцій

$$\{\bar{r} \vee \bar{t} \vee p, q, r, \bar{p} \vee \bar{r} \vee t, \bar{q} \vee t, \bar{p} \vee \bar{q}\}.$$

У заданій множині є три точних хорнівських диз'юнкти, один негативний та два однолітеральні. Відповідно, у програмі є три правила, одне питання та два факти. Перепишемо диз'юнкти заданої множини у вигляді правил відповідної логічної програми.

Правила:

- (1)  $p: -r, t.$
- (2)  $t: -p, r.$
- (3)  $t: -q.$

Факти:

- (4)  $q: -.$
- (5)  $r: -.$
- Питання:
- (6)  $? - p, q.$

Спростування множини хорнівських диз'юнктив проводять за методом SLD-резолюції. Покажемо отримання резольвенти двох хорнівських диз'юнктив у разі запису їх у формі правил логічної програми. Розглянемо правила:

- (1)  $a: -b, c.$
- (2)  $d: -a, f.$

Ці правила є записом хорнівських диз'юнктивів

$$(b \wedge c) \rightarrow a = (\neg(b \wedge c)) \vee a = \bar{b} \vee \bar{c} \vee a \text{ та}$$

$$(a \wedge f) \rightarrow d = (\neg(a \wedge f)) \vee d = \bar{a} \vee \bar{f} \vee d.$$

Оскільки диз'юнкти містять пару контрарних літералів  $\{a, \bar{a}\}$ , то їхня резольвента має вигляд  $\bar{b} \vee \bar{c} \vee \bar{f} \vee d = (\neg(b \wedge c \wedge f)) \vee d = (b \wedge c \wedge f) \rightarrow d$ . Перепишемо її логічним правилом

- (3)  $d: -b, c, f.$

Зазначимо, що резольвенту (3) отримано із правил (1) та (2) так: у тілі правила (2) узятो літерал  $a$ , який збігається із заголовком правила (1). Резольвенту (3) отримують із правила (2) заміною літерала  $a$  у тілі правила (2) на тіло правила (1).

Такий спосіб побудови резольвенти складає *принцип логічного програмування*. Логічну програму, яку виконують за цим принципом, називають *декларативною*, на відміну від традиційної програми, яку називають *процедурною*. Логічні програми

реалізують на комп'ютерах із використанням спеціальної мови логічного програмування *Пролог*, для якої створено велику кількість трансляторів.

Спростування множини хорнівських диз'юнктив за методом *SLD*-резолюцій виконують побудовою резольвент для пар хорнівських диз'юнктив, один із яких точний, а другий – цільовий. Такі диз'юнкти повинні мати контрарну пару літералів. Резольвенту отримують об'єднанням множин літералів цих диз'юнктив та виключенням з отриманої множини контрарної пари літералів. Іншими словами, до множини літералів цільового диз'юнкту додають усі літерали точного, а потім викреслюють контрарну пару літералів. Отриманий диз'юнкт теж цільовий.

Спростування множини диз'юнктив виконанням відповідної логічної програми починають з цілі програми, цілі якої замінюють тілами правил програми. Процес спростування вважають завершеним успішно, якщо отримана порожня ціль програми, яка є порожньою множиною цілей.

Спростування множини хорнівських диз'юнктив, записаної у вигляді логічної програми, виконують за алгоритмом, який називають *інтерпретатором логічної програми*. Якщо літерали в правилах є атомами логіки першого ступеня, то під час побудови резольвент виконують їхню уніфікацію.

### Алгоритм інтерпретатора логічної програми

Задано логічну програму  $P$  та ціль програми  $g$ . У результаті виконання програми  $P$  отримують відповідь „так” або відповідь „ні” та приклад цілі  $g\theta$ .

Крок 1. Вибрати в  $g$  ціль  $h$  та таке правило  $H: -b_1, b_2, \dots, b_n$  у програмі, щоб  $h$  та  $h'$  уніфікувалися з найзагальнішим уніфікатором  $\theta$ . Якщо такого правила немає, то відповідь „ні”. Кінець.

Крок 2. Вилучити ціль  $h$  із  $g$  та замінити її рядком  $b_1, b_2, \dots, b_n$ . Уніфікувати за допомогою уніфікатора  $\theta$  ціль програми  $g$ .

Крок 3. Якщо ціль програми не порожня, то повернутись до кроку 1. Інакше, відповідь „так” та результат виконання алгоритму  $g\theta$ . Кінець.

Виконання логічної програми здійснюють у такій послідовності. Для кожної цілі шукають у базах даних та знань правило, заголовок якого можна уніфікувати із цією ціллю. Правила переглядають зверху вниз у порядку запису, а цілі в тілі програми – зліва направо. Прикладом тіла знайденого правила замінюють ціль у тілі питання.

**Приклад 2.101.** Задано множину диз'юнктив

$$\{p \vee \bar{r}, q \vee \bar{t}, q \vee \bar{p} \vee \bar{r}, r \vee \bar{p}, t \vee \bar{s}, \bar{q}\}.$$

Спростування множини диз'юнктив виконаємо за методом *SLD*-резолюції:

$$(1) p \vee \bar{r};$$

$$(2) q \vee \bar{t};$$

$$(3) q \vee \bar{p} \vee \bar{r};$$

$$(4) r;$$

$$(5) r \vee \bar{p};$$

$$(6) t \vee \bar{s};$$

$$(7) \bar{q}.$$

Випишемо множину резольвент:

$$(8) \bar{t} \quad (2), (7);$$

$$(9) \bar{s} \quad (6), (8);$$

$$(10) \bar{p} \vee \bar{r}, (3) (7);$$

$$(11) \bar{r} \quad (1), (10);$$

$$(12) \square \quad (4), (11).$$

Кожна з побудованих резольвент – цільовий диз'юнкт, який використано для побудови нового цільового диз'юнкту.

Побудуємо відповідну логічну програму та виконаємо її, використовуючи інтерпретатор логічної програми. Усі літерали є атомами логіки висловлювань і уніфікація не потрібна. Ця множина містить п'ять точних, один точний однолітеральний та один цільовий диз'юнкт. Цільовий диз'юнкт містить лише один літерал.

Перепишемо множину диз'юнктив у формі логічної програми та виконаємо її.

$$(1) \bar{p}: -r.$$

$$(2) q: -t.$$

$$(3) q: -p, r.$$

$$(4) r.$$

$$(5) r: -p.$$

$$(6) t: -s.$$

$$(7) ? - q.$$

Розглянемо послідовність кроків виконання логічної програми.

- ◆ Починаємо з цілі  $q$  у цілі програми (тілі питання) та знаходимо правило із заголовком  $q$ . Таким правилом є правило (2). Замінімо тілом цього правила ціль  $q$ . Тілом питання буде  $t$ .
- ◆ Тепер у тілі питання є одна ціль  $t$ , з якою збігається заголовок правила (6). Тілом правила (6) заміняємо ціль  $t$  та отримуємо  $s$  – нову ціль програми.
- ◆ Оскільки правил із заголовком  $s$  немає, то виконання продовжувати неможливо. Це означає, що необхідно повернутись до питання з ціллю  $t$ .
- ◆ Оскільки іншого правила із заголовком  $t$  у програмі немає, то повертаємось до питання з тілом  $q$  та знаходимо інше правило із заголовком  $q$ . Цим правилом є правило (3), а його тілом –  $(p, r)$ . Тепер ціллю програми буде рядок  $p, r$ .
- ◆ Вибираємо ціль  $p$  та знаходимо правило (1) із заголовком  $p$ . Його тіло – буква  $r$ . Тепер ціллю програми буде рядок  $r, r$ .
- ◆ Вибираємо першу букву  $r$  та знаходимо правило із заголовком  $r$ . Таким правилом є факт (4), який має порожнє тіло. Цим тілом замінимо букву  $r$ . Аналогічно вчинимо з другою буквою  $r$ . Тепер питання має у тілі порожній рядок. Це означає, що одержано відповідь „так”, програму виконано, а відповідну множину спростовано. Наведене спростування полягає у побудові такої послідовності цілей програми:

$$(8) t \quad (7), (2),$$

$$(9) s \quad (8), (6),$$

$$(10) p, r \quad (7), (3),$$

$$(11) r, r \quad (10), (1),$$

$$(12) r \quad (11), (4),$$

$$(13) \emptyset \quad (12), (4).$$

Процес виконання логічної програми наведено на рис. 2.7 за допомогою відповідного *дерева пошуку*. Кожну вершину цього дерева позначено номером застосованого правила та отриманою ціллю програми. Дерево пошуку будують від кореневої вершини обходом зверху вниз.

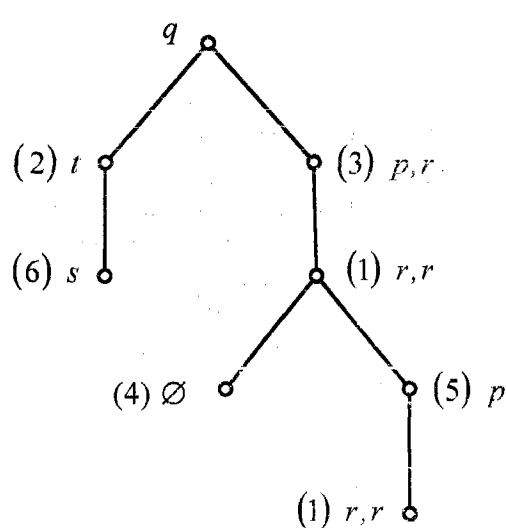


Рис. 2.7

Зазначимо, що піддерево з коренем у вершині, яку позначено як (5)  $p$ , одержимо, коли при виконанні логічної програми спочатку виконаємо правило (5), а після нього — правило (4). Дерево пошуку будують у процесі виконання алгоритму інтерпретатора логічної програми.

На рис. 2.8 зображено дерево пошуку, яке одержано виконанням заданої логічної програми, у якій правила (4) та (5) поміняно місцями. Ці правила позначено відповідно (5') та (4'). Тепер у разі виконання програми буде утворений нескінченний цикл, тобто програму не буде виконано.

Наступний приклад ілюструє застосування алгоритму інтерпретатора логічної програми з уніфікацією цілі програми. Правила програми утворені з формул логіки першого ступеня.

**Приклад 2.102.** Застосуємо інтерпретатор логічної програми для доведення силіогізму про Сократа (див. приклад 2.89). Розв'язування силіогізму зведено до спростування множини хорнівських диз'юнктивів  $S = \{ \bar{H}(x) \vee M(x), H(C), \bar{M}(C) \}$ .

- (1)  $\bar{H}(x) \vee M(x)$ ,
- (2)  $H(C)$ ,
- (3)  $\bar{M}(C)$ .

Доведення полягає у виконанні кроків:

- (4)  $M(C)$  (1), (2),  $\sigma = \{C/x\}$ ,
- (5)  $\square$  (3), (4).

Диз'юнкти множини  $S$  перепишемо у вигляді логічних формул:

- (1')  $H(x) \rightarrow M(C)$ ,
- (2')  $H(C)$ ,

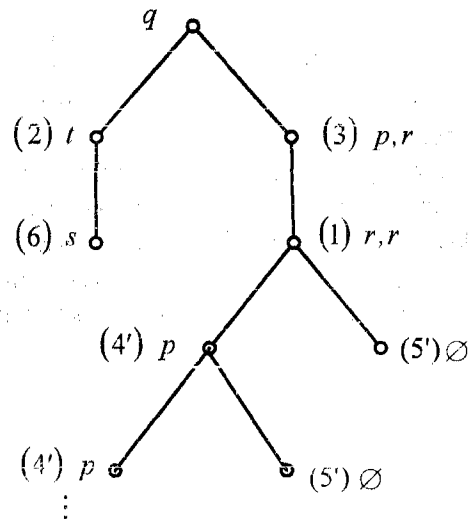


Рис. 2.8

$$(3') M(C) \rightarrow F.$$

З урахуванням вимог до запису змінних та правил (усі імена з малої букви, крім імен змінних), відповідна логічна програма набуває вигляду:

$$(1'') m(c) : - h(X).$$

$$(2'') h(c).$$

$$(3'') ? - m(c).$$

Виконання програми дає відповідь „так” на питання (3''). Якщо питання сформулювати так: « $? - m(X)$ », що означає „Хто смертний?”, то результат виконання програми — значення змінної  $X=c$ , яке отримуємо з підстановки  $\theta = \{c/X\}$ .



## ЗАДАЧІ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

1. Задано формулу  $f = (p \rightarrow q) \rightarrow (p \rightarrow r) \rightarrow (p \rightarrow s)$ . Побудувати кілька її часткових інтерпретацій та обчислити значення заданої формули в цих інтерпретаціях.

2. Побудувати семантичне дерево для базису  $P = \{p, q, r, s\}$ .

3. Для семантичного дерева, зображеного на рис. 2.1, побудувати часткові інтерпретації для вершин  $P, Q, R, S$ .

4. Визначити значення істинності формул  $\forall x \bar{P}(x)$  та  $\exists x P(x)$  в інтерпретації, яка складається із предметної області  $D = \{1, 2\}$  та таких значень одномісного предиката  $P(1) = F, P(2) = T$ .

5. Знайти значення істинності формули  $\exists x \forall y P(x, y)$  в інтерпретації, яка складається із предметної області  $D = \{1, 2\}$  та таких значень двомісного предиката  $P(1, 1) = T, P(1, 2) = T, P(2, 1) = F, P(2, 2) = F$ .

6. Знайти значення істинності формули  $\forall x (P(a) \rightarrow Q(f(a), x))$  в інтерпретації, яка складається із предметної області  $D = \{1, 2\}$ , значення сталої  $a=2$ , значень одномісної функції  $f: f(1)=1, f(2)=2$ , та значень предикатів  $P$  та  $Q$ :  $P(1)=F, P(2)=T, Q(1, 1)=T, Q(1, 2)=F, Q(2, 1)=T, Q(2, 2)=F$ .

7. Знайти значення істинності наведених нижче формул в інтерпретації  $D = \{1, 2\}$ ,  $a=1, b=2, P(1, 1)=T, P(1, 2)=T, P(2, 1)=F, P(2, 2)=F, f(1)=2, f(2)=1$ :

$$a) P(a, f(a)) \wedge P(b, f(b)); \quad б) \forall x \exists y P(y, x); \quad в) \forall x \forall y (P(y, x) \rightarrow P(f(x), f(y))).$$

8. Показати, що формула  $A \rightarrow ((A \rightarrow (B \rightarrow A)) \rightarrow A)$  вивідна у численні висловлювань.

9. Знайти, які правила виведення використано у наведених нижче твердженнях:

а) Якщо піде дощ, то басейн буде зачинено. Пішов дощ, отже, басейн зачинено.

б) Якщо піде сніг, то університет буде зачинено. Університет не зачинено. Отже, сніг не пішов.

в) Якщо я піду плавати, то буду довго на сонці. Якщо я буду довго на сонці, то засмагну. Отже, якщо я піду плавати, то засмагну.

г) Кенгуру живуть в Австралії, мають сильні нижні кінцівки, роблять довгі стрибки і є сумчастими. Отже, кенгуру — сумчасті.

д) На вулиці спека більше 40 градусів або небезпечне забруднення повітря. Сьогодні менше 40 градусів, отже, забруднення повітря небезпечне.

10. Дано гіпотези: „Іван тяжко працює”, „Якщо Іван тяжко працює, то він пасивний



хлопець” та „Якщо Іван пасивний хлопець, то він не знайде кращої роботи”. Використати правила виведення для обґрунтування такого висновку з цих гіпотез: „Іван не знайде кращої роботи”.

11. Дано гіпотези: „Логіка складна або небагато студентів люблять логіку” та „Якщо математика складна, то логіка не складна”. Визначити, чи можна з цих гіпотез отримати висновок:

- Математика не легка, якщо багато студентів люблять логіку.
- Небагато студентів люблять логіку, якщо математика не складна.
- Математика не легка або логіка складна.
- Логіка не складна або математика не легка.
- Якщо небагато студентів люблять логіку, то або математика не легка або логіка не складна.

12. Довести, що у наведених нижче прикладах висновок можна вивести з гіпотез.

- Гіпотези: „Усі леви жорстокі істоти”, „Деякі леви не п'ють кави”. Висновок: „Деякі жорстокі істоти не п'ють кави”.
- Гіпотези: „Усі колібрі мають яскраве пір'я”, „Жодний великий птах не їсть меду і не має яскравого пір'я”. Висновок: „Колібрі – маленькі птахи”.
- Гіпотези: „Кожний атлет сильний”, „Кожний, хто сильний і розумний, доб'ється успіху”, „Петро – атлет”, „Петро – розумний”. Висновок: „Петро доб'ється успіху”.

13. Для кожного з наведених нижче логічних виведень визначити коректність висновку та зробити необхідні пояснення.

- Всі студенти цієї групи розуміють логіку. Дмитро – студент цієї групи. Отже, Дмитро розуміє логіку.
- Кожний студент, який вчиться на комп'ютерних науках і є студентом старшого курсу, прослухав курс дискретної математики. Наталка прослухала курс дискретної математики. Отже, Наталка є студенткою старшого курсу і вчиться на комп'ютерних науках.
- Кожна папуга виглядає як фрукт. Моя пташка не папуга. Отже, моя пташка не виглядає як фрукт.
- Роман любить дивитись бойовики. Роман любить фільм „Третій зайвий”. Отже, фільм „Третій зайвий” – бойовик.
- Кожний студент університету повинен жити у гуртожитку. Михайло не живе у гуртожитку. Отже, Михайло не студент університету.
- Якщо геометрична фігура – квадрат, то її діагоналі взаємно перпендикулярні і в точці перетину діляться пополам. Ця фігура не квадрат. Отже, її діагоналі не перпендикулярні та не діляться пополам.
- Формула логіки висловлювань є тавтологією, якщо всі її диз'юнкти містять контрарні пари літералів. Задана формула не тавтологія. Отже, принаймні один її диз'юнкт не містить контрарної пари літералів.
- Якщо число має дільник 6, то воно має дільниками числа 2 і 3. Якщо число має дільниками числа 2 та 3, то воно має дільник 6. Отже, число має дільник 6 тоді і лише тоді, коли воно має дільниками числа 2 і 3.

з) Якщо Петро поїде до Харкова, то Іван поїде до Києва. Петро поїде або до Харкова, або до Львова. Якщо Петро поїде до Львова, то Ольга залишиться у Полтаві. Ольга не залишилась в Полтаві. Отже, Іван поїхав до Києва.

14. Довести логічні теореми:

- $\bar{p} \vee q, \bar{p} \vee r, \bar{q} \vee \bar{r} \vdash \bar{p}$ ; б)  $\bar{h}, \bar{h} \rightarrow (p \vee q), p \rightarrow c, q \rightarrow c \vdash c$ .

15. За алгоритмом Девіса–Патнема показати, що множина  $S = \{\bar{p} \vee q \vee \bar{r}, \bar{p} \vee \bar{q}, p, r, u\}$  невиконанна.

16. За алгоритмом Девіса–Патнема показати, що наведені нижче множини виконанні:

- $S = \{p \vee \bar{q}, q, \bar{p} \vee \bar{q} \vee \bar{r}\}$ ;
- $S = \{\bar{p} \vee \bar{q}, p \vee q, q \vee \bar{r}, \bar{q} \vee \bar{r}\}$ ;
- $S = \{p \vee \bar{q}, p \vee q, r \vee \bar{q}, r \vee q\}$ .

17. Перевірити правильність наведених нижче тверджень:

- $p \rightarrow q, \bar{p} \rightarrow q \vdash q$ ; б)  $p \rightarrow q, \bar{q} \rightarrow \bar{r} \vdash r \rightarrow p$ ;
- $p \rightarrow q, r \rightarrow \bar{q} \vdash p \rightarrow \bar{r}$ ; в)  $p \rightarrow q, \bar{p} \rightarrow r \vdash q \rightarrow r$ ;
- $p \rightarrow r, q \rightarrow r \vdash q \rightarrow p$ ; г)  $p \rightarrow q, r \rightarrow s, p \wedge r \vdash q \wedge s$ ;
- $(p \vee q) \rightarrow r, (p \wedge r) \rightarrow \bar{q} \vdash \bar{r} \rightarrow \bar{p}$ .

18. За алгоритмом Девіса–Патнема довести виконанність наведених нижче формул:

- $p \wedge q \wedge r$ ; б)  $(p \vee q) \wedge (\bar{p} \vee q) \wedge r$ ;
- $(p \vee q) \wedge \bar{q}$ ; в)  $(\bar{p} \vee q) \wedge \bar{q} \wedge p$ ;
- $(p \vee q) \wedge (r \vee q) \wedge \bar{r} \wedge \bar{q}$ ; г)  $(p \vee q) \wedge (\bar{p} \vee q) \wedge (\bar{r} \vee \bar{q}) \wedge (r \vee \bar{q})$ .

19. Побудувати резольвенту диз'юнктивів  $d_1$  та  $d_2$ :

- $d_1 = p \vee r, d_2 = \bar{p} \vee q$ ; б)  $d_1 = \bar{p} \vee q \vee r, d_2 = \bar{q} \vee s$ .

20. Перевірити наведені нижче твердження, використовуючи метод резолюцій:

- $p \vee q, p \rightarrow r, q \rightarrow r \vdash r$ ; б)  $p \vee q \vee s, p \rightarrow r, q \rightarrow r, s \rightarrow r \vdash r$ ;
- $(p \wedge \bar{q}) \rightarrow \bar{p}, (q \wedge \bar{r}) \rightarrow r \vdash p \rightarrow r$ ; в)  $p \vee q, p \rightarrow r, q \rightarrow s \vdash r \vee s$ ;
- $p \rightarrow r, \bar{p} \rightarrow s, r \rightarrow q, s \rightarrow \bar{t}, t \vdash q$ ; г)  $p \rightarrow q, \bar{p} \rightarrow r, q \rightarrow s \vdash r \vee s$ .

21. За методом резолюцій перевірити, чи можна спростувати наведені нижче множини:

- $\{p \vee q \vee r, \bar{p} \vee r, \bar{q}, \bar{r}\}$ ; б)  $\{p \vee q, \bar{p} \vee q, \bar{q} \vee r\}$ ;
- $\{\bar{p} \vee q, \bar{q}, p\}$ ; в)  $\{p \vee q, \bar{p} \vee \bar{q}, \bar{p} \vee q, p \vee \bar{q}\}$ .

22. Побудувати випереджену нормальну форму для наведених нижче формул:

- $\forall x(P(x) \rightarrow \exists y Q(x, y))$ ;
- $\exists x(\neg((\exists y P(x, y)) \rightarrow (\exists z Q(z) \rightarrow R(x))))$ ;
- $\exists x(\neg((\exists y P(x, y)) \rightarrow (\exists z Q(z) \rightarrow R(x))))$ ;
- $\forall x \forall y (\exists z P(x, y, z) \wedge (\exists u Q(x, u) \rightarrow \exists v Q(y, v)))$ .

23. Визначити, чи мають склейки такі диз'юнкти:

- $P(x) \vee Q(y) \vee P(f(x))$ ; б)  $P(x) \vee P(a) \vee Q(f(x)) \vee Q(f(a))$ ;
- $P(x, y) \vee P(a, f(a))$ ; в)  $P(a) \vee P(b) \vee P(x)$ ;

$$d) P(x) \vee P(f(y)) \vee Q(x, y).$$

24. Визначити, чи можна уніфікувати наведені нижче множини:

$$a) W = \{Q(a), Q(b)\};$$

$$b) W = \{Q(a, x), Q(a, a)\};$$

$$e) W = \{Q(a, x, f(x)), Q(a, y, a)\}; \quad z) W = \{Q(x, y, z), Q(u, h(v, v), u)\};$$

$$d) W = \{Q(b, y), Q(x, f(a))\};$$

$$e) W = \{P(x_1, g(x_1), x_2, h(x_1, x_2), x_3, k(x_1, x_2, x_3)), P(y_1, y_2, e(y_2), y_3, f(y_2, y_3), y_4)\}.$$

25. Довести, що коли  $\theta, \lambda, \mu$  – підстановки, то  $(\theta \circ \lambda) \circ \mu = \theta \circ (\lambda \circ \mu)$ .

26. Задано підстановку  $\theta = \{a/x, b/y, g(x, y)/z\}$  та формулу  $E = P(h(x), z)$ .

Знайти  $E\theta$ .

27. Задано підстановки  $\theta_1 = \{a/x, f(z)/y, y/z\}$  та  $\theta_2 = \{b/x, z/y, g(x)/z\}$ .

Побудувати  $\theta_2 \circ \theta_1$  і  $\theta_1 \circ \theta_2$ .

28. Застосувавши алгоритм уніфікації, знайти найзагальніший уніфікатор диз'юнкту  $W = \{P(a, y, g(f(x))), P(z, g(z), g(u))\}$ .

29. Визначити, чи можна уніфікувати множини  $W = \{Q(f(a), g(y)), Q(x, x)\}$ .

30. Побудувати скелемівські нормальні форми для наведених нижче формул:

$$a) \neg(\forall x P(x) \rightarrow \exists y \forall z Q(y, z));$$

$$b) \neg(\forall x P(x) \rightarrow \exists y P(y));$$

$$e) \forall x (\neg E(x, 0) \rightarrow \exists y (E(y, g(x)) \wedge \forall z (E(z, g(x)) \rightarrow E(y, z))));$$

$$z) \forall x \exists y \forall z \exists u \forall v \exists w P(x, y, z, u, v, w).$$

31. Отримати клаузальну нормальну форму для

$$\forall x \exists y \exists z ((\bar{P}(x, y) \wedge G(x, z)) \vee R(x, y, z)).$$

32. Методом резолюцій довести невиконанність наведених нижче множин:

$$a) S = \{p, \bar{p} \vee q, \bar{p} \vee r, \bar{p} \vee \bar{q} \vee \bar{r}\};$$

$$b) S = \{p, q, r, w, \bar{p} \vee \bar{q} \vee \bar{r} \vee \bar{w}\};$$

$$e) S = \{R(a) \vee R(b), \neg D(y) \vee L(a, y), \neg R(x) \vee \neg Q(y) \vee \neg L(x, y), \\ D(a) \vee \neg Q(a), Q(b) \vee \neg R(b)\}.$$

33. Дано гіпотези  $\forall x (P(x) \rightarrow Q(x))$  та  $\neg Q(a)$ . Довести, що  $\neg P(a)$  є їхнім логічним наслідком.

34. Використавши алгоритм SLD-резолюцій, спростувати множини наведених нижче хорнівських диз'юнктів:

$$a) S = \{\{q\}, \{\bar{p}, \bar{q}, \bar{s}\}, \{p, \bar{q}, \bar{r}\}, \{\bar{q}, \bar{r}, s\}, \{\bar{q}, r\}, \{p, \bar{q}\}\};$$

$$b) S = \{\bar{p} \vee \bar{q} \vee \bar{r}, \bar{p} \vee q \vee \bar{t}, p \vee \bar{r} \vee \bar{t}, \bar{q} \vee t, q, \bar{q} \vee r\};$$

$$e) S = \{\bar{p} \vee \bar{q}, \bar{r} \vee \bar{t}, \bar{p} \vee r \vee \bar{t}, \bar{q} \vee t, q\}$$

35. Задано множину диз'юнктивів  $\{p \vee r, q \vee \bar{t}, q \vee \bar{p} \vee r, \bar{r}, \bar{r} \vee \bar{p}, t \vee \bar{s}, \bar{q}\}$ .

Побудувати відповідну логічну програму. Використовуючи інтерпретатор логічної програми, виконати її.



## КОМП'ЮТЕРНІ ПРОЕКТИ

Скласти програми із зазначеними вхідними даними та результатами

1. Задано формулу логіки висловлювань. Знайти її значення в довільній повній або частковій інтерпретаціях.

2. Задано множину диз'юнктивів логіки висловлювань. Методом Куайна визначити, чи виконання ця множина.

3. Задано множину диз'юнктивів логіки висловлювань. Методом Девіса Патієма визначити, чи виконання ця множина.

4. Задано множину диз'юнктивів логіки висловлювань. Методом резолюцій визначити, чи виконання ця множина.

5. Задано скінченну множину атомів логіки першого ступеня. Знайти найзагальніший уніфікатор цієї множини.

6. Задано скінченну множину хорнівських диз'юнктивів. Методом резолюцій для хорнівських диз'юнктивів визначити, чи виконання ця множина.

7. Задано скінченну множину диз'юнктивів логіки першого ступеня. Методом резолюцій визначити, чи виконання ця множина.

8. Задано множину диз'юнктивів логіки висловлювань. Реалізувати алгоритм інтерпретатора логічної програми для спростування цієї множини.

9. Задано множину диз'юнктивів логіки першого ступеня. Реалізувати алгоритм інтерпретатора логічної програми для спростування цієї множини.



- ◆ *Інтуїтивне розуміння навчання*
- ◆ *Означення навчання*
- ◆ *Програми, що навчаються*
- ◆ *Мотивація до навчання*
- ◆ *Таксономія машинного навчання*
- ◆ *Споріднені галузі*
- ◆ *Навчання як розділ штучного інтелекту*
- ◆ *Загальне формулювання задачі навчання за прецедентами*
- ◆ *Основні поняття та означення*
- ◆ *Типологія завдань навчання за прецедентами*
- ◆ *Задачі з описом об'єктів на основі ознак*
- ◆ *Приклади задач машинного навчання*
- ◆ *Навчання в штучному інтелекті*
- ◆ *Дерево рішень*
- ◆ *Навчання на основі зв'язків*
- ◆ *Метод опорних векторів*
- ◆ *Мережі, що самоорганізуються*

### 3.1. ІНТУЇТИВНЕ РОЗУМІННЯ НАВЧАННЯ

На початку цього розділу, спираючись здебільшого на інтуїцію, з'ясуємо, що розумітимемо під поняттями „навчання”. У подальшому розглянемо деякі системи, які самонавчаються, та покажемо, чому такі системи заслуговують уваги. Також побачимо, як за допомогою кількох критеріїв можна виділити багато видів навчання, котрим задовольняють відповідні алгоритми.

Під час навчання людина здобуває навички до мовлення, читання, уміння програмувати, а також здатність ходити, їсти, їздити за кермом, плавати тощо. Ми вчимося розпізнавати обличчя знайомих людей, вік людини на основі її вигляду, настроїв на основі виразу обличчя та жестикуляції. Ми вчимося з допомогою вчителів та підручників. Ми узагальнимо наші спостереження та відкриваємо залежності між ними. Роблячи спроби, ми допускаємо помилки, на які звертають увагу тренери або ми самі, ми намагаємось уникати подальшого повторення цих помилок. Без перебільшення можна сказати, що весь наш досвід створений за допомогою навчання. Основні наші психологічні, інтелектуальні та фізичні риси – наслідок довготривалого та неперервного процесу навчання.

Тварини також навчаються. Собака впізнає свого господаря та інших людей, реагує на своє ім'я та знає, коли та в якому місці може сподіватись на порцію їжі. Більш розвинуті представники наших гавкаючих друзів дивують нас проявами набагато складніших знань чи навичок, не вроджених, а набутих. Саме на здібності до навчання спирається їхнє використання поліцією, прикордонниками та іншими службами. Щурі чудово вчаться уникати отрути, що розкладають у підвалах. Групу шимпанзе навіть удалося навчити спілкуватись мовою жестів.

Яке відношення мають приклади навчання, які зустрічаються в людей і тварин, до навчання в штучному інтелекті? Ми маємо на увазі штучні системи, якими є комп'ютерні програми, і яким можна надавати певну властивість. Цю властивість, при всіх відмінностях її від властивостей натуральних біологічних систем, найкраще можна назвати здатністю до навчання. Наша мета – навчитись створювати комп'ютерні програми, які володіють цією властивістю. Спочатку спробуємо краще зрозуміти природу цієї властивості та показати доцільність урахування її в алгоритмічному застосуванні, а не лише в біологічному чи психічному.

### 3.2. ОЗНАЧЕННЯ НАВЧАННЯ

Перша проблема, яку будемо обговорювати, полягає в тому, як серед усіх щойно розглянутих прикладів навчання знайти спільні риси. Іншими словами, у чому полягає здатність до навчання, притаманна певним біологічним системам, і котру хочемо надати створюваним штучним системам? При великій різниці способів навчання, які застосовуємо ми самі або спостерігаємо в інших людей чи тварин, імовірно зіштовхнемось із проблемою точного визначення поняття „навчання”. У певних випадках можна казати, чи маємо справу з навчанням, чи ні, але формулювання загальних властивостей цього поняття може викликати труднощі. Незважаючи на це, спробуємо зробити кілька перших кроків у подоланні цих труднощів.

Першим елементом спрощеного означення поняття „навчання” є зміна. Не можна казати, що система навчається, якщо вона залишається незмінною в часі. Але не кожна зміна в системі має характер навчання, наприклад, зміною є забування.

Розглядаючи зміни, вимагатимемо, щоб вони були корисними, тобто несли певне покращення. Тут ідеться про покращення ефективності системи з точки зору її функціональності. Функцію людського навчання, незважаючи на біологічну обумовленість, визначає цивілізація, суспільство й світогляд, навчання тварин – природа або дресирувальник, а навчання програм – програміст чи користувач. Може статись, що зміна в системі буде корисна з погляду на одні її функції, але не корисна з погляду на інші. Зміни зі змішаним типом корисності також називатимемо навчальними, та відмовимо в такій назві змінам, які негативні для функціонування системи. Для кожної конкретної системи шукатимемо можливість оцінити якість функціонування, що вможливить розрізняти корисні та некорисні зміни.

Однак не кожну корисну зміну можна назвати навчанням. Тварини, яким дають більше їжі, стають спритнішими, але нічому не навчаються. Нічому не навчається програма, яка має покращене обчислення кількох функцій або яка скомпільована на

кращому компіляторі, хоча в обох випадках можна спостерігати корисні зміни. Не вважаємо також за навчання не вроджені здатності, які отримує людина, наприклад, за допомогою нейрохірургічного втручання або вживання психотропних засобів. Наведені приклади розширюють визначення вимогою автономності корисних змін у системі. Отже, система, що навчається, змінюється на краще сама, без зовнішнього втручання.

Потрібно розрізнити самостійні та ймовірні зміни від змін, які, хоча й автономні, але відбуваються під впливом певних зовнішніх чинників. Щоб корисну автономну зміну в системі назвати навчанням, додатково вимагатимемо розглядати ці зовнішні чинники як досвід, який набуває система. Цю умову задовольняють результати спостережень за системою або отримання інформації щодо її функціонування, покращення яких є зовнішньою характеристикою навчання системи. Для тварини – це нагорода за вдале виконання завдання дресирувальника та покарання – за невдале.

У природі досвід тварини, на основі якого вона навчається, походить із таких вимог навколишнього середовища, як пошук їжі, втеча від загрози, спостереження та наслідування інших представників того самого виду. Для навчання на шкільній лаві існує більше форм, у яких набувається досвід. Це – учитель, який наводить приклади виконання завдань, та самостійні спроби розв'язування інших завдань, які оцінює вчитель. На підставі всього сказаного так сформулюємо поняття „навчання”. *Навчання системи* – це будь-яка автономна зміна в системі, яка відбувається на основі досвіду та веде до покращення якості її функціонування.

Систему, у якій за певний проміжок часу відбуваються зміни, відповідні наведеному вище означенню, називають *учнем*. Оскільки зміни в системі оцінюють відносно об'єктивно, то результати функціонування залежать від суб'єктивного погляду. Зміна нічого не варта, якщо не визначити аспектів діяльності системи та не сформулювати критерій якості, за яким хочемо її оцінювати. Ще важче оцінити автономність змін, і зовсім важко вирішити, чи наступили вони на основі досвіду. Зазначені проблеми свідчать не тільки про недоліки означення поняття „навчання”, а швидше про неточність нашого розуміння навчання, штучне уточнення якого мало що покращує. Тому трактуватимемо їх як підказку, яка дає змогу зрозуміти зміст поняття „навчання”. Це поняття краще розглядати разом із різними конкретними прикладами. У подальшому такі приклади стосуватимуться комп'ютерних програм, що навчаються.

### 3.3. ПРОГРАМИ, ЩО НАВЧАЮТЬСЯ

Спроби написання програм, що навчаються, не зумовлені бажанням зменшити зусилля програмістів. Вони не є альтернативою традиційним технологіям програмування, так само, як і здатність людини чи тварини до навчання не замінює відповідних механізмів еволюції організмів. Причиною таких спроб не є нездатність писати програми, які використовують сучасні методи аналізу та проектування. Нею є певні складні задачі, для яких складно або неможливо побудувати правильні та повні алгоритми. Програму, що навчається, можна вважати також програмою, яка використовує абстрактний, „параметричний” алгоритм виконання завдання. Навчання полягає у виборі на основі досвіду відповідних „параметрів”, які цей абстрактний алгоритм переводять у конкретний алгоритм, що задовольняє вимоги постановника завдання.

„Параметри”, які отримують у результаті навчання, називають *знанням* або *вмінням*, залежно від характеру завдання та прийнятої точки зору. Оскільки вони не надані *учневі* безпосередньо, а визначені учнем внаслідок автономного навчання, їх часто називають його гіпотезою. Гіпотеза походить із певної множини – *простору гіпотез*, який містить усі гіпотези, які може використати учень для вирішення завдання. Така термінологія підкреслює, з одного боку, непадійність знань та вмінь учня, котрі найчастіше не можуть бути визнані за надійні та можуть вимагати перевірки, а, з іншого боку, можуть відігравати роль автономних суб'єктів.

Різниця між знанням та вмінням є розмитою та має умовний характер. Швидше про вміння, ніж про знання йде мова у випадку завдань, коли програмі потрібно виконати певну послідовність операцій, котра визначена в результаті навчання. Вона є відповіддю на питання, як потрібно діяти аби досягнути бажаної мети. Про знання треба говорити швидше тоді, коли йдеться про вибір одного з варіантів певного рішення, яке найчастіше стосується інтерпретації вхідних даних. Відповідає вона на питання, що відомо про об'єкт дослідження, наприклад, що він собою являє або як пов'язаний з іншими об'єктами. Часто й знання в такому вузькому сенсі, і вміння разом називають *знанням у ширшому сенсі*. Одночасно про знання у *вузькому сенсі* можна сказати як про *декларативне знання*. Воно описує певні об'єкти, ситуації, зв'язки. Тоді про вміння говорять як про *продукційне знання*. Воно визначає стратегію досягнення певних цілей. Алгоритми пошуку або вдосконалення знань та вмінь називають *алгоритмами навчання*.

У процесі навчання відбувається зміна прихованих та використовуваних програмою „параметрів” алгоритму виконання завдання, або знань та вмінь. Можна сподіватися, що це є корисна зміна, зрештою, якщо задані спочатку значення цих „параметрів” не були відповідно підібрані. Якби існувала можливість такого вибору, то навчання не було б потрібне. Ця зміна автономна, бо виконує її сам алгоритм, а не програміст чи оператор. Зрештою, діє на підставі експериментів, котрі будемо в цьому випадку визначати також за допомогою *тренувальної інформації*. Так само, як і біологічні системи, програми, що навчаються, можуть володіти вродженими знаннями, які внесені програмістом. Ці знання в процесі навчання можуть удосконалюватись та корегуватись.

Наведемо кілька конкретних прикладів навчання, які можна використати для написання комп'ютерних програм.

**Приклад 3.1. Гра в шашки.** Розглянемо програму, яка грає в шашки. Спочатку вона грає формально правильно, згідно правил гри, але посередньо. Тут знаннями системи може бути функція, яка оцінює якість різних можливих позицій на дошці з погляду на шанси виграти. Зміну цієї функції можна здійснити на основі результатів зіграних партій. Якщо програмі, завдяки цій зміні, вдасться краще оцінювати якість позицій, що виникають у процесі гри, то їй вдасться краще вибирати свої ходи, і рівень гри підвищиться. Цей приклад показує відмінність між знанням та вмінням: знання, що подані функцією оцінювання позицій, перетворюються на вміння вибирати ходи.

**Приклад 3.2. Медична діагностика.** Нехай відділення лікарні, яке спеціалізується на певному виді хвороб, нагромадило в базі даних записи історій хвороби своїх пацієнтів. Там можуть бути результати обстеження окремих пацієнтів на основі різних діагностичних тестів, витяги з історій хвороб, визначені симптоми, діагнози лікарів, а також інформація про застосування терапії та її результати. Дані такого типу, якщо

вони охоплюють доволі багато пацієнтів і є достатньо повними та достовірними, можуть бути потенційно цінним джерелом знань, яке може допомогти лікарям у діагностиці та лікуванні нових пацієнтів. Можна припустити, що на основі таких знань програма побудує правила розпізнавання хвороб і знаходження терапії на основі симптомів і результатів тестів. Така програма може працювати як дорадча система, жодною мірою не заміняючи компетентного лікаря, а лише надаючи йому допомогу.

**Приклад 3.3. Викриття зловживань.** Нехай розрахунок у торгівельній або сервісній установі здійснюється за допомогою банківської картки. Транзакцію здійснює працівник такої установи за допомогою терміналу, який зчитує дані з магнітної стрічки картки. Одночасно термінал може зв'язатись зі спеціальним центром для виконання авторизації. У найпростішому випадку перевіряється, чи картка дійсна, чи вона платоспроможна, і чи квота транзакції не перевищує межі, яка встановлена для цієї картки її емітентом (для дебетних карт розміру рахунку). Транзакція відбудеться в разі успішної авторизації. Проте кредитними картками цікавляться любителі чужої власності. Вони можуть спробувати здійснити транзакцію із чужого рахунку вкраденою або загубленою карткою. Вони можуть виготовити фальшиву картку, або скористатись номером картки. Якщо авторизація таких транзакцій пройде успішно, клієнт може довідатись записно. Бажано викривати злочинні транзакції на етапі авторизації. Можна уявити використання із цією метою програми, що навчається, яка на основі аналізу історичних даних, що стосуються „хороших” та „поганих” транзакцій, окреслить певні риси, які дозволять розрізнити їх максимально правильно. Свої знання із цієї проблеми ця програма буде періодично оновлювати, використовуючи нові порції даних. Така адаптивна система викриття зловживань мала б істотні переваги над системами з постійними, простими правилами визначення небезпеки, які підготовлені фахівцями. Варто додати, що в деяких інших галузях проблема викриття зловживань також актуальна, зокрема, у мобільній телефонії, де мова йде про розмови за чужий рахунок, у комп'ютерних мережах, де йдеться про підозрілу активність хакерів.

**Приклад 3.4. Класифікація документів.** Розвиток комп'ютерних технологій збереження та пересилання інформації вимагає методів її автоматичної класифікації та фільтрування. Користувачі лавиноподібно зростаючих засобів, доступних у мережі Інтернет, могли б заощадити багато часу, користуючись інструментальними засобами, які читали би та в розумний спосіб відбирали найцікавіші з їхньої точки зору сторінки або статті з дискусійних груп. Корисними могли б бути програми, котрі на основі кількох документів, класифікованих користувачем на тематичні категорії, або на категорії цікаві й не цікаві, навчилися б відповідно класифікувати нові документи. Бажаним може бути також удосконалення знань програми в процесі її використання, коли після читання нових документів, користувач підтверджує їхню класифікацію, здійснену програмою.

**Приклад 3.5. Керування поїздкою.** Напевно, неохоче відали б своє життя й здоров'я під опіку електронних водіїв таксі, котрі возили б нас по заповнених автомобілями вулицях міста. Проте можна розглядати принаймні кілька ситуацій, коли комп'ютеру вдається певною мірою керувати поїздками. Це може бути автоматичне паркування автомобілів на паркінгах, перевезення товарів і осіб по простих трасах на обмежені відстані, наприклад, усередині виробничого підприємства, або допомога й часткова заміна водія-людини під час поїздки на деяких відрізках траси, наприклад, на автострадах. Системи такого типу могли б приймати рішення про правильний стан

засобів керування, наприклад, вибирати кут повороту керма автомобіля на основі ситуації на дорозі, отриманої з однієї або кількох розміщених на машині камер і додаткових спеціальних датчиків. Як джерело навчальної інформації, що дає змогу набутися таке вміння, можна використати вчителя, що показує правильне керування поїздкою в різних умовах.

**Приклад 3.6. Пересування в офісі.** Деякі фірми та установи займають великі будинки з мережею коридорів і численними кімнатами на кожному поверсі. Оскільки поступ інформаційних технологій не вивільнив від потреби в інтенсивному обігу паперів у таких фірмах та установах, постає потреба призначення частини часу для того, щоб висококваліфікований персонал бігав по коридорах із папками документів у руках або наймати кур'єрів. Альтернативою можна визнати механічних кур'єрів – рухомих роботів, керованих програмами, що вможливають їхнє пересування. Одним з умінь, яким має володіти така програма, є вміння навігації – оминати людей та перешкоди, проходити скрізь двері, просто пересуватися вздовж коридору. Така програма має також керувати моторами, за допомогою яких робот рухається, на основі інформації, що надходить із його сенсорів. Необхідність навчання впливає зі складності формування готового, повного, конкретного алгоритму для вирішення цих проблем. труднощів, які додатково з'являються через неточність вимірів, що надходять від сенсорів. Один із можливих підходів до навчання може полягати в дозволі роботу експериментувати – робити спроби рухатись й здійснювати помилки, які мають призвести до знаходження задовільної стратегії керування.

Реальні задачі навчання, які мають відношення до наведених прикладів та сформульовані тут у спрощеному вигляді, насправді є доволі складними й не такими, щоб їх можна було описати в цьому розділі в готовому до реалізації вигляді. Для деяких із них відомі розв'язання, що не є цілком прийнятними. У цьому розділі будуть сформульовані алгоритми та надані поради до їхнього застосування. Уміле й творче використання цих алгоритмів дасть змогу проектувати системи, що навчаються, для багатьох важливих практичних задач.

### 3.4. МОТИВАЦІЯ ДО НАВЧАННЯ

Людині легко визначити свою мотивацію до навчання. Якщо дивитись ширше, то здатність до навчання є властивістю біологічних видів. Ця здатність забезпечує виживання, розмноження та пристосування, зміни ознак та поведінки. Тоді що розуміти під мотивацією для спеціальної комп'ютерної програми, що навчається?

Складається враження, що в цьому випадку доволі близькою є біологічна аналогія. Як такому „програмісту”, яким є еволюція, вигідно „параметризувати” організми деяких видів, щоб вони впродовж свого життя набували знань і умінь, так і програмістам у деяких практичних застосуваннях вигідно використовувати програми, що навчаються, замість того щоб витрачати свій час, зусилля та кошти на розроблення строго визначених алгоритмів, котрі можуть виявитися в застосуваннях неправильними, або неповними, тобто такими, що не задовольняють усі вимоги до них, або виконуються з помилками.

Розвиваючи цю думку, можна навести такі причини важливості навчання комп'ютерних програм.

1. Для насправді складних завдань важко сформулювати цілком визначені, правильні та повні алгоритми їхнього розв'язання. Стосується це реальних проектів, у яких потрібно зважати на недетермінізм і змінність середовища, у якому виконується програма. Пошук задовільних алгоритмів може бути неможливим або економічно невиправданим, оскільки він вимагатиме значних зусиль або засобів. Такі середовища складні для опису, часто або не мають прийнятних теоретичних моделей, або їх пошук дуже дорогий. Навіть, якщо такі моделі побудовані, то з погляду на їхню спрощеність або непередбачувану змінність середовища, створені на їх основі алгоритми можуть виявитися малоефективними.

2. Інтелектуальні інформаційні системи в багатьох сучасних застосуваннях мають бути максимально автономними, тобто здатними функціонувати без втручання людини. Стосується це ефективності систем керування складними процесами, дією промислових та офісних роботів, засобів пересування без екіпажів. Потрібний ступінь автономії неможливий без адаптивності, тобто здатності до пристосування до змін середовища та вимог. Програму, що навчається, можна використати в середовищі, у якому відбуваються важкі до передбачення зміни, або в багатьох різних середовищах.

3. Множини даних, які одержано на основі вимірювань, спостережень, документів часто надзвичайно великі й складні, аби можна було знаходити в них залежності та класифікувати їх неавтоматизованим способом. Програми навчання дають змогу імовірного процесу виведення на основі історичних даних. Сучасний аналіз даних став одним із найважливіших застосувань машинного навчання.

### 3.5. ТАКСОНОМІЯ МАШИННОГО НАВЧАННЯ

Види навчання, а, швидше, види систем, що навчаються, можна класифікувати за низкою критеріїв. Серед них найбільше значення мають такі критерії.

1. Метод подання знань та вмінь.
2. Спосіб застосування знань та вмінь.
3. Джерело та вигляд навчальної інформації.
4. Механізм набування та вдосконалення знань та вмінь.

Щодо подання знань та вмінь, то відомі різні їх способи, які формують програми, що навчаються. Окреслена галузь застосування проектованої системи, що навчається, на загал містить інформацію, з якої потрібно вибрати щонайбільше кілька можливостей, з яких кожна має власні переваги й обмеження. У цьому посібнику не ставимо за мету систематичне вивчення методів подання знань, але це буде зроблено під час вивчення конкретних методів, які використовують відповідні подання. Такими методами є дерева рішень, правила виведення, логіки першого ступеня, розподілів імовірностей.

Не завжди вибраний метод подання знань однозначно визначає спосіб їх використання, хоча часто для цього немає великого вибору. Спосіб застосування знань визначає метод подання знань та мету, якій ці знання мають слугувати. До найтипівіших задач належать класифікація та апроксимація. *Класифікація* встановлює належність об'єктів певним категоріям, або класам, а *апроксимація* – відображення об'єктів у множину дійсних чисел. У випадку класифікації говоритимемо про навчання понять, розуміючи під поняттям знання про належність об'єкта певної галузі визначеної категорії.

Джерела та вигляд відомої навчальної інформації в найпростішому випадку

поділяють традиційно на навчання з учителем та навчання без учителя. Наразі достатньо прийняти, що система, що навчається, має завдання спостерігати за вхідними даними та отримувати на їх основі вихідні дані. Процес навчання полягає у формулюванні алгоритму генерування цих вихідних даних. Навчальні дані впливають певним способом на учня, опосередковано чи безпосередньо, для формування відповідної відповіді.

У випадку навчання з учителем, у якому джерело навчальних даних називають учителем, учень отримує інформацію, яка для певної сукупності вхідних даних визначає його правильні відповіді як приклади поведінки, якої від нього очікують. У разі навчання без учителя, така навчальна інформація взагалі недоступна. На основі лише вхідних даних учень має навчитися формувати відповіді винятково на основі їхньої послідовності. Таке навчання використовують у випадку групування або інших перетворень множини вхідних даних, які не походять від учителя, а є лише інтегральною характеристикою алгоритму навчання. Часом кажуть, що система навчання без учителя має вбудованого вчителя.

Окрім навчання з учителем і навчання без учителя існують види навчання, які з огляду на джерело та характер навчальних даних важко віднести до жодної з тих груп. Близьким до навчання з учителем є навчання на основі запитань: тут навчальні дані також надходять від учителя, але у формі відповідей на явні запитання, які задані учнем. Учитель не може активно вибирати приклади, а є лише тим, до кого звертається учень.

Про навчання шляхом експериментування говоримо тоді, коли учень нагромаджує досвід на основі експериментів із середовищем, що може полягати у виконанні певних дій і спостереженні за їх результатом. Іноді такий вид навчання трактують як особливий випадок навчання без учителя, але такий підхід може викликати непорозуміння.

Нарешті, розглянемо навчання з підкріпленням, яке спирається на експериментування, але використовує для цього додаткове джерело навчальної інформації, його іноді називають критиком, який оцінює якість дій учня. Навчальна інформація має в цьому випадку не характер інструкцій, оскільки не вказує учневі, що він має робити, а лише характер оцінок, бо дає інформацію учневі, хорошими чи поганими є його дії. Жодного з наведених тут видів навчання свідомо не визначаємо формально точно, оскільки відмінності між ними не є чіткими, а в певних суміжних ситуаціях можуть порізно трактуватися з різних міркувань. Тому не будемо конкретизувати зазначених видів навчання до розгляду характерних алгоритмів.

За допомогою отриманої навчальної інформації система, що навчається, генерує нові або вдосконалені вже відомі знання або вміння, які в певний спосіб подані та призначені для виконання сформульованого завдання. Механізм накопичення та вдосконалення знань найчастіше однозначно визначений методами подання знань або виглядом навчальної інформації. Існує багато алгоритмів навчання, які виконують часткові механізми набуття знання. Найпопулярнішим механізмом навчання є індукція – підхід до процесу виведення, який полягає в узагальненні конкретної навчальної інформації з метою узагальнення знання. Варто говорити про цілу сім'ю індукційних механізмів.

Вивчення систем, що навчаються, у міру зростання обсягу досліджень та їх урізноманітнення, а також популярності, стало причиною розвитку нової наукової дисципліни. Науку про навчання штучних систем називають часом *машинним навчанням*, або *навчанням машин*. Її трактують як напрям штучного інтелекту або



як розділ інформатики. З поглибленням спеціалізації в сучасній науці зростає також значення міждисциплінарних досліджень, і щоразу важче будувати прості ієрархічні класифікації. Істотний внесок у дослідження систем, що навчаються, вносять окрім інформатиків також математики, фахівці з автоматизації, психологи й біологи.

Можна визначити три основні напрямки в наукових працях із машинного навчання, які можна назвати теоретичним, біологічним та системним. Теоретичний напрям займається розвитком теоретичних основ алгоритмів навчання. Основними проблемами, зокрема, тут є визначення складності задач навчання, оцінювання часу та кількості навчальної інформації, потрібної для навчання, визначення якості знання для навчання, а також створення однорідного теоретичного підходу для побудови міркувань для різних механізмів навчання.

Біологічний напрям оснований на конструюванні обчислювальних моделей процесів навчання шляхом аналогій із процесами, що існують у природних біологічних системах на різних рівнях їх організації – від клітини до центральної нервової системи.

У значній частині цей розділ посібника присвячено систематичному вивченню алгоритмів навчання, а також питанням конструювання, дослідження й застосування систем, що навчаються.

### 3.6. СПОРІДНЕНІ ГАЛУЗІ

У сучасній науці очевидно немає ізольованих досліджень. Різноманітні дисципліни впливають одна на одну, а значна частина дослідницьких проєктів має міждисциплінарний характер. Саме так є й із машинним навчанням. Варто згадати принаймні про деякі інші галузі, з якими машинне навчання поєднане спільними рисами, що проявляються у використанні їхніх доробків, корисних теоретичних або обчислювальних знарядь. До таких галузей слід віднести:

- 1) теорію ймовірностей;
- 2) теорію інформації;
- 3) формальну логіку;
- 4) математичну статистику;
- 5) теорію керування;
- 6) психологію;
- 7) нейрофізіологію.

Теорія ймовірностей має істотний вплив одночасно як на теоретичний, так і на системний напрям у машинному навчанні. У першому випадку використовують її математичний апарат для аналізу алгоритмів навчання. Стосується це, у першу чергу, обчислювальної теорії навчання. У другому випадку вони є основою різних механізмів імовірнісного виведення, зацікавленість якими зростає, що дозволяє отримувати важливі практичні результати в багатьох галузях.

На дослідження в системному напрямку машинного навчання, а точніше, на ідею деяких алгоритмів навчання, вплинула теорія інформації. Іноді при виборі гіпотез застосовують критерій якості з теорії інформації. Проблему індукційного навчання також можна розглядати як проблему відповідного кодування навчальної інформації.

Формальна логіка вплинула на дослідження систем, що навчаються, як на основу багатьох символічних методів подання знань, зокрема, методів, які використовують

правила. Однак певні види навчання безпосередньо пов'язані з формальною логікою. Зокрема, це стосується індукційного логічного програмування, де знання, яке створене в результаті навчання, подають логічними формулами, а також навчання шляхом з'ясування, де використовують елементи логічної дедукції.

Математична статистика займається методами аналізу даних та виведеннями на основі цих даних. Із цією метою вона використовує їхні розподіли, середні значення, а також апарат теорії ймовірностей. Її мета збігається з метою значної частини алгоритмів машинного навчання, що іноді спонукає до трактування машинного навчання як певного різновиду статистики. Єдина відмінність полягає в тому, що статистика використовує інші засоби. Види знань, які можуть бути досліджені системами, що навчаються, виходять далеко поза статистичний опис даних, які є навчальною інформацією, а тому мають незрівнянно ширші можливості виведення на основі такої інформації. Одночасно деякі види систем, що навчаються, використовують засоби статистики для аналізу навчальних даних і отримання на їх основі висновків, застосованих у навчанні. У першу чергу, це стосується характеристики похибок, які робить учень, та тестів статистичного оцінювання гіпотез, які можуть бути використані для їхнього вибору, а також емпіричного оцінювання якості поведінки систем навчання.

Предметом теорії керування є методи проєктування систем автоматичного керування для різних класів об'єктів та процесів. Завданням регулятора є вплив на керований об'єкт чи процес для збереження чи отримання ним потрібного стану. Оскільки керування полягає у виконанні регулятором послідовності операцій, що мають приводити до визначеної мети, його найкраще пов'язати з навчанням умінь. Передусім, цей зв'язок стосується адаптивного керування, у якому немає інформації про модель керованого об'єкта, а допускається можливість її зміни в часі. У зв'язку із цим можна розглядати відповідні типи систем, що навчаються, як адаптивні регулятори. Можливо також, що висновки теорії керування стануть основою досліджень деяких видів систем навчання.

Феномен навчання людей і високоорганізованих тварин – одна із центральних проблем, які досліджує психологія. Концепції навчання, які розвиваються на цьому ґрунті, впливають не лише на біологічний напрям машинного навчання. Серед практичних навчальних алгоритмів можна виділити такі, що мають психологічне підґрунтя. У сучасному машинному навчанні психологічне коріння має, передусім, навчання з підкріпленням. Кількісна навчальна інформація у вигляді числових нагород нагадує підхід, який застосовують психологи-експериментатори в їх досліджах із навчанням тварин, а термін „підкріплення” походить із психологічної літератури.

Нейрофізіологія, тобто наука про нервову систему людей і тварин, вплинула на дослідження в рамках біологічного напрямку машинного навчання. Ідеться тут, передусім, про нейронні мережі.

### 3.7. НАВЧАННЯ ЯК РОЗДІЛ ШТУЧНОГО ІНТЕЛЕКТУ

Навчання, або машинне навчання (*Machine Learning*) – великий розділ штучного інтелекту, що вивчає методи побудови алгоритмів, здатних навчатись. Розрізняють два типи навчання. *Навчання за прецедентами*, або *індуктивне навчання*, ґрунтується на виявленні загальних закономірностей на основі часткових емпіричних даних.

Дедуктивне навчання передбачає формалізацію знань експертів та перенесення їх у комп'ютер у вигляді бази знань. Дедуктивне навчання прийнято відносити до області експертних систем, тому терміни „машинне навчання” та „навчання за прецедентами” можна вважати синонімами.

Машинне навчання знаходиться на стику математичної статистики, методів оптимізації та класичних математичних дисциплін, але має також і власну специфіку, пов'язану із проблемами обчислювальної ефективності та перенавчання. Багато методів індуктивного навчання розроблялися як альтернатива класичним статистичним підходам. Значна частина методів також тісно пов'язана з видобуванням даних і інтелектуальним аналізом даних (Data Mining).

Найбільш теоретично обґрунтовані розділи машинного навчання об'єднані в окремий напрям – *теорію обчислювального навчання* (COmputational Learning Theory, COLT).

Машинне навчання — не лише математична, але й практична, інженерна дисципліна. Чиста теорія, як правило, не дає змогу одразу формулювати методи та алгоритми, які можна застосовувати практично. Аби змусити їх добре працювати, доводиться винаходити додаткові евристичні, компенсуючи невідповідність умов реальних завдань зробленим теоретичним припущенням. Практично жодне дослідження в машинному навчанні не обходиться без експерименту на модельних або реальних даних, які підтверджують практичну працездатність методу.

### 3.8. ЗАГАЛЬНЕ ФОРМУЛЮВАННЯ ЗАДАЧІ НАВЧАННЯ ЗА ПРЕЦЕДЕНТАМИ

Дана скінченна множина прецедентів (об'єктів, ситуацій), по кожній з яких зібрані (виміряні) певні дані. Дані про прецедент також називають його *описом*. Сукупність усіх наявних описів прецедентів називають *навчальною вибіркою*. Вимагається на основі цих часткових даних виявити загальні залежності, закономірності, взаємозв'язки, властиві не лише цій конкретній вибірці, але взагалі всім прецедентам, у тому числі тим, які ще не спостерігалися. Говорять також про *відновлення залежностей за емпіричними даними* — цей термін був введений у роботах Вапніка й Червоненкіса.

Найбільш поширеним способом опису прецедентів є ознаковий опис. Фіксується сукупність  $n$  показників, виміряних у всіх прецедентах. Якщо всі  $n$  показників числові, то ознакові описи є числовими векторами розмірністю  $n$ . Можливі й складніші випадки, коли прецеденти описують часовими рядами або сигналами, зображеннями, відеорядами, текстами, попарними відношеннями подібності або інтенсивності взаємодії тощо.

Для розв'язування задачі навчання за прецедентами в першу чергу фіксується *модель залежності*, що відновлюється. Після цього вводиться *функціонал якості*, значення якої показує, як добре модель описує спостережувані дані. *Алгоритм навчання* (learning algorithm) шукає такий набір параметрів моделі, при якому функціонал якості на заданій навчальній вибірці набуває оптимального значення. Процес *налаштування* (fitting) моделі за вибіркою даних у більшості випадків зводиться до застосування чисельних методів оптимізації.

У зарубіжних публікаціях термін algorithm уживається в сенсі обчислювальної

процедури, яка за навчальною вибіркою здійснює налаштування моделі. Виходом навчального алгоритму є функція, яка апроксимує невідому (відновлювану) залежність. У задачах класифікації апроксимуючу функцію прийнято називати *класифікатором* (classifier), *концептом* (concept), або *гіпотезою* (hypothesis). У задачах відновлення регресії — *функцією регресії*, або *функцією*. У вітчизняній літературі апроксимуючу функцію також називають *алгоритмом*, підкреслюючи, що й вона має допускати ефективну комп'ютерну реалізацію.

### 3.9. ОСНОВНІ ПОНЯТТЯ ТА ОЗНАЧЕННЯ

Нехай маємо множину об'єктів  $X$ , множину відповідей  $Y$ , та існує цільова функція (target function)  $y^*: X \rightarrow Y$ , значення якої  $y_i = y^*(x_i)$  відомі на скінченній підмножині об'єктів  $\{x_1, x_2, \dots, x_l\} \subset X$ . Пари об'єкт-відповідь  $(x_i, y_i)$  називають *прецедентами*.

Множину пар  $X^l = (x_i, y_i)_{i=1}^l$  називають *навчальною вибіркою* (training sample).

Задача навчання за прецедентами полягає в тому, щоб відновити функціональну залежність між об'єктами та відповідями, тобто побудувати відображення  $a: X \rightarrow Y$ , яке задовольняє таку сукупність вимог.

- ◆ Відображення  $a$  має допускати ефективну комп'ютерну реалізацію. Із цієї причини будемо його називати *алгоритмом*.
- ◆ Алгоритм  $a(x)$  має відновлювати на об'єктах вибірки задані відповіді  $a(x_i) = y_i, i = 1, 2, \dots, l$ . Рівність тут можна розуміти як точну або наближену, залежно від конкретної задачі.
- ◆ На алгоритм  $a(x)$  можуть накладатися різні апіорні обмеження, наприклад, вимога неперервності, гладкості, монотонності тощо, або комбінація кількох вимог. У певних випадках може задаватися *модель алгоритму* — функціональний вигляд відображення  $a(x)$ , визначений з точністю до параметрів.
- ◆ Алгоритм  $a(x)$  має володіти *узагальнюючою властивістю* (generalization ability), тобто достатньо точно наближати цільову функцію  $y^*(x)$  не тільки на об'єктах навчальної вибірки, але й на всій множині  $X$ .

### 3.10. ТИПОЛОГІЯ ЗАДАЧ НАВЧАННЯ ЗА ПРЕЦЕДЕНТАМИ

Навчання з учителем (supervised learning) — найпоширеніший випадок. Кожний прецедент є парою „об'єкт-відповідь”. Потрібно знайти функціональну залежність відповідей від описів об'єктів і побудувати алгоритм, який приймає на вході опис об'єкту й видає на виході відповідь. Функціонал якості зазвичай визначають як середню помилку відповідей, виданих алгоритмом, по всіх об'єктах вибірки. Перелік задач навчання можна сформулювати так.

1. *Задача класифікації (classification).*
2. *Задача регресії (regression).*
3. *Задача прогнозування (forecasting).*
4. *Навчання без учителя (unsupervised learning).*
5. *Задача кластеризації (clustering).*
6. *Задача пошуку асоціативних правил (association rules learning).*
7. *Задача частково навчання (semi-supervised learning).*
8. *Трансдуктивне навчання (transduction learning).*
9. *Навчання з підкріпленням (reinforcement learning).*
10. *Динамічне навчання (online learning).*
11. *Активне навчання (active learning).*
12. *Метанавчання (meta-learning або learning-to-learn).*
13. *Багатозадачне навчання (multi-task learning).*
14. *Індуктивне перенесення (inductive transfer).*

Для задачі класифікації характерне те, що множина допустимих відповідей скінченна. Їх називають *мітками класів (class label)*. Клас – це множина всіх об'єктів із даним значенням мітки. Задача регресії відрізняється тим, що допустимою відповіддю є дійсне число або числовий вектор. У задачі прогнозування об'єктами є відрізки часових рядів, що обриваються в той момент, коли потрібно зробити прогноз на майбутнє. Для розв'язування задач прогнозування вдається пристосувати методи регресії або класифікації, причому в другому випадку йдеться швидше про задачу прийняття рішень.

У випадку навчання без учителя відповіді – невідомі залежності між об'єктами, і їх потрібно шукати. Задача кластеризації полягає в групуванні об'єктів у кластери на основі даних про їх попарну схожість. Функціонали якості можуть визначатися по-різному, наприклад, як відношення середніх відстаней між кластерами і середніх відстаней усередині кластерів.

У задачах пошуку асоціативних правил вихідні дані подаються у вигляді ознакових описів. Потрібно знайти такі набори ознак і такі значення цих ознак, які найчастіше зустрічаються в ознакових описах об'єктів.

Часткове навчання займає проміжне положення між навчанням з учителем і навчанням без учителя. Кожний прецедент є парою „об'єкт-відповідь”, але відповіді відомі лише на частині прецедентів. Приклад прикладної задачі – автоматична рубрикація великої кількості текстів за умови, що деякі з них уже віднесені до певних рубрик.

У трансдуктивному навчанні задано скінченну навчальну вибірку прецедентів. За цими частковими даними вимагається зробити передбачення відносно інших часткових даних – тестової вибірки. На відміну від стандартного формулювання тут не потрібно виявляти загальну закономірність, оскільки відомо, що нових тестових прецедентів не буде. З іншого боку, з'являється можливість поліпшити якість передбачень за рахунок аналізу всієї тестової вибірки в цілому, наприклад, шляхом її кластеризації. У багатьох застосуваннях трансдуктивне навчання практично не відрізняється від часткового навчання.

У навчанні з підкріпленням роль об'єктів відіграють пари „ситуація, прийняте рішення”, а відповідями є значення функціонала якості, що характеризує правильність

прийнятих рішень (реакцію середовища). Як і в задачах прогнозування, тут суттєву роль відіграє чинник часу. Приклади прикладних задач: формування інвестиційних стратегій, автоматичне управління технологічними процесами, самонавчання роботів тощо.

Динамічне навчання може бути як навчанням з учителем, так і без учителя. Його специфіка полягає в тому, що прецеденти надходять у потоці. Потрібно негайно приймати рішення по кожному прецеденту й одночасно донавчати модель алгоритму з урахуванням нових прецедентів. Як і в задачах прогнозування, тут істотну роль відіграє чинник часу.

Активне навчання відрізняється тим, що той, хто навчається має можливість самостійно призначати наступний прецедент.

Метанавчання відрізняється тим, що прецедентами є вже розв'язані задачі навчання. Потрібно визначити, які з використаних у них евристик працюють найефективніше. Кінцева мета такого навчання полягає в забезпеченні постійного автоматичного вдосконалення алгоритму навчання в часі. Іноді до метанавчання помилково відносять побудову алгоритмічних композицій, зокрема, бустінг. Проте в композиціях декілька алгоритмів розв'язують одну й ту саму задачу, тоді як метанавчання передбачає, що розв'язується багато різних задач.

У багатозадачному навчанні множина взаємопов'язаних або схожих задач навчання розв'язується одночасно за допомогою різних алгоритмів навчання, які мають схоже внутрішнє подання. Інформація про подібність задач між собою дає змогу ефективніше вдосконалювати алгоритм навчання й підвищувати якість розв'язування основної задачі.

Для індуктивного перенесення характерний досвід розв'язування окремих часткових задач навчання за прецедентами переноситься на розв'язування подальших часткових задач навчання. Для формалізації та збереження цього досвіду застосовують реляційні або ієрархічні структури подання знань.

У процесі розв'язування задач машинного навчання виникають деякі допоміжні задачі. Допоміжні задачі, як правило, не є цікавими із прикладної точки зору, але використовуються для підвищення якості розв'язку. До допоміжних задач відносять такі задачі.

1. *Скорочення розмірності (dimension reduction).*
2. *Відбір інформативних ознак (features selection).*
3. *Фільтрація викидів (outliers detection).*
4. *Доповнення відсутніх даних (missing data).*

Зокрема, задачі скорочення розмірності або відбору інформативних ознак часто застосовують у задачах класифікації та регресії, оскільки зайві ознаки ускладнюють отримання вхідних даних, ускладнюють процес навчання та призводять до перенавчання.

Фільтрація викидів – виявлення в навчальній вибірці невеликої кількості нетипових об'єктів (викидів, шуму), які можуть з'являтися внаслідок помилок спостереження, несподіваної зміни цільової функції, або неадекватності моделі. Як правило, видалення таких об'єктів або надання їм меншої ваги призводить до побудови надійніших (робастних) алгоритмів.

Доповнення відсутніх даних – заміна відсутніх значень в ознакових описах певними прогнозованими значеннями.

Оскільки певні задачі, які виникають у прикладних галузях, мають риси одразу кількох стандартних типів задач навчання, то їх важко однозначно віднести до одного типу. Прикладом такої задачі є задача формування інвестиційного портфеля (portfolio selection). Ця задача є задачею динамічного навчання з підкріпленням, у якій дуже важливим є відбір інформативних ознак. Ознаками тут є фінансові інструменти. Склад оптимального набору ознак (портфеля) може змінюватися в часі. Функціоналом якості є довгостроковий прибуток від інвестування в дану стратегію управління портфелем.

Колаборативна фільтрація (collaborative filtering) – це прогнозування переваг користувачів на основі їх колишніх переваг і переваг схожих користувачів. У таких задачах застосовують елементи класифікації, кластеризації та доповнення відсутніх даних.

### 3.11. ЗАДАЧІ З ОПИСОМ ОБ'ЄКТІВ НА ОСНОВІ ОЗНАК

Ознакою (feature) називають відображення  $f: X \rightarrow D_f$ , яке описує результат вимірювання деякої характеристики об'єкту, де  $D_f$  – задана множина.

Залежно від множини допустимих значень  $D_f$  ознаки поділяють на такі типи:

- ◆ бінарна ознака  $D_f = \{0,1\}$ ;
- ◆ номінальна ознака  $D_f$  – скінченна множина;
- ◆ порядкова ознака – скінченна впорядкована множина;
- ◆ кількісна ознака.

У прикладних задачах зустрічаються і більш складні випадки. Значеннями ознак можуть бути числові послідовності, растрові зображення, функції, графи, результати запитів до баз даних тощо.

Якщо всі ознаки мають однаковий тип, то початкові дані називають однорідними, інакше – різнорідними.

Нехай маємо множину ознак  $f_1, \dots, f_n$ . Якщо всі ознаки мають один тип, тобто  $D_{f_1} = \dots = D_{f_n}$ , то початкові дані називають *однорідними*, інакше – *різнорідними*. Вектор  $(f_1(x), \dots, f_n(x))$  називають *описом об'єкту*  $x \in X$  на основі ознак. У подальшому будемо отождествлювати об'єкт з його описом. Сукупність описів на основі ознак усіх об'єктів вибірки обсягу  $l$  з множини  $X$ , записану у вигляді таблиці розміру  $l \times n$  називають *матрицею об'єктів-ознак*.

$$F = [f_j(x_i)]_{l \times n} = \begin{pmatrix} f_1(x_1) & \dots & f_n(x_1) \\ \dots & \dots & \dots \\ f_1(x_l) & \dots & f_n(x_l) \end{pmatrix}. \quad (3.1)$$

Матриця об'єктів-ознак являє собою стандартний та найпоширеніший спосіб подання початкових даних. У той же час, на практиці зустрічаються задачі, у яких дані мають складнішу структуру, наприклад, описи об'єктів можуть мати змінну довжину. У таких випадках за наявними початковими даними обчислюють перетворені дані, які мають стандартний вигляд (3.1). Цей прийом називають *видобуванням ознак із даних* (feature extraction).

## 3.12. ПРИКЛАДИ ЗАДАЧ МАШИННОГО НАВЧАННЯ

У цьому підрозділі розглянуто приклади задач машинного навчання.

### 3.12.1. Задачі класифікації

**Приклад 3.7.** У задачах медичної діагностики об'єктами виступають пацієнти. Ознаки характеризують результати обстежень, симптоми захворювань та застосовані методи лікування. Прикладами бінарних ознак є стать, наявність головного болю, слабкості, нудоти тощо. Порядкові ознаки – важкість стану (задовільний, середня важкість, важкий, дуже важкий). Кількісні ознаки – вік, пульс, артеріальний тиск, уміст гемоглобіну в крові, дози препаратів тощо. Опис пацієнта за допомогою ознак є, фактично, формалізованою історією хвороби. Накопиченням достатньої кількості прецедентів можна розв'язувати різні задачі: класифікувати вид захворювання (диференційна діагностика); визначати найдоцільніший спосіб лікування; прогнозувати тривалість та результат захворювання; оцінювати ризик ускладнень; знаходити синдроми – найхарактерніші для даного захворювання сукупності симптомів. Цінність побудови відповідних систем на основі розв'язування задач класифікації полягає в тому, що вони здатні миттєво аналізувати та узагальнювати велику кількість прецедентів – можливість, яка недоступна людині.

**Приклад 3.8.** Задачі технічної діагностики виникають у випадку керування складними технічними комплексами. Наприклад, у разі моніторингу процесу буріння дуже важливо вчасно виявити передаварійну ситуацію, щоб зупинити процес та здійснити ремонт бурового інструменту. Тут ознаками є покази датчиків технологічних параметрів буріння: глибини свердловини, обертів ротора, тиску та витрат промивної рідини тощо. Об'єкти відповідають моментам часу, у які відбувається реєстрація показів датчиків. Складність такої задачі полягає в тому, що опис об'єктів (ситуацій) на основі ознак суттєво багатовимірний. Кількість різних типів несправностей також достатньо велика. У цих умовах навіть досвідченому оператору буває важко вчасно виявити та класифікувати причину несправності. Для автоматичного моніторингу можна використати навчальні алгоритми. Для їх застосування спочатку накопичується навчальна вибірка – ситуації, які виявлені та правильно класифіковані експертами, а лише потім за прецедентами відбувається навчання алгоритму розпізнавання.

**Приклад 3.9.** Задача передбачення властивостей кінцевої продукції за властивостями складових та параметрів технологічного процесу. Сучасні хімічні, металургійні, текстильні та інші виробництва є такими складними, що побудувати доволі точну математичну модель залежності якості продукції від множини факторів, які впливають, практично неможливо. У цьому випадку також застосовують навчання за прецедентами. Тут об'єктами є експериментальні партії продукції, які виготовлені у визначених технологічних умовах. Ознаками є параметри складових та технологічного процесу. Цільовими (тими, що передбачаються) ознаками є показники якості продукції. Це можуть бути якісні показники (тоді мова йде про задачу класифікації) або кількісні (тоді вирішується регресійна задача). Знайдені залежності в подальшому можна використати для керування технологічним процесом.

Близькі задачі планування експериментів та активного навчання. Випуск експериментальної партії продукції, як правило, є дорогою справою. Потрібно так спланувати послідовність експериментів, щоб знайти оптимальну сукупність

технологічних параметрів та витратити якомога менше коштів на експерименти. У наших термінах це означає, що формування кожного наступного об'єкту навчання відбувається з урахуванням результатів, отриманих на попередніх етапах.

**Приклад 3.10.** У задачах пошуку родовищ корисних копалин, якщо їх розглядати як задачі класифікації, ознаками є дані геологічної розвідки. Наявність або відсутність певних порід на території району кодується бінарними ознаками. Фізико-хімічні властивості цих порід можна описувати як кількісними, так і якісними ознаками. Навчальна вибірка складається із прецедентів двох класів: районів відомих родовищ та районів, у яких шукані копалини не були виявлені. У разі пошуку рідкісних копалин кількість об'єктів може виявитись значно меншою від кількості ознак. У цій ситуації класичні статистичні методи не працюють, і задачу розв'язують шляхом пошуку логічних залежностей, які містять масиви даних. Під час розв'язування задачі виділяють короткі множини ознак, які володіють найбільшою інформативністю, здатністю найкраще поділяти класи. За аналогією з медичною діагностикою, можна сказати, що відшукують „синдроми” родовищ.

**Приклад 3.11.** У задачах автоматичного розпізнавання спаму потрібно поділити потік вхідних текстових повідомлень чи листів електронної пошти на два класи — звичайні та небажані, тобто, спам. Навчальними об'єктами є такі повідомлення, які самостійно класифікував конкретний користувач. На відміну від попередніх задач тут не існує хорошого опису на основі ознак — ці ознаки треба винаходити. Прикладом бінарної ознаки може бути наявність певного ключового слова або речення в тексті листа, наприклад, „цілком безкоштовно” „віагра”; номінальної ознаки — країна-відправник; кількісної ознаки — розмір листа в байтах, розмір вкладення в байтах, кількість адресатів у розсилці, кількість уже отриманих листів від цього відправника, кількість повторень певного символу тощо. Складність такої класифікації полягає в тому, що система має сама генерувати ознаки без багажу загальних знань, завдяки якому людина легко класифікує листи та текстові документи. Наразі системи виявлення спаму працюють недостатньо точно та допускають забагато помилок. Див. також приклад 4.29 із розділу 4.

**Приклад 3.12.** Схожою, але складнішою задачею є задача рубрикації текстів. Вона виникає у випадку роботи з великою кількістю текстових документів, особливо коли їх накопичує та використовує великий колектив людей, наприклад, редакція журналу або інформаційна агенція. У деяких випадках від середнього часу пошуку документу залежить робота всієї компанії.

Системи рубрикації проходили у своєму розвитку кілька етапів. Якщо документів небагато, то можна пам'ятати, де шукати потрібний матеріал. У певний момент кількість документів перевищує критичну масу, і їх розкладають за рубриками. Потім відбувається усвідомлення того, що кожному співробітнику потрібна своя рубрикація, і, можливо, не єдина. У результаті створюється спільне сховище документів, яке має багато персональних рубрикаторів. У цьому випадку виникає потреба автоматизувати процес розкладання та пошуку документів.

Автоматичне розміщення нового документу в потрібну рубрику є задачею класифікації з величезною кількістю класів-рубрик, які можуть перетинатись. Навчальний прецедент створюється користувачем у той момент, коли він поміщає документ у потрібну рубрику. Алгоритм класифікації має врахувати логіку користувача, і в подальшому класифікувати тексти за цим же принципом. Зокрема, він має знайти

критерії, за якими користувач вважає документи подібними. Від користувача не вимагають у явному вигляді формулювати ці принципи. Більш того, він може користуватись ними інтуїтивно, навіть не підозрюючи про їх існування.

Одним із можливих підходів тут є порівняння текстів за складом ключових слів. Окремі слова та сталі словосполучення вважають ключовими, якщо їх зустрічають у певеликій підмножині документів і дуже рідко — у всіх інших. Зазвичай, це спеціальні терміни, власні імена, географічні назви тощо. Документи вважають подібними, якщо множини їх ключових слів суттєво перетинаються. Підмножини ключових слів вважають стійкими, якщо їх зустрічають у великій кількості документів. Ці документи вважають подібними, і вони утворюють окрему тематичну рубрику.

Принцип подібності можна використовувати й для пошуку документів. Типова ситуація, коли користувач не цілком знає, що саме він хоче знайти. Часто запит виглядає так: „що ще відомо на цю тему?” або „знайти всі документи, подібні до заданого”. Це також задача класифікації із двома класами: „подібні документи” та „решта документів”, і її теж можна розв'язувати оцінюванням близькості текстів.

**Приклад 3.13.** Задача розпізнавання окремих рукописних символів розв'язується під час автоматичної обробки анкет, заповнених від руки за спеціальною формою: бланків перепису населення, податкових декларацій тощо. Це задача класифікації з кількістю класів, що дорівнює кількості допустимих символів: букв, цифр та знаків пунктуації. Спочатку скановане растрове зображення символу — це матриця розмірів, наприклад,  $50 \times 50$ , яка складається із чорних та білих пікселів. „Сирий” опис на основі 2500 бінарних ознак не дуже зручний для класифікації. Один із підходів до виявлення ознак полягає в розбитті зображення на певну кількість областей (можливо таких, що перетинаються) та підрахуванні частки чорних пікселів у кожній області. Такі ознаки більш інформативні й уже дають змогу робити певні висновки. Наприклад, якщо чорні піксели групуються по горизонталі зверху, а по вертикалі в центрі, то символ є схожим на букву „Г”. Інший підхід полягає в побудові за растром контуру зображення у вигляді ламаної лінії. Тоді ознаками є її характеристики, наприклад, положення, нахил та довжина кожної ланки. Побудова таких ознак вимагає проведення великої експериментальної роботи з відбору ознак.

**Приклад 3.14.** Задачі ідентифікації підписів є двох типів: off-line та on-line. У випадку off-line підпис вводять у комп'ютер за допомогою сканера. Підпис є растровим зображенням, і тому ця задача подібна до попередньої. У випадку on-line людина вводить підпис за допомогою електронного пера. Цю задачу вважають простішою в порівнянні з off-line, бо крім зображення підпису в розпорядженні системи розпізнавання є додаткова інформація про послідовність та швидкість руху пера. Отже, об'єкти (підписи) описані не растровим зображенням, а парою функцій часу  $x(t)$  та  $y(t)$ , які є координатами пера. У цій задачі доцільно взагалі не переходити до ознак, а порівнювати підписи безпосередньо як часові ряди. Задача зводиться до побудови такої міри подібності, у якій усі підписи, що належать одній людині, є подібними, а підписи різних людей суттєво відрізняються.

**Приклад 3.15.** Задача розпізнавання розмовної мови є, можливо, однією із найскладніших серед задач класифікації. Незважаючи на значні зусилля, які докладають багато лабораторій у всьому світі, наразі не вдається побудувати алгоритм, який здатний розпізнавати людську мову із прийнятним для комерційних застосувань рівнем помилок.



Складність задачі полягає в тому, що окремі звукові сигнали можуть суттєво змінюватись в залежності від положення звуку в слові, слова в реченні, зовнішніх шумів та власної мови диктора. Успішне розпізнавання мови практично неможливе без використання знань із декількох суміжних областей: фізики (акустики), лінгвістики (фонетики), теорії класифікації, теорії марковських ланцюгів.

На відміну від перетворення розмовної мови в текст, розпізнавання голосових команд із заданого списку або розпізнавання ключових слів у мові є відносно нескладними задачами. Загалом задачі класифікації суттєво спрощуються у випадку зменшення кількості класів.

**Приклад 3.16.** Задачі оцінювання позичальників розв'язують банки під час видачі кредитів. Потреба в автоматизації процедури видачі кредитів уперше виникла в період буму кредитних карт 60-70-х років у США та інших розвинутих країнах. Об'єктами в цьому випадку є позичальники – фізичні або юридичні особи, які претендують на отримання кредиту. У разі фізичних осіб опис ознак складається з анкети, яку заповнює сам позичальник, і, можливо, додаткової інформації, яку банк збирає про нього із власних джерел. Приклади бінарних ознак: стать, наявність телефону. Номінальні ознаки – місце проживання, професія, роботодавець. Порядкові ознаки – освіта, займана посада. Кількісні ознаки – вік, стаж роботи, дохід сім'ї, розмір заборгованостей в інших банках, сума кредиту. Навчальна вибірка складається з позичальників із відомою кредитною історією. У найпростішому випадку прийняття рішень зводиться до класифікації позичальників на два класи: „хороших” та „поганих”. Кредити видають позичальникам лише першого класу. У більш складних випадках оцінюють сумарну кількість балів (score) позичальника, яку він набрав за сукупністю інформаційних ознак. Що вище оцінка, то надійнішим вважають позичальника. Звідси й назва – кредитний скорінг (credit scoring). На стадії навчання здійснюють синтез та відбір інформаційних ознак та визначають, скільки балів призначити за кожну ознаку, щоб ризик прийнятих рішень був мінімальним. Наступна задача – вирішити, на яких умовах видавати кредит: визначити процентну ставку, термін погашення, та інші параметри кредитної угоди. Ця задача також зводиться до навчання за прецедентами.

**Приклад 3.17.** Задача оцінювання привабливості інвестиційних проектів подібна до задачі оцінювання позичальників. З нею зустрічаються інвестиційні компанії, банки, венчурні фонди, крупні корпорації, які фінансують дослідницькі, інноваційні та ризиковані бізнес-проекти. Об'єктами є заявки на проекти, які постачають у вигляді анкет. Типова кількість пунктів анкети біля сотні, кількість анкет – сотні або тисячі. Основна складність такої задачі – низька формалізованість анкет. Найважливіша з точки зору експертів інформація знаходиться, як правило, у декількох текстових полях, які змістовно описують суть проекту. Зрозуміло, що повна автоматизація процесу розгляду заявок неможлива. Однак, можна розв'язувати задачу попередньої класифікації заявок на „хороші”, що будуть детально вивчатись експертами, та „погані” – які з високою ймовірністю не пройдуть конкурсний відбір. Погані заявки можна відсіяти попередньо та повністю автоматично, що значно скоротить завантаженість експертів. Навчальна вибірка складається з раніше класифікованих анкет, наприклад, заявок минулих років. Методи розв'язування подібні до методів оцінювання позичальників. Як і в інших задачах аналізу текстів (виявлення спаму, рубрикації), найскладнішим є етап попереднього опрацювання даних, у ході якого створюють числові ознаки заявки. Ознаками можуть

бути як початкові характеристики заявки: термін виконання проекту, обсяги потрібних інвестицій, прибутковість, рентабельність, кількість виконавців, галузь тощо, а також обчислені характеристики, які відображають якість заповнення анкети: обсяг анотації та інших текстових полів, ступінь заповнення полів, відповідність анкети формальним вимогам тощо.

**Приклад 3.18.** Задача передбачення втрати клієнтів (churn prediction) випливає в крупних та середніх компаній, які працюють із великою кількістю клієнтів, як правило, з фізичними особами. Особливо актуальна ця задача для сучасних телекомунікаційних компаній. Коли ринок перебуває в стані, близькому до насичення, основні зусилля компанії скеровують не на залучення нових клієнтів, а на втримання старих. Для цього необхідно якнайскоріше віділити клас клієнтів, схильних до відходу в найближчий час. Класифікація проводиться на основі інформації, яку зберігають у компанії: клієнтські анкети, дані про частоту користування послугам компанії, склад послуг, тарифні плани, регулярність платежів тощо. Найінформативнішими є дані про те, що саме змінилось в поведінці клієнта за останній час. Тому об'єктами, строго говорячи, є не самі клієнти, а пари „клієнт  $x$  у момент  $t$ ”. Необхідно передбачити, чи піде клієнт до моменту  $t + \Delta t$ . Навчальна вибірка формується із клієнтів, про яких точно відомо, коли вони пішли. Цю задачу характеризує велика кількість помилок, оскільки передбачувати поведінку людей є непростю справою. Задачу формулюють прагматично – мінімізувати сумарні втрати компанії. Для цього виділяють відносно невелику цільову групу клієнтів із найбільшою ймовірністю відходу та найбільшою ймовірністю того, що клієнт змінить своє рішення після надання йому знижки або додаткових послуг. Ураховують також співвідношення вартості маркетингового впливу та перспективного прибутку від клієнта.

### 3.12.2. Задача відновлення регресії

**Приклад 3.18.** Термін „регресія” було введено 1886 р. антропологом Френсісом Гальтоном під час вивчення статистичних закономірностей спадковості зросту. Повсякденний досвід свідчить, що в середньому зріст дорослих дітей то більший, що вищими є їхні батьки. Однак Гальтон виявив, що часто сини дуже високого батька мають не дуже високий зріст. Він зібрав вибірку з 928 пар „батько-син”. Кількісну залежність непогано описувала лінійна функція  $y = 2x/3$ , де  $x$  – відхилення зросту батька від середнього, а  $y$  – відхилення зросту сина від середнього. Гальтон назвав це явище „регресією до середнього”, тобто до середнього значення в популяції. Термін „регресія”, рух назад, натякав також на нестандартний для того часу хід дослідження. Спочатку були зібрані дані, а потім по них „угадана” модель залежності, тоді як традиційно робили навпаки: дані використовували лише для підтвердження теоретичних моделей. Це був один із перших випадків застосування інформаційного моделювання, оснований виключно на аналізі даних. Пізніше термін, який виник у частковій прикладній задачі, закріпився за широким класом методів відновлення залежностей.

### 3.12.3. Задачі прогнозування та прийняття рішень

**Приклад 3.19.** Задачу прогнозування споживчого попиту розв'язують сучасні супермаркети та торговельні роздрібні мережі. Для ефективного керування торговельною мережею потрібно спрогнозувати обсяги продажів для кожного товару на задану



кількість днів наперед. На основі цих прогнозів здійснюється планування закупівель, керування асортиментом, формування цінової політики, планування рекламних кампаній. Специфіка задачі полягає в тому, що кількість товарів може вимірюватись десятками, а може й сотнями тисяч. Прогнозування та прийняття рішень по кожному товару „вручну” просто неможливе. Початковими даними для прогнозування є часові послідовності цін та обсягів продажів по товарах та окремих магазинах. Сучасні технології дають змогу отримувати ці дані безпосередньо з касових апаратів. Для збільшення точності прогнозу треба також урахувувати різні зовнішні чинники, які впливають на споживчий попит: рівень інфляції, погодні умови, рекламні кампанії, соціально-демографічні умови, активність конкурентів. У залежності від цілей аналізу об'єктами виступають або товари, або магазини, або пари „магазин-товар”. Ще одна особливість задачі – несиметричність функції втрат. Якщо прогноз роблять із метою планування закупівель, то втрати від заниженого прогнозу суттєво вищі ніж від завищеного.

**Приклад 3.20.** Прийняття інвестиційних рішень на фінансовому ринку – це задача, у якій уміння добре прогнозувати безпосередньо перетворюється на прибуток. Якщо інвестор припускає, що ціна акції виросте, то він купує акції з надією продати їх пізніше за вищою ціною. І, навпаки, прогножуючи падіння цін, інвестор продає акції, щоб у майбутньому викупити їх назад за нижчою ціною. Задача інвестора-спекулянта полягає в тому, щоб правильно передбачити напрямок майбутньої зміни ціни – зростання або спадання. У цьому випадку інвестор сильно ризикує: відомо, що на фінансовому ринку з позитивним прибутком залишаються лише 35% гравців. Тим не менше, інвестори готові ризикувати з надією на високий прибуток. Прийнято вважати, що разом вони виконують важливу економічну функцію – забезпечують ліквідність ринку, тобто можливість у будь-який момент купити або продати доволі великий пакет акцій.

Дуже популярні алгоритми, які приймають торгові рішення без участі людини. Розробка такого алгоритму – це задача навчання за прецедентами. Об'єктами є моменти часу, а опис об'єкту – це передісторія змін цін та обсягів торгів, зафіксованих у певні моменти. У найпростішому випадку об'єкти необхідно класифікувати на три класи, які відповідають можливим рішенням: купити, продати або вичікувати. Навчальною вибіркою для вироблення торговельних стратегій є історичні дані про рух цін та обсягів за певний проміжок часу. Критерій якості в цьому випадку суттєво відрізняється від стандартного функціонала середньої похибки прогнозів або класифікацій, оскільки інвестора цікавить не точність прогнозування, а максимізація кінцевого прибутку. Сучасний біржовий технічний аналіз нараховує сотні параметричних торговельних стратегій, які прийнято налагоджувати за критерієм максимуму прибутку на вибраному проміжку історії.

Біржова торгівля має й зворотний бік. Задачі навчання за прецедентами виникають також і в самій біржі як організатора торгів, головним чином, у разі проведення біржового нагляду (market surveillance). Мета нагляду – підтримання справедливого та ефективного ринку шляхом виявлення та розслідування навмисного маніпулювання цінами та інсайдерської торгівлі. Налагоджена служба нагляду, яка періодично супроводжується „показовими процесами”, значно підвищує довіру до біржі в основній масі рядових інвесторів. У світовій практиці відомі випадки, коли введення нагляду дало змогу значно збільшити обсяги торгів.

**Приклад 3.21.** Задача виявлення інтервенцій на фінансовому ринку та оцінювання їх впливу на ціну. Практично жодна біржова маніпуляція не буває без фінансових інтервенцій. Інтервенція – це коли один з учасників торгів (або кілька учасників за змовою) проводять короткочасні односторонні операції на ринку: або купує крупний пакет, що призводить до зростання ціни, або продає крупний пакет, що призводить до падіння ціни. Інтервенцію можна проводити по-різному. „М'яка” інтервенція власне такою не є, оскільки її мета – дійсно багато купити або продати, по можливості, не здійснюючи тиску на ринок. При цьому трейдери „акуратно” розподіляють свої операції в часі, діють багатократно малими партіями, даючи можливість після кожної операції повернутись ринку в стан рівноваги. Такі інтервенції не є протизаконними або підозрілими. „Жорстка” інтервенція, навпаки, переслідує мету максимально зрушити ціну в напрямку, вигідному маніпулятору, витрачаючи на це мінімальну кількість грошей. Як правило, вона проводиться доволі швидко в добре підібраний момент часу, коли ринок найслабший і є частиною наперед спланованого сценарію. Якщо маніпуляторам вдасться реалізувати свій сценарій, рядові інвестори несуть значні збитки. Маніпуляції не вигідні ринку в цілому і їх треба переслідувати за законом.

Для автоматичного виявлення фінансових інтервенцій у потоці біржових даних потрібні строгі формальні критерії. Вироблення критерію – це типова задача навчання за прецедентами. Тут об'єктами є доволі короткі проміжки часу, на протязі яких певний учасник торгів зробив значний обсяг односторонніх операцій. Опис ознак об'єкту складається приблизно з десяти величин, які характеризують цей інтервал та дії підозрюваного учасника. Зокрема: сальдо операцій учасника (тобто, різниця обсягів його покупок та продажів), частина учасника в торговельному обороті, розподіл операцій учасника в часі, отриманий учасником прибуток та інші. Цільовою величиною є стрибок цін, обчислений як різниця середніх цін після інтервенції й до неї.

Пошук залежності стрибка ціни від опису ознак інтервенції – це типова задача відновлення регресії. Якщо залежність буде знайдено, то наявність інтервенції на визначеному проміжку часу можна обґрунтувати статистично, наприклад, так: „у 95% аналогічних інтервенцій стрибок ціни помітно відрізнявся від нуля й складав  $(17 \pm 10)$  пунктів”. Регресія дає змогу кількісно оцінити, як сильно дії підозрюваного учасника вплинули на ціну, та вказати довірчий інтервал цього впливу.

**Приклад 3.22.** Задача нестандартної поведінки учасників торгів багато в чому подібна до задачі виявлення інтервенцій. Відмінність у тому, що критерії нестандартності в цьому випадку формують експерти наперед, у вигляді простих зрозумілих правил. Наприклад: „операція є перехресною (сама з собою), і її обсяг перевищує  $\epsilon_1$  % від середнього денного обігу, її ціна відхиляється від середньої на  $\epsilon_2$  % або більше”. Задача підбору порогів  $\epsilon_i$  – це типова задача класифікації. Навчальна вибірка складається з перевірених випадків стандартної (клас 1) та нестандартної (клас 0) поведінки. Крім того, накладають додаткові обмеження на середню кількість сигналів, які видають на протязі торговельного дня. Їх не має бути занадто багато, щоб експерти служби нагляду мали фізичну можливість вручну дослідити знайдені випадки. Прийнято вважати, що в цій задачі повна автоматизація недоцільна, оскільки дослідження кожного випадку вимагає залучення неформальних знань експертів.

### 3.12.4. Задачі кластеризації

Задача кластеризація, або навчання без учителя, відрізняється від задач класифікації тим, що в них не задають відповіді  $y_i = y^*(x_i)$ . Відомі лише об'єкти  $x_i$ , і треба розбити вибірку на підмножини, що не перетинаються (кластери) так, щоб кожний кластер складався з подібних об'єктів, а об'єкти різних кластерів суттєво відрізнялись. Кількість кластерів може бути відомою наперед, але найчастіше її треба визначити.

**Приклад 3.23.** Основним інструментом проведення соціологічних та маркетингових досліджень є проведення опитувань. Щоб результати опитування були об'єктивними, необхідно забезпечити, щоб вибірка респондентів була представницькою. З іншого боку, необхідно мінімізувати вартість проведення опитування. Тому в разі планування опитувань виникає така задача: відібрати якомога менше респондентів, щоб вони утворювали репрезентативну вибірку, тобто подавали весь спектр громадської думки. Наприклад, у разі формування множини точок опитування (це можуть бути міста, райони, магазини тощо) спочатку складають описи ознак доволі великої кількості точок. Це можна зробити шляхом використання недорогих способів збирання інформації – пробних опитувань та фіксації характеристик точок. Потім розв'язують задачу кластеризації й із кожного кластера вибирають по одній представницькій точці. Тільки у відібраній множині точок проводять основне, найбільш трудомістке, опитування.

**Приклад 3.24.** Задача тематичної кластеризації новин. Потoki новин у мережі Інтернет стали такими великими, що орієнтуватись та оцінювати інформацію, яка надходить, стає дуже важко, а часто практично неможливо. Одна новина може породжувати ланцюжок новин, які значною мірою дублюють одна одну. Тому поряд зі знаходженням усіх подібних новин виникає задача визначення першого повідомлення, визначення зв'язків між повідомленнями, і, зрештою, відслідковування всього ланцюжка подій. Виділений кластер повідомлення замінюють не одним типовим представником, а доволі складною деревовидною структурою, яка описує сценарій розвитку подій. Цей опис має бути одночасно повним та не надлишковим.

### 3.12.5. Задачі аналізу клієнтських середовищ

Клієнтське середовище – це сукупність клієнтів певної компанії або сервісу, які регулярно користуються фіксованим набором послуг або ресурсів. Клієнтські середовища виникають у різних сферах бізнесу й не тільки бізнесу. Можна говорити про клієнтські середовища торгівельних мереж, операторів зв'язку, емітентів пластикових карт, бібліотек, електронних магазинів, інтернет-порталів тощо.

Сучасні інформаційні технології дають змогу детально протоколювати дії клієнтів у електронному вигляді. Ці протоколи містять цінну інформацію, яка необхідна для розв'язування широкого спектру задач, об'єднаних модною назвою *керування взаємостосунками із клієнтом* (Customer Relationship management, CRM). Мається на увазі виявлення цільових груп клієнтів, персоналізація роботи із клієнтами, зокрема, Розв'язування персональних пропозицій, прогнозування можливого відтоку клієнтів (churn prediction), виявлення шахрайств (fraud detection) або незвичної поведінки клієнтів. Вирішення цих та інших аналітичних задач скероване, зрештою, на підвищення якості послуг, що надаються, автоматизації зворотного зв'язку із клієнтами, ефективнішого залучення та втримання клієнтів.

**Приклад 3.25.** Задача передбачення рейтингів розв'язується в інтернет-магазинах, особливо книжкових, та в мережах відеопрокату. Якщо клієнт купує через сайт певний предмет (книгу, фільм тощо), то він має можливість висловити своє ставлення до нього виставленням рейтингу, зазвичай цілого числа в діапазоні від 1 до 5. Система використовує інформацію про всі виставлені рейтинги для персоналізації пропозицій. Коли клієнт бачить на сайті сторінку з описом предмету, йому показують також список усіх схожих предметів, які отримали високий рейтинг у схожих клієнтів. Головна задача полягає в тому, щоб швидко знайти схожих клієнтів та схожі предмети у великому обсязі даних, а потім прогнозувати їх рейтинги для даного клієнта. Формально задачу можна сформулювати і як задачу класифікації. Роль матриці об'єктів-ознак відіграє матриця клієнтів-предметів, яка заповнена значеннями рейтингів. Як правило, вона сильно розріджена та має до 90% порожніх комірок. Фіксованої цільової ознаки в цій задачі немає. Алгоритм класифікації має вміти передбачити рейтинг для довільної незаповненої комірки матриці.

Про складність та актуальність цієї задачі свідчить такий факт. У жовтні 2006 року найкрупніша американська компанія Netflix, яка займається відеопрокатом через Інтернет, оголосила міжнародний конкурс із призом в один мільйон доларів тому, хто зможе на 10% покращити точність прогнозування рейтингів, порівняно із системою Cinematch, яку експлуатує Netflix. Зазначимо, що прогнози Cinematch лише на 10% точніші елементарних середніх рейтингів фільмів. Компанія надзвичайно зацікавлена в збільшенні точності прогнозів, оскільки більш ніж 70% замовлень отримує через рекомендувану систему.

**Приклад 3.26.** Задача персоналізації пошуку. Не є секретом, що Інтернет у його сучасному стані часто називають „інформаційним звалищем”. Інформації так багато, що пошукові машини вимагають надзвичайно точно формулювати критерій пошуку. На жаль, звузити запит можна десятком різних способів, про деякі з яких користувач може й не знати. Хотілося б, щоб система сама впізнавала користувача за його минулою поведінкою та ранжувала результати пошуку персонально для нього, орієнтуючись на коло його звичних інтересів. Наприклад, різні користувачі можуть зробити запит „chip prediction”, однак фахівця з IT-департаменту цікавитимуть питання запровадження існуючих на ринку систем, ученого – алгоритми класифікації із застосуванням у бізнес-аналітиці, менеджера із продажів – презентації компаній-конкурентів, а студента – реферати та огляди на цю тему. Можливе вирішення проблеми полягає в ідеї персоналізації. Результати пошуку сортують так, щоб на початку списку опинились документи, схожі на ті, які дивились схожі користувачі. Найскладніше в цій задачі – побудова адекватної міри подібності. Користувачів можна вважати схожими, якщо вони проглядають схожі документи. Документи є схожими, якщо їх проглядають схожі користувачі.

## 3.13. НАВЧАННЯ ПОНЯТЬ В ШТУЧНОМУ ІНТЕЛЕКТІ

Ніколи, властиво, не було, й, напевно, у найближчому майбутньому також не буде загальної згоди щодо того, чим займається штучний інтелект та які ставить перед собою цілі. Якщо всі, прихильники й супротивники, погодяться з тим, що основною його метою є конструювання штучних інтелектуальних систем, то можна окреслити принаймні такі чотири головні елементи в розумінні того, якою є або якою має бути інтелектуальна система:

- ◆ система, яка мислить як людина;
- ◆ система, яка мислить раціонально;
- ◆ система, яка поводить себе як людина;
- ◆ система, яка поводить себе раціонально.

Спрощено можна сказати, що фахівцям, які займаються штучним інтелектом, ближче положення про поведінку та раціональність, а психологам і філософам про мислення й наслідування людині. Кожне з наведених положень може бути предметом дискусії: є складним питання однозначного визначення того, які раціональні критерії можна застосовувати щодо поведінки систем, а більше того, що означає мислити і як можна це перевірити. Також не дуже зрозумілим є те, які аспекти поведінки людини можуть і мають наслідувати штучні системи.

Доволі часто штучний інтелект поділяють лише на два види – сильний і слабкий. Сильний штучний інтелект ставить на меті створення штучних систем, що мислять на такому інтелектуальному рівні, як і людина, або на ще вищому.

Слабкий штучний інтелект займається досить широким колом часткових проблем з добре визначеними цілями й критеріями їх досягнення. Проблеми ці стосуються виконання інформаційними системами задач, складність яких вимагає від них інтелекту, якщо їх виконує людина. Розв'язок цих проблем може бути корисним у сильному штучному інтелекті, але поштовхом над праці над ними є швидше їх практичне застосування. Можна казати, що слабкий штучний інтелект є предметом зацікавлення інформатиків, а сильний штучний інтелект – філософів.

У подальшому не будемо схилитись до жодного з поглядів у суперечці, чи досяжні цілі сильного штучного інтелекту. Немає сумніву, що вдалось розв'язати багато задач слабого штучного інтелекту та можливий подальший поступ у цій галузі. Говорячи про штучний інтелект, будемо маги на увазі слабкий штучний інтелект, а машинне навчання трактувати як один із його найважливіших і найперспективніших розділів.

### 3.13.1. Задача навчання понять – пошук у просторі гіпотез

Задача побудови функцій на основі конкретних навчальних прикладів є центральною в навчанні. Розглянемо навчання понять як знаходження функції за множиною позитивних і негативних навчальних прикладів. *Навчання понять* можна сформулювати як пошук гіпотези, яка найкраще відповідає навчальним прикладам серед усіх можливих гіпотез. У багатьох випадках цей пошук можна організувати шляхом упорядкування гіпотез від загальної до конкретної. Розглянемо кілька алгоритмів навчання для отримання наближення до правильної гіпотези. Також розглянемо природу індуктивного навчання та умови, за яких будь-яка програма може успішно робити узагальнення поза наведеними навчальними даними.

Навчання передбачає отримання загальних понять із навчальних прикладів. Люди постійно вивчають загальні поняття, наприклад, „птахи”, „автомобіль”, „ситуації, у яких я повинен учитись, щоб скласти іспит” тощо. Кожне таке поняття можна розглядати як опис певної підмножини об'єктів чи подій, визначених на більш загальній множині. Прикладом може бути підмножина птахів у множині тварин. З іншого боку, кожне поняття можна розглядати як функцію зі значеннями  $\{0, 1\}$ , визначену на цій більшій множині. Наприклад, ця функція може бути визначена на всій множині тварин, а її значення бути істинним для птахів і хибним для інших тварин.

Розглядатимемо також проблему автоматичного виведення певного поняття на основі прикладів, які відповідають або не відповідають заданому поняттю. Цю задачу зазвичай формулюють як задачу навчання поняття, або апроксимацію зазначеної вище функції на прикладах.

**Приклад 3.27.** Розглянемо задачу навчання поняття „Дні, у які мій друг Михайло насолоджується улюбленим водним спортом” (далі *НасСпортом*). Приклад формулювання задачі навчання поняття *НасСпортом* у загальному вигляді подано в табл. 3.1. У цій таблиці наведено перелік днів, у які Михайло мав можливість займатись спортом, а обставини, які спричинили таку можливість, подано множиною *атрибутів* – назв стовпців таблиці. Атрибут *НасСпортом* указує, чи насолоджується Михайло його улюбленим водним спортом у відповідний день. Завдання навчання цільового поняття полягатиме в тому, щоб навчитися „передбачати” значення атрибуту *НасСпортом* для будь-якого іншого дня на основі значень атрибутів цього дня. Отже, задача полягає в побудові гіпотез, які будуть використані для навчання.

Почнемо з розгляду простого подання гіпотези, у якому кожна гіпотеза є кон'юнкцією значень атрибутів прикладів. Кожна гіпотеза зображатиметься вектором, який містить значення шести атрибутів: *Небо*, *ТемпПовітря*, *Вологість*, *Вітер*, *Вода* і *Прогноз*.

Для кожного атрибута обмеженням (або умовою) гіпотези вважатимемо:

- ◆ „?” – якщо атрибут допускає будь-яке значення;
- ◆ „ $\emptyset$ ” – якщо жодне значення атрибута недопустиме;
- ◆ задане значення атрибута, наприклад, *Тепла*.

Якщо якийсь приклад  $x$  задовольняє всі обмеження гіпотези  $h$ , то  $h$  класифікує  $x$  як позитивний приклад ( $h(x) = 1$ ).

**Приклад 3.28.** Подамо як  $\langle ?, \text{Холодна}, \text{Висока}, ?, ?, ? \rangle$  гіпотезу про те, що Михайло насолоджується спортом тільки в холодні дні з високою вологістю. Ця гіпотеза не залежить від значень інших атрибутів.

Таблиця 3.1

| День | Небо    | ТемпПовітря | Вологість | Вітер   | Вода    | Прогноз   | НасСпортом |
|------|---------|-------------|-----------|---------|---------|-----------|------------|
| 1    | Сонячно | Тепла       | Нормальна | Сильний | Тепла   | Той самий | Так        |
| 2    | Сонячно | Тепла       | Висока    | Сильний | Тепла   | Той самий | Так        |
| 3    | Дощ     | Холодна     | Висока    | Сильний | Тепла   | Зміна     | Ні         |
| 4    | Сонячно | Тепла       | Висока    | Сильний | Холодна | Зміна     | Так        |

Гіпотеза  $\langle ?, ?, ?, ?, ?, ? \rangle$  – найзагальніша і, відповідно до неї, кожний день позитивний. Гіпотеза  $\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$  – найконкретніша і, відповідно до неї, жоден день не є позитивним. Тоді задача навчання поняття *НасСпортом* вимагає знаходження множини днів, для яких *НасСпортом* = так. Цю множину можна описати кон'юнкцією обмежень на атрибути такого дня. Отже, будь-яку задачу навчання поняття можна задати цільовою функцією, множиною випадків на яких вона визначена, множиною можливих гіпотез, що розглядаються учнем, і множиною відомих прикладів навчання.

Під час обговорення проблем навчання понять використаємо таку термінологію. Множину елементів, для яких поняття визначено, назовемо множиною *прикладів* та позначатимемо її  $X$ . У табл. 3.1  $X$  – це множина днів, поданих атрибутами *Небо*, *ТемпПовітря*, *Вологість*, *Вітер*, *Вода* і *Прогноз*. Поняття чи функцію, яку навчають, називають *цільовим поняттям* та позначають  $c$ . Узагалі,  $c$  може бути функцією, зі значеннями  $\{0, 1\}$ , визначеною на прикладах  $X$ , тобто  $c: X \rightarrow \{0, 1\}$ . Для заданого прикладу цільове поняття є значенням атрибуту *НасСпортом*, тобто,  $c(x) = 1$  для *НасСпортом* = Так та  $c(x) = 0$  для *НасСпортом* = Ні.

У разі навчання цільового поняття учню надають набір *навчальних прикладів*, кожний з яких є прикладом  $x$  із  $X$  та має значення цільового поняття  $c(x)$ . Приклади, у яких  $c(x) = 1$ , називають *позитивними прикладами* або членами цільового поняття. Приклади, у яких  $c(x) = 0$ , називають *негативними прикладами* цільового поняття. Упорядкована пара  $(x, c(x))$  описує навчальний приклад  $x$  і значення його цільового поняття  $c(x)$ , а символ  $D$  позначає множину всіх навчальних прикладів.

Для множини навчальних прикладів та цільового поняття  $c$  задача навчання полягає у формулюванні гіпотези про значення  $c$  та оцінювання цього значення. Символ  $H$  використаємо для позначення множини *всіх можливих гіпотез*, які розглядатимемо щодо цільового поняття. Як правило, подання гіпотез множини  $H$  визначає людина – проектувальник системи прийняття рішень. Узагалі, кожній гіпотезі з  $H$  ставиться у відповідність функція  $h: X \rightarrow \{0, 1\}$ . Метою навчання є знаходження гіпотези  $h$ , для якої  $h(x) = c(x)$  для всіх  $x$  із  $X$ .

Незважаючи на те, що задача навчання полягає у визначенні гіпотези  $h$ , ідентичної цільовому поняттю  $c$  на множині всіх прикладів  $X$ , єдиною доступною інформацією про  $c$  є його значення на навчальних прикладах. Тому індуктивні алгоритми навчання можуть гарантувати лише те, що отримана гіпотеза відповідатиме цільовому поняттю на навчальних даних. За відсутності будь-якої додаткової інформації, припущення полягає в тому, що найкращою гіпотезою для всіх прикладів є така гіпотеза, яка найкраще відповідає навчальним прикладам. Це припущення є фундаментальним в індуктивному навчанні.

Навчання понять можна розглядати як задачу пошуку в просторі гіпотез. Мета такого пошуку полягає в знаходженні такої гіпотези, яка найкраще відповідає навчальним прикладам. Важливо зазначити, що спосіб подання гіпотези неявно визначає простір усіх гіпотез, які програма може зобразити, і, отже, на яких навчатиметься.

**Приклад 3.29.** Розглянемо приклади  $X$  і гіпотези  $H$  у задачі навчання *НасСпортом*. З огляду на те, що атрибут *Небо* має три можливі значення, а кожний з атрибутів *ТемпПовітря*, *Вологість*, *Вітер*, *Вода* й *Прогноз* по два значення, то простір прикладів  $X$  міститиме точно  $3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 96$  різних прикладів. Аналогічні підрахунки показують, що існує  $5 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 5120$  синтаксично різних гіпотез в  $H$ . Такі гіпотези отримані з прикладів додаванням до доменів атрибутів значень „Ø” та „?”. Зазначимо, що кожна гіпотеза, яка містить принаймні один символ „Ø”, подає порожньою множиною прикладів, тобто класифікує кожний такий приклад як негативний. Отже, кількість семантично різних гіпотез дорівнює лише  $1 + (4 \cdot 3 \cdot 3 \cdot 3 \cdot 3) = 973$ .

Задача *НасСпортом* є дуже простою задачею навчання з відносно невеликим та скінченним простором гіпотез. Реальні задачі навчання мають набагато більший, іноді нескінченний, простір гіпотез.

Розглядатимемо навчання як пошук, а дослідження алгоритмів навчання продовжимо аналізом різних стратегій пошуку в просторі гіпотез. Нас будуть цікавити алгоритми ефективного пошуку в дуже великих чи нескінченних просторах для знаходження гіпотез, які найкраще відповідають навчальним прикладам.

### 3.13.2. Упорядкування гіпотез „від загальної до конкретної”

Багато алгоритмів навчання понять, організовують пошук у просторі гіпотез використовуючи структуру, яка застосовна для довільної проблеми навчання поняття – упорядкування гіпотез за принципом „від загальної до конкретної” (general-to-specific). Так можна проектувати алгоритми навчання, що здійснюють пошук навіть у нескінченних просторах гіпотез без явного перерахування кожної з них.

**Приклад 3.30.** Для ілюстрації впорядкування гіпотез „від загальної до конкретної” розглянемо дві гіпотези  $h_1 = \langle \text{Сонячно}, ?, ?, \text{Сильний}, ?, ? \rangle$ ,  $h_2 = \langle \text{Сонячно}, ?, ?, ?, ?, ? \rangle$  та приклади, які ці гіпотези класифікують як позитивні. Оскільки гіпотеза  $h_2$  накладає менше обмежень на приклади, то вона класифікує як позитивні більшу кількість прикладів. Фактично, будь-який приклад, класифікований гіпотезою  $h_1$  як позитивний, буде також класифікований як позитивний і гіпотезою  $h_2$ . Тому говорять, що *гіпотеза  $h_2$  загальніша ніж  $h_1$* .

Інтуїтивне відношення „загальніша” на множині гіпотез можна строго означити так: для довільного прикладу  $x$  з множини  $X$  і гіпотези  $h$  з множини  $H$  говорять, що  $x$  *задовольняє  $h$* , якщо  $h(x) = 1$ . Тепер визначимо відношення „загальніша або еквівалентна” в термінах множини прикладів, що задовольняють гіпотези  $h_j$  та  $h_k$ . Говорять, що гіпотеза  $h_j$  „загальніша або еквівалентна” гіпотези  $h_k$ , якщо будь-який приклад, який задовольняє гіпотезу  $h_k$ , також задовольняє гіпотезу  $h_j$ . Нехай  $h_j$  та  $h_k$  – функції зі значеннями  $\{0, 1\}$ , визначені на множині  $X$ . Тоді говорять, що гіпотеза  $h_j$  „загальніша або еквівалентна” гіпотези  $h_k$ , якщо  $(h_k(x) = 1) \rightarrow (h_j(x) = 1)$ ,  $\forall x \in X$ , та записують  $h_j \geq h_k$ . Корисними є випадки, коли одна гіпотеза строго загальніша іншої. Говорять, що  $h_j$  „строго загальніша”  $h_k$ , якщо  $(h_j \geq h_k) \wedge \neg(h_k \geq h_j)$ , та записують  $h_j > h_k$ . Нарешті, іноді будемо говорити, що гіпотеза  $h_j$  „конкретніша” ніж гіпотеза  $h_k$ , якщо гіпотеза  $h_k$  „загальніша” ніж гіпотеза  $h_j$ .

**Приклад 3.31.** Для ілюстрації введених понять розглянемо гіпотези  $h_1$ ,  $h_2$  та  $h_3$  з прикладу *НасСпортом*. Ці гіпотези зображено на рис. 3.1 Прямокутник ліворуч зображає множину  $X$  усіх прикладів, прямокутник праворуч – множину  $H$  усіх гіпотез. Кожній гіпотезі відповідає деяка підмножина прикладів множини  $X$ , які ця гіпотеза класифікує як позитивні.

Стрілки між гіпотезами зображають відношення „загальніша”. Стрілка напрямлена від загальнішої гіпотези до конкретнішої. Зазначимо, що підмножина прикладів, які відповідають гіпотезі  $h_2$  містить підмножину прикладів, які відповідають гіпотезі  $h_1$ , отже, гіпотеза  $h_2$  „загальніша” гіпотези  $h_1$ .

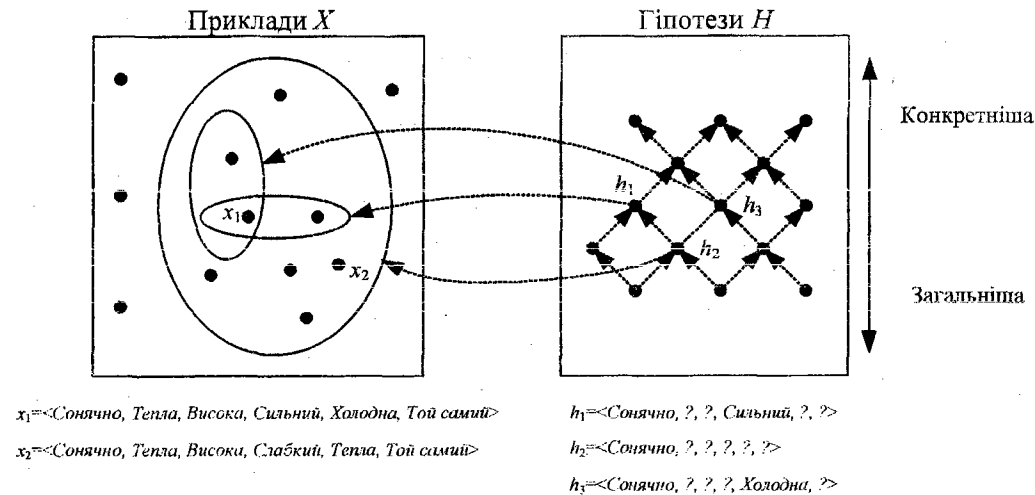


Рис. 3.1

Покажемо, як  $h_1$ ,  $h_2$  та  $h_3$  пов'язані відношенням „ $\geq$ ”. Гіпотеза  $h_2$  загальніша гіпотези  $h_1$ , оскільки кожний приклад, який задовольняє гіпотезу  $h_1$ , також задовольняє гіпотезу  $h_2$ . Аналогічно, гіпотеза  $h_2$  загальніша за гіпотезу  $h_3$ . Ані  $h_1$ , ані  $h_3$  не є взаємно загальнішими. Незважаючи на те, що приклади, які відповідають цим гіпотезам, мають непорожній перетин, жодна множина не містить іншої. Зазначимо, що відношення „ $\geq$ ” та „ $>$ ” не за означенням залежать від цільового поняття. Вони залежать лише від прикладів, які задовольняють ці гіпотези, і не залежать від класифікації цих прикладів цільовим поняттям. Відношення „ $\geq$ ” визначає частковий порядок у просторі гіпотез  $H$ , оскільки воно рефлексивне, антисиметричне та транзитивне. Якщо говоримо, що множина є частково впорядкованою, то припускаємо, що можуть бути такі пари гіпотез  $h_1$  та  $h_3$ , що не виконується ані  $h_1 \geq h_3$ , ані  $h_3 \leq h_1$ . Відношення „ $\geq$ ” дає змогу ввести корисну структуру на просторі гіпотез  $H$  для будь-якої задачі навчання понять.

### 3.13.3. Алгоритм Find-S пошуку максимально конкретної гіпотези

Використаємо часткове впорядкування для пошуку гіпотези, яка відповідатиме наданим навчальним прикладам. Говорять, що *гіпотеза накриває позитивний приклад*, якщо вона правильно класифікує цей приклад як позитивний. Один із можливих шляхів – розпочати пошук гіпотези з найконкретнішої гіпотези в  $H$  та узагальнювати її, якщо вона не накриває позитивні навчальні приклади. Розглянемо алгоритм Find-S, який використовує часткове впорядкування.

#### Алгоритм Find-S

Крок 1. Ініціалізувати  $h$  найконкретнішою гіпотезою з  $H$

Крок 2. Для кожного позитивного навчального прикладу  $x$  виконати

- Для кожного обмеження атрибуту  $a_i$  в  $h$  виконати

Якщо обмеження  $a_i$  задовольняється  $x$

То не виконувати нічого

Інакше замінити  $a_i$  в  $h$  на наступне більш загальне обмеження, яке задовольняється  $x$

Крок 3. Кінець; результат – максимально конкретна гіпотеза  $h$ , яка відповідає навчальним прикладам

**Приклад 3.32.** Для ілюстрації алгоритму Find-S припустимо, що учню дано послідовність навчальних прикладів для задачі НасСпортом. Перший крок алгоритму Find-S полягає в ініціалізації  $h$  найконкретнішою гіпотезою з  $H$ :  $h = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$ . Після розгляду першого навчального прикладу з табл. 3.1, який є позитивним прикладом, стає очевидним, що вибрана гіпотеза є занадто конкретною. Зокрема, жодне обмеження Ж” у гіпотезі  $h$  не задовольняється цим прикладом. Тому кожне з них замінимо наступним, загальнішим обмеженням, яке буде задовольнятися прикладом. У результаті отримаємо  $h = \langle \text{Сонячно, Тепла, Нормальна, Сильний, Тепла, Той самий} \rangle$ .

Ця гіпотеза  $h$  усе ще є дуже конкретною. Для неї всі приклади, крім одного, є негативними. Другий навчальний приклад (теж позитивний) вимагає від алгоритму продовжити узагальнювати  $h$  та постановити “?” замість кожного значення атрибута в  $h$ , яке не задовольняє новий приклад. Отримана гіпотеза тепер матиме вигляд

$$h = \langle \text{Сонячно, Тепла, ?, Сильний, Тепла, Той самий} \rangle.$$

У результаті аналізу третього навчального прикладу – на цей раз негативного – алгоритм не робить жодних змін у гіпотезі  $h$ . Фактично, алгоритм Find-S просто ігнорує кожний негативний приклад. Це може здатися дивним, але якщо зауважити, що гіпотеза  $h$  вже сумісна з негативним прикладом (тобто,  $h$  правильно класифікує цей приклад як негативний), то жодних змін не потрібно. Якщо припустити, що простір гіпотез містить гіпотезу, яка правильно описує цільове поняття  $c$  і навчальні приклади не містять помилок, то гіпотеза  $h$  ніколи не вимагатиме корекції у разі аналізу негативних прикладів. Пояснюється це тим, що поточна гіпотеза  $h$  – максимально конкретна гіпотеза в  $H$ , несуперечлива з розглянутими на поточний момент позитивними прикладами. Оскільки припускаємо, що цільове поняття  $c$  також має міститись у  $H$  і узгоджуватись з позитивними навчальними прикладами, то  $c$  має бути загальнішим або еквівалентним  $h$ . Але цільове поняття  $c$  ніколи не буде накривати негативні приклади, і, отже, жодний з них не задовольнятиме  $h$  (за означенням загальніша або еквівалентна”). Тому, жодний негативний приклад не може вимагати корекції  $h$ . Для завершення виконання алгоритму Find-S розглянемо четвертий, позитивний, приклад. Це призведе до подальшого узагальнення гіпотези  $h$ , яка набуде вигляду  $h = \langle \text{Сонячно, Тепла, ?, Сильний, ?, ?} \rangle$ .

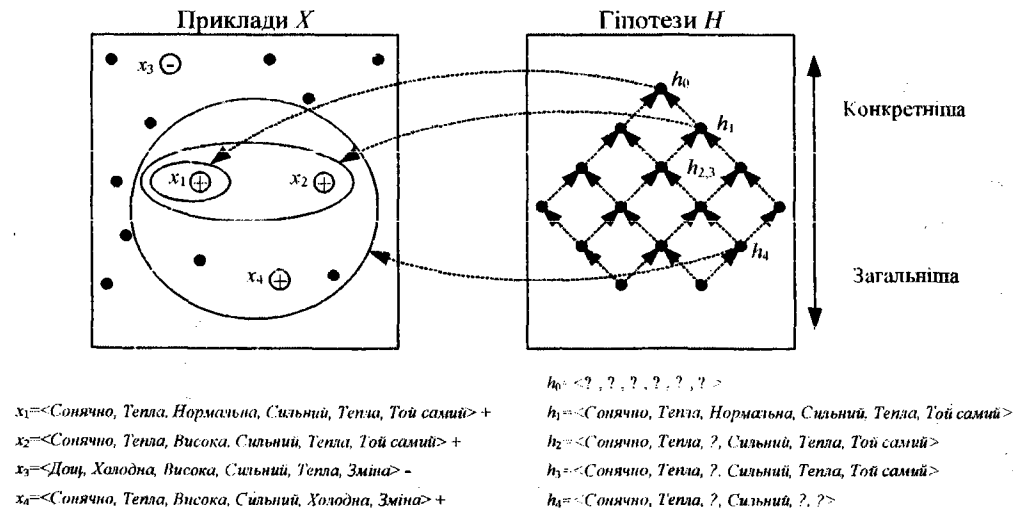


Рис. 3.2



Алгоритм Find-S ілюструє один зі шляхів, за яким часткове впорядкування „загальніша” можна використати для організації пошуку прийнятної гіпотези. Пошук виконують передами від гіпотези до гіпотези, від найконкретнішої до більш загальної, утворюючи ланцюжок частково впорядкованих гіпотез. Схема на рис. 3.2 ілюструє пошук у термінах просторів гіпотез і прикладів. Пошук починають із найконкретнішої в  $H$  гіпотези  $h_0$ , розглядають усе загальніші гіпотези, починаючи з  $h_1$ , і закінчуючи  $h_n$ , як цього вимагає послідовність навчальних прикладів. На схемі, яка зображає простір прикладів, позитивні навчальні приклади позначено знаком „+”, а негативні – знаком „-”. Приклади, які не були навчальними, показано чорними кружечками.

На кожному кроці гіпотеза узагальнюється настільки, наскільки це необхідно для того, щоб накрити новий позитивний приклад. Тому гіпотеза, яку знайдено на кожному кроці, є максимально конкретною з точки зору відповідності навчальним прикладам на всіх, з останнім включно, уже виконаних кроках.

Головна властивість алгоритму Find-S полягає в тому, що для просторів гіпотез, описаних кон'юнкціями значень атрибутів, Find-S гарантовано дасть максимально конкретну гіпотезу в  $H$ , яка відповідає позитивним навчальним прикладам. Зразок такого простору  $H$  розглянуто в задачі прогнозування *НасСпортом*. Кінцева гіпотеза буде також узгоджена з негативними прикладами за умови, що шукане цільове поняття міститься в  $H$ , а навчальна вибірка є коректною.

Є низка питань, на які не дає відповіді застосування алгоритму навчання Find-S. Сформулюємо ці питання.

1. Як добре наблизився учень до шуканого цільового поняття? Якщо алгоритм Find-S знаходить гіпотезу, яка узгоджується з навчальними прикладами, то немає жодного способу визначити, чи знайдено *єдину* гіпотезу в  $H$ , яка узгоджується з прикладами, тобто, чи знайдена гіпотеза є цільовим поняттям. Також залишається невідомим, чи існують інші гіпотези, які узгоджуються з навчальними прикладами. Добре мати такий алгоритм навчання, який може визначати, чи знайдено поняття є цільовим, а, якщо ні, то принаймні оцінити ступінь його наближення до шуканого цільового поняття.

2. Чому перевагу надано максимально конкретній гіпотезі? Якщо є кілька гіпотез, що задовольняють навчальні приклади, то знайде Find-S знайде максимально конкретну з них. Залишається незрозумілим, якій гіпотезі потрібно віддати перевагу: цій максимально конкретній, найзагальнішій чи певній іншій гіпотезі із проміжним ступенем загальності.

3. Чи є несуперечливими навчальні приклади? У більшості практичних задач навчання існує певна ймовірність того, що навчальні приклади містять помилки чи неточності. Така властивість навчальних прикладів може доволі сильно позначитись на достовірності роботи Find-S, оскільки цей алгоритм ігнорує негативні приклади. Бажано мати алгоритм, який може виявляти суперечливість навчальних даних, а ще краще – коректно опрацьовувати приклади такого роду.

4. Як діяти в разі існування кількох максимально конкретних несуперечливих гіпотез? У прикладі 3.32 для простору гіпотез  $H$  задачі *НасСпортом* завжди існує єдина максимально конкретна гіпотеза, несуперечлива з будь-якою множиною позитивних прикладів. У той же час, для інших просторів гіпотез може бути кілька максимально конкретних гіпотез, які узгоджуються із цими прикладами. У такому випадку треба так розширити Find-S, щоб мати можливість повторного пошуку по точках прийняття

рішення, ну яких визначити шлях узагальнення гіпотези. Це надає можливість урахувати, що цільове поняття може знаходитись на іншому ланцюжку часткового впорядкування, відмінному від уже вибраного. Крім того, можна визначити простір гіпотез, у якому немає максимально конкретної несуперечливої гіпотези.

### 3.13.4. Алгоритм „вилучення кандидата”.

Інший підхід до навчання понять реалізовано в *алгоритмі вилучення кандидата* (*candidate elimination*, CE), який дає змогу усунути низку обмежень алгоритму Find-S. Зазначимо, що Find-S знаходить гіпотезу в просторі гіпотез  $H$  і ця гіпотеза узгоджується з навчальними прикладами. Але це лише одна з багатьох гіпотез простору  $H$ , що можуть задовольняти навчальні приклади. Головна ідея алгоритму CE полягає в знаходженні множини всіх гіпотез, які узгоджуються з навчальними прикладами. Алгоритм CE дає змогу знайти цю множину без явного перерахування всіх її елементів. Це можна здійснити частковим упорядкуванням за відношенням „загальніша” для компактного подання множини несуперечливих гіпотез і поступового уточнення цього подання під час обчислення нового навчального прикладу.

Алгоритм CE був практично використаний для вирішення проблеми навчання регулярності в хімічній спектроскопії має і правил керування евристичним пошуком. Однак, практичне застосування алгоритмів CE та Find-S обмежене тим, що вони погано працюють із прикладами, які мають неточності в навчальних даних. Більш важливо, що алгоритм CE забезпечує хорошу концептуальну основу для подання низки фундаментальних проблем машинного навчання.

Алгоритм CE знаходить усі гіпотези, що узгоджуються з навчальними прикладами. Для точного опису алгоритму введемо кілька понять. Гіпотеза *узгоджується* з навчальним прикладом, якщо вона правильно класифікує цей приклад. Гіпотеза  $h$  *узгоджується* з множиною навчальних прикладів  $D$ , якщо  $h(x) = c(x)$  для кожного прикладу  $\langle x, c(x) \rangle$  із  $D$ .

Зазначимо ключову відмінність між термінами *узгоджується* та *задовольняє*. Говорять, що  $x$  *задовольняє* гіпотезу  $h$ , якщо  $h=1$  незалежить від того, яким є  $x$ , позитивним чи негативним прикладом цільового поняття. Однак, чи *узгоджується* такий приклад з  $h$ , залежить від цільового поняття, зокрема, чи  $h(x) = c(x)$ .

Алгоритм CE формує підмножину *всіх* гіпотез, що узгоджуються з навчальними прикладами. Цю підмножину називають *простором версій* (*version space*) відносно простору гіпотез  $H$  і навчальних прикладів  $D$ , бо вона містить усі можливі варіанти цільового поняття. Отже, простір версій, який позначимо  $VS_{H,D}$ , відносно простору

гіпотез  $H$  і множини навчальних прикладів  $D$ , є підмножиною множини гіпотез з  $H$ , що узгоджуються з навчальними прикладами з  $D$ .

Очевидним способом подання простору версій є просте перерахування всіх його елементів. Це дає змогу сформулювати простий алгоритм навчання, який можна назвати „перелічи та вилучи” (*list-then-eliminate*, LTE).

#### Алгоритм LTE

Крок 1. ПростірВерсій := список усіх гіпотез з  $H$ .

Крок 2. Для кожного навчального прикладу  $\langle x, c(x) \rangle$  виконати



•Видалити з ПростірВерсій всяку гіпотезу  $h$ , для якої  $h(x) \neq c(x)$

Крок 3. Вивести список гіпотез ПростірВерсій

Алгоритм LTE спершу організує простір версій так, щоб він містив усі гіпотези з  $H$ , а після цього вилучає з нього всяку гіпотезу, для якої знайдено неузгодженість з певним навчальним прикладом. Отже, простір версій звужується під час перегляду прикладів доти, поки в ідеальному випадку не залишиться точно одна гіпотеза, яка узгоджуватиметься з усіма навчальними прикладами. Очевидно, що це й буде шуканим цільовим поняттям.

Якщо кількість прикладів недостатня для звуження простору версій до однієї гіпотези, то алгоритм виводить повну множину гіпотез, що узгоджуються з поданими даними.

Алгоритм LTE можна застосовувати для скінченного простору гіпотез  $H$ . Цей алгоритм має багато переваг, зокрема, він гарантує знаходження всіх гіпотез, які узгоджуються з навчальними прикладами. Але це вимагає повного перебору всіх гіпотез із  $H$ , що є нереальною вимогою для довільних просторів гіпотез.

Виконання алгоритму CE ґрунтується на тому самому принципі, що й алгоритм LTE, але використовує значно компактніше подання простору версій. Тут простір версій подають його максимально загальними та максимально конкретними елементами. Вони визначають граничні множини загальності та конкретності, що формують межі простору версій для часткового впорядкування простору гіпотез.

Для ілюстрації такого подання просторів версій знову розглянемо задачу навчання поняття *НасСпортом*, описану в прикладі 3.27. Нагадаємо, що для чотирьох навчальних прикладів алгоритм Find-S буде гіпотезу  $h = \langle \text{Сонячно}, \text{Тепла}, ?, \text{Сильний}, ?, ? \rangle$ .

Фактично, це – лише одна із шести різних гіпотез простору  $H$ , що узгоджуються з заданими навчальними прикладами. Ці шість гіпотез наведено на рис. 3.3. Вони утворюють простір версій щодо заданої множини даних і обраного простору гіпотез.

Простір версій можна подати простіше за допомогою множин  $S$  і  $G$ . Стрілки зображають відношення „загальніша”.

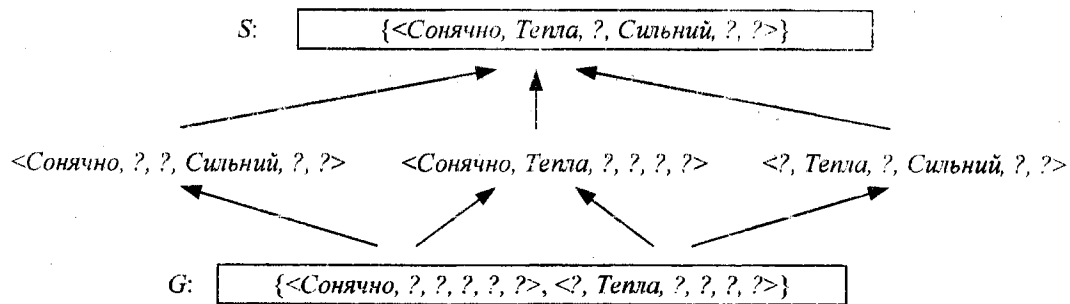


Рис. 3.3

Алгоритм CE задає простір версій множинами  $G$  і  $S$  його максимально загальних та максимально конкретних елементів, відповідно. За лише цими двома множинами  $S$  і  $G$  можна знайти решту необхідних елементів простору версій. Це здійснюється

генеруванням гіпотез, які знаходяться між цими двома множинами гіпотез, частково впорядкованими за відношенням „загальніша”.

Інтуїтивно зрозуміло, що можна подати простір версій у термінах його максимально загальних і максимально конкретних елементів. Для цього визначимо граничні множини  $S$  і  $G$  формально та покажемо, що ці множини дають змогу задати простір версій.

Межею загальності  $G$  простору гіпотез  $H$  для навчальних даних  $D$  є множина максимально загальних елементів множини  $H$ , які узгоджуються з  $D$ . Межею конкретності  $S$  простору гіпотез  $H$  для навчальних даних  $D$  є множина максимально конкретних елементів множини  $H$ , які узгоджуються з  $D$ . Якщо множини  $S$  і  $G$  визначені коректно, то вони повністю визначають простір версій. Зокрема, простір версій є множиною гіпотез, що містяться в  $S$  і  $G$  разом із гіпотезами, які містяться між  $S$  і  $G$  у частково впорядкованому просторі гіпотез. Цей факт стверджує теорема 3.1.

**Теорема 3.1.** Нехай  $X$  – довільна множина прикладів, а  $H$  – множина гіпотез зі значеннями  $\{0,1\}$ , визначених на  $X$ . Нехай  $c: X \rightarrow \{0,1\}$  є довільним цільовим поняттям на  $X$ , а  $D$  – довільна множина навчальних прикладів  $\{ \langle x, c(x) \rangle \}$ . Для всіх  $X$ ,  $H$ ,  $c$  та  $D$  таких, що  $S$  і  $G$  є визначеними, виконується співвідношення

$$VS_{H,D} = \{ h \in H \mid (\exists s \in S)(\exists g \in G)(g \geq h \geq s) \}$$

**Доведення.** Для доведення теореми достатньо показати, що виконуються такі умови:

1) кожна гіпотеза  $h$ , що задовольняє праву частину наведеного виразу, міститься у  $VS_{H,D}$ ;

2) кожен елемент  $VS_{H,D}$  задовольняє праву частину виразу.

Для того щоб показати виконання умови 1, припустимо, що  $g$  – довільний елемент  $G$ ,  $s$  – довільний елемент  $S$ , а  $h$  – довільний елемент  $H$  та  $g \geq h \geq s$ . Тоді за означенням  $S$  усі позитивні приклади з  $D$  мають задовольняти  $s$ . Оскільки  $h \geq s$ , то  $h$  мають задовольняти всі позитивні приклади з  $D$ . Аналогічно за визначенням  $G$  жоден з негативних прикладів у  $D$  не може задовольняти  $g$  і через те, що  $g \geq h$ ,  $h$  не може задовольняти жодний негативний приклад із  $D$ . Оскільки  $h$  задовольняє всі позитивні приклади і жоден негативний приклад з  $D$ , то  $h$  узгоджена з  $D$ . Тому  $h$  є елементом  $VS_{H,D}$ . Отже, умову 1 доведено. Доведення умови 2 дещо складніше. Її можна довести в припущенні, що існує гіпотеза  $h$  з  $VS_{H,D}$ , яка не задовольняє праву частину виразу, а потім показати, що це призводить до суперечності.

Алгоритм CE знаходить простір версій, що містить усі гіпотези з  $H$ , узгоджені із заданою множиною навчальних прикладів. Виконання алгоритму починають з ініціалізації простору версій множиною всіх гіпотез із  $H$ ; це означає, що межа загальності  $G$  ініціалізується множиною, що містить єдиний елемент – найзагальнішу гіпотезу з  $H$ ,  $G_0 := \{ \langle ?, ?, ?, ?, ? \rangle \}$ , а межа конкретності  $S$  – найконкретнішу гіпотезу з  $H$ ,  $S_0 := \{ \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle \}$ .

Ці дві множини задають увесь простір гіпотез, бо кожна гіпотеза з  $H$  загальніша ніж  $S_0$  та конкретніша ніж  $G_0$ . Розгляд кожного навчального прикладу узагальнює і конкретизує границі  $S$  і  $G$ , відповідно, та вилучає з простору версій будь-які гіпотези, що не узгоджуються з цим навчальним прикладом. Після того як опрацьовано всі приклади, знайдений простір версій містить усі гіпотези, які узгоджені з усіма наданими прикладами, і не містить жодних інших.

### Алгоритм СЕ

Крок 1. Ініціалізувати  $G$  множиною максимально загальних гіпотез із  $H$

Ініціалізувати  $S$  множиною максимально конкретних гіпотез із  $H$

Крок 2. Для кожного навчального прикладу  $d$  виконати

• Якщо  $d$  – позитивний приклад

• Вилучити з  $G$  всяку гіпотезу, яка не узгоджується з  $d$

• Для кожної гіпотези  $s$  із  $S$ , яка не узгоджується з  $d$ , виконати

• Вилучити  $s$  із  $S$

• Додати до  $S$  усі такі мінімальні узагальнення  $h$  гіпотези  $s$ , що

$h$  узгоджується з  $d$  та якийсь елемент із  $G$  загальніший ніж  $h$

• Вилучити з  $S$  всяку гіпотезу, яка є більш загальною ніж інші гіпотези в  $S$

• Якщо  $d$  – негативний приклад

• Вилучити з  $S$  всяку гіпотезу, яка не узгоджується з  $d$

• Для кожної гіпотези  $g$  із  $G$ , яка не узгоджується з  $d$ , виконати

• Вилучити  $g$  із  $G$

• Додати до  $G$  усі такі мінімальні конкретизації  $h$  гіпотези  $g$ , що  $h$  узгоджується з  $d$  та якийсь елемент із  $S$  конкретніший ніж  $h$

• Вилучити з  $G$  всяку гіпотезу, яка є менш загальною ніж інші гіпотези в  $G$

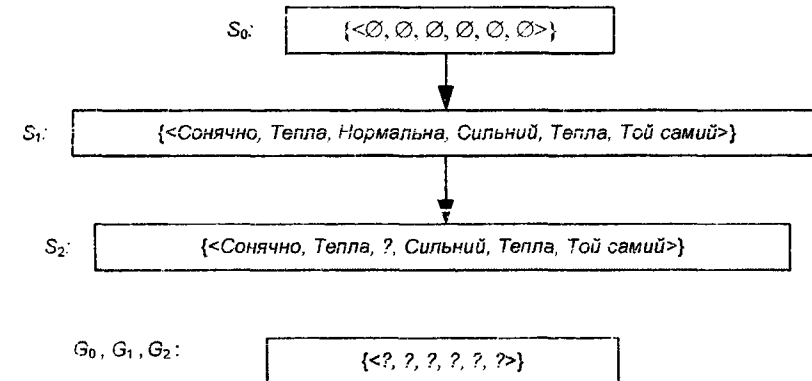
Крок 3. Кінець; результат – множини  $G$  та  $S$  (межі загальності та конкретності простору гіпотез  $H$ )

Зазначимо, що алгоритм описано в термінах операцій обчислення мінімальних узагальнень та мінімальних конкретизацій заданих гіпотез та ідентифікації немінимальних і немаксимальних гіпотез. Детальна реалізація цих операцій залежатиме від вибраного способу подання прикладів і гіпотез, але сам алгоритм можна застосовувати до будь-якої задачі навчання понять і довільного простору гіпотез, для якого визначено ці операції.

**Приклад 3.33.** Простежимо за виконанням алгоритму СЕ для задачі навчання *НасСпортом* із прикладу 3.27. На рис. 3.4 наведено перебіг цього алгоритму для перших двох навчальних прикладів. Згідно з алгоритмом спочатку граничні множини ініціалізуються значеннями  $G_0$  і  $S_0$  – максимально загальною й максимально конкретною гіпотезами в  $H$ , відповідно.

Розглянемо перший навчальний приклад, який є позитивним. Алгоритм СЕ перевіряє межу  $S$  і виявляє, що вона є занадто конкретною та не накриває позитивного прикладу.

Через це межу переглядаємо та узагальнюємо до найменш загальної гіпотези, що накриває цей новий приклад. Ця переглянута межа показана як  $S_1$  на рис. 3.4. Жодна модифікація межі  $G$  не потрібна для цього навчального прикладу, бо  $G_0$  накриває цей приклад. Для другого навчального прикладу (також позитивного) відбувається аналогічне узагальнення  $S$  до  $S_2$ , а  $G$  знову залишається незмінною,  $G_2 = G_1 = G_0$ . Зазначимо, що опрацювання перших двох позитивних прикладів схоже на те, що виконує алгоритм Find-S.



Навчальні приклади:

1. <Сонячно, Тепла, Нормальна, Сильний, Тепла, Той самий>; НасСпортом=Так
2. <Сонячно, Тепла, Бисока, Сильний, Тепла, Той самий>; НасСпортом=Так

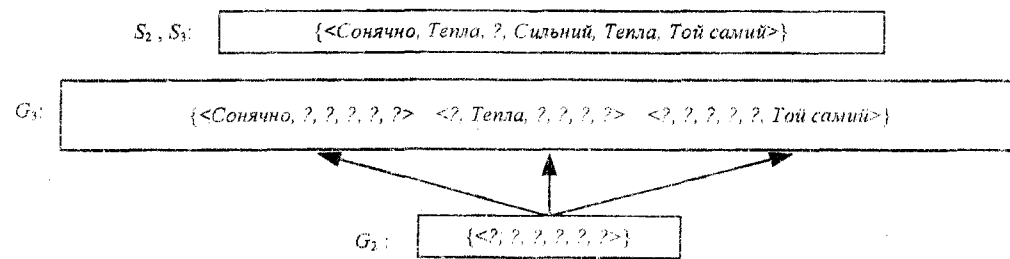
Рис. 3.4

Перші два кроки показали, що позитивні навчальні приклади змушують межу  $S$  простору версій ставати загальнішою. Негативні навчальні приклади відіграють аналогічну роль для межі  $G$ , змушуючи її ставати конкретнішою.

Розглянемо третій навчальний приклад, виконання алгоритму для якого наведено на рис. 3.5. Цей приклад негативний, і алгоритм показує, що межа  $G$  простору версій є надто загальною, але гіпотеза у  $G$  помилково стверджує, що цей приклад позитивний. Тому гіпотезу з межею  $G$  потрібно конкретизувати так, щоб правильно класифікувати цей негативний приклад. Як показано на рис. 3.5 є кілька альтернативних варіантів конкретизації гіпотези, які є елементами нової граничної множини  $G_3$ .

Залишається незрозумілою така ситуація. Є шість атрибутів, які можна визначити для конкретизації  $G_2$ , але  $G_3$  містить лише три нові гіпотези. Наприклад, гіпотеза  $h = \langle ?, ?, \text{нормальна}, ?, ?, ? \rangle$  є мінімальною конкретизацією  $G_2$ , що правильно класифікує новий приклад як негативний, але вона не включена до  $G_3$ . Причиною цього є те, що вона суперечить позитивним прикладам, які розглянуто перед тим. Алгоритм визначає, що гіпотеза  $h$  не є загальнішою від поточного значення межі конкретності  $S_2$ . Насправді межа  $S$  простору версій підсумовує вже розглянуті позитивні приклади, що можна використати для визначення сумісності гіпотези з цими прикладами. Будь-яка гіпотеза, загальніша від  $S$ , накриває приклади, уже накриті  $S$ . Отже, вона накриває й будь-який уже розглянутий позитивний приклад. Аналогічним чином границя  $G$

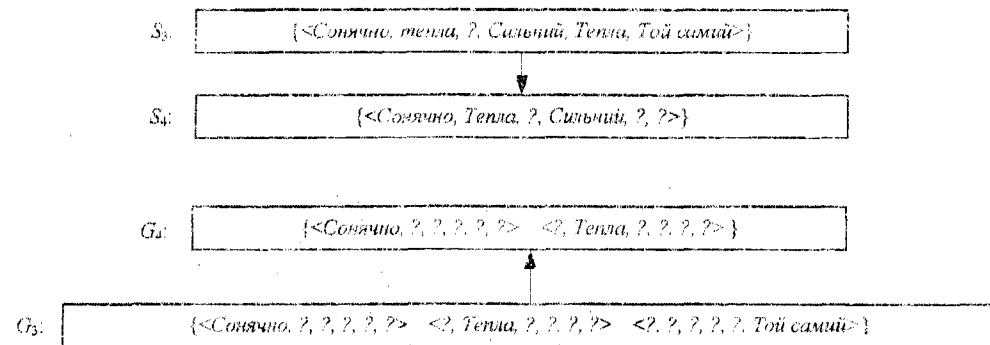
підсумовує інформацію про вже розглянуті негативні приклади. Будь-яка гіпотеза конкретніша від  $G$  буде гарантовано сумісною з усіма переглянутими негативними прикладами. Це твердження істинне, бо будь-яка гіпотеза не може накривати приклади, не накріті  $G$ .



Навчальний приклад:

3. <Дош, Холодна, Висока, Сильний, Тепла, Зміна>; НасСпортотом=Ні

Рис. 3.5



Навчальний приклад:

4. <Сонячно, Тепла, Висока, Сильний, Холодна, Зміна>; НасСпортотом=Так

Рис. 3.6

Четвертий, позитивний, навчальний приклад, як показано на рис. 3.6, призводить до подальшого узагальнення межі  $S$  простору версій. Це призводить також до вилучення одного елемента з межі  $G$ , бо він не може накріти новий позитивний приклад. Таку дію задано в рядку, що знаходиться після умови Якщо  $d$  – позитивний приклад” в описі алгоритму СЕ. Щоб зрозуміти зміст цього кроку, варто розглянути, чому некоректну гіпотезу треба вилучити з  $G$ . Її неможливо конкретизувати, бо довільна конкретизація цієї гіпотези не зможе накріти новий приклад. Її також неможливо узагальнити, оскільки, за визначенням  $G$ , будь-яка загальніша гіпотеза буде накривати принаймні один негативний навчальний приклад. Отож, гіпотезу потрібно вилучити з межі загальності  $G$ , що призведе до вилучення цілої гілки часткового впорядкування з простору версій гіпотез, який конструюється.

Після обробки чотирьох прикладів, граничні множини  $G_4$  і  $S_4$  визначають простір усіх гіпотез, сумісних з розглянутою множиною навчальних прикладів. Повний простір гіпотез наведено на рис. 3.7. Він містить ці гіпотези та обмежений  $G_4$  і  $S_4$ . Знайдений

простір не залежить від послідовності, у якій опрацьовані навчальні приклади – у результаті виконання алгоритму він містить усі гіпотези, сумісні із множиною прикладів. Розгляд наступних навчальних прикладів пересуне межі  $S$  та  $G$  ближче одну до однієї, і визначить межі все меншого простору можливих гіпотез.

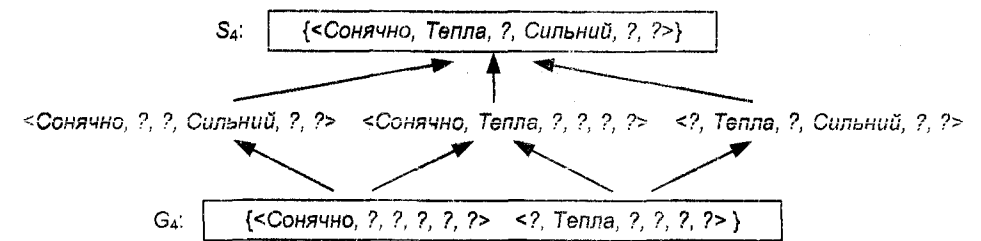


Рис. 3.7

Простір версій, знайдений алгоритмом СЕ, наблизитиметься до гіпотези, яка правильно описує цільове поняття, якщо:

- ◆ немає жодних помилок у навчальних прикладах;
- ◆ у множині гіпотез  $H$  є певна гіпотеза, яка коректно описує цільове поняття.

Із розглядом нових навчальних прикладів простір версій може переглядатись для виявлення, чи залишилися неоднозначності щодо істинності цільового поняття та визначення, чи опрацьовано достатню кількість навчальних прикладів, щоб однозначно ідентифікувати цільове поняття. Цільове поняття точно вивчене, коли множини границь  $S$  та  $G$  сходяться лише до однієї гіпотези.

Що станеться, коли навчальні приклади містять помилки? Припустимо, що другий із наведених навчальних прикладів помилково подано як негативний замість того, щоб бути позитивним. Нажаль, у цьому випадку алгоритм видалить правильне цільове поняття із простору версій. Оскільки він видаляє кожну гіпотезу, що суперечить будь-якому навчальному прикладу, алгоритм вилучить істинне цільове поняття із простору версій, щойно розгляне цей хибний негативний приклад. Звичайно, розглядаючи багато додаткових навчальних прикладів та зауваживши, що граничні множини  $S$  і  $G$  збігаються до порожнього простору версій, учень врешті решт знайде неузгодженість. Такий порожній простір версій свідчить, що немає жодної гіпотези в  $H$ , сумісної з усіма навчальними прикладами. Подібна ознака проявиться й тоді, коли навчальні приклади правильні, але цільове поняття не можна описати обраним поданням гіпотез. Прикладом цього є випадок, коли цільові поняття є диз'юнкціями атрибутів, а простір гіпотез підтримує описи тільки у вигляді кон'юнкцій.

До цього моменту припускалося, що навчальні приклади учень отримує від деякого зовнішнього вчителя. Нехай учневі дозволено проводити експерименти, у яких він вибирає приклад, а потім одержує правильну класифікацію цього прикладу від зовнішнього достовірного джерела. Такий сценарій охоплює ситуації, у яких учень може проводити експерименти або є вчитель, який може правильно класифікувати ситуації. Використаємо термін „запит” для позначення прикладів, класифікованих зовнішнім достовірним джерелом.

Знову розглянемо простір версій, сформований на основі чотирьох навчальних прикладів для поняття *НасСпортотом* та наведений на рис. 3.3. Який хороший запит може сформулювати учень у такому випадку? Що вважати хорошою стратегією запитів

у загальному випадку? Очевидно, що учень повинен намагатися виділяти одні та відкидати інші гіпотези серед усіх альтернативних конкуруючих гіпотез у просторі версій. Для цього він повинен вибрати приклади, які одні з цих гіпотез класифікують як позитивні, а інші – як негативні. Одним із таких прикладів є <Сонячно, Тепла, Нормальна, Слабкий, Тепла, Той самий>.

Цей приклад задовольняє три із шести гіпотез простору версій, наведеному на рис. 3.3. Якщо вчитель класифікує цей приклад як позитивний, то межу  $S$  простору версій можна узагальнити. Аналогічно, якщо вчитель укаже, що це негативний приклад, то межу  $G$  можна конкретизувати. У будь-якому випадку учень успішно навчився більше щодо значення істинності цільового поняття й цим зменшив простір версій удвічі – із шести гіпотез до трьох.

Оптимальна стратегія запитів навчання понять полягає в генеруванні прикладів, які задовольняють точно половину гіпотез простору версій. Якщо таке можливе, то потужність простору версій кожним новим прикладом зменшуватиметься вдвічі. Отже, для знаходження коректного цільового поняття достатньо  $\lceil \log_2 |\mathcal{H}| \rceil$  експериментів, де  $\mathcal{H} = VS_{H,D}$  – простір версій.

Описана ситуація аналогічна грі в двадцять запитань. У цій грі потрібно задавати запитання, які вимагають відповідей лише „так” чи „ні”. За цими відповідями потрібно визначити правильну гіпотезу. Оптимальна стратегія гри в двадцять запитань полягає у такому їх формулюванні, яке порівну розбиває потенційну множину гіпотез між відповідями „так”, „ні”. Хоча теоретично можна побудувати приклад, який задовольнятиме точно половину гіпотез у просторі версій, як це наведено на рис. 3.3, проте в реальній ситуації утворити такий приклад неможливо. У цьому випадку буде потрібно більше запитів ніж  $\lceil \log_2 |\mathcal{H}| \rceil$ .

Нехай для вирішення проблеми навчання поняття немає жодних додаткових навчальних прикладів окрім наведених чотирьох. Незважаючи на це, учневі потрібно класифікувати нові приклади, які ще не розглядалися. Тобто простір версій із рис. 3.3 містить ще кілька гіпотез. Це показує, що цільове поняття ще не вивчено повністю. Тим не менше, певні приклади можна класифікувати з тією самою впевненістю, ніби цільове поняття вже однозначно ідентифіковано. Для ілюстрації припустимо, що учневі треба класифікувати чотири нових приклади, які наведено в табл. 3.2.

Таблиця 3.2

| Приклад | Небо    | Темп Повітря | Вологість | Вітер   | Вода    | Прогноз   | НасСпортом |
|---------|---------|--------------|-----------|---------|---------|-----------|------------|
| A       | Сонячно | Тепла        | Нормальна | Сильний | Холодна | Зміна     | ?          |
| B       | Дощ     | Холодна      | Нормальна | Слабкий | Тепла   | Той самий | ?          |
| C       | Сонячно | Тепла        | Нормальна | Слабкий | Тепла   | Той самий | ?          |
| D       | Сонячно | Холодна      | Нормальна | Сильний | Тепла   | Той самий | ?          |

Зазначимо, що хоча приклад A не був серед навчальних прикладів, він класифікується як позитивний кожною гіпотезою простору версій із рис. 3.3. Оскільки гіпотези простору версій однозначно стверджують, що це – позитивний приклад, то учень може впевнено класифікувати цей приклад, ніби правильне цільове поняття вже знайдене. Незалежно від того, яка гіпотеза в просторі версій виявиться врешті решт правильним

цільовим поняттям, уже очевидно, що вона класифікуватиме цей приклад як позитивний. Не потрібно розглядати кожну гіпотезу в просторі версій, щоб перевірити, чи класифікує вона приклад як позитивний. Ця умова буде виконуватись, тоді й лише тоді, коли приклад задовольняє кожний елемент  $S$ . Таке твердження правильне тому, що решта гіпотез простору версій є щонайменше настільки ж загальні, як і елементи  $S$ . Якщо новий приклад за означенням відношення „загальніша” задовольняє всі елементи  $S$ , то він також має задовольнити будь-яку загальнішу гіпотезу.

Так само, кожна гіпотеза простору версій класифікує приклад B як негативний. Тому він його можна гарантовано класифікувати як негативний частково вивченим поняттям. Ефективною перевіркою цього є те, що приклад B не задовольняє жоден елемент  $G$ .

Приклад C описує іншу ситуацію. Половина гіпотез простору версій класифікує його як позитивний, а друга половина – як негативний. Отже, учень не зможе впевнено класифікувати цей приклад доти, доки не будуть надані додаткові навчальні приклади. Зазначимо, що приклад C уже розглянуто трохи вище як оптимальний запит для учня. Це не випадково, бо приклади, класифікація яких найбільш неоднозначна, надають найбільше нової інформації для уточнення простору версій.

Нарешті, приклад D дві гіпотези простору версій класифікують як позитивний, а решта чотири гіпотези – як негативний. У цьому випадку є менше впевненості в класифікації, ніж у прикладах A та B. Однак, можна віддати перевагу негативній класифікації. Один із можливих методів полягає у виборі варіанту класифікації, який підтриманий більшістю гіпотез. У цьому випадку припускаємо, що всі гіпотези в  $H$  є апріорно однаково ймовірними, і тоді такий підхід дає найімовірнішу класифікацію нового прикладу.

Алгоритм SE наближатиметься до істинного цільового поняття в тому випадку, коли він опрацює правильні навчальні приклади, а заданий простір гіпотез містить цільове поняття. Що станеться, якщо цільове поняття не міститься в просторі гіпотез? Чи можна уникнути цих труднощів використанням простору гіпотез, що включатиме всі гіпотези? Як потужність такого простору гіпотез впливає на здатність алгоритму узагальнювати неопрацьовані випадки? Як потужність простору гіпотез впливає на кількість навчальних прикладів, необхідних для опрацювання? Це фундаментальні питання, які стосуються індуктивного виведення. Будемо досліджувати їх у контексті алгоритму SE. Побачимо, що висновок, отриманий у результаті такого аналізу, стосується будь-якої системи навчання понять, який виводить будь-яку гіпотезу, узгоджену з навчальними даними.

Нехай потрібно перевірити, чи містить простір гіпотез невідоме цільове поняття. Очевидне рішення полягає в збільшенні потужності простору гіпотез включенням усіх можливих гіпотез. Проілюструємо це на прикладі. Для цього знову розглянемо приклад НасСпортом, простір гіпотез якого обмежений лише кон'юнкціями значень атрибутів. Через це обмеження простір гіпотез неспроможний подати навіть прості диз'юнктивні цільові поняття вигляду „Небо=Сонячне або Небо=Хмарне”. Тому для трьох навчальних прикладів, наведених у табл. 3.3 для цієї диз'юнктивної гіпотези, алгоритм видасть, що у просторі версій немає жодної гіпотези.

Таблиця 3.3

| Приклад | Небо    | ТемпПовітря | Вологість | Вітер   | Вода    | Прогноз | НасСпорт |
|---------|---------|-------------|-----------|---------|---------|---------|----------|
| 1       | Сонячно | Тепла       | Нормальна | Сильний | Холодна | Зміна   | Так      |
| 2       | Хмарно  | Тепла       | Нормальна | Сильний | Холодна | Зміна   | Так      |
| 3       | Дощ     | Тепла       | Нормальна | Сильний | Холодна | Зміна   | Ні       |

Щоб переконатися в тому, що немає жодної гіпотези, узгодженої з цими трьома прикладами, розглянемо максимально конкретну гіпотезу, узгоджену з першими двома прикладами. У цьому просторі гіпотез  $H$  нею є гіпотеза

$\langle ?, \text{Тепла}, \text{Нормальна}, \text{Сильний}, \text{Холодна}, \text{Зміна} \rangle$ .

Хоча ця гіпотеза і є найконкретнішою в  $H$  та сумісною з першими двома прикладами, вона є занадто загальною, бо неправильно подає третій (негативний) навчальний приклад. Це зумовлено тим, що учень зорієнтований на розгляд лише кон'юнктивних гіпотез. Тому цей випадок вимагає більш представницького простору гіпотез.

Щоб гарантувати внесення цільового поняття в простір гіпотез  $H$ , потрібно надати простору гіпотез можливість представляти кожне здатне до навчання поняття. Отже, треба надати простору гіпотез спроможність представляти довільну підмножину множини прикладів  $X$ . Множину всіх підмножин множини  $X$  називають *булеаном*  $X$ . Наприклад, у задачі навчання поняття *НасСпорт* потужність простору прикладів  $X$  для даних із шістьма атрибутами складає 96. Скільки можливих понять можна визначити для цієї множини прикладів? Іншими словами, яку потужність має булеан  $X$ ? Потужність булеану  $X$  дорівнює  $2^{|X|}$ . Отже, існує  $2^{96}$ , або приблизно  $10^{28}$  різних цільових понять, які можна визначити на цьому просторі прикладів і на яких учень може навчатись. Згадаємо, що кон'юнктивний простір гіпотез може представляти лише 973 із них, що є свідченням дуже упередженого підходу до формування простору гіпотез.

Переформулюємо задачу навчання *НасСпорт* так, щоб вона була неупередженою. Визначимо новий простір гіпотез  $H'$ , який може представити довільну підмножину прикладів, тобто нехай  $H'$  відповідає булеану  $X$ . Один із способів визначення такого  $H'$  полягає у тому, щоб дозволити довільні диз'юнкції, кон'юнкції та заперечення попередніх гіпотез. Наприклад, цільове поняття „Небо=Сонячне” або „Небо=Хмарне” у такому випадку можна записати  $\langle \text{Сонячне}, ?, ?, ?, ?, ? \rangle \vee \langle \text{Хмарне}, ?, ?, ?, ?, ? \rangle$ .

Із таким простором гіпотез можна використовувати алгоритм СЕ, не турбуючись про те, що цільове поняття не буде зображено. Однак, виникає нова складна проблема – алгоритм навчання поняття тепер нездатний робити узагальнення на підставі відмінних від розглянутих прикладів. Щоб зрозуміти, чому так відбувається, припустимо, що учень отримав три позитивні  $(x_1, x_2, x_3)$  і два негативні  $(x_4, x_5)$  приклади. У цьому випадку, межа  $S$  простору версій буде складатися з гіпотези, яка є диз'юнкцією позитивних прикладів  $S : \{ (x_1 \vee x_2 \vee x_3) \}$ , бо це єдина можлива максимально конкретна гіпотеза, яка накриває ці три приклади. Аналогічно, межа  $G$  буде складатися з гіпотези, яка заперечує тільки надані негативні приклади  $G : \{ \neg(x_4 \vee x_5) \}$ .

Проблема полягає в тому, що із цим дуже „виразним” поданням гіпотез, межа  $S$  завжди буде просто диз'юнкцією позитивних прикладів, а межа  $G$  – запереченням диз'юнкції негативних прикладів. Тому, приклади, які буде однозначно класифікувати  $S$  і  $G$  – задані навчальні приклади. Знаходження єдиного, остаточного цільового поняття змусить кожен приклад з  $X$  вважати навчальним.

Може здатися, що можна уникнути цієї проблеми простим використанням частково навченого простору версій та механізму голосування на елементах простору версій. Нажаль, одноголосно будуть обрані вже розглянуті навчальні приклади. Для всіх інших випадків голосування буде безрезультатним: кожен ще не розглянутий приклад буде класифіковано як позитивний точно половиною гіпотез у просторі версій і негативним іншою половиною. Щоб зрозуміти, чому саме так відбудеться, зверніть увагу на те, що коли  $H$  булеан  $X$ , а  $x$  – якийсь приклад, що ще не розглядався, то для будь-якої гіпотези  $h$  з простору версій, що накриває  $x$ , у булеані міститиметься інша гіпотеза  $h'$ , ідентична до  $h$  за винятком її класифікації прикладу  $x$ . Звичайно, якщо  $h$  знаходиться у просторі версій, то і  $h'$  також міститиметься в ньому, бо убігається з  $h$  на усіх раніше розглянутих навчальних прикладах.

Усе це ілюструє фундаментальну властивість індуктивного навчання: учень, що не робить жодних апріорних припущень щодо шуканого цільового поняття, не має жодних розумних підстав для класифікації будь-яких ще не розглянутих прикладів. Фактично єдина причина, через яку алгоритм СЕ був здатний робити висновки на підставі розглянутих навчальних прикладів у початковому формулюванні завдання *НасСпорт* полягає в тому, що було зроблено неявне припущення – цільове поняття можна подати кон'юнкцією значень атрибутів. У випадках, коли це припущення правильне, а навчальні приклади не містять помилок, класифікація алгоритмом нових прикладів також буде правильною. Натомість, якщо це припущення хибне, алгоритм СЕ завжди неправильно класифікуватиме принаймні окремі приклади з  $X$ .

Оскільки індуктивне навчання вимагає певних попередніх припущень, або упередженості індукції, розглянемо можливі підходи. Для цього уточнімо поняття упередженості індукції. Ключова ідея, на яку треба звернути увагу – це принцип, за яким учень робить узагальнення на підставі навчальних даних для подальшої класифікації нових прикладів. Розглянемо загальний випадок, у якому довільний навчальний алгоритм  $L$  застосовано до довільної множини навчальних даних  $D_c = \{ \langle x, c(x) \rangle \}$  із деяким довільним цільовим поняттям  $c$ . Після закінчення навчання потрібно класифікувати за допомогою алгоритму  $L$  новий приклад  $x_i$ . Позначимо як  $L(x_i, D_c)$  результат класифікації, позитивний чи негативний, який за допомогою алгоритму  $L$  отримано для  $x_i$  в результаті навчання на навчальних даних  $D_c$ . Можна описати індуктивне виведення, виконане алгоритмом  $L$  так:  $(D_c \wedge x_i) \succ L(x_i, D_c)$ . Тут  $y \succ z$  означає, що приклад  $z$  індуктивно виведено з прикладу  $y$ . Наприклад, якщо взяти як  $L$  алгоритм СЕ, замість  $D_c$  – навчальні дані з табл. 3.1, а як  $x_i$  – перший приклад з табл. 3.1, то в результаті індуктивного виведення отримаємо висновок  $L(x_i, D_c) = (\text{НасСпорт} = \text{так})$ .

Оскільки  $L$  – алгоритм індуктивного навчання, то отриманий результат  $L(x_i, D_c)$ , взагалі кажучи, може виявитись хибним. Тобто, класифікація  $L(x_i, D_c)$  дедуктивно не виводиться з навчальних даних  $D_c$  та опису нового прикладу  $x_i$ . Виникає питання, які додаткові припущення необхідно додати до  $D_c \wedge x_i$ , щоб  $L(x_i, D_c)$  виводився дедуктивно. Означимо *упередженість* індукції для  $L$  як множину таких додаткових припущень. Точніше, *упередженість* індукції  $L$  – це така множина припущень  $B$ , що для всіх нових прикладів  $x_i$   $(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)$ . Тут запис  $u \vdash z$  означає, що  $z$  є логічним наслідком  $u$ . Отже, *упередженість* індукції учня означено як множину додаткових припущень  $B$ , достатніх для підтвердження його індуктивного виведення як дедуктивного.

Тепер означимо, що є *упередженням* індукції для алгоритму СЕ. Для цього надамо формальне означення  $L(x_i, D_c)$ . На заданій множині даних  $D_c$  алгоритм СЕ спочатку обчислює простір версій  $VS_{H, D_c}$ , а потім класифікує новий приклад  $x_i$  голосуванням на гіпотезах цього простору. Нехай алгоритм класифікуватиме  $x_i$  лише тоді, коли голосування на гіпотезах простору версій є одногосно позитивним чи негативним. У всіх інших випадках він не класифікуватиме. Тепер *упередженням* індукції для алгоритму СЕ є припущення, що  $c \in H$ . Використовуючи його, кожне індуктивне виведення, зроблене алгоритмом СЕ, підтверджене дедуктивно.

Щоб показати, чому класифікація  $L(x_i, D_c)$  дедуктивно слідує з  $B = \{c \in H\}$  разом з даними  $D_c$  і описом прикладу  $x_i$ , розглянемо такі міркування. Значимо, що з припущення  $c \in H$  дедуктивно випливає, що  $c \in VS_{H, D_c}$ . Це випливає з  $c \in H$ , означення простору версій  $VS_{H, D_c}$  як множини усіх гіпотез у  $H$ , узгоджених з  $D_c$  та з означення  $D_c = \{\langle x, c(x) \rangle\}$  як навчальних даних, узгоджених із цільовим поняттям  $c$ . Класифікація  $L(x_i, D_c)$  також визначена як одногосне прийняття усіх гіпотез з простору версій. Отже, якщо  $L$  класифікує  $L(x_i, D_c)$ , то всі гіпотези з  $VS_{H, D_c}$ , включно з гіпотезою  $c \in VS_{H, D_c}$ , класифікуються так само, тому  $c(x_i) = L(x_i, D_c)$ .

Ця ситуація схематично узагальнена на рис. 3.8. Тут індуктивний алгоритм СЕ, зображений у верхній частині рисунку, має на входах навчальні приклади та новий приклад для класифікації. Система дедуктивного доведення теорем, яка наведена в нижній частині рисунку, має ті самі входи плюс твердження „ $H$  містить цільове поняття”. Хід процесу введення-виведення алгоритму СЕ, який використовує простір гіпотез  $H$ , ідентичний до дедуктивної системи доведення теорем, якщо його ініціалізувати твердженням „ $H$  включає цільове поняття”. Тому це твердження називають *упередженням індукції* алгоритму СЕ. Характеристика індуктивних систем через їхнє *упередження індукції* дає змогу моделювати їх еквівалентними дедуктивними системами. Це забезпечує спосіб порівняння індуктивних систем відповідно до їх принципу узагальнення поза навчальними прикладами. Ці системи виконують ідентичне виведення для кожної можливої вхідної множини навчальних прикладів і кожного можливого нового прикладу з  $X$ . *Упередження індукції*, яке явно показано для системи доведення теорем, насправді існує як частина коду алгоритму СЕ та описана як множина явних тверджень.



Рис. 3.8

Розгляд індуктивної системи виведення в термінах *упередження індукції* дає змогу застосовувати непроедурні засоби опису їхніх принципів узагальнення за межами навчальних даних. Це є першою перевагою такого підходу. Другою перевагою є те, що з'являється можливість порівнювати різних учнів відповідно до сили *упередження індукції*, яку вони використовують.

Для прикладу розглянемо навчальні алгоритми, які розташовані за зростанням сили *упередження індукції* – від найслабшого до найсильнішого.

1. ROTE-LEARNER. Це певний гіпотетичний алгоритм (від англійського *learn by rote* – зубрити, вчити напам'ять), в якому навчання полягає в простому збереженні кожного навчального прикладу в пам'яті. Нові приклади класифікують переглядом пам'яті. Якщо такий приклад знайдено в пам'яті, то алгоритм видає збережений варіант класифікації. У протилежному випадку нового прикладу алгоритм не класифікує.
2. Алгоритм СЕ. Нові приклади класифікують тільки у випадку, коли всі елементи поточного простору версій згідні з класифікацією. У протилежному випадку алгоритм не класифікує нового прикладу.
3. FIND-S. Цей алгоритм знаходить максимально конкрету гіпотезу, узгоджену з навчальними прикладами. Далі він використовує цю гіпотезу, щоб класифікувати всі наступні приклади.

Алгоритм ROTE-LEARNER не має *упередження індукції* та класифікує нові приклади, використовуючи дедуктивне виведення з уже розглянутих навчальних прикладів без жодних додаткових припущень.



Алгоритм СЕ має сильніше упередження індукції, а цільове поняття може бути представленим у його просторі гіпотез. Оскільки він має сильніше упередження, то класифікуватиме деякі приклади, які не зможе класифікувати ROTE-LEARNER. Звичайно, правильність таких класифікацій цілком залежатиме від правильності цього упередження індукції.

Алгоритм FIND-S має ще більше упередження індукції. На додаток до припущення, що цільове поняття може бути описане в його просторі гіпотез, він ще має й припущення упередженості індукції, що всі приклади є негативними прикладами, якщо протилежне не випливає зі знань, отриманих під час навчання.

Отже, головні ідеї цього підрозділу такі:

- ◆ навчання поняття можна подати як пошук у великому наперед визначеному просторі потенційних гіпотез;
- ◆ часткове впорядкування гіпотез „загальна-до-конкретної”, яке можна визначити для будь-якої проблеми навчання поняття, організує пошук у просторі гіпотез;
- ◆ алгоритм FIND-S використовує впорядкування „загальне-до-конкретного” для проведення пошуку в просторі гіпотез по одній із гілок цього часткового впорядкування та знаходить максимально конкретну гіпотезу, узгоджену з навчальними прикладами;
- ◆ алгоритм СЕ використовує впорядкування „загальна-до-конкретної” для визначення простору версій (множини всіх гіпотез, які узгоджені з навчальними даними), для чого поступово будує множини максимально конкретних  $S$  та максимально загальних  $G$  гіпотез;
- ◆ оскільки множини  $S$  та  $G$  визначають повну множину гіпотез, узгоджених з даними, вони забезпечують унiю механiзм опису його невпевненості щодо шуканого цільового поняття. Цей простір версій альтернативних гіпотез можна дослідити для того, щоб визначити, чи знайшов учень цільове поняття, чи навчальні дані є суперечливими, для генерування ефективних запитів із метою подальшого вдосконалення простору версій, і для визначення того, які нові приклади можна однозначно класифікувати за частково вивченим поняттям;
- ◆ простір версій та алгоритм СЕ забезпечують корисну концептуальну структуру для дослідження проблеми вивчення понять. Однак цей алгоритм навчання не є стійким до неточних даних та до ситуацій, у яких невідоме цільове поняття не можна виразити в заданому просторі гіпотез;
- ◆ індуктивні навчальні алгоритми здатні класифікувати нові приклади тільки через їх неявне упередження індукції, що використовується для вибору однієї несуперечливої гіпотези з інших. Упередження алгоритму СЕ полягає в тому, що цільове поняття можна знайти в заданому просторі гіпотез ( $c \in H$ ). Результуючі гіпотези та класифікація наступних прикладів дедуктивно впливають із цього припущення разом із навчальними даними;
- ◆ якщо простір гіпотез розширено так, що кожній можливій підмножині прикладів відповідатиме гіпотеза із простору гіпотез, то це дасть змогу видалити будь-яке упередження індукції з алгоритму СЕ. Нажаль, це також

приведе до втрати здатності класифікувати будь-який приклад поза розглянутими навчальними прикладами. Неупереджений учень не зможе виконувати індуктивні кроки для того, щоб класифікувати нові приклади.

### 3.14. ДЕРЕВО РІШЕНЬ

Дерево рішень використовують як математичну модель у задачах класифікації та прогнозування. До них відносяться проблеми медичного діагностування, оцінювання кредитного ризику, визначення тенденцій на фінансових ринках тощо.

#### 3.14.1. Означення дерева рішень

Дерева рішень використовують техніку „поділяй і володарюй” для послідовного поділу простору пошуку (тут *простір пошуку* – це множина об’єктів, серед яких шукають потрібний). Це нагадує гру „Двадцять запитань”, розглянуту в наведеному нижче прикладі.

**Приклад 3.34.** Нехай є два гравці, і перший із них має задумати якийсь предмет, чи особу, а другий – задавати запитання та за відповідями першого гравця вгадати, який предмет чи яку особу задумано. Для цього дозволено задати не більше ніж двадцять запитань. Нехай перше запитання таке: „Чи цей об’єкт – жива істота?”. Залежно від відповіді буде задано наступне запитання. Якщо відповідь на перше запитання – „так”, то другим запитанням може бути „Чи це людина?”. У разі відповіді „так” можна запитати „Чи це хтось із моїх друзів?”. Якщо відповідь „ні”, то наступним запитанням може бути таке: „Чи ця людина – член моєї родини?”. Після відповіді „так” можна називати імена членів родини. Зменшуючи в такий спосіб кількість об’єктів, ми зменшуємо простір пошуку й, зрештою, можемо визначити, хто чи що було задумано. Хід гри проілюстровано деревом, наведеним на рис. 3.9.

*Дерево рішень* – це кореневе дерево, у якому кожну внутрішню вершину позначено запитанням. Ребра, які виходять із кожної такої вершини, подають можливі відповіді на запитання, асоційоване із цією вершиною. Кожний листок являє собою прогноз розв’язку розглядуваної проблеми.

Бінарне дерево пошуку – частковий випадок дерева рішень. Зазначимо, що дерево рішень – не обов’язково бінарне.

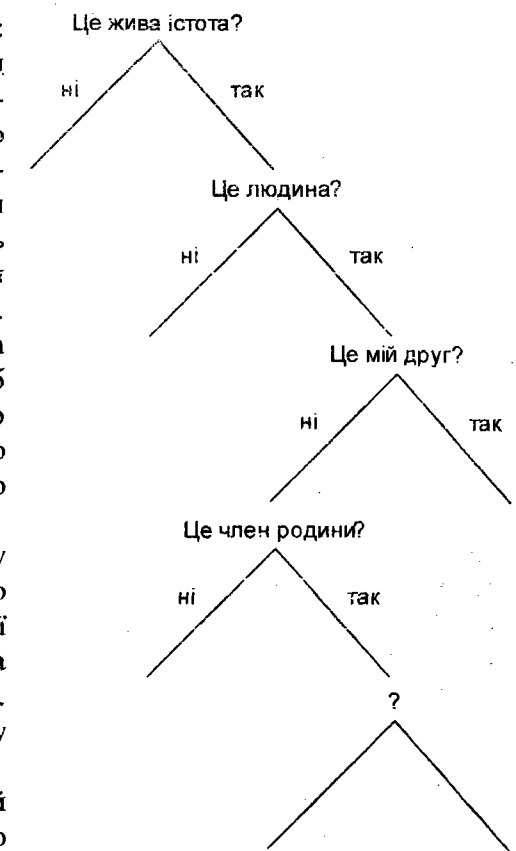


Рис. 3.9

**Приклад 3.35.** Серед восьми монет є одна фальшива, вона має меншу вагу. Потрібно знайти цю монету послідовністю зважувань на балансових терезах. Під час кожного зважування на балансових терезах є три можливі відповіді на запитання „Яка чаша важча?” А саме, чаші мають однакову вагу, перша чаша легша або друга чаша легша. Отже, дерево рішень для послідовності зважувань 3-арне. Це дерево має щонайменше вісім листків, бо є вісім можливих варіантів розв’язку (кожна з восьми наявних монет може бути фальшивою). Найменша кількість зважувань, необхідних для виявлення фальшивої монети, дорівнює висоті дерева рішень. Із наслідку [25, стор 154] випливає, що висота дерева рішень складає щонайменше  $\lceil \log_3 8 \rceil = 2$ . Отже, потрібно не менше двох зважувань. У цьому прикладі фальшиву монету можна виявити, використавши два зважування. Розв’язок подано на рис. 3.10. Занумеруємо монети числами 1, 2, ..., 8. Розділимо монети на три групи: першу групу складуть монети з номерами 1, 2, 3, другу – монети з номерами 4, 5, 6, а третю – монети з номерами 7 та 8. Порівнюємо вагу монет першої й другої груп. Якщо одна із цих груп виявиться легшою, то це означає, що фальшива монета є в цій групі, і її можна виявити другим зважуванням. У разі балансу терезів фальшива одна з монет третьої групи, що також можна виявити другим зважуванням. Кількість зважувань дорівнює висоті дерева, тобто двом (див. рис. 3.10).

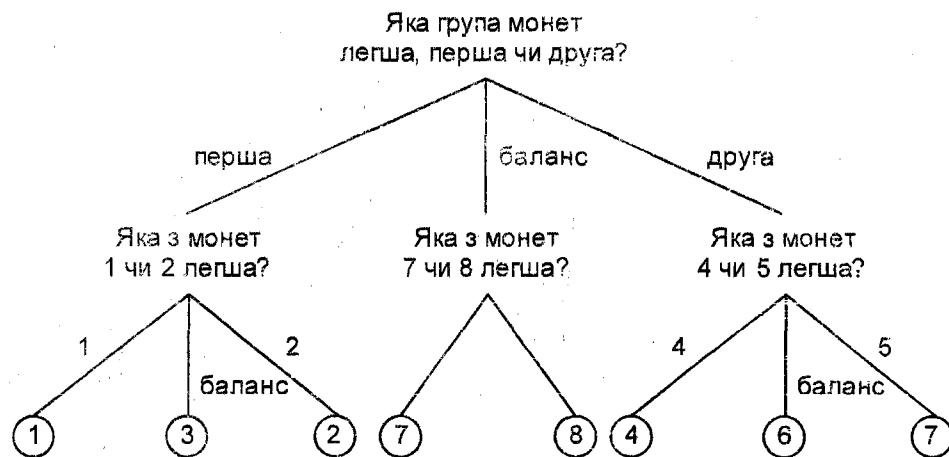


Рис. 3.10

Підхід, який ґрунтується на деревах рішень, особливо корисний для задач класифікації. За цією технікою дерево рішень будують, щоб змодельовати процес класифікації.

Об’єкти, що підлягають класифікації, характеризуються множиною *атрибутів*  $A = \{a_1, a_2, \dots, a_m\}$ . Кожний атрибут  $a \in A$  має *множину значень*, позначимо її як  $Value(a)$ . Наприклад, якщо об’єкти – це дні, то атрибутами можуть бути *Погода*, *Температура*, *Вологість*, *Вітер*. Значення атрибута *Погода* – „Сонце”, „Хмари” та „Дош”; значення атрибута *Температура* – „Спека”, „Помірно” та „Холод”; значення атрибута *Вологість* – „Норма” та „Висока”; значення атрибута *Вітер* – „Слабкий” та „Сильний”. Об’єкти різняться значеннями атрибутів, ці значення записують у рядки таблиці. Кожний її рядок описує один об’єкт, і являє собою кортеж значень його атрибутів.

Інакше кажучи, об’єкт задано кортежем значень його атрибутів (див. табл. 4.3). Множина об’єктів, які потрібно класифікувати, утворює *простір пошуку*. Дерево рішень розбиває цей простір на класи, віднесення об’єкту до певного класу виконується приписуванням цьому об’єкту певної мітки (мітки класу).

Коли дерево побудовано, його можна застосувати до кожного кортежу, який відповідає об’єкту, і результатом буде віднесення об’єкта до певного класу. Техніка дерева рішень для задач класифікації складається із двох етапів: побудови дерева на основі даних із відомими мітками класів (такі дані називають *навчальними*) та класифікації нових об’єктів за допомогою цього дерева. Означення дерева рішень у разі його використання для задач класифікації, наведено нижче.

Задано множину об’єктів  $D = \{D_1, D_2, \dots, D_n\}$ , кожний елемент якої – кортеж  $D_i = (d_{i1}, \dots, d_{im})$ . Елементи кортежу  $D_i$  являють собою, відповідно, значення атрибутів  $a_1, a_2, \dots, a_m$ . Окрім того, задано множину міток класів  $C = \{c_1, c_2, \dots, c_k\}$ . *Дерево рішень* або *класифікаційне дерево* – це кореневе дерево, асоційоване з множиною  $D$ , яке задовольняє такі умови:

- ◆ Кожну внутрішню вершину позначено символом атрибута  $a_i$ .
- ◆ Кожне ребро позначено значенням атрибута, асоційованого з вершиною, з якої це ребро виходить.
- ◆ Кожний листок позначено міткою класу  $c_j$ .

Інакше кажучи, кожна внутрішня вершина дерева рішень асоційована з якимось атрибутом, а кожному можливому значенню цього атрибута відповідає піддерево. Листки відображають результати класифікації.

### 3.14.2. Алгоритми побудови дерева рішень

Розв’язування проблеми класифікації з використанням дерева рішень – процес, що складається із двох етапів.

1. Побудувати дерево рішень із використанням навчальних даних.
2. Для кожного  $D_i \in D$  визначити, до якого класу належить об’єкт.

Існує багато алгоритмів побудови дерев рішень. Усі ці алгоритми – рекурсивні та будують дерево рішень від кореня. Наведемо найпростіший алгоритм побудови дерев рішень – алгоритм **ID3**, який запропоновано 1986 р. Дж. Куїнланом (J. Quinlan).

Параметри **рекурсивного** алгоритму **ID3**( $D, C, A$ ) мають такий зміст:

- ◆  $D$  – множина кортежів зі значеннями властивостей атрибутів об’єктів;
- ◆  $C = \{c_1, c_2, \dots, c_k\}$  – множина міток класів;
- ◆  $A = \{a_1, a_2, \dots, a_m\}$  – множина атрибутів.

Будемо використовувати також такі позначення:

- ◆  $D(a = v)$  – множина об’єктів, для яких значення атрибута  $a$  дорівнює  $v$  (очевидно,  $D(a = v) \subset D$ );
- ◆  $Value(a)$  – множина значень атрибута  $a$ .

#### Алгоритм побудови дерева рішень – ID3( $D, C, A$ )

- Утворити кореневу вершину  $R$
- Якщо всі кортежі із множини  $D$  мають однакові мітки  $c \in C$ , то  $R$  – листок, позначити його міткою  $c$
- Інакше якщо  $A = \emptyset$ , то  $R$  – листок, позначити його міткою, яка найчастіше зустрічається серед міток кортежів із множини  $D$
- Інакше початок

- Вибрати атрибут  $a \in A$  та позначити вершину  $R$  символом цього атрибуту
- Для кожного значення  $v \in \text{Value}(a)$  виконати
  - Додати нове ребро до кореневої вершини  $R$
  - Побудувати множину  $D(a = v)$
  - Якщо  $D(a = v) = \emptyset$ , то кінець доданого ребра – листок, позначити його міткою, що найчастіше зустрічається серед міток кортежів із множини  $D$
  - Інакше в кінцевій вершині доданого ребра побудувати дерево за допомогою  $\text{ID3}(D(a = v), C, A \setminus \{a\})$

кінець

- Повернути кореневу вершину  $R$

Алгоритм  $\text{ID3}(D, C, A)$  утворює та повертає вершину  $R$  дерева, яка може бути або листком, або коренем нового піддерева. Якщо ця вершина – листок, то алгоритм позначає її міткою  $c$  ( $c \in C$ ); якщо ж вона – корінь нового піддерева, то алгоритм позначає цю вершину символом атрибуту  $a$  ( $a \in A$ ). Алгоритм завершує роботу тоді, коли всі листки будованого дерева одержать мітки класів (у такому разі нових звертань до алгоритму вже не буде).

**Приклад 3.36.** У табл. 3.4 наведено результати участі певного гравця в тенісних турнірах та погодні умови під час проведення змагань. Результати змагань – це мітки класів („так” – виграв, „ні” – програв). Імена об’єктів містяться в стовпці **Об’єкти**. Множина атрибутів  $A = \{\text{Погода}, \text{Температура}, \text{Вологість}, \text{Вітер}\}$ , значення кожного атрибуту містяться у відповідному стовпці. Наприклад, об’єкт  $D1$  характеризується таким кортежем значень атрибутів:  $D1 = (\text{Сонце}, \text{Спека}, \text{Висока}, \text{Слабкий})$ , цей об’єкт позначено міткою „ні”. На основі даних про тринадцять ігор (навчальні дані) треба побудувати дерево рішень, і прогнозувати результат чотирнадцятої гри, тобто присвоїти об’єкту  $D14$  одну із двох міток „так” або „ні”. Дерево рішень побудуємо, застосувавши алгоритм  $\text{ID3}$ .

Таблиця 3.4

| Об'єкти<br>(тут об'єкт – це день) | Атрибути           |             |           |         | Мітки класів<br>(Так – виграв,<br>Ні – програв) |
|-----------------------------------|--------------------|-------------|-----------|---------|---|
|                                   | Погода             | Температура | Вологість | Вітер   |   |
|                                   | Значення атрибутів |             |           |         |   |
| D1                                | Сонце              | Спека       | Висока    | Слабкий | Ні  |
| D2                                | Сонце              | Спека       | Висока    | Сильний | Ні  |
| D3                                | Хмари              | Спека       | Висока    | Слабкий | Так   |
| D4                                | Дощ                | Помірно     | Висока    | Слабкий | Так   |
| D5                                | Дощ                | Холод       | Норма     | Слабкий | Так   |
| D6                                | Дощ                | Холод       | Норма     | Сильний | Ні  |
| D7                                | Хмари              | Холод       | Норма     | Сильний | Так   |
| D8                                | Сонце              | Помірно     | Висока    | Слабкий | Ні  |
| D9                                | Сонце              | Холод       | Норма     | Слабкий | Так   |
| D10                               | Дощ                | Помірно     | Норма     | Слабкий | Так   |
| D11                               | Сонце              | Помірно     | Норма     | Сильний | Так   |
| D12                               | Хмари              | Помірно     | Висока    | Сильний | Так   |
| D13                               | Хмари              | Спека       | Норма     | Слабкий | Так   |
| D14                               | Дощ                | Помірно     | Висока    | Сильний |   |

Під час першого звертання алгоритм утворює кореневу вершину дерева рішень, вибирає атрибут з усієї множини атрибутів  $A = \{\text{Погода}, \text{Температура}, \text{Вологість}, \text{Вітер}\}$ . Нехай першим вибрано атрибут *Погода*, символом цього атрибуту буде позначено корінь. Усі можливі значення атрибуту *Погода* – „Сонце”, „Хмари” та „Дош”, тому для цієї вершини утворимо три ребра, яким припишемо вказані значення. Тепер множину об’єктів розіб’ємо на три підмножини за значеннями атрибуту *Погода* (рис. 3.11), вилучимо цей атрибут із множини  $A$  та отримаємо зменшену множину атрибутів  $A = \{\text{Температура}, \text{Вологість}, \text{Вітер}\}$ , яку розглядатимемо під час наступного звертання до алгоритму.

У вершині 2 на рис. 3.11 об’єкти  $\{D9, D11\}$  мають значення „так” результату гри, а об’єкти  $\{D1, D2, D8\}$  – значення „ні”. Отже, ця вершина – не листок, тому зі зменшеної множини атрибутів  $A$  виберемо атрибут, нехай це буде *Температура*. Цим атрибутом позначимо вершину 2. Побудуємо ребра зі значеннями *Температури* („Спека”, „Помірно” та „Холод”) і розіб’ємо множину об’єктів у вершині 2 на три підмножини так, щоб усі об’єкти в кожній із них мали однакове значення цього атрибуту.

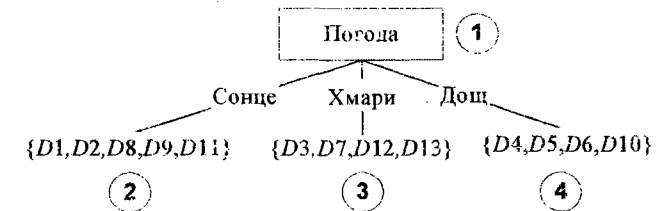


Рис. 3.11

У вершині 5 із рис. 3.12 усі об’єкти ( $D1$  та  $D2$ ) мають значення „ні” результату гри. Тому вершина 5 – листок, його позначимо міткою „ні”. Вершині 7 відповідає лише один об’єкт ( $D9$ ) зі значенням „так” результату гри, тому вершина 7 – листок, який позначимо міткою „так”. Вершині 6 відповідають об’єкти  $D8$  і  $D11$ , вони мають різні значення результатів гри, тому у вершині 6 потрібно будувати дерево. Атрибут *Температура* вилучимо із множини  $A$ , тому тепер  $A = \{\text{Вологість}, \text{Вітер}\}$ . Виберемо атрибут *Вологість* та позначимо символом цього атрибуту вершину 6 – корінь піддерева, яке будуватиметься при рекурсивному звертанні до алгоритму для побудови дерева у вершині 6. Атрибут *Вологість* має два значення: „Норма” та „Висока”.

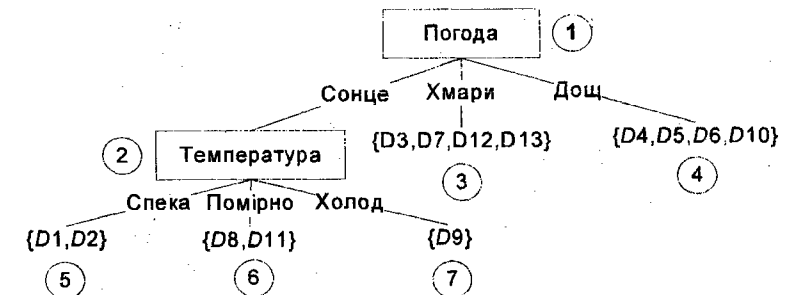


Рис. 3.12

На рис. 3.13 об'єктам  $D11$  та  $D8$  відповідають листки, які, відповідно, одержать позначки „так” і „ні”. Вилучаємо атрибут *Вологість*, отже,  $A = \{\text{Вітер}\}$ .

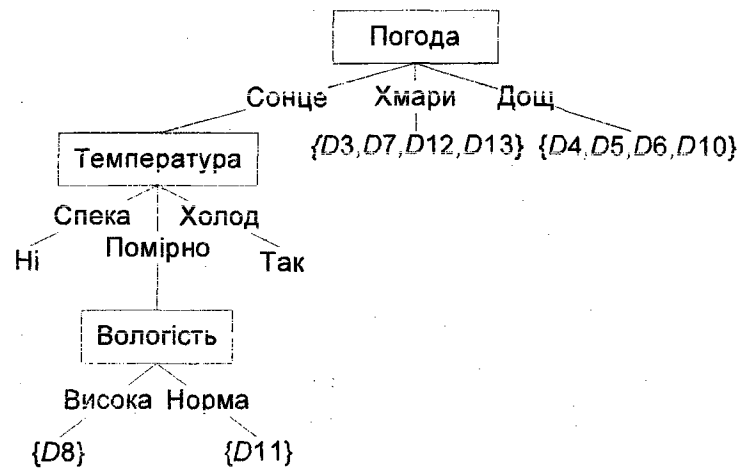


Рис. 3.13

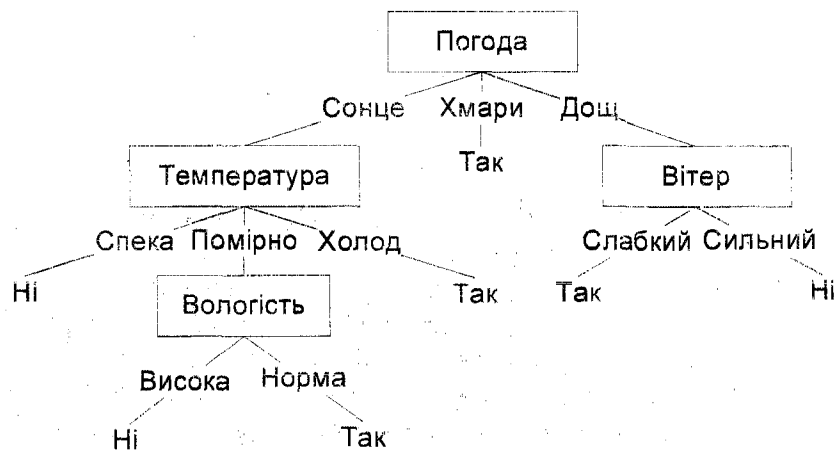


Рис. 3.14

Усі об'єкти, що розглядаємо у вершині 3 із рис. 3.11, мають значення „так” результатів гри. Отже, вершина 3 – листок, його позначимо „так”. У вершині 4 із рис. 3.11 об'єкти мають різні значення результатів гри. Тому виберемо для цієї вершини атрибут *Вітер* (єдина, що залишився) із двома значеннями – „Слабкий” і „Сильний”. Цими значеннями позначимо ребра, які утворимо у вершині 4. Множину об'єктів розіб'ємо на підмножини  $\{D4, D5, D10\}$  зі значеннями результатів гри „так” та  $\{D6\}$  – зі значенням „ні”. Позаяк усі листки одержали мітки класів, то виконання алгоритму закінчене. Дерево рішень подано на рис. 3.14. Користуючись цим деревом, можна прогнозувати результат гри для чотирнадцятого рядка таблиці (програш). Розгляд прикладу завершено.

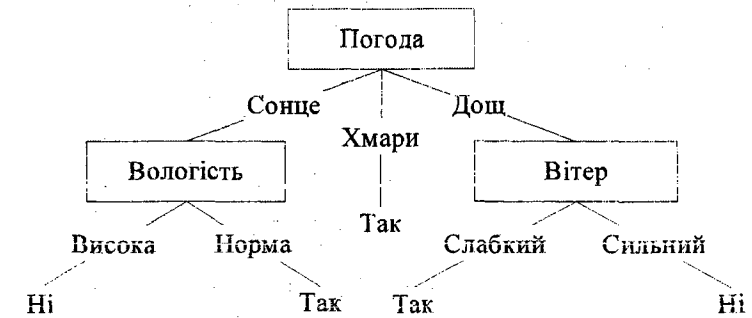


Рис. 3.15

**Приклад 3.37.** Використовуючи дані із прикладу 3.36, можна побудувати інше дерево рішень, якщо атрибути вибирати в іншій послідовності. Таке дерево зображено на рис. 3.15; воно має менше вершин – відсутній атрибут *Температура* – та меншу висоту.

Порівняння прикладів 3.36 та 3.37 показує вплив послідовності, за якою вибирались атрибути, на висоту дерева рішень.

Простий спосіб здійснити класифікацію об'єктів – це побудувати на основі дерева рішень *правила вигляду ЯКЩО – ТО*. Кількість таких правил дорівнює кількості листків дерева. Правила формують на шляхах від кореня до кожного листка дерева. Кожне правило має вигляд

ЯКЩО умова ТО рішення,

де умова – це кон'юнкція речень вигляду (значення атрибуту =  $v$ ), а рішення – це речення вигляду (мітка класу =  $c$ ).

**Приклад 3.38.** Наведемо правила, побудовані на основі цього дерева рішень, зображеного на рис. 4.26.

ЯКЩО (погода = сонце)  $\wedge$  (вологість = висока) ТО (клас = ні);  
 ЯКЩО (погода = сонце)  $\wedge$  (вологість = норма) ТО (клас = так);  
 ЯКЩО (погода = хмари) ТО (клас = так);  
 ЯКЩО (погода = дощ)  $\wedge$  (вітер = слабкий) ТО (клас = так);  
 ЯКЩО (погода = дощ)  $\wedge$  (вітер = сильний) ТО (клас = ні).

За допомогою цих правил дістаємо прогноз для чотирнадцятого рядка табл. 4.3. Оскільки (*Погода* = Дош) і (*Вітер* = Сильний), то (клас = ні), тобто прогноз гри – програш.

Існує багато різних алгоритмів побудови дерев рішень. Зокрема, широко застосовують у задачах прийняття рішень алгоритми **C4.5**, **See5**, **CART**, **SPRINT** та інші. Ці алгоритми відрізняються від **ID3** тим, що в них інакше виконуються вибір атрибуту для позначення внутрішньої вершини, побудова ребер, приписування класів листкам тощо. Окрім того, у ці алгоритми вводять певні обчислення для оптимізації висоти дерева.

Приклад 3.37 показав, що може існувати не єдине дерево рішень, яке розв'язує ту саму задачу класифікації, але мати меншу кількість внутрішніх вершин, відповідно, і перевірок у цих вершинах. Крім того, побудови такого дерева вимагає меншої кількості атрибутів таблиці. Тому розглянемо, як модифікувати алгоритм побудови дерева рішень для того, щоб скоротити кількість перевірок та зменшити висоту дерева рішень.

У штучному інтелекту одним із важеливих принципів є так званий методологічний „принцип бритви Оккама”. Цей принцип отримав назву за ім'ям англійського монаха-францисканця, філософа Вільяма Оккама (Ockham, Ockam, або Occam), який жив у 1285—1349 роках). У спрощеному вигляді цей принцип має такі формулювання: „Не варто помножувати сутність без необхідності”, або „Не варто залучати нові сутності без необхідності”.

Принцип бритви Оккама застосовують у науці в такому сенсі: якщо певне явище можна пояснити двома способами: наприклад, першим – через застосування сутностей (термінів, факторів, перетворень тощо)  $A, B$  і  $C$ , та другим – через  $A, B, C$  та  $D$ , і при цьому обидва способи дають однаковий результат, то сутність  $D$  є зайвою, а правильним є перший спосіб, який може обійтись без залучення зайвої сутності.

У застосуванні до дерев рішень принцип бритви Оккама покладено в основу евристичного підходу до побудови дерева: найкращим є таке дерево, яке дає змогу розв'язати задачу класифікації з найменшою кількістю порівнянь.

Тому основною проблемою, яку треба вирішувати при побудові кожної наступної вершини дерева рішень є вибір атрибута таблиці, найкращого в певному сенсі.

Для вирішення проблеми вибору атрибута будемо на кожному кроці обчислювати певну кількісну оцінку його якості. Для цього введемо статистичну оцінку інформаційного прибутку, яка визначатиме, як добре кожний атрибут вибирає навчальні приклади у відповідності до вимоги розв'язати задачу класифікації.

Інформаційний прибуток будемо визначати за допомогою ентропії, яку використовують у теорії інформації, та яка характеризує ступінь чистоти (чи зашумленості) довільної множини прикладів. Якщо дано множину прикладів  $D$ , яка складається з позитивних та негативних прикладів для деякого цільового поняття, то ентропія  $D$  відносно цієї двійкової класифікації – це величина

$$Entropy(D) = -p_+ \log_2 p_+ - p_- \log_2 p_-,$$

де  $p_+$  – частина позитивних (мітка класу „так”), а  $p_-$  – частина негативних (мітка класу „ні”) прикладів у  $D$ . У всіх обчисленнях, які використовують ентропію, вважають за означенням  $0 \log 0 = 0$ .

**Приклад 3.39.** Розглянемо ще раз табл. 3.4 із прикладу 3.36. Із 14 прикладів 9 є позитивними (класифіковані як „так”), а 5 – негативними (класифіковані як „ні”); тут додано 14-й приклад, який було класифіковано після побудови множини правил як негативний. Позначимо цю множину прикладів як  $[9+;5-]$ . Тоді ентропія множини прикладів  $D$  по відношенню до двійкової класифікації („так”-„ні”) становить

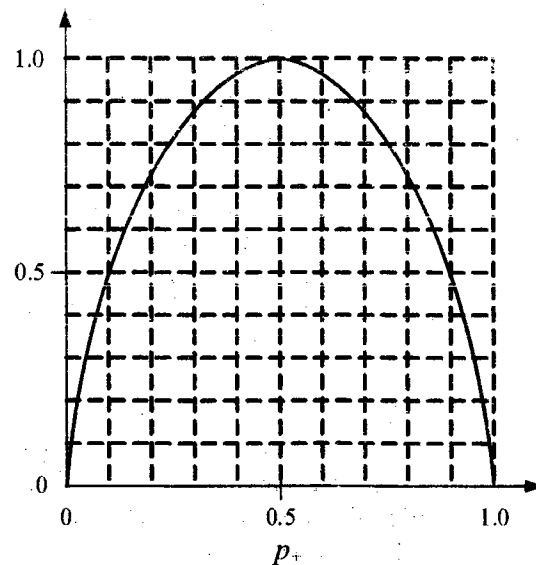


Рис. 3.16

$$Entropy([9+;5-]) = -(9/14) \log_2 (9/14) - (5/14) \log_2 (5/14) = 0.940.$$

Зазначимо, що ентропія дорівнює нулю, коли всі приклади віднесено до одного класу, та дорівнює одиниці, коли множина прикладів містить однакову кількість позитивних та негативних прикладів. В інших випадках значення ентропії знаходиться в проміжку між нулем та одиницею. На рис. 3.16 наведено графік залежності ентропії для двійкової класифікації, тут  $p_+$  змінюється від 0 до 1.

Тут розглянуто ентропію для спеціального випадку, коли цільова класифікація двійкова. У загальнішому випадку, коли класифікація  $c$ -значна, ентропію визначають так

$$Entropy(D) = \sum_{i=1}^c -p_i \log_2 p_i,$$

де  $p_i$  – відносна кількість елементів у множині  $D$ , які належать до класу  $i$ .

Використовуючи ентропію як міру зашумленості множини навчальних прикладів, ми тепер можемо означити міру ефективності атрибута для класифікації навчальних прикладів. Атрибут для класифікації слід вибирати так, щоб після класифікації ентропія відносно цільової функції стала якомога меншою. Нехай множину об'єктів  $D$  класифіковано за допомогою атрибуту  $a$ . Тоді інформаційний прибуток визначають так

$$Gain(D, a) = Entropy(D) - \sum_{v \in Value(a)} \frac{|D_v|}{|D|} Entropy(D_v),$$

де  $Value(a)$  – множина всіх можливих значень атрибуту  $a$ ,  $D_v$  – підмножина множини  $D$  об'єктів, для яких атрибут  $a$  має значення  $v$ ; при описі алгоритмів побудови дерева для множини  $D_v$  із методичних міркувань використано позначення  $D(a=v)$ .

**Приклад 3.40.** Обчислимо інформаційний прибуток для атрибуту Вітер, який має значення Слабкий і Сильний (табл. 3.4 із прикладу 3.36). Із 14 прикладів множини  $D$  є 6 позитивних та 2 негативних, які мають значення Вітер = Слабкий, а в решті – Вітер = Сильний, тобто  $D_{\text{Слабкий}} = [6+, 2-]$ ,  $D_{\text{Сильний}} = [3+, 3-]$ . Інформаційний прибуток від сортування за атрибутом Вітер тепер можна знайти так.

Обчислимо

$$\begin{aligned} Gain(D, \text{Вітер}) &= Entropy(D) - \sum_{v \in \{\text{Слабкий}, \text{Сильний}\}} \frac{|D_v|}{|D|} Entropy(D_v) = \\ &= Entropy(D) - (8/14) Entropy(D_{\text{Слабкий}}) - (6/14) Entropy(D_{\text{Сильний}}), \\ Entropy(D_{\text{Слабкий}}) &= Entropy([6+, 2-]) = -(6/8) \log_2 (6/8) - (2/8) \log_2 (2/8) = 0.811, \\ Entropy(D_{\text{Сильний}}) &= Entropy([3+, 3-]) = -(3/6) \log_2 (3/6) - (3/6) \log_2 (3/6) = 1.0. \end{aligned}$$

Отже, остаточно маємо

$$Gain(D, \text{Вітер}) = 0.940 - (8/14) \cdot 0.811 - (6/14) \cdot 1.0 = 0.048.$$

В алгоритмі C4.5 інформаційний прибуток дає змогу визначити найкращий атрибут, який треба вибрати при утворенні наступної вершини.

**Приклад 3.41.** Розглянемо, який з двох атрибутів Вітер чи Вологість краще вибрати для класифікації. На рис. 3.17 наведено значення ентропії (позначено  $E$ ), обчисленої для атрибутів Вологість та Вітер. На рисунку наведено розбиття об'єктів (елементів множини  $D$ ) за значеннями цих атрибутів на позитивні та негативні. Тепер обчислимо інформаційний прибуток від вибору цих атрибутів:

$$Gain(D, \text{Вітер}) = 0.940 - (8/14) \cdot 0.811 - (6/14) \cdot 1.0 = 0.048,$$

$$Gain(D, \text{Вологість}) = 0.940 - (7/14) \cdot 0.985 - (7/14) \cdot 0.592 = 0.151.$$

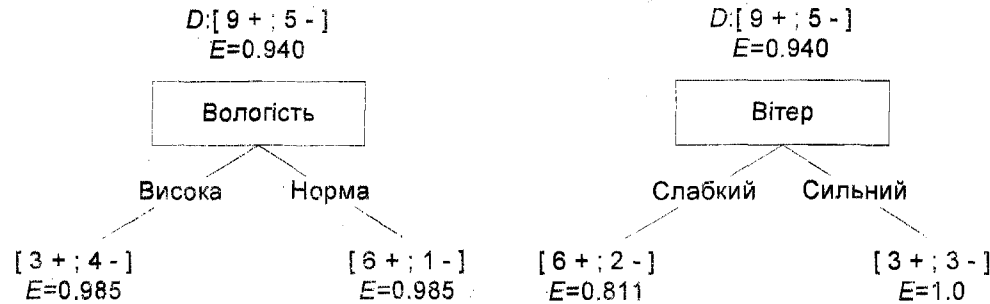


Рис. 3.17

Отже, атрибут Вологість дає більший інформаційний прибуток, він краще класифікує приклади, тобто його треба обрати при утворенні нової вершини дерева рішень.

Відмінність алгоритму C4.5 від алгоритму ID3 полягає в способі вибору атрибута.

#### Алгоритм побудови дерева рішень – C4.5(D, C, A)

- Утворити кореневу вершину  $R$
- Якщо всі кортежі із множини  $D$  мають однакові мітки  $c \in C$ , то  $R$  – листок, позначити його міткою  $c$
- Інакше якщо  $A = \emptyset$ , то  $R$  – листок, позначити його міткою, яка найчастіше зустрічається серед міток кортежів із множини  $D$
- Інакше початок
  - Обчислити інформаційний прибуток кожного атрибута з  $A$
  - Вибрати атрибут  $a \in A$  із найбільшим інформаційним прибутком та позначити вершину  $R$  символом цього атрибута
  - Для кожного значення  $v \in \text{Value}(a)$  виконати
    - Додати нове ребро до кореневої вершини  $R$
    - Побудувати множину  $D(a = v)$
    - Якщо  $D(a = v) = \emptyset$ , то кінець доданого ребра – листок, позначити його міткою, що найчастіше зустрічається серед міток кортежів із множини  $D$
    - Інакше в кінцевій вершині доданого ребра побудувати дерево за допомогою C4.5( $D(a = v)$ ,  $C$ ,  $A \setminus \{a\}$ )

кінець

- Повернути кореневу вершину  $R$

**Приклад 3.42.** Розв'яжемо приклад 4.36 за алгоритмом C4.5 для випадку, коли

14-й об'єкт класифіковано як ні". Знайдемо інформаційний прибуток для кожного атрибута, щоб визначити, який із них буде вершиною дерева.

$$Entropy(D) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14) = 0.940,$$

$$Gain(D, \text{Погода}) = 0.940 - \frac{5}{14} \left( -\frac{2}{5} \cdot \log_2\left(\frac{2}{5}\right) - \frac{3}{5} \cdot \log_2\left(\frac{3}{5}\right) \right) -$$

$$-\frac{4}{14} \left( \frac{4}{4} \cdot \log_2\left(\frac{4}{4}\right) - \frac{0}{4} \cdot \log_2\left(\frac{0}{4}\right) \right) - \frac{5}{14} \left( -\frac{3}{5} \cdot \log_2\left(\frac{3}{5}\right) - \frac{2}{5} \cdot \log_2\left(\frac{2}{5}\right) \right) = 0.246,$$

$$Gain(D, \text{Температура}) = 0.940 - \frac{4}{14} \left( -\frac{2}{4} \cdot \log_2\left(\frac{2}{4}\right) - \frac{2}{4} \cdot \log_2\left(\frac{2}{4}\right) \right) -$$

$$-\frac{6}{14} \left( -\frac{4}{5} \cdot \log_2\left(\frac{4}{5}\right) - \frac{2}{5} \cdot \log_2\left(\frac{2}{5}\right) \right) - \frac{4}{14} \left( -\frac{3}{4} \cdot \log_2\left(\frac{3}{4}\right) - \frac{1}{4} \cdot \log_2\left(\frac{1}{4}\right) \right) = 0.165,$$

$$Gain(D, \text{Вологість}) = 0.940 - \frac{7}{14} \left( -\frac{3}{7} \cdot \log_2\left(\frac{3}{7}\right) - \frac{4}{7} \cdot \log_2\left(\frac{4}{7}\right) \right) -$$

$$-\frac{7}{14} \left( -\frac{6}{7} \cdot \log_2\left(\frac{6}{7}\right) - \frac{1}{7} \cdot \log_2\left(\frac{1}{7}\right) \right) = 0.151,$$

$$Gain(D, \text{Вітер}) = 0.940 - \frac{8}{14} \left( -\frac{6}{8} \cdot \log_2\left(\frac{6}{8}\right) - \frac{2}{8} \cdot \log_2\left(\frac{2}{8}\right) \right) -$$

$$-\frac{6}{14} \left( -\frac{3}{6} \cdot \log_2\left(\frac{3}{6}\right) - \frac{3}{6} \cdot \log_2\left(\frac{3}{6}\right) \right) = 0.048.$$

Відповідно до міри інформаційного прибутку атрибут  $a = \text{Погода}$  надає найкращий варіант для вибору як атрибут для розбиття множини прикладів. На рис. 3.18 наведено фрагмент дерева рішень, одержаного на першому кроці.

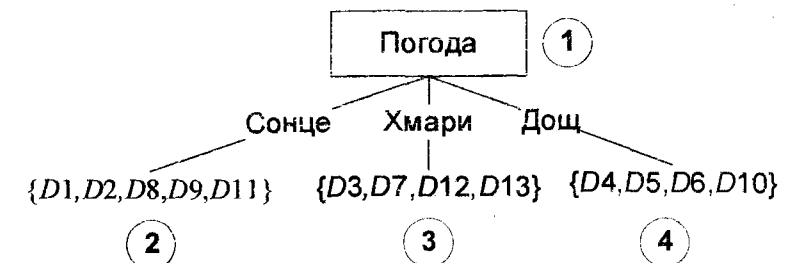


Рис. 3.18



Розглянемо вхідні дані для наступного кроку виконання алгоритму (табл. 3.5).

Таблиця 3.5

| День | Температура | Вологість | Вітер   | Гра |
|------|-------------|-----------|---------|-----|
| D1   | Спека       | Висока    | Слабкий | Ні  |
| D2   | Спека       | Висока    | Сильний | Ні  |
| D8   | Помірно     | Висока    | Слабкий | Ні  |
| D9   | Холод       | Норма     | Слабкий | Так |
| D11  | Помірно     | Норма     | Сильний | Так |

На цьому кроці  $D_{\text{Сонячно}} = \{D1, D2, D8, D9, D11\}$  і ентропія становить

$$Entropy(D_{\text{Сонячно}}) = -\frac{2}{5} \log_2\left(\frac{2}{5}\right) - \frac{3}{5} \log_2\left(\frac{3}{5}\right) = 0.971.$$

Знайдемо інформаційні прибутки для атрибутів, що залишилися:

$$Gain(D_{\text{Сонячно}}, \text{Температура}) = 0.971 - \frac{2}{5} \left( -\frac{2}{2} \cdot \log_2\left(\frac{2}{2}\right) - \frac{0}{2} \cdot \log_2\left(\frac{0}{2}\right) \right) -$$

$$-\frac{2}{5} \left( -\frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) - \frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) \right) - \frac{1}{5} \left( -\frac{1}{1} \cdot \log_2\left(\frac{1}{1}\right) - \frac{0}{1} \cdot \log_2\left(\frac{0}{1}\right) \right) = 0.585,$$

$$Gain(D_{\text{Сонячно}}, \text{Вологість}) = 0.971 - \frac{3}{5} \left( -\frac{3}{3} \cdot \log_2\left(\frac{3}{3}\right) - \frac{0}{3} \cdot \log_2\left(\frac{0}{3}\right) \right) -$$

$$-\frac{2}{5} \left( -\frac{0}{2} \cdot \log_2\left(\frac{0}{2}\right) - \frac{2}{2} \cdot \log_2\left(\frac{2}{2}\right) \right) = 0.971,$$

$$Gain(D_{\text{Сонячно}}, \text{Вітер}) = 0.971 - \frac{3}{5} \left( -\frac{1}{3} \cdot \log_2\left(\frac{1}{3}\right) - \frac{2}{3} \cdot \log_2\left(\frac{2}{3}\right) \right) -$$

$$-\frac{2}{5} \left( -\frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) - \frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) \right) = 0.02.$$

Максимальний прибуток припадає на атрибут  $a = \text{Вологість}$ . Отже, дерево рішень на другому кроці набуде вигляду, наведеному на рис. 3.19.

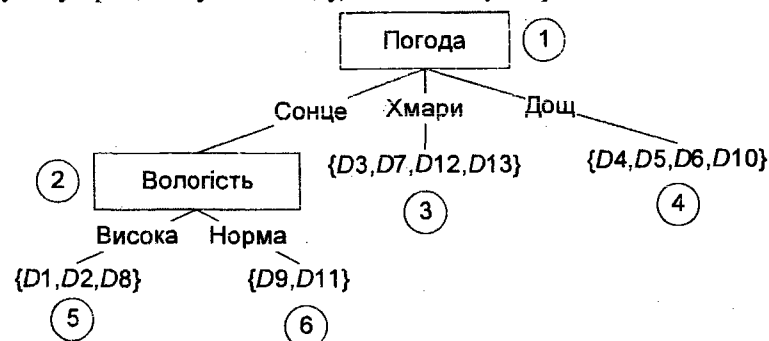


Рис. 3.19

При розгляді  $\{D1, D2, D8\}$  цільовий атрибут набуває значення „ні”, а  $\{D9, D11\}$  – „так”, тому необхідності шукати інформаційний прибуток для вершин цих гілок немає, бо вони – листки. Отже, дерево рішень набуває вигляду, наведеному на рис. 3.20.

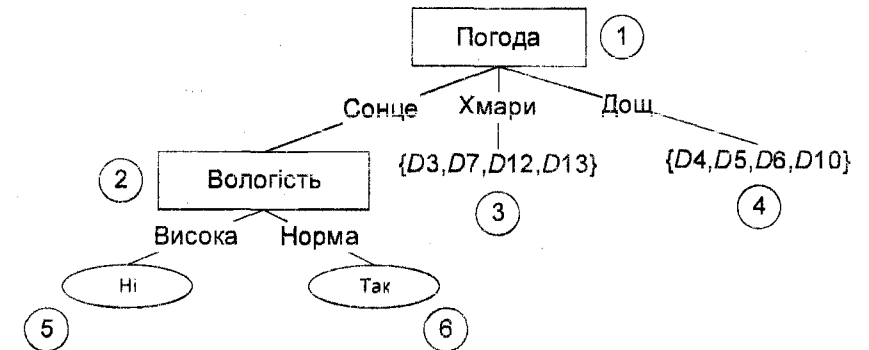


Рис. 3.20

У табл. 3.6 наведено дані для наступного кроку.

Таблиця 3.6

| День | Температура | Вологість | Вітер   | Гра |
|------|-------------|-----------|---------|-----|
| D3   | Спека       | Висока    | Слабкий | Так |
| D7   | Холод       | Норма     | Сильний | Так |
| D12  | Помірно     | Висока    | Сильний | Так |
| D13  | Спека       | Норма     | Слабкий | Так |

Оскільки цільовий атрибут для  $\{D3, D7, D12, D13\}$  набуває одне значення „так”, то в кінці ребра „Хмари” отримуємо листок (рис. 3.21).

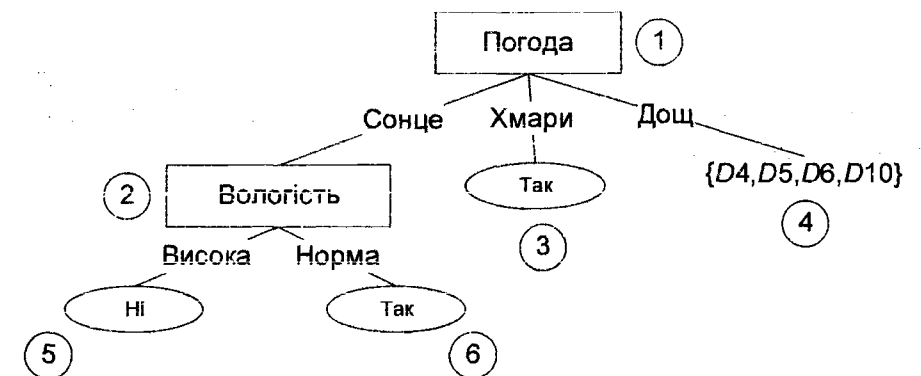


Рис. 3.21

Не розглянутою залишилася гілка „Погода = Дош”. Вхідні дані на цьому кроці зведено в табл. 3.7.

Таблиця 3.7

| День | Температура | Вологість | Вітер   | Гра |
|------|-------------|-----------|---------|-----|
| D4   | Помірно     | Висока    | Слабкий | Так |
| D5   | Холод       | Норма     | Слабкий | Так |
| D6   | Холод       | Норма     | Сильний | Ні  |
| D10  | Помірно     | Норма     | Слабкий | Так |
| D14  | Помірно     | Висока    | Сильний | Ні  |

На цьому кроці  $D_{\text{Дош}} = \{D4, D5, D6, D10\}$  і ентропія становить

$$Entropy(D_{\text{Дош}}) = -\frac{3}{5} \log_2\left(\frac{3}{5}\right) - \frac{2}{5} \log_2\left(\frac{2}{5}\right) = 0.971.$$

Знайдемо інформаційні прибутки для атрибутів Температура, Вологість, Вітер.

$$Gain(D_{\text{Дош}}, \text{Температура}) = 0.971 - \frac{3}{5} \left( -\frac{2}{3} \cdot \log_2\left(\frac{2}{3}\right) - \frac{1}{3} \cdot \log_2\left(\frac{1}{3}\right) \right) -$$

$$-\frac{2}{5} \left( -\frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) - \frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) \right) = 0.02,$$

$$Gain(D_{\text{Дош}}, \text{Вологість}) = 0.971 - \frac{2}{5} \left( -\frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) - \frac{1}{2} \cdot \log_2\left(\frac{1}{2}\right) \right) -$$

$$-\frac{3}{5} \left( -\frac{2}{3} \cdot \log_2\left(\frac{2}{3}\right) - \frac{1}{3} \cdot \log_2\left(\frac{1}{3}\right) \right) = 0.02,$$

$$Gain(D_{\text{Дош}}, \text{Вітер}) = 0.971 - \frac{3}{5} \left( -\frac{3}{3} \cdot \log_2\left(\frac{3}{3}\right) - \frac{0}{3} \cdot \log_2\left(\frac{0}{3}\right) \right) -$$

$$-\frac{2}{5} \left( -\frac{0}{2} \cdot \log_2\left(\frac{0}{2}\right) - \frac{2}{2} \cdot \log_2\left(\frac{2}{2}\right) \right) = 0.971.$$

Максимальний інформаційний прибуток дорівнює 0.971 (атрибут Вітер). Отже, для вершини 4 вибираємо атрибут Вітер. Дерево рішень після цього етапу виконання алгоритму наведено на рис. 3.22.

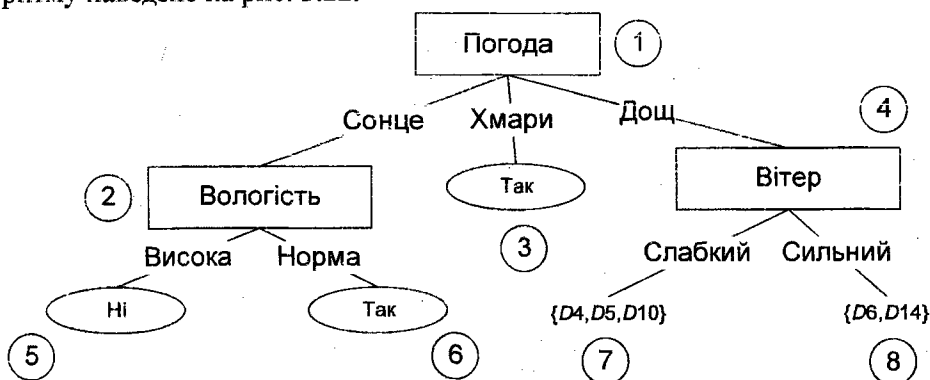


Рис. 3.22

## Розділ 3. Машинне навчання

При розгляді вхідних даних можна побачити, що у вершині 7 цільовий атрибут набуває значення „так”, а у вершині 8 – „ні”. Отже, ці вершини – листки. Остаточний результат побудови дерева за алгоритмом C4.5 наведено на рис. 3.23.

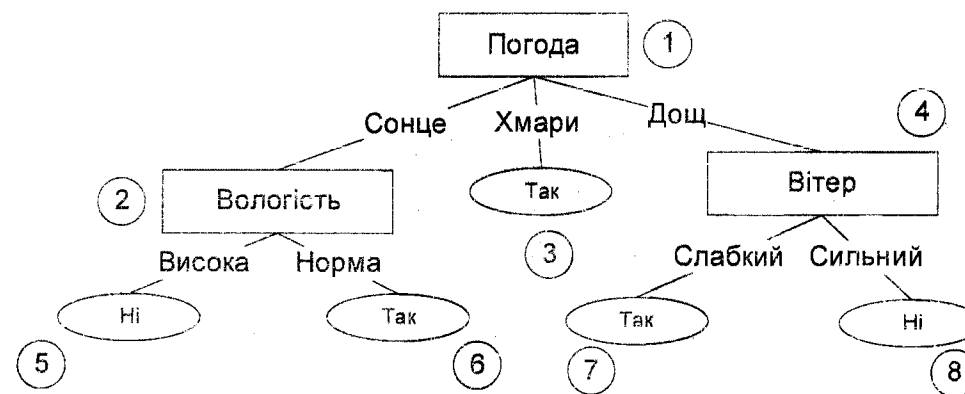


Рис. 3.23

Ще один алгоритм для побудови дерев рішень – алгоритм **CART** (Classification and Regression Trees). Цей алгоритм будує **бінарне дерево рішень**, тобто в кожній його вершині відбувається розбиття множини прикладів на дві підмножини. Цей алгоритм уперше опубліковано 1984 року. Для вибору чергової вершини застосовують критерій на основі індексу Gini. Якщо множина об'єктів  $D$  містить об'єкти  $n$  класів, то *індекс Gini* обчислюють за формулою

$$Gini(D) = 1 - \sum_{i=1}^n p_i^2,$$

де  $p_i$  – відносна частота класу  $i$  в множині  $D$ . Якщо множину розбивають на дві підмножини  $D_1$  і  $D_2$  з кількостями прикладів  $n_1$  та  $n_2$ , відповідно, то показник якості розбиття обчислюють за формулою

$$Gini(D) = \frac{n_1}{n} Gini(D_1) + \frac{n_2}{n} Gini(D_2).$$

Найкращим вважають розбиття, у якого  $Gini(D)$  є найменшим. Для побудови дерева рішень за алгоритмом **CART** застосовують таку процедуру. Позначимо як  $N$  кількість об'єктів у певній вершині дерева, як  $L$  та  $R$  – кількість об'єктів у лівому та правому піддеревах, відповідно,  $l_i$  та  $r_i$  – кількість об'єктів  $i$ -го класу у лівому та правому піддеревах, відповідно. Тоді якість розбиття визначають за критерієм мінімуму

$$Gini = \frac{L}{N} \left( 1 - \sum_{i=1}^n \left( \frac{l_i}{L} \right)^2 \right) + \frac{R}{N} \left( 1 - \sum_{i=1}^n \left( \frac{r_i}{R} \right)^2 \right),$$

по всіх можливих розбиттях. Останню формулу можна спростити до такого вигляду

$$Gini = \frac{1}{N} \left( L \cdot \left( 1 - \frac{1}{L^2} \sum_{i=1}^n l_i^2 \right) + R \cdot \left( 1 - \frac{1}{R^2} \sum_{i=1}^n r_i^2 \right) \right).$$

Можна виписати альтернативні формули для мінімізації, бо множення на сталу не впливає на результат мінімізації

$$Gini = L - \frac{1}{L^2} \sum_{i=1}^n l_i^2 + R - \frac{1}{R^2} \sum_{i=1}^n r_i^2 = N - \left( \frac{1}{L^2} \sum_{i=1}^n l_i^2 + \frac{1}{R^2} \sum_{i=1}^n r_i^2 \right)$$

Остаточно, задачу можна звести до максимізації функції

$$Gini = \frac{1}{L^2} \sum_{i=1}^n l_i^2 + \frac{1}{R^2} \sum_{i=1}^n r_i^2$$

по всіх можливих розбиттях множини об'єктів на дві підмножини.

### 3.15. НАВЧАННЯ НА ОСНОВІ ЗВ'ЯЗКІВ

У цьому підрозділі розглянуто нейроподібні, або біологічні підходи до навчання. Нейроподібні моделі, котрі також називають системами паралельного розподіленого опрацювання інформації, або системами зв'язків, не передбачають явного використання символічного подання в задачі навчання. Інтелектуальні властивості цих систем забезпечуються взаємодією простих компонентів (біологічних або штучних нейронів) і налагодженням зв'язків між ними в процесі навчання, або адаптації. Нейрони організовані в декілька шарів, тому такі системи є розподіленими. Інформація опрацьовується паралельно, тобто всі нейрони одного шару одночасно й незалежно один від одного одержують і перетворюють вхідні дані.

Однак, у нейроподібних моделях символічне подання відіграє важливу роль під час формування вхідних векторів та інтерпретації вхідних значень. Наприклад, для створення нейронної мережі розробник має визначити схему кодування образів для їхнього передавання в нейронну мережу. Вибір схеми кодування може стати важливим чинником здатності мережі до навчання.

Нейронні мережі виконують паралельне й розподілене опрацювання інформації. Вхідну інформацію із предметної області перетворюють у числові вектори. Зв'язки між елементами (нейронами) також подають числовими значеннями. Нарешті, перетворення образів – це наслідок числових операцій, як правило, векторно-матричного множення.

Алгоритми та архітектури, які реалізують цей підхід, не передбачають явного програмування. Їх просто вибирають для навчання мережі. У цьому й полягає головна перевага такого підходу: інваріантні властивості вхідної інформації виявляються завдяки вибору відповідної архітектури й методу навчання.

**Нейронні мережі найкраще застосовні для розв'язування таких задач:**

- ◆ класифікація – визначення категорії або групи, до якої належать вхідні значення;
- ◆ розпізнавання образів – ідентифікація структури або шаблону даних;
- ◆ реалізація пам'яті, зокрема задача контекстної адресації пам'яті;
- ◆ прогнозування, наприклад, діагностування хвороби за її симптомами, визначення наслідків на основі відомих причин;

- ◆ оптимізація – пошук оптимальних значень;
- ◆ фільтрація – виділення корисного сигналу з фоновому шуму, відкидання неістотних компонент сигналу.

Описані в цьому підрозділі методи найкраще застосовні для розв'язування задач, котрі важко описати символічними моделями. До них належать задачі, предметна область яких вимагає усвідомлення або погано формалізується за допомогою явно заданого синтаксису.

#### 3.15.1. Біологічні нейронні мережі

Нейрон, або нервова клітина, є особливою біологічною клітиною, яка опрацьовує інформацію (рис. 3.24). Вона складається з тіла клітини, чи соми, та двох типів вихідних деревоподібних відростків: аксона та дендритів. Тіло клітини містить ядро з інформацією про успадковані властивості, та плазму, яка містить усе для продукування необхідних нейрону речовин. Нейрон отримує сигнал від інших нейронів, який передається йому по дендритах (приймачі), і передає вироблений тілом його клітини сигнал через аксон, кінець якого розгалужений на волокна. Кінцями цих волокон є синапси.

Синапс є елементарною структурою й функціональним зв'язком між двома нейронами. Коли імпульс досягає синапсу, то виділяються хімічні речовини, які називають нейротрансмітерами. Нейротрансмітери збуджують чи гальмують, залежно від типу синапсу, здатність нейрона-приймача до породження електричного імпульсу.

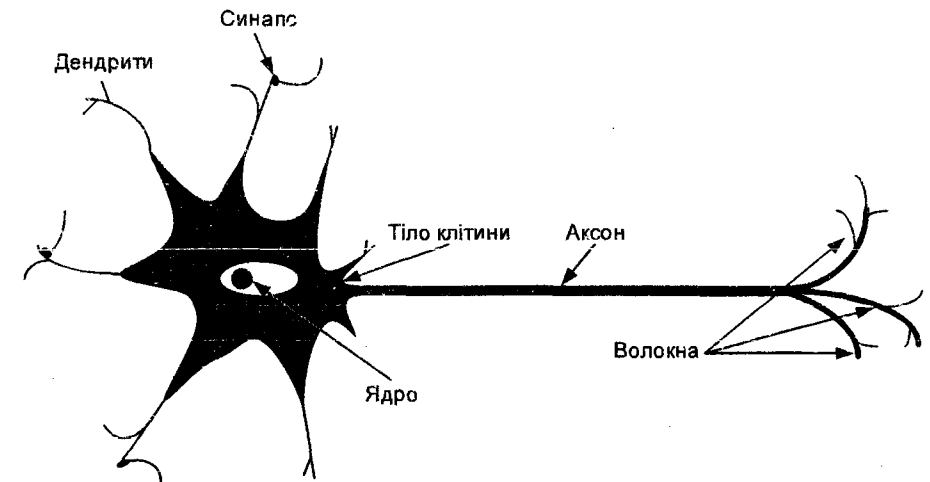


Рис. 3.24

#### 3.15.2. Модель штучного нейрона

Штучною нейронною мережею, або нейромережею, називають масово паралельний та розподілений процесор, який складається із простих обчислювальних елементів, і який має природну властивість до збереження емпіричних знань та забезпечує можливість їхнього використання.

Нейромережа моделює роботу мозку в такий спосіб: знання отримуються з деякого середовища засобами навчання, а міжнейронні зв'язки (синаптичні ваги) використовуються для збереження набутого знання.

Штучним нейроном називають обчислювальний елемент, який є фундаментальним

для функціонування нейромережі. Модель штучного нейрона вперше була запропонована Мак-Каллоком (McCulloch) та Пітсом (Pitts).

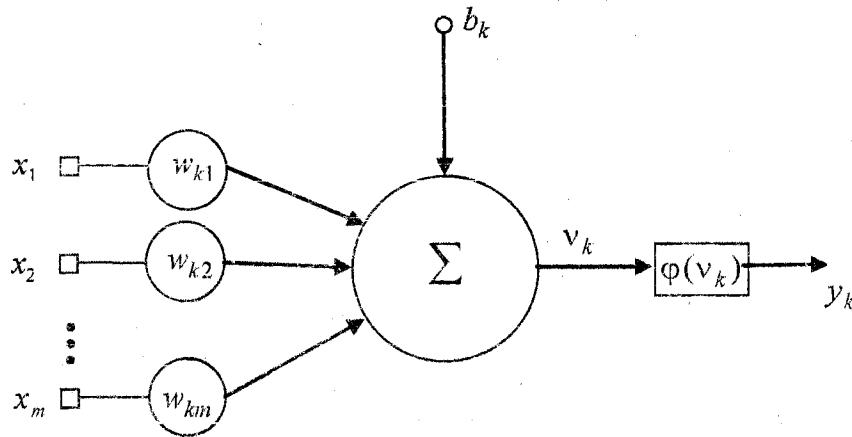


Рис. 3.25

Модель штучного нейрона з індексом  $k$  (читають „ $k$ -й нейрон”) зображено на рис. 3.25; вона має такі головні елементи.

- ♦ Значення  $x_j$  ( $j=1, \dots, m$ ) називають *вхідними значеннями нейрона*, вони формують  $m$ -вимірний вхідний вектор  $x = (x_1, \dots, x_m)^T$ .
- ♦ Існує *множина синапсів*, або *синаптичних зв'язків*. Приклад синаптичного зв'язку у вигляді фрагменту рис. 3.25 зображено на рис. 3.26.



Рис. 3.26

- ♦ Кожен із синапсів характеризується *ваговим коефіцієнтом*  $w_{kj}$ , який визначає силу зв'язку (див. рис. 3.25).
- ♦ У сукупності вагові коефіцієнти синаптичних зв'язків нейрона формують вектор  $w_k = (w_{k1}, \dots, w_{km})$ , який називають *ваговим вектором  $k$ -го нейрона*.
- ♦ *Суматор* обчислює суму зважених елементів вхідного вектора.
- ♦ Величину  $b_k$  називають *порогом нейрона*.
- ♦ Величину  $v_k$  називають *підсиленням локальним полем нейрона (induced local field)*.
- ♦ Функцію  $\phi(v_k)$  називають *активаційною функцією* штучного нейрона.
- ♦ Величину  $y_k$  називають *вихідним значенням, або реакцією* нейрона.

Модель штучного нейрона можна записати в такому вигляді:

$$\begin{cases} u_k = \sum_{j=1}^m w_{kj} x_j, \\ y_k = \phi(v_k), \\ v_k = u_k + b_k. \end{cases}$$

У своїй моделі нейрона Мак-Каллок та Пітс використовували *порогову* активаційну функцію

$$\phi(v) = \begin{cases} 1, & \text{якщо } v \geq 0, \\ 0, & \text{якщо } v < 0. \end{cases} \quad (3.2)$$

Саме тому величину  $b_k$  називають *пороговим значенням*, або *порогом*. Геометрична інтерпретація порогу штучного нейрона така: оскільки  $v_k = u_k + b_k$ , то  $b_k$  являє собою афінне перетворення зваженої суми  $u_k$ , завдяки чому підсилене локальне поле  $v_k$  не проходить через початок координат. Проілюструємо це на прикладі.

**Приклад 3.43.** Розглянемо зміст порогового значення в моделі нейрона.

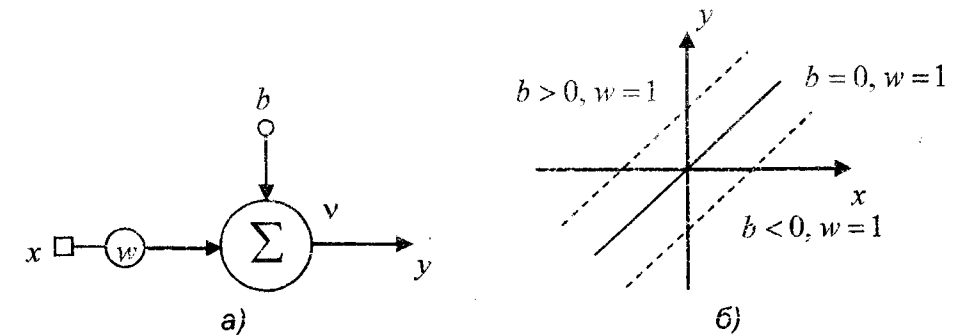


Рис. 3.27

Зображений нейрон на рис. 3.27 (а) має такі властивості:

- 1) одне вхідне значення;
- 2) активаційна функція  $\phi(v) = v$ ;
- 3) локальне підсилене поле  $v = w \cdot x + b$ .

З огляду на те, що  $\phi(v) = v$ , можна записати  $y = w \cdot x + b$ , що є рівнянням прямої (див. рис. 3.27 (б)). Отже, запропонований нейрон моделює пряму, ваговий коефіцієнт  $w$  визначає кут нахилу прямої, а поріг  $b$  визначає переміщення вздовж осі  $OY$ .

У наведеному прикладі показано, що штучний нейрон моделює пряму на площині. У загальному випадку, у разі довільної кількості вхідних значень, штучний нейрон моделює гіперплощину.

Для зручності, поріг у моделі нейрона прийнято трактувати як додаткове вхідне значення (рис. 3.28).

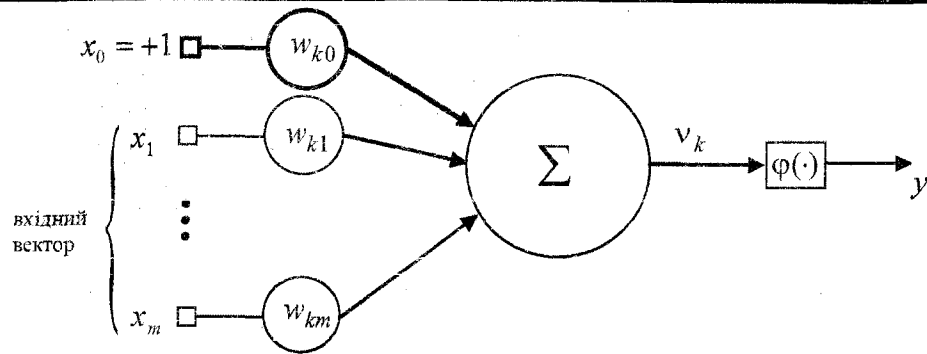


Рис. 3.28

Таку модель нейрона можна подати так:

$$\begin{cases} v_k = \sum_{j=0}^m w_{kj} x_j, \\ y_k = \varphi(v_k), \\ x_0 = +1, \\ w_{k0} = b_k. \end{cases}$$

Приклад 3.44. Обчислимо реакцію нейрона, зображеного на рис. 3.29.

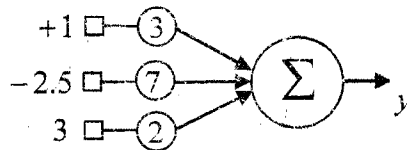


Рис. 3.29

Використаний тут нейрон має:

- ♦ вхідний вектор  $x = (+1, -2.5, 3)$ ;
- ♦ ваговий вектор  $w = (3, 7, 2)$ ;
- ♦ порогову активаційну функцію, тобто

$$\varphi(v) = \begin{cases} 1, & \text{якщо } v \geq 0; \\ 0, & \text{якщо } v < 0. \end{cases}$$

Обчислення реакції нейрона  $y$  складається із двох кроків:

1) обчислення підсиленого локального поля

$$v = \sum_{j=0}^2 w_j x_j = 3 \cdot (+1) + 7 \cdot (-2.5) + 2 \cdot 3 = -8.5;$$

2) обчислення значення активаційної функції  $\varphi(v) = 0$  (оскільки  $-8.5 < 0$ ). Звідси отримаємо реакцію нейрона  $y = 0$ .

Активаційну функцію використовують для обмеження реакції нейрона, а також для внесення нелінійності. Типова активаційна функція набуває значення з проміжку  $[0, 1]$ . Зазначимо, що поріг нейрона використовують для збільшення чи зменшення вхідного значення для активаційної функції.

Найпоширенішими активаційними функціями є порогова (3.2), кусково-лінійна (3.3), сигмоїд (3.4) та гаусіан (3.5). Графіки цих функцій зображено на рис. 3.30 (а – порогова, б – кусково-лінійна, в – сигмоїд, г – гаусіан).

$$\varphi(v) = \begin{cases} 1, & \text{якщо } v \geq 0.5, \\ v + 0.5, & \text{якщо } -0.5 < v < 0.5, \\ 0, & \text{якщо } v \leq -0.5; \end{cases} \quad (3.3)$$

$$\varphi(v) = \frac{1}{1 + \exp(-\alpha v)}; \quad (3.4)$$

$$\varphi(v) = \exp\left(-\frac{v}{\alpha}\right). \quad (3.5)$$

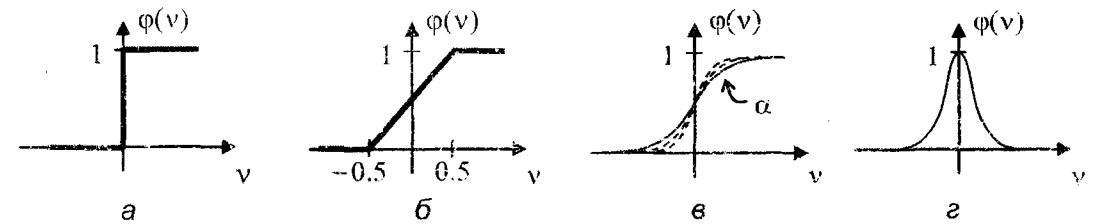


Рис. 3.30

### 3.15.3. Подання нейромереж та їхньої архітектури

Подання нейромереж за допомогою орієнтованих графів. Графом потоку сигналів називають орієнтований граф, у якого вершину  $j$  асоційовано із сигналом  $x_j$ , а дуга, яку називають напрямленим зв'язком, починається у вершині  $j$  і закінчується у вершині  $k$ . Цей зв'язок має передавальну функцію, котра визначає залежність сигналу  $y$  у вершині  $k$  від сигналу  $x_j$  у вершині  $j$ . Потік сигналів у різних частинах графа визначається трьома правилами.

1. Сигнал передається лише в напрямку стрілки. Розрізняють два типи зв'язків: синаптичні та активаційні. Поведінка синаптичних зв'язків характеризується лінійним входом/виходом: сигнал  $y$  дорівнює добутку вхідного сигналу  $x_j$  на значення синаптичної ваги  $w_j$  (рис. 3.31, а). Поведінка активаційних зв'язків характеризується нелінійним входом/виходом (рис. 3.31, б).
2. Сигнал вершини дорівнює алгебраїчній сумі всіх сигналів, які передаються по вхідних зв'язках у цю вершину як приймаючу (рис. 3.31, в). Таку конфігурацію ще називають синаптичною збіжністю або синаптичним об'єднанням по входу.
3. Сигнал  $x_j$  вершини  $j$  передається по кожному вихідному зв'язку (рис.

3.31, з). Зазначимо, що передача сигналу не залежить від передавальних функцій вихідних зв'язків. Таку конфігурацію називають *синаптичною розбіжністю*, або *синаптичним розгалуженням по виходу*.

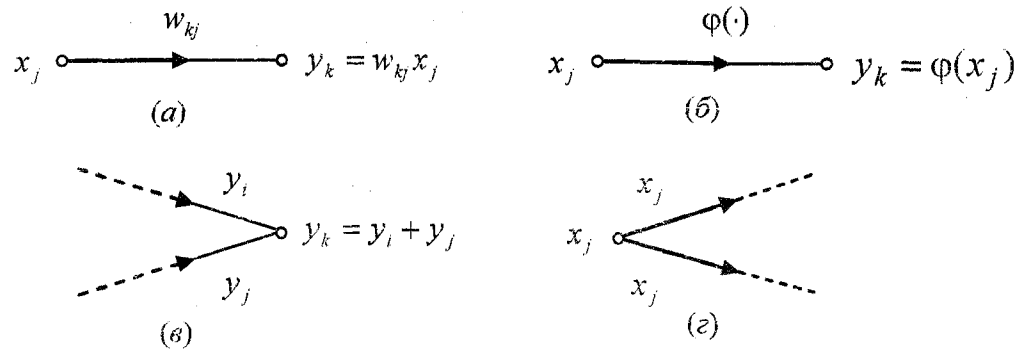


Рис. 3.31

**Приклад 3.45.** Побудуємо модель штучного нейрона Мак-Каллока та Пітса у вигляді графа потоку сигналів. Для цього використаємо модель нейрона, зображену на рис. 3.28, та введені правила побудови графа потоку сигналів. Результат зображено на рис. 3.32.

**Приклад 3.46.** На рис. 3.33 зображено нейромережу з прикладу 3.44 у вигляді графа потоку сигналів.

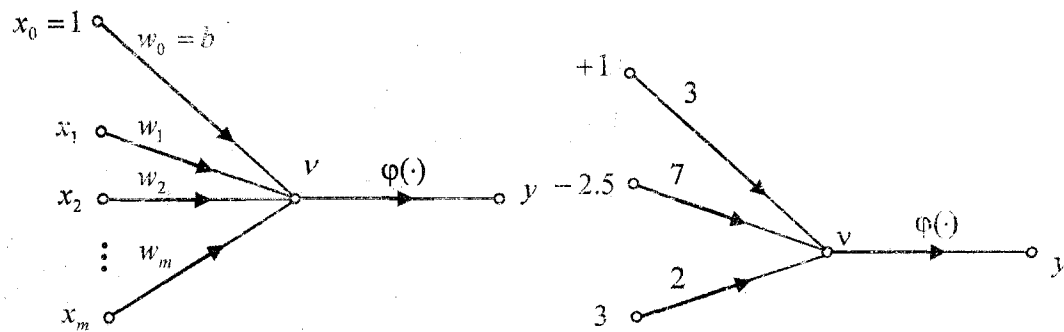


Рис. 3.32

Рис. 3.33

Нейронну мережу можна подати у вигляді орієнтованого графа, вершинам якого відповідають сигнали, а дугам – синаптичні та активаційні зв'язки. Цей граф характеризується такими властивостями:

- ◆ кожен нейрон поданий набором синаптичних зв'язків, порогом, а також активаційним зв'язком (необов'язково). Поріг зображається у вигляді синаптичного зв'язку, на який подається фіксований сигнал +1;
- ◆ синаптичні зв'язки нейрона зважують відповідні їм вхідні сигнали;
- ◆ зважена сума вхідних сигналів визначає підсилене локальне поле нейрона;
- ◆ активаційний зв'язок стискає підсилене локальне поле нейрона для породження його реакції.

Подання нейромереж у вигляді графа потоку сигналів вважається *повним* у тому розумінні, що цей граф описує не лише потік сигналів між нейронами, але й усередині самих нейронів.

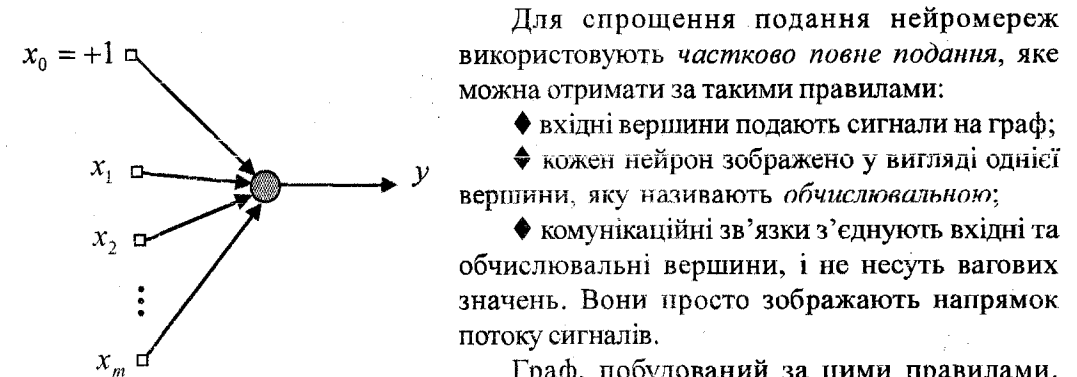


Рис. 3.34

Граф, побудований за цими правилами, називають *графом архітектури* нейромережі, або *архітектурним графом*.

**Приклад 3.47.** На рис. 3.34 зображено модель штучного нейрона Мак-Каллока та Пітса у вигляді архітектурного графа.

Підсумуємо подані способи зображення нейронів та нейромереж.

- ◆ Блок-діаграма (рис. 3.25 та 3.28) забезпечує функціональний опис нейромережі.
- ◆ Граф потоку сигналів (приклад 3.45) забезпечує повний опис потоку сигналів у нейромережі.
- ◆ Архітектурний граф (приклад 3.47) описує структуру нейромережі.

### 3.15.4. Сучасні архітектури нейромереж

Спосіб, у який нейрони утворюють нейромережу, безпосередньо пов'язаний із методом її навчання. Виділяють три фундаментальні класи архітектури нейромережі.

**1. Одношарові нейромережі прямого поширення.** Нейрони в нейромережах цього класу організовані в прошарки. У найпростішому випадку матимемо вхідний прошарок, що складається із вхідних нейронів (активаційна функція тотожна до вхідного значення), які подають сигнал на нейрони вихідного прошарку (обчислювальні елементи), і ніколи навпаки, що й означає „пряме поширення”.

Коли мова йде про кількість прошарків, то переважно йдеться про прошарки, які складаються з обчислювальних елементів. Отже, одношарові нейромережі прямого поширення складаються з одного прошарку обчислювальних елементів (рис. 3.35).

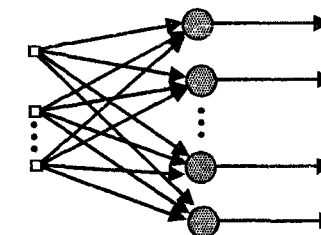


Рис. 3.35



**Приклад 3.48.** На рис. 3.36 у вигляді графа потоку сигналів зображено одношарову нейромережу прямого поширення. Обчислимо її реакцію.

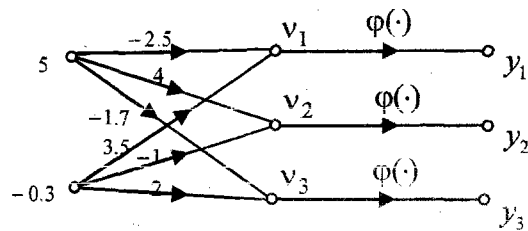


Рис. 3.36

Ця нейромережа має такі властивості:

- ◆ два вхідних елементи; поріг рівний нулю;
- ◆ три обчислювальні елементи, які є й вихідними елементами;
- ◆ активаційна функція для всіх трьох обчислювальних елементів є пороговою, тобто

$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$$

- ◆ вхідний вектор  $x = (5, -0.3)$ ;
- ◆ ваговим вектором першого обчислювального елемента є  $w_1 = (-2.5, 3.5)$ ,

другого –  $w_2 = (4, -1)$ , третього –  $w_3 = (-1.7, 2)$ ;

◆ реакція нейромережі є вектором, компоненти якого є реакціями обчислювальних елементів, тобто  $y = (y_1, y_2, y_3)$ .

Для вхідного вектора  $x = (5, -0.3)$  реакцією нейромережі є вектор  $y = (0, 1, 0)$ . Його обчислюють так.

1. Реакція першого вихідного нейрона:

$$v_1 = \sum_{j=1}^2 w_{1j} x_j = 5 \cdot (-2.5) + (-0.3) \cdot 3.5 = -13.55,$$

$$\varphi(v_1) = \begin{cases} 1, & v_1 \geq 0 \\ 0, & v_1 < 0 \end{cases} = \begin{cases} 1, & -13.55 \geq 0 \\ 0, & -13.55 < 0 \end{cases} = 0 \Rightarrow y_1 = 0.$$

2. Реакція другого вихідного нейрона:

$$v_2 = \sum_{j=1}^2 w_{2j} x_j = 5 \cdot 4 + (-0.3) \cdot (-1) = 21.3,$$

$$\varphi(v_2) = \begin{cases} 1, & v_2 \geq 0 \\ 0, & v_2 < 0 \end{cases} = \begin{cases} 1, & 21.3 \geq 0 \\ 0, & 21.3 < 0 \end{cases} = 1 \Rightarrow y_2 = 1.$$

3. Реакція третього нейрона:

$$v_3 = \sum_{j=1}^2 w_{3j} x_j = 5 \cdot (-1.7) + (-0.3) \cdot 2 = -9.1,$$

$$\varphi(v_3) = \begin{cases} 1, & v_3 \geq 0 \\ 0, & v_3 < 0 \end{cases} = \begin{cases} 1, & -9.1 \geq 0 \\ 0, & -9.1 < 0 \end{cases} = 0 \Rightarrow y_3 = 0.$$

**2. Багатшарові нейромережі прямого поширення.** Нейромережі цього класу мають ту особливість, що вони можуть мати один або декілька *прихованих прошарків*. Додавання нових прихованих прошарків збільшує здатність до виявлення закономірностей вищого порядку складності.

Нейрони вхідного прошарку подають відповідні елементи вхідного вектора на нейрони першого прихованого прошарку, який, як і у випадку одношарової нейромережі прямого поширення, складається з обчислювальних нейронів. Виходи нейронів першого прихованого прошарку подаються на нейрони другого прихованого прошарку, і так далі. Зазначимо, що переважно нейрони в кожному із прихованих прошарків одержують сигнал лише від нейронів прошарку, який знаходиться безпосередньо перед ним. Вихідні значення нейронів у останньому вихідному прошарку є загальною реакцією нейромережі.

**Приклад 3.49.** На рис. 3.37 наведено приклад багатшарової нейромережі прямого поширення. Вона складається із трьох прошарків: вхідного, прихованого та вихідного. Таку нейромережу можна описати як 3-4-2, де 3 – кількість нейронів у вхідному прошарку, 4 – кількість нейронів у прихованому прошарку, 2 – кількість нейронів у вихідному прошарку.

У загальному випадку багатшарову нейромережу прямого поширення позначають як  $m - h_1 - h_2 - \dots - h_n - q$ . Нейромережу типу розглянутої в прикладі 3.49 називають *повнозв'язною*, бо кожен нейрон кожного із прошарків з'єднаний із кожним нейроном наступного прошарку (це правило не поширюється на нейрони вихідного прошарку). Якщо принаймні один нейрон не з'єднаний із принаймні одним нейроном у наступному прошарку, то таку нейромережу називають *частково зв'язною*.

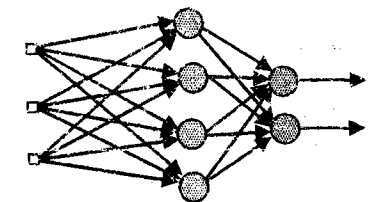


Рис. 3.37

**Приклад 3.50.** На рис. 3.38 зображено багатшарову нейромережу. Ця нейромережа має такі властивості:

- ◆ вона є багатшаровою та частково зв'язною (нейрони прихованого прошарку не зв'язані із другим нейроном у вихідному прошарку);
- ◆ вхідний прошарок складається із двох вхідних елементів; поріг рівний нулю;
- ◆ прихований прошарок складається із двох обчислювальних елементів, які з'єднані з першим нейроном у вихідному прошарку;
- ◆ вихідний прошарок містить два обчислювальні елементи;
- ◆ активаційна функція для всіх трьох обчислювальних елементів порогова, тобто

$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$$

- ♦ вхідний вектор  $x = (5, -0.3)$ ;
- ♦ ваговий вектор першого обчислювального елемента в прихованому прошарку  $w_1^{(1)} = (-2.5, 3.5)$ ; другого –  $w_2^{(1)} = (-1.7, -1)$ ;
- ♦ реакцією першого нейрона в прихованому прошарку є  $y_1^{(1)}$ , другого –  $y_2^{(1)}$ ;
- ♦ ваговий вектор першого обчислювального елемента у вихідному прошарку  $w_1^{(2)} = (1.7, -1.5)$ ; другого –  $w_2^{(2)} = (4, 2)$ ;
- ♦ реакція нейронмережі є вектором, компоненти якого є реакціями обчислювальних елементів вихідного прошарку, тобто  $y = (y_1^{(2)}, y_2^{(2)})$ .

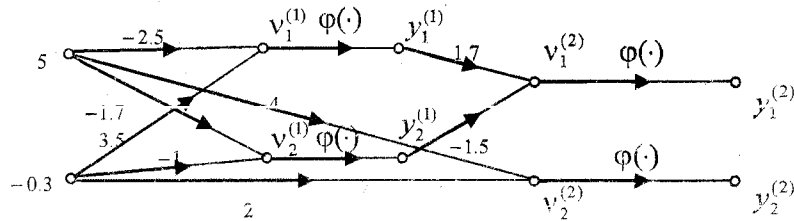


Рис. 3.38

**3. Рекурентні нейронмережі.** Відмінність рекурентних нейронмереж від нейронмереж прямого поширення полягає в існуванні хоча б одного зворотного зв'язку. Така нейронмережа може мати довільну кількість прихованих прошарків. На рис. 3.39 зображено рекурентну нейронмережу, яка складається з одного обчислювального прошарку. Кожний нейрон цього прошарку передає породжений ним сигнал до решти нейронів прошарку.

Говорять, що в динамічній системі існує зворотний зв'язок, коли вихідне значення деякого елемента системи частково впливає на вхідне значення цього ж елемента, що породжує один чи більше циклічних шляхів або петель передавання сигналів у системі.

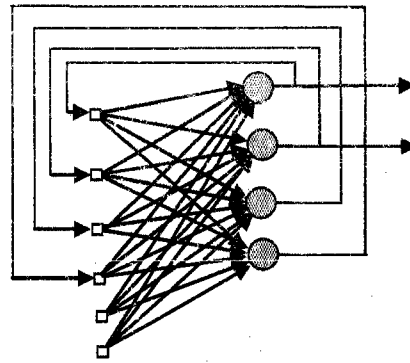


Рис. 3.39

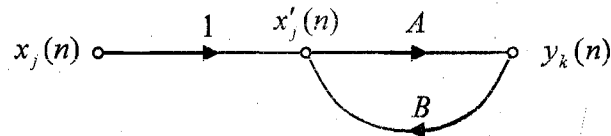


Рис. 3.40

На рис. 3.40 схематично зображено зворотний зв'язок, де вхідний сигнал  $x_j(n)$ , внутрішній сигнал  $x_j'(n)$  та вихідний сигнал  $y(n)$  є функціями дискретної змінної часу  $n$ . Система складається із прямого та зворотного зв'язків, які характеризуються операторами  $A$  та  $B$ , відповідно. Вихід прямого зв'язку частково визначається сам через себе завдяки зворотному зв'язку:

$$y(n) = A[x_j'(n)],$$

$$x_j'(n) = x_j(n) + B[y(n)].$$

Підставляючи другу із цих формул у першу, одержимо:

$$y(n) = \frac{A}{1 - AB} [x_j(n)]. \quad (3.6)$$

Оператор  $\frac{A}{1 - AB}$  називають закрито-петельним оператором системи, а оператор  $AB$  – відкрито-петельним.

**Приклад 3.51.** Наведемо приклад однопетельної системи зворотного зв'язку із затримкою сигналу. Нехай оператор  $A$  має ваговий коефіцієнт  $w$ ;  $B$  – є оператором одиничної затримки  $z^{-1}$ , чий вихід є затриманий по відношенню до входу на одну одиницю часу. Отже, можемо записати закрито-петельний оператор

$$\frac{A}{1 - AB} = \frac{w}{1 - wz^{-1}}.$$

За формулою суми нескінченної геометричної прогресії (за умови, що  $wz^{-1} < 0$ ) можемо записати:

$$\frac{A}{1 - AB} = w \sum_{l=0}^{\infty} w^l z^{-l}. \quad (3.7)$$

Підставимо (3.7) у (3.6) і одержимо

$$y(n) = w \sum_{l=0}^{\infty} w^l z^{-l} [x_j(n)].$$

За означенням оператора одиничної затримки маємо:

$$z^{-l} [x_j(n)] = x_j(n-l),$$

де  $x_j(n-l)$  – вхідне значення, затримане на  $l$  одиниць часу. Отже, остаточно можемо записати:

$$y(n) = \sum_{l=0}^{\infty} w^{l+1} x_j(n-l). \quad (3.8)$$

Зі співвідношення (3.8) легко побачити, що поведінка системи контролюється значенням вагового коефіцієнта  $w$ . Якщо  $|w| < 1$ , то вихід системи  $y(n)$  збігається, а система є *стабільною* (рис. 3.41, а). Якщо  $|w| \geq 1$ , то вихід системи  $y(n)$  розбігається, а система є *нестабільною*. Якщо  $|w| > 1$ , то розбіжність експоненціальна (рис. 3.41, в), якщо  $|w| = 1$ , то розбіжність лінійна (рис. 3.41, б).

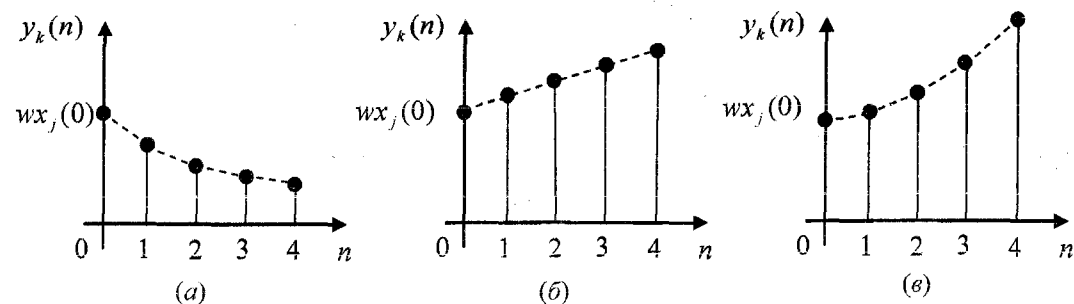


Рис. 3.41

Властивості стабільності є одними з найважливіших, що досліджують у системах зі зворотними зв'язками. Випадок  $|w| < 1$  відповідає системі з нескінченною пам'яттю в тому розумінні, що вихід системи залежить від вхідних значень із нескінченного минулого. Більш того, пам'ять „згасає” в тому розумінні, що вплив минулих вхідних значень зменшується експоненціально зі збільшенням часу  $n$ .

### 3.15.5. Навчання одношарових нейромереж прямого поширення

Головна властивість штучних нейромереж – їхня здатність навчатись з оточення та покращувати свою дієвість за допомогою навчання. Штучна нейронна мережа навчається в деякий інтерактивний спосіб, впродовж якого коректуються її ваги та пороги. В ідеалі нейромережа здобуває все більше знання про навколишнє середовище з кожною ітерацією навчального процесу.

Навчанням називають процес, упродовж якого параметри нейромережі адаптуються в результаті стимуляції середовища, у якому перебуває ця нейромережа. Тип навчання визначають способом зміни параметрів нейромережі. Означення поняття навчання передбачає таку низку подій:

- ◆ нейромережа стимулюється (або збуджується) середовищем;
- ◆ параметри нейромережі змінюються як результат стимуляції;
- ◆ нейромережа починає реагувати на середовище в інший спосіб, що зумовлено змінами її структури.

Правила проведення навчання називають *навчальним алгоритмом*. Для кожної архітектури нейромережі може існувати кілька навчальних алгоритмів, які відрізняються способом модифікації синаптичних ваг. Кожний із таких алгоритмів може мати свої переваги в навчанні.

**Навчання з коректуванням помилкою.** Нехай маємо один обчислювальний нейрон у вихідному прошарку нейромережі прямого поширення (рис. 3.42). Цей нейрон реагує на деякий вхідний вектор сигналів  $x(n) = (x_1, \dots, x_m)$ , який передається через елементи вхідного прошарку. Змінна  $n$  – це номер ітерації навчального алгоритму. Вихідне значення нейрона позначимо як  $y(n)$ . Це вихідне значення порівнюють із бажаним вихідним значенням  $d(n)$ . Отже, *сигнал помилки* можемо записати так  $e(n) = d(n) - y(n)$ .

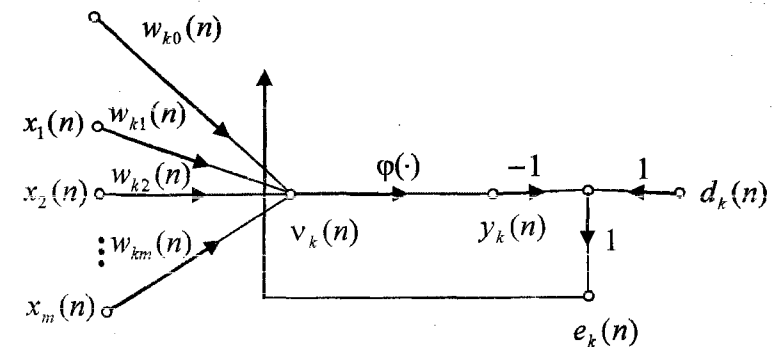


Рис. 3.42

Сигнал  $e(n)$  фактично виконує роль контролю з метою коректування синаптичних ваг нейрона. Відповідні зміни мають бути такими, щоб наблизити вихідні значення  $y(n)$  із еталонним значенням  $d(n)$ . Цього досягають мінімізації *функції ціни*

$$E(n) = \frac{1}{2} e^2(n).$$

Покрокове коректування вагових коефіцієнтів продовжують доти, доки система не досягне *стабільного стану* – це відбудеться, коли синаптичні ваги досягнуть певного граничного значення. Мінімізацію функції ціни реалізує навчальний алгоритм який називають *дельта-правилом*, або правилом Відрова (Widrow) та Гоффа (Hoff). Це правило формулюють так. Нехай  $w_j(n)$  – значення вагового коефіцієнта  $w_j$  нейрона, який на ітерації  $n$  отримав на вхід компоненту  $x_j(n)$  вектора  $x(n)$ . Величина корекції  $\Delta w_j(n)$  вагового коефіцієнта  $w_j(n)$  на ітерації  $n$  обчислюють за формулою

$$\Delta w_j(n) = \eta e(n) x_j(n),$$

де  $\eta$  – крок алгоритму.

Отже, коректування значення вагового коефіцієнта відбувається за формулою

$$w_j(n+1) = w_j(n) + \Delta w_j(n).$$

**Приклад 3.52.** Розглянемо використання нейромережі для розв'язання задачі лінійної регресії. Нехай є множина ізольованих точок, розкиданих у координатній площині. Потрібно побудувати пряму найкращого наближення для цих точок.

Це є задачею про найменші квадрати, яку в загальному вигляді можна записати співвідношенням

$$S(\beta) = \frac{1}{2} \sum_{i=1}^n (f(\beta, x_i) - d_i)^2 \rightarrow \min_{\beta \in R^p}, \quad (3.9)$$

де  $f(\beta, x)$  – шукана функція найкращого наближення загального вигляду,  $\beta \in R^p$  – вектор параметрів шуканої функції,  $(x_i, d_i)$  – множина даних, які потрібно наблизити,  $x_i \in R^m$ ,  $d_i \in R$ ,  $m \geq 1$ .

Розглянемо випадок  $x_i \in R$ ,  $d_i \in R$ , а функція  $f(\beta, x)$  є прямою, тобто  $f(\beta, x) = mx + c$ .

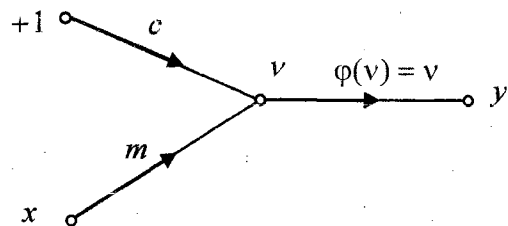


Рис. 3.43

Для побудови  $f(\beta, x)$  використаємо модель нейрона, зображену на рис. 3.43, оскільки вона моделює пряму  $y = mx + c$ . Формулу (3.9) можна записати так:

$$\frac{1}{2} \sum_{i=1}^n (y_i - d_i)^2 \rightarrow \min_{m, c}, \quad (3.10)$$

де  $y_i$  – вихід нейромережі при вхідному значенні  $x_i$ ,  $d_i$  – очікуване вихідне значення. Підставимо рівняння прямої в (3.10), тоді одержимо:

$$\frac{1}{2} \sum_{i=1}^n (mx_i + c - d_i)^2 \rightarrow \min_{m, c}.$$

Табл. 3.8 містить координати заданих точок площини. Як початкові значення коефіцієнтів шуканої прямої візьмемо  $m = 0.5$ ,  $c = 0.53$ .

Таблиця 3.8

| $x_i$ | $d_i$ |
|-------|-------|
| 0.30  | 1.60  |
| 0.35  | 1.40  |
| 0.40  | 1.40  |
| 0.50  | 1.60  |
| 0.60  | 1.70  |
| 0.80  | 2.00  |
| 0.95  | 1.70  |
| 1.10  | 2.10  |

Для моделі нейрона з рис. 3.43 рекурентні формули дельта правила для обчислення вагових коефіцієнтів можна записати так:

$$c(n+1) = c(n) + \Delta c(n), \quad \Delta c(n) = \eta e(n)1 = \eta(d(n) - y(n)),$$

$$m(n+1) = m(n) + \Delta m(n), \quad \Delta m(n) = \eta e(n)x(n) = \eta(d(n) - y(n))x(n).$$

Тут враховано, що вхідний елемент, який відповідає коефіцієнту  $c$ , дорівнює одиниці. Величина  $y(n)$  є вихідним значенням нейромережі при вхідному значенні  $x(n) \in \{x_1, \dots, x_n\}$ , а  $d(n)$  – еталонне вихідне значення. Перші сім ітерацій алгоритму навчання за дельта правилом із кроком навчання  $\eta = 0.3$  наведено нижче. Номери ітерацій виділено напівжирним шрифтом.

|  |  |
|--|--|
| 0. $y = 0.5x + 0.5$ ,<br>$x(0) = 0.30$ , $y(0) = 0.65$ , $d(0) = 1.60$ ,<br>$e(0) = 0.95$ ,<br>$\Delta c(0) = 0.3 \cdot 0.95 = 0.285$ , $c(0) = 0.5 + 0.285 = 0.785$ ,<br>$\Delta m(0) = 0.3 \cdot 0.95 \cdot 0.3 = 0.0855$ , $m(0) = 0.586$ . | 1. $y = 0.586x + 0.785$ ,<br>$x(1) = 0.35$ , $y(1) = 0.9901$ , $d(1) = 1.40$ ,<br>$e(1) = 0.4099$ ,<br>$\Delta c(1) = 0.12297$ , $c(1) = 0.90797$ ,<br>$\Delta m(1) = 0.052395$ , $m(1) = 0.638395$ ,            |
| 2. $y = 0.638395x + 0.90797$ ,<br>$x(2) = 0.40$ , $y(2) = 1.16333$ , $d(2) = 1.40$ ,<br>$e(2) = 0.23667$ ,<br>$\Delta c(2) = 0.071$ , $c(2) = 1.0507$ ,<br>$\Delta m(2) = 0.0284004$ , $m(2) = 0.6668$ .                                       | 3. $y = 0.6668x + 1.0507$ ,<br>$x(3) = 0.50$ , $y(3) = 1.3841$ , $d(3) = 1.60$ ,<br>$e(3) = 0.2159$ ,<br>$\Delta c(3) = 0.6476994$ , $c(3) = 1.115470$ ,<br>$\Delta m(3) = 0.032385$ , $m(3) = 0.699185$ .       |
| 4. $y = 0.699185x + 1.115470$ ,<br>$x(4) = 0.60$ , $y(4) = 1.534981$ , $d(4) = 1.70$ ,<br>$e(4) = 0.165019$ ,<br>$\Delta c(4) = 0.049506$ , $c(4) = 1.649757$ ,<br>$\Delta m(4) = 0.029703$ , $m(4) = 0.728888$ .                              | 5. $y = 0.728888x + 1.649757$ ,<br>$x(5) = 0.80$ , $y(5) = 2.23287$ , $d(5) = 2.00$ ,<br>$e(5) = -0.232867$ ,<br>$\Delta c(5) = -0.069860$ , $c(5) = 1.579864$ ,<br>$\Delta m(5) = -0.055888$ , $m(5) = 0.673$ . |
| 6. $y = 0.673x + 1.579864$ ,<br>$x(6) = 0.95$ , $y(6) = 2.187654$ , $d(6) = 1.70$ ,<br>$e(6) = -0.519214$ ,<br>$\Delta c(6) = -0.155764$ , $c(6) = 1.424099$ ,<br>$\Delta m(6) = -0.147976$ , $m(6) = 0.525024$ .                              | 7. $y = 0.525024x + 1.424099$ ,<br>$x(7) = 1.10$ , $y(7) = 2.001626$ , $d(7) = 2.10$ ,<br>$e(7) = 0.098374$ ,<br>$\Delta c(7) = 0.029512$ , $c(7) = 1.453612$ ,<br>$\Delta m(7) = 0.032463$ , $m(7) = 0.55749$ . |

У результаті отримано пряму  $y = 0.56x + 1.45$ . Для порівняння, метод найменших квадратів дає пряму  $y = 0.72x + 1.24$ .

**Лінійна та нелінійна роздільність даних.** Розглянемо приклади використання одношарових нейронних мереж для реалізації булевих функцій та обмеження, які існують у застосовності одношарових нейромереж.

**Приклад 3.53.** Обчислимо логічну функцію AND за допомогою нейромережі. Функція AND є булевою функцією двох змінних, яку задано табл. 3.9. На рис. 3.44 зображено геометричну інтерпретацію функції AND. Вхідні значення зображені точками площини, координати яких записано в круглих дужках. Значення самої функції записано в квадратних дужках. На рис. 3.44 можна побачити, що точки, у яких функція AND набуває значення 1, і точки, у яких вона набуває значення 0, можна відділити прямою лінією. Дані, які можна розділи прямою, а в загальному випадку, гіперплощиною, називають *лінійно роздільними*.

Таблиця 3.9

| $x$ | $y$ | $z$ |
|-----|-----|-----|
| 0   | 0   | 0   |
| 0   | 1   | 0   |
| 1   | 0   | 0   |
| 1   | 1   | 1   |

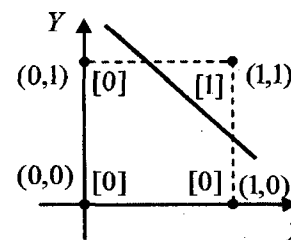


Рис. 3.44

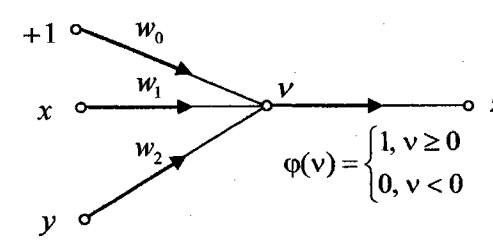


Рис. 3.45

Отже, для обчислення функції AND потрібно побудувати нейромережу, яка моделює пряму та надає значення 1 для точок, які лежать вище прямої, та 0 для точок, які лежать нижче прямої. На рис. 3.45 зображено модель нейромережі, яка має два вхідні та один обчислювальний нейрони, який є вихідним, з пороговою активаційною функцією. Така нейромережа моделює функцію AND за умови правильного підбору вагових коефіцієнтів.

**Приклад 3.54.** Обчислимо логічну функцію OR за допомогою нейромережі. Функцію OR задано табл. 3.10. На рис. 3.46 подано геометричну інтерпретацію функції OR. З рис. 3.5 видно, що точки, у яких функція OR набуває значення 1, і точки, у яких вона набуває значення 0, можна розділити прямою.

Для обчислення функції OR побудуємо нейромережу, яка б моделювала пряму та надавала значення 1 для точок, які лежать вище прямої та 0 для точок, які лежать нижче прямої. Модель нейромережі, яка була використана в прикладі 3.53 для реалізації функції AND, можна використати й для реалізації функції OR за умови зміни активаційної функції.

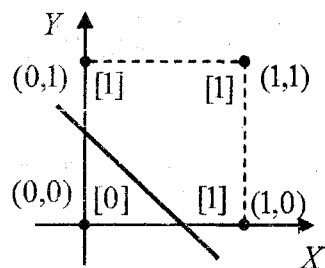


Рис. 3.46

Таблиця 3.10

| x | y | z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Приклад 3.55.** Булева функція XOR, задана табл. 3.11, є прикладом даних, які не можна розділити прямою. На рис. 3.47 подано геометричну інтерпретацію функції XOR.

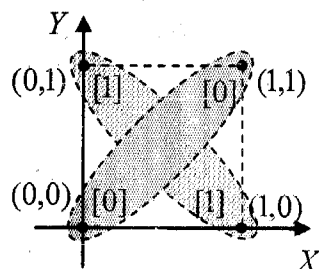
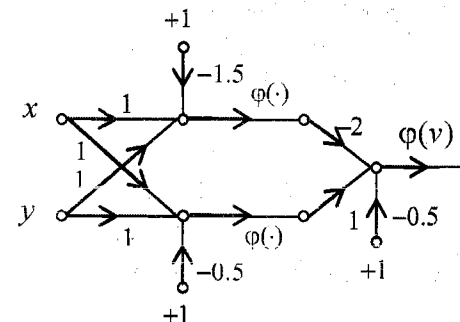


Рис. 3.47

З рис. 3.47 можна побачити, що точки, у яких функція XOR набуває значення 1, і точки, у яких вона набуває значення 0, не можна розділити прямою. Дані, які не можна розділити прямою, називають *лінійно нероздільними*, або *нелінійно роздільними*.

Для реалізації функції XOR недостатньо одношарової нейромережі, яка може моделювати лише пряму. Віддільність пари точок (0,1) та (1,0) від пари точок (0,0) та (1,1) може забезпечити багатошарова нейромережа. Одну з можливих конструкцій нейромережі для обчислення функції XOR зображено на рис. 3.48.



$$\phi(v) = \begin{cases} 1, & \text{якщо } v \geq 0, \\ 0, & \text{якщо } v < 0. \end{cases}$$

Рис. 3.48

Перевірити те, що запропонована нейромережа справді реалізовує функцію XOR, можна звичайною підстановкою всіх чотирьох вхідних значень із таблиці 1.4.

Покажемо, як саме запропонована нейромережа реалізує функцію XOR.

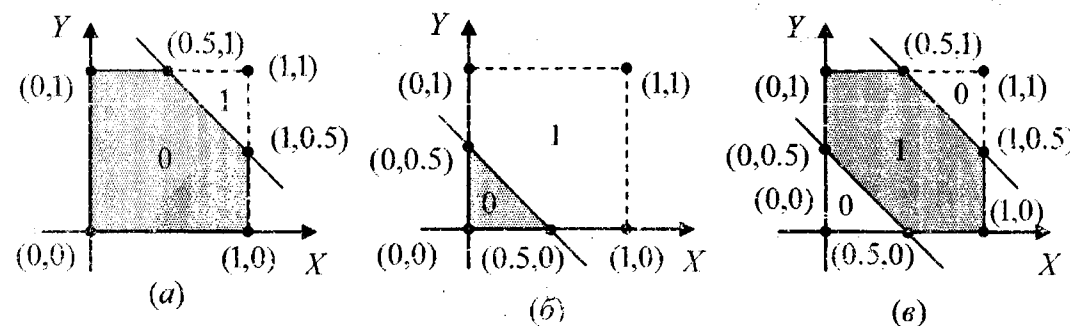


Рис. 3.49

Підсилене локальне поле першого нейрона в прихованому прошарку  $v_1^{(1)} = x + y - 1.5$ , тобто нейрон моделює пряму  $y = -x + 1.5$ . Активаційна функція є пороговою, отже, цей нейрон розбиває вхідний простір на дві півплощини – уверх від прямої  $y = -x + 1.5$ , де вихід нейрона дорівнює 1, та вниз від прямої, де вихід нейрона дорівнює 0 (рис. 3.49, а).

Підсилене локальне поле другого нейрона в прихованому прошарку  $v_2^{(1)} = x + y - 0.5$ , тобто нейрон моделює пряму  $y = -x + 0.5$ . З огляду на це та те, що активаційна функція є пороговою, цей нейрон розбиває вхідний простір на дві півплощини – уверх від прямої  $y = -x + 0.5$ , де вихід нейрона дорівнює 1, та вниз від прямої, де вихід нейрона дорівнює 0 (рис. 3.49, б).

Вихідний нейрон нейромережі будує лінійну комбінацію областей прийняття рішень прихованих нейронів так, що утворюється область прийняття рішень, зображена на рис. 3.49, в. Другий нейрон у прихованому прошарку має додатний зв'язок із вихідним нейроном, його називають *збудником*. Перший же нейрон має від'ємний зв'язок із вихідним нейроном, у такому разі кажуть, що нейрон має *функцію пригнічення*.

Коли обидва приховані нейрони мають вихід 0 при вхідному значенні (0, 0), вихідний нейрон також набуває значення 0. Коли обидва приховані нейрони мають вихід 1 при вхідному значенні (1, 1), вихідний нейрон знову набуває значення 0, оскільки пригнічення першим прихованим нейроном, спричинене більшим від'ємним значенням ваги,

пересилює збуджувальний сигнал, спричинений другим прихованим нейроном, який передається по зв'язку з додатною вагою. Для вхідних значень (0, 1) чи (1, 0) вихід першого прихованого нейрона дорівнює нулю, а другого – одиниці, відтак реакція вихідного нейрона дорівнює 1.

### 3.15.6. Навчання багатозарових нейромереж прямого поширення

Розглянемо алгоритм зворотного поширення помилки, який є першим алгоритмом, що був запропонований для навчання багатозарових нейромереж прямого поширення, і залишається одним із найбільш уживаних та важливих навчальних алгоритмів. Спершу розглянемо деякі важливі результати.

**Метод найшвидшого спуску навчання одношарових нейромереж прямого поширення.** Нехай  $\varepsilon(w)$  є дійснозначною та неперервно-диференційовною функцією векторного аргументу  $w$ . Цю функцію використовують для обчислення значень вектора  $w$  з метою оптимального функціонування нейромережі, для якої вектор  $w$  є множиною синаптичних ваг.

Наше завдання – знайти такий вектор  $w^*$ , що виконується умова

$$\varepsilon(w^*) \leq \varepsilon(w).$$

Це означає, що потрібно розв'язати задачу оптимізації без обмежень, тобто

$$\varepsilon(w) \rightarrow \min_w.$$

Запишемо необхідну умову мінімуму

$$\nabla \varepsilon(w^*) = 0,$$

де  $\nabla$  – оператор градієнта

$$\nabla = \left[ \frac{\partial}{\partial w_1}, \dots, \frac{\partial}{\partial w_m} \right]^T.$$

Для  $\nabla \varepsilon(w)$  маємо

$$\nabla \varepsilon(w) = \left[ \frac{\partial \varepsilon}{\partial w_1}, \dots, \frac{\partial \varepsilon}{\partial w_m} \right]^T.$$

Ідея розв'язування задач мінімізації без обмежень ґрунтується на застосуванні ітеративної процедури локального спуску, яка, починаючи з певного початкового значення  $w(0)$ , будує послідовність векторів  $w(1), w(2), \dots$ , та для якої виконується умова

$$\varepsilon(w(n+1)) < \varepsilon(w(n)). \quad (3.11)$$

Існують різні ітераційні методи, які, у загальному випадку, забезпечують локальну збіжність, тобто знайдений розв'язок  $w^*$  забезпечує мінімальне значення функції лише у певному локальному, а не глобальному розумінні. Одним із таких методів є метод найшвидшого спуску.

У методі найшвидшого спуску вагові вектори коректують у напрямку, протилежному до напрямку градієнту

$$q = \nabla \varepsilon(w).$$

Обчислення виконують за формулою

$$w(n+1) = w(n) - \eta q(n), \quad (3.12)$$

де  $\eta > 0$  – стала величина, яка визначає довжину кроку;  $q(n)$  – значення градієнту у точці  $w(n)$ . Приріст вектора ваг  $\Delta w(n)$  має вигляд

$$\Delta w(n) = w(n+1) - w(n) = -\eta q(n). \quad (3.13)$$

Покажемо, що метод найшвидшого спуску забезпечує виконання умови (3.11). Для цього розкладемо функцію  $\varepsilon(w)$  у ряд Тейлора першого порядку в околі точки  $w(n+1)$

$$\varepsilon(w(n+1)) \approx \varepsilon(w(n)) + q^T(n) \Delta w(n). \quad (3.14)$$

Підставимо (3.13) в (3.14) та одержимо

$$\varepsilon(w(n+1)) \approx \varepsilon(w(n)) - \eta q^T(n) q(n) = \varepsilon(w(n)) - \eta \|q(n)\|^2. \quad (3.15)$$

З формули (3.15) видно, що для додатного значення  $\eta$  виконана умова (3.11).

**Навчання одношарових нейромереж методом найменших квадратів.** У методі найменших квадратів (Least-Mean-Square, LMS) функцію помилки  $\varepsilon(w)$  обчислюють за формулою

$$\varepsilon(w) = \frac{1}{2} e^2(n),$$

де  $e(n)$  – значення помилки в момент часу  $n$ . Диференціюванням  $\varepsilon(w)$  за  $w$  отримаємо

$$\frac{\partial \varepsilon(w)}{\partial w} = e(n) \frac{\partial e(n)}{\partial w}. \quad (3.16)$$

Алгоритм LMS використовують для нейромереж із лінійним виходом нейронів (без активаційної функції), отже

$$e(n) = d(n) - x^T(n)w(n), \quad (3.17)$$

де  $x^T(n)w(n)$  – вихід нейромережі. Використовуючи (3.17), співвідношення (3.16) можна переписати так

$$\frac{\partial \varepsilon(w)}{\partial w} = -x(n)e(n).$$

Оскільки  $q = \nabla \varepsilon(w)$ , то

$$q(n) = -x(n)e(n). \quad (3.18)$$

Підстановкою (3.18) в (3.12) одержимо

$$w(n+1) = w(n) + \eta x(n)e(n). \quad (3.19)$$

Формула (3.19) визначає метод LMS, і за змістом є методом дельта правила.



Зазначимо, що обчислення градієнта з використанням (3.18) є наближеним порівняно з тим, яке використовують у методі найшвидшого спуску.

**Модель багат шарових нейромереж прямого поширення, придатних для навчання алгоритмом зворотного поширення помилки.** На рис. 3.50. зображено приклад багат шарової нейромережі прямого поширення. Вона складається з одного вхідного прошарку, одного вихідного прошарку та двох прихованих прошарків нейронів. Ця нейромережа є повнозв'язною, оскільки кожний нейрон будь-якого прошарку з'єднаний із кожним нейроном наступного. Потік сигналів у такій нейромережі прямує зліва направо.

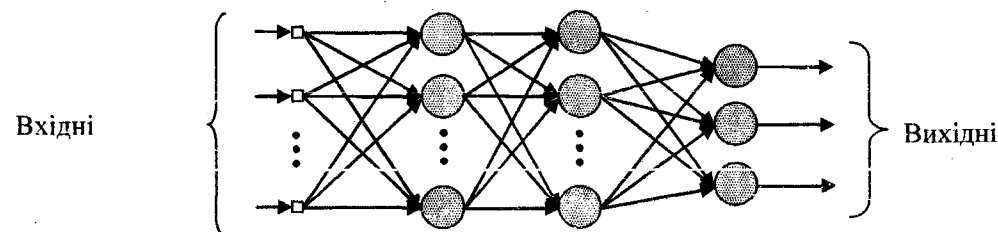


Рис. 3.50

У зображеної багат шарової нейромережі де є два типи зв'язків.

♦ **Функціональні сигнали.** Функціональним сигналом є вхідний сигнал який входить у нейромережу через вхідні нейрони, проходить через нейрони прихованих прошарків по синаптичних зв'язках, виходить через вихідні нейрони і є виходом нейромережі. Цей сигнал називають *функціональним* із двох причин: по-перше, вважається, що ці сигнали відіграють певну функцію на виході нейромережі; по-друге, у кожному нейроні, через який ці сигнали проходять, обчислюється функція входів, яка зважується ваговими коефіцієнтами.

♦ **Сигнали помилки.** Сигнал помилки утворюється на виході вихідних нейронів, і поширюється в зворотному напрямку (справа наліво) через нейромережу. Цей сигнал називається *сигналом помилки*, оскільки його обчислення кожним нейроном у нейромережі залежить від уведеної функції помилки.

На рис. 3.51 зображено фрагмент багат шарової нейромережі прямого поширення із зображенням потоку сигналів помилки та функціональних сигналів.

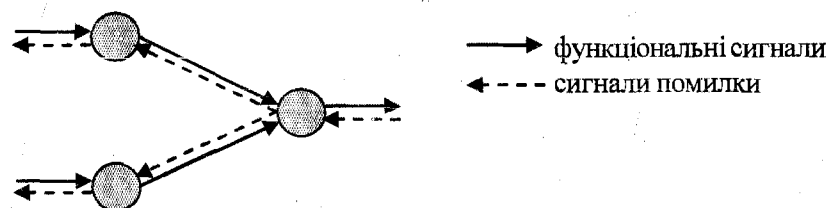


Рис. 3.51

Вихідні нейрони, які є обчислювальними, утворюють вихідний прошарок. Решту обчислювальних нейронів, які не містяться у вихідному прошарку, називають

*прихованими*, вони утворюють приховані прошарки. Кожний прихований чи вихідний нейрон виконує два види обчислень:

- ♦ обчислення функціональних сигналів як неперервної та нелінійної активаційної функції суми зважених вхідних сигналів;
- ♦ обчислення наближеного значення градієнту зі зворотним проходженням сигналу помилки через нейромережу.

**Метод зворотного поширення помилки.** Уведемо такі позначення.

♦  $i, j$  та  $k$  – індекси для позначення нейронів нейромережі. Якщо сигнал поширюється зліва направо, то вважатимемо, що нейрон  $j$  розташований у прошарку праворуч по відношенню до прошарку з нейроном  $i$ , а нейрон  $k$  – у прошарку праворуч від нейрона  $j$  для прихованого нейрона  $j$ .

- ♦  $n$  – момент часу, у який на нейромережу подано  $n$ -й навчальний приклад.
- ♦  $d_j(n)$  – еталонне значення виходу вихідного нейрона  $j$  на ітерації  $n$ .
- ♦  $y_j(n)$  – функціональний вихід нейрона  $j$  на ітерації  $n$ .
- ♦  $e_j(n)$  – похибка нейрона  $j$  у вихідному прошарку на ітерації  $n$ , яку обчислюють за правилом

$$e_j(n) = d_j(n) - y_j(n), \quad (3.20)$$

♦  $\varepsilon(n)$  – сумарна квадратична похибка нейромережі на кроці  $n$  алгоритму навчання, яку ще називають енергією помилки та обчислюють за формулою

$$\varepsilon(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n), \quad (3.21)$$

де  $C$  – множина вихідних нейронів

♦  $E$  – узагальнена похибка нейромережі для усієї навчальної множини обчислюватимемо за формулою

$$E = \frac{1}{N} \sum_{n=1}^N \varepsilon(n).$$

♦  $w_{ji}(n)$  – ваговий коефіцієнт зв'язку, який з'єднує нейрон  $i$  із нейроном  $j$  на ітерації  $n$ . Корекцію ваги  $w_{ji}(n)$  на ітерації  $n$  позначатимемо  $\Delta w_{ji}(n)$ .

♦  $v_j(n)$  – підсилене локальне поле нейрона  $j$  на ітерації  $n$ .

♦  $\phi_j(n)$  – активаційна функція нейрона  $j$ .

♦  $b_j$  – порогове значення нейрона  $j$  фактично є ваговим коефіцієнтом  $w_{j0} = b_j$  зв'язку по якому передається фіксований сигнал  $+1$ .

♦  $x(n)$  – вхідний вектор на ітерації  $n$ ,  $x_i(n)$  – його компонента.

♦  $o(n)$  – вихідний вектор нейромережі на ітерації  $n$ , який складається із реакцій нейронів у вихідному прошарку,  $o_k(n)$  – його компонента.

♦  $\eta$  – крок навчання.

♦  $m_l$  – кількість нейронів у прошарку  $l = 0, 1, \dots, L$ , де  $L$  – глибина нейромережі;  $m_0$  – кількість нейронів у вхідному прошарку;  $m_1$  – кількість нейронів у першому прихованому прошарку;  $m_L$  – кількість нейронів у вихідному прошарку.

Зазначимо, що функція  $\varepsilon(n)$  є сумарною помилкою усіх нейронів у вихідному прошарку. Для заданої навчальної множини функція  $E$  є функцією помилки, яка визначає

якість навчання нейромережі. Ціль навчання полягає в пошуку таких значень синаптичних ваг, які мінімізують цільову функцію.

Розглядатимемо метод навчання, у якому кожне вхідне значення з навчальної множини модифікує ваги мережі. Опрацювання всіх значень із навчальної множини називають *epoxoю*. Нехай нейрон  $j$  приймає сигнали від нейронів прошарку, розташованого зліва від нього (див. рис. 3.52). Підсилене локальне поле цього нейрона обчислимо за формулою

$$v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n), \quad (3.22)$$

де  $m+1$  – кількість вхідних сигналів разом із сигналом порогового нейрона. Сигнал нейрона  $j$  на ітерації  $n$  обчислюється так

$$y_j(n) = \phi(v_j(n)). \quad (3.23)$$

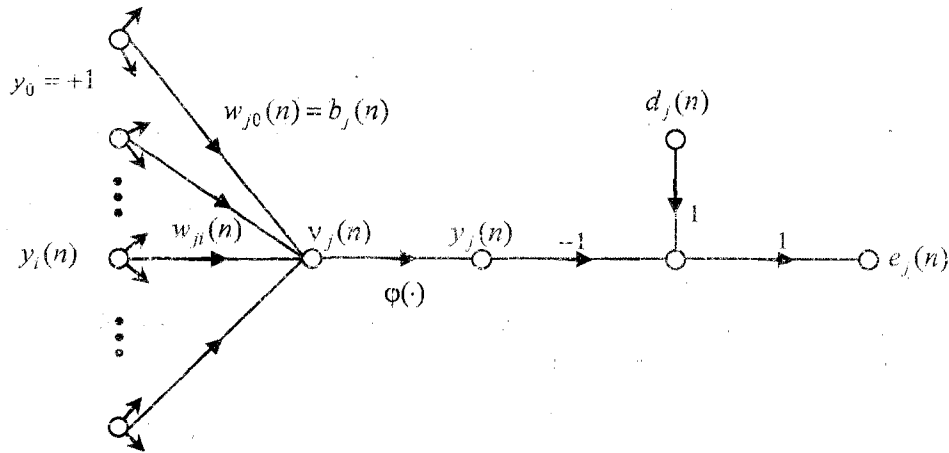


Рис. 3.52

Метод зворотного поширення помилки використовує  $\Delta w_{ji}(n)$  для коректування ваги  $w_{ji}(n)$ , значення якої обчислюють пропорційно частинній похідній  $\frac{\partial \epsilon(n)}{\partial w_{ji}(n)}$ . За правилом диференціювання суперпозиції функцій одержимо

$$\frac{\partial \epsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \epsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}. \quad (3.24)$$

Частинна похідна  $\frac{\partial \epsilon(n)}{\partial w_{ji}(n)}$ , яку називають *фактором чутливості*, визначає напрямок у просторі вагових коефіцієнтів для пошуку оптимального значення  $w_{ji}(n)$ . Обчислимо складові формули (3.24).

Величини  $\frac{\partial \epsilon(n)}{\partial e_j(n)}$  обчислимо диференціюванням (3.21)

$$\frac{\partial \epsilon(n)}{\partial e_j(n)} = e_j(n). \quad (3.25)$$

Диференціюванням (3.20) по  $y_j(n)$ , одержимо:

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1, \quad (3.26)$$

диференціюванням (3.23) по  $v_j(n)$  одержимо

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \phi'_j(v_j(n)), \quad (3.27)$$

і, нарешті, диференціюванням (3.22), одержимо

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n). \quad (3.28)$$

Підстановкою (3.25) – (3.28) в (3.24) одержимо

$$\frac{\partial \epsilon(n)}{\partial w_{ji}(n)} = -e_j(n) \phi'_j(v_j(n)) y_i(n). \quad (3.29)$$

Величину  $\Delta w_{ji}(n)$  визначимо за дельта-правилом

$$\Delta w_{ji}(n) = -\eta \frac{\partial \epsilon(n)}{\partial w_{ji}(n)}, \quad (3.30)$$

де  $\eta$  – крок навчання. Знак мінус у співвідношенні (3.30) використано для утворення антиградієнту. З (3.29) та (3.30) можна отримати

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n), \quad (3.31)$$

де величину  $\delta_j(n)$  називають *локальним градієнтом* і обчислюють за формулою

$$\delta_j(n) = -\frac{\partial \epsilon(n)}{\partial v_j(n)} = -\frac{\partial \epsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \phi'_j(v_j(n)). \quad (3.32)$$

Локальний градієнт указує напрямок зміни значення вагового коефіцієнта. Відповідно до (3.32) локальний градієнт для нейрона  $j$  обчислюють як добуток сигналу помилки цього нейрона на похідну його активаційної функції, яка є функцією однієї змінної.

З (3.31) та (3.32) можна побачити, що головна складова в обчисленні  $\Delta w_{ji}(n)$  – це сигнал помилки  $e_j(n)$ . Окремо розглядають обчислення сигналу помилки для нейронів вихідного прошарку та для прихованих прошарків.

**Випадок 1: нейрон  $j$  – вихідний.** Для вихідних нейронів існує набір еталонних значень, тому сигнал помилки можна обчислити за співвідношенням (3.20). Отже, обчислити локальний градієнт за формулою (3.32) не складно.

**Випадок 2: нейрон  $j$  – прихований.** Для нейронів прихованих прошарків не існує заданих явно еталонів вихідних сигналів. Тому в цьому випадку знаходження сигналу помилки виконують рекурсивно, використовуючи сигнали помилок нейронів з якими вони безпосередньо з'єднані. Розглянемо рис. 3.53, на якому зображено граф потоку сигналів від прихованого нейрона  $j$  до вихідного нейрона  $k$ . За формулою (3.32) локальний градієнт для прихованого нейрона  $j$  можна записати у вигляді

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \varphi'_j(v_j(n)). \quad (3.33)$$

Для обчислення частинної похідної  $\frac{\partial \varepsilon(n)}{\partial y_j(n)}$  скористаємось співвідношенням (3.21),

у якому замінимо індекс  $j$  на  $k$  для узгодження з рис. 3.53, у якому вихідні нейрони позначено індексом  $k$ :

$$\varepsilon(n) = \frac{1}{2} \sum_{k \in L} e_k^2(n). \quad (3.34)$$

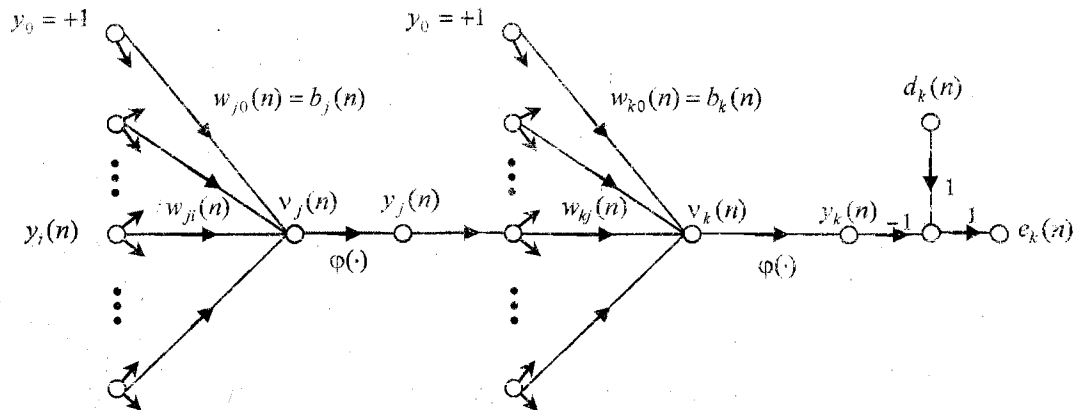


Рис. 3.53

Диференціюванням (3.34) по  $y_j(n)$  одержимо

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)}. \quad (3.35)$$

За правилом диференціювання суперпозиції функцій для обчислення  $\frac{\partial e_k(n)}{\partial y_j(n)}$

запишемо

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}. \quad (3.36)$$

Оскільки для вихідних нейронів  $e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n))$ , то

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi'_k(v_k(n)). \quad (3.37)$$

Зазначимо, що відповідно до рис. 1.53 підсилене локальне поле вихідного нейрона  $k$  обчислюють так

$$v_k(n) = \sum_{j=0}^m w_{kj}(n) y_j(n). \quad (3.38)$$

Диференціюванням (3.38) по  $y_j(n)$  одержимо

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n). \quad (3.39)$$

Підстановкою (3.37) та (3.39) у (3.36) отримаємо

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = -\sum_k e_k(n) \varphi'_k(v_k(n)) w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n). \quad (3.40)$$

і для отримання остаточного виразу залишилось скористатися формулою (3.32).

З (3.33) та (3.40) для прихованого нейрона  $j$  остаточного запишемо формулу для обчислення помилки

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n). \quad (3.41)$$

На рис. 3.54 формулу (3.41) схематично подано у вигляді потоку сигналів, де  $m_L$  – кількість нейронів у вихідному прошарку.

Отже, підсумовуючи сказане, сформулюємо метод зворотного поширення помилки.

1. Ваговий коефіцієнт синаптичного зв'язку  $w_{ji}(n)$  між нейронами  $i$  та  $j$  модифікується додаванням величини  $\Delta w_{ji}(n)$ , яку обчислюють за узагальненим дельта правилом

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n), \quad (3.42)$$

де  $\eta$  – крок методу,  $\delta_j(n)$  – локальний градієнт нейрона  $j$ ,  $y_i(n)$  – вхідний сигнал нейрона  $j$ .

2. Обчислення локального градієнту  $\delta_j(n)$  залежить від того, прихований чи вихідний нейрон  $j$ :

◆ якщо нейрон  $j$  вихідний, то  $\delta_j(n)$  обчислюють за формулою (3.32) як добуток похідної активаційної функції нейрона  $\phi'_j(v_j(n))$  і сигналу помилки  $e_j(n)$ , останній знаходять як різницю еталонної та фактичної реакції;

◆ якщо нейрон  $j$  прихованим, то  $\delta_j(n)$  обчислюють за формулою (3.41) як добуток похідної активаційної функції нейрона  $\phi'_j(v_j(n))$  і зваженої суми  $\sum_k \delta_k(n) w_{kj}(n)$ , обчисленої для нейронів наступного прошарку, з якими з'єднаний нейрон  $j$  (цей прошарок може бути як прихованим, так і вихідним).

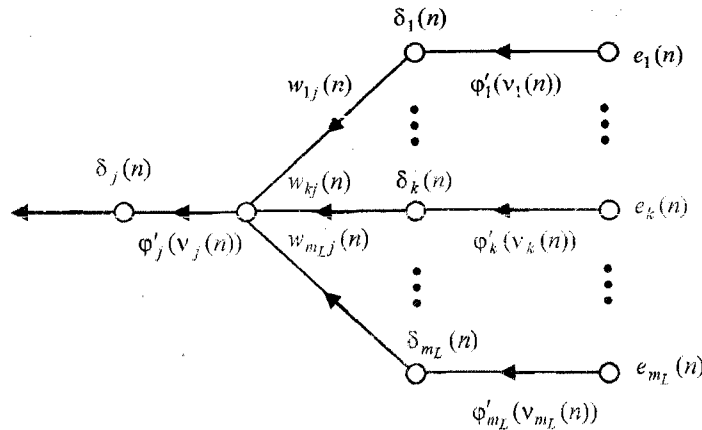


Рис. 3.54

**Приклад 3.56.** Виконаємо одну ітерацію навчання нейромережі за методом зворотного поширення помилки. Вигляд нейромережі подана на рис. 3.55. Вагові вектори нейронів прихованого прошарку –  $(w_{11}^{(1)}, w_{12}^{(1)}) = (0.5, 1.0)$  та  $(w_{21}^{(1)}, w_{22}^{(1)}) = (1.0, 0.5)$  для першого та другого нейронів відповідно. Ваговий вектор вихідного нейрона –  $(w_{11}^{(2)}, w_{12}^{(2)}) = (0.5, -0.5)$ . Зазначимо, що верхній індекс у дужках означає номер прошарку.

Активаційною функцією всіх нейронів є логістична функція  $\phi(v) = \frac{1}{1 + \exp(-av)}$ ; iii

похідна  $\phi'(v) = \frac{a \exp(-av)}{[1 + \exp(-av)]^2} = a\phi(v)(1 - \phi(v))$ . Оскільки реакція нейрона  $y = \phi(v)$ ,

то

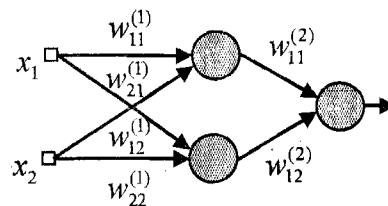


Рис. 3.55

похідну можна записати в термінах реакції нейрона як  $\phi'(v) = ay(1 - y)$ .

Виконаємо крок алгоритму зворотного поширення за таких умов:

◆ вхідний вектор  $(0.5, 1)$ ; еталонне вихідне значення  $0.7$ .

◆ параметр активаційної функції  $a = 1$ ; крок алгоритму  $\eta = 0.3$ .

**Хід уперед (обчислення реакції нейромережі): номер ітерації  $n = 1$ .**

Реакція першого нейрона прихованого прошарку:

$$\begin{cases} v_1^{(1)}(1) = 0.5 \cdot 0.5 + 1 \cdot 1 = 1.25, \\ y_1^{(1)}(1) = \phi(v_1^{(1)}(1)) = \frac{1}{1 + \exp(-1.25)} = 0.7773. \end{cases}$$

Реакція другого нейрона прихованого прошарку:

$$\begin{cases} v_2^{(1)}(1) = 0.5 \cdot 1 + 1 \cdot 0.5 = 1.0, \\ y_2^{(1)}(1) = \phi(v_2^{(1)}(1)) = \frac{1}{1 + \exp(-1)} = 0.7311. \end{cases}$$

Маючи реакції нейронів прихованого прошарку, обчислимо реакцію вихідного нейрона:

$$\begin{cases} v_1^{(2)}(1) = 0.5 \cdot 0.7773 - 0.5 \cdot 0.7311 = 0.0231, \\ y_1^{(2)}(1) = \phi(v_1^{(2)}(1)) = \frac{1}{1 + \exp(-0.0231)} = 0.5058. \end{cases}$$

**Хід назад (обчислення помилки, та її поширення назад для коректування ваг).**

Обчислення зворотного ходу розпочинаємо з обчислення помилки реакції нейромережі (3.20), локального градієнту (3.32) та коректування ваг вихідного нейрона (3.42):

$$\begin{cases} e_1(1) = 0.7 - 0.5058 = 0.1942, \\ \delta_1^{(2)}(1) = 0.1942 \cdot (0.5058 \times (1 - 0.5058)) = 0.1942 \cdot 0.25 = 0.0486; \\ \Delta w_{11}^{(2)}(1) = 0.3 \cdot 0.0486 \cdot 0.7773 = 0.01132, \\ w_{11}^{(2)}(2) = 0.5 + 0.01132 = 0.51132; \\ \Delta w_{12}^{(2)}(1) = 0.3 \cdot 0.0486 \cdot 0.7311 = 0.0106, \\ w_{12}^{(2)}(1) = -0.5 \cdot 0.0106 = -0.4894. \end{cases}$$

Далі, обчислимо локальний градієнт для першого нейрона в прихованому прошарку (3.41) та проведемо коректування його вагового вектора (3.42):

$$\begin{cases} \delta_1^{(1)}(1) = \varphi'(v_1^{(1)}(1)) \cdot \left( \sum_{k=1}^1 \delta_k^{(2)}(1) \cdot w_{k1}^{(2)}(1) \right) = 0.1731 \cdot (0.0486 \cdot 0.5) = 0.0042, \\ \begin{cases} \Delta w_{11}^{(1)}(1) = \eta \cdot \delta_1^{(1)}(1) \cdot y_1^{(0)}(1) = 0.3 \cdot 0.0042 \cdot 0.5 = 0.00063, \\ w_{11}^{(1)}(2) = 0.5 + 0.00063 = 0.50063; \end{cases} \\ \begin{cases} \Delta w_{12}^{(1)}(1) = \eta \cdot \delta_1^{(1)}(1) \cdot y_2^{(0)}(1) = 0.3 \cdot 0.0042 \cdot 1 = 0.00126, \\ w_{12}^{(1)}(1) = 1 + 0.00126 = 1.00126. \end{cases} \end{cases}$$

Тепер обчислимо локальний градієнт для другого нейрона в прихованому прошарку (3.41) та проведемо коректування його вагового вектора (3.42):

$$\begin{cases} \delta_2^{(1)}(1) = \varphi'(v_2^{(1)}(1)) \cdot \left( \sum_{k=1}^1 \delta_k^{(2)}(1) \cdot w_{k2}^{(2)}(1) \right) = 0.1966 \cdot (0.0486 \times (-0.5)) = -0.0048; \\ \begin{cases} \Delta w_{21}^{(1)}(1) = \eta \cdot \delta_2^{(1)}(1) \cdot y_1^{(0)}(1) = 0.3 \cdot (-0.0048) \cdot 0.5 = -0.00072, \\ w_{21}^{(1)}(2) = 1 - 0.00072 = 0.9993; \end{cases} \\ \begin{cases} \Delta w_{22}^{(1)}(1) = \eta \cdot \delta_2^{(1)}(1) \cdot y_2^{(0)}(1) = 0.3 \cdot (-0.0048) \cdot 1 = -0.00144, \\ w_{22}^{(1)}(2) = 0.5 - 0.00144 = 0.4986. \end{cases} \end{cases}$$

Виконані обчислення відкоригували всі вагові коефіцієнти нейромережі. Перевіримо, чи покращилось після модифікації ваг реакція нейромережі при тому самому входному значенні, яке використовувалось для навчання. Для цього виконаємо ще раз прямий хід:

$$\begin{cases} v_1^{(1)}(2) = 0.5 \cdot 0.50063 + 1 \cdot 1.00126 = 1.251575, \\ y_1^{(1)}(2) = \varphi(v_1^{(1)}(2)) = 0.7776; \\ v_2^{(1)}(2) = 0.5 \cdot 0.9993 + 1 \cdot 0.4986 = 0.99825, \\ y_2^{(1)}(2) = \varphi(v_2^{(1)}(2)) = 0.7307; \\ v_1^{(2)}(2) = 0.51132 \cdot 0.7776 - 0.4894 \cdot 0.7307 = 0.04, \\ y_1^{(2)}(2) = \varphi(v_1^{(2)}(2)) = 0.51. \end{cases}$$

Отже,  $e = 0.7 - 0.51 = 0.19 < 0.1942$ , що свідчить про покращення результату, який обчислює нейромережа.

### 3.16. МЕТОД ОПОРНИХ ВЕКТОРІВ

Розглядається задача навчання за прецедентами  $\langle X, Y, y^*, X^l \rangle$ , де  $X$  – множина об'єктів,  $Y$  – множина відповідей,  $y^* : X \rightarrow Y$  – цільова залежність, значення якої

відомі тільки на об'єктах навчальної вибірки  $X^l = (x_i, y_i)_{i=1}^l$ ,  $y_i = y^*(x_i)$ . Потрібно побудувати алгоритм  $a : X \rightarrow Y$ , який апроксимує цільову залежність на усьому просторі  $X$ . Розглянутий у цьому підрозділі підхід називають методом опорних векторів [5]; використовують також аббревіатуру SVM (англ. *Support Vector Machines*).

Розглянемо задачу класифікації на два класи, що не перетинаються, у якій об'єкти описані  $n$ -вимірними дійсними векторами  $X = R^n$ ,  $Y = \{-1, +1\}$ . Будуватимемо лінійний пороговий класифікатор за правилом

$$a(x) = \text{sign} \left( \sum_{j=1}^n w_j x^j - w_0 \right) = \text{sign}(\langle w, x \rangle - w_0), \quad (3.43)$$

де  $x = (x^1, \dots, x^n)$  – опис ознак об'єкта  $x$ , вектор  $w = (w^1, \dots, w^n) \in R^n$  та скалярний поріг  $w_0 \in R$  є параметрами алгоритму.

Рівняння  $\langle w, x \rangle = w_0$  описує гіперплощину, яка розділяє класи у просторі  $R^n$ . Правило класифікації в точності співпадає з моделлю нейрона МакКаллока–Пітса, але критерій та методи налагодження параметрів у SVM докорінно відрізняються від перцептронних (градієнтних) методів навчання.

На рис. 3.56 прямі  $L_1$  та  $L_2$  розділяють об'єкти, а пряма  $L_3$  не розділяє.

Нехай вибірка є лінійно роздільною, тобто існують такі значення параметрів  $w$  та  $w_0$ , при яких функціонал кількості помилок

$$Q(w, w_0) = \sum_{i=1}^l [y_i (\langle w, w_i \rangle - w_0) < 0]$$

приймає нульове значення. Але тоді розділяюча гіперплощина не єдина, оскільки існують інші положення розділяючої гіперплощини, яка реалізує те саме розбиття вибірки. Ідея методу полягає в тому, щоб розумно використати цю свободу вибору. На рис. 3.57 показано, як кілька прямих (гіперплощин) можуть розділяти множину об'єктів.

Вимагатимемо, щоб розділяюча гіперплощина знаходилась максимально далеко від найближчих до неї точок обох класів. Початково такий тип класифікації виник з евристичних міркувань: розумно припускати, що максимізація зазору (margin) між класами має сприяти більш упевненій класифікації.

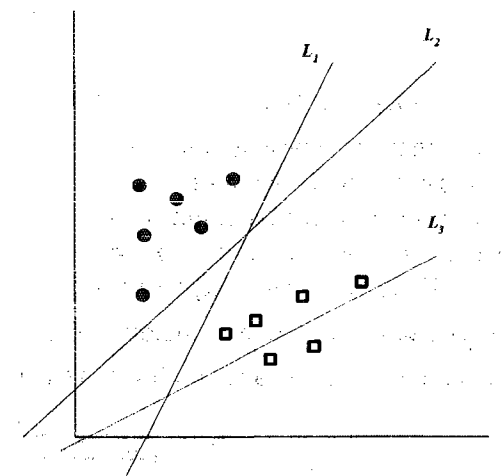


Рис. 3.56.

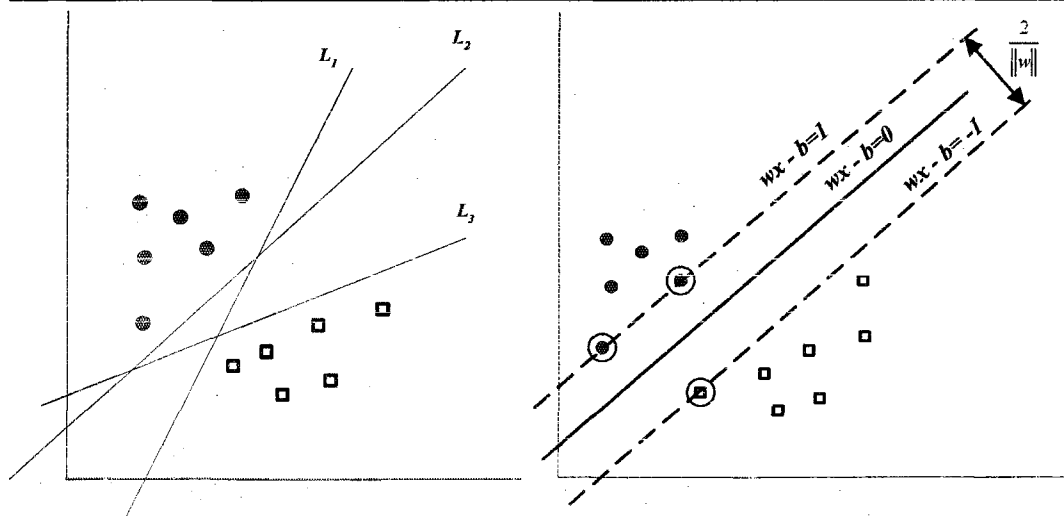


Рис. 3.57.

Параметри лінійного порогового класифікатора визначені з точністю до нормування: алгоритм  $a(x)$  не зміниться, якщо  $w$  та  $w_0$  одночасно помножити на одну й ту саму позитивну сталу. Зручно вибирати цю сталу так, щоб для всіх, близьких до розділяючої гіперплощини об'єктів  $x_i$  з  $X^l$  виконувались умови  $\langle w, x_i - w_0 \rangle = y_i$ .

Зробити це можна, оскільки в разі оптимального положення розділяючої гіперплощини всі об'єкти, які близькі до границі, знаходяться від неї на однаковій відстані, а решта — далі. Отже, для всіх  $x_i \in X^l$

$$\langle w, x_i \rangle - w_0 \begin{cases} \leq -1, & \text{якщо } y_i = -1; \\ \geq 1, & \text{якщо } y_i = +1; \end{cases} \quad (3.44)$$

Умова  $-1 < \langle w, x \rangle - w_0 < 1$  задає смугу, яка розділяє класи. Жодна з точок навчальної вибірки не може знаходитись всередині цієї смуги. Границями смуги є дві паралельні гіперплощини з направляючим вектором  $w$ . Точки, які є найближчими до розділяючої гіперплощини, лежать точно на границях смуги. При цьому розділяюча гіперплощина лежить точно посередині смуги.

Щоб розділяюча гіперплощина знаходилась якнайдалше від точок вибірки, ширина смуги має бути якнайширшою. Нехай  $x_+$  та  $x_-$  — дві довільні точки класів  $+1$  та  $-1$  відповідно, які лежать на межі смуги. Тоді шириною смуги є

$$\left\langle x_+ - x_-, \frac{w}{\|w\|} \right\rangle = \frac{(\langle w, x_+ \rangle - \langle w, x_- \rangle)}{\|w\|} = \frac{((w_0 + 1) - (w_0 - 1))}{\|w\|} = \frac{2}{\|w\|}.$$

Ширина смуги є максимальною, якщо норма вектора  $w$  є мінімальною. Отже, у випадку, коли вибірка лінійно роздільна, доволі прості геометричні міркування призводять

до такої задачі: потрібно знайти такі значення параметрів  $w$  та  $w_0$ , для яких норма вектора  $w$  мінімальна за умови (3.44). Ця задача є задачею квадратичного програмування.

Побудова оптимальної роздільної гіперплощини зводиться до мінімізації квадратичної форми для  $l$  обмежень-нерівностей вигляду (3.44) відносно  $n+1$  змінної  $w$  та  $w_0$ :

$$\begin{cases} \langle w, w_0 \rangle \rightarrow \min; \\ y_i (\langle w, x_i \rangle - w_0) \geq 1, \quad i = 1, \dots, l. \end{cases} \quad (3.45)$$

За теоремою Куна–Таккера [5] ця задача еквівалентна двоїстій задачі пошуку сідлової точки функції Лагранжа

$$\begin{cases} L(w, w_0; \lambda) = \frac{1}{2} \langle w, w \rangle - \sum_{i=1}^l \lambda_i (y_i (\langle w, x_i \rangle - w_0) - 1) \rightarrow \min_{w, w_0} \max_{\lambda}; \\ \lambda_i \geq 0, \quad i = 1, \dots, l; \\ \lambda_i = 0, \quad \text{або} \quad \langle w, x_i \rangle - w_0 = y_i, \quad i = 1, \dots, l; \end{cases}$$

де  $\lambda = (\lambda_1, \dots, \lambda_l)$  — вектор двоїстих змінних. Останню із трьох умов називають умовою доповняльної нежорсткості. Необхідною умовою сідлової точки є рівність нулю похідних Лагранжіана. Звідси випливають два корисних співвідношення

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^l \lambda_i y_i x_i = 0 \quad \Rightarrow \quad w = \sum_{i=1}^l \lambda_i y_i x_i; \quad (3.46)$$

$$\frac{\partial L}{\partial w_0} = -\sum_{i=1}^l \lambda_i y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^l \lambda_i y_i = 0; \quad (3.47)$$

З (3.46) випливає, що шуканий вектор ваг  $w$  є лінійною комбінацією векторів навчальної вибірки, причому тільки для тих, для яких  $\lambda_i \neq 0$ . З умови доповняльної нежорсткості на цих векторах  $x_i$  обмеження-нерівності перетворюються у рівності  $\langle w, x_i \rangle - w_0 = y_i$ . Отже, ці вектори знаходяться на межі розділяючої гіперплощини. Решта векторів знаходяться на певній відстані від межі, для них  $\lambda_i = 0$  і вони не приймають участі у сумі (3.46). Функція (3.43) не зміниться, якщо цих векторів взагалі не буде в навчальній вибірці.

Якщо  $\lambda_i > 0$  та  $\langle w, x_i \rangle - w_0 = y_i$ , то об'єкт  $x_i$  навчальної вибірки називають *опорним вектором* (support vector). На рис. 3.58 наведено оптимальну розділяючу гіперплощину. Опорні вектори обведені.



Підстановкою (3.46) та (3.47) у Лагранжіван отримаємо еквівалентну задачу квадратичного програмування, яка містить лише двоїсті змінні

$$\begin{cases} -L(\lambda) = -\sum_{i=1}^l \lambda_i + \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle \rightarrow \min_{\lambda}; \\ \lambda_i \geq 0, \quad i = 1, \dots, l; \\ \sum_{i=1}^l \lambda_i y_i = 0. \end{cases} \quad (3.48)$$

Тут мінімізуємо квадратичний функціонал, який має невід'ємно визначену квадратичну форму. Отже, він опуклий. Область, яка визначена обмеженнями - нерівностями та однією рівністю, також опукла. Звідси випливає, що ця задача має єдиний розв'язок.

Нехай цю задачу вже розв'язано. Тоді вектор  $w$  обчислюють за формулою (3.46). Для визначення порогу  $w_0$  достатньо взяти довільний опорний вектор  $x_i$  та виразити  $w_0$  з рівності  $w_0 = \langle w, x_i \rangle - y_i$ . На практиці для підвищення чисельної стійкості рекомендують брати за  $w_0$  середнє значення по всіх опорних векторах, а ще краще - медіану  $w_0 = \text{med} \{ \langle w, x_i \rangle - y_i : \lambda_i > 0, i = 1, \dots, l \}$ .

Остаточно алгоритм класифікації можна записати так

$$a(x) = \text{sign} \left( \sum_{i=1}^l \lambda_i y_i \langle x_i, x \rangle - w_0 \right). \quad (3.49)$$

Звернімо увагу, що реально сумування йде не по всій вибірці, а лише по опорних векторах, для яких  $\lambda_i \neq 0$ . Саме ця властивість *розрідженості* (scarcity) відрізняє SVM від інших лінійних роздільників - дискримінанти Фішера, логістичної регресії та одношарового персептрона.

Остаточно зауважимо, що залишаються відкритими два питання: як бути, якщо класи лінійно нероздільні та як розв'язувати двоїсту задачу.

Для узагальнення SVM на випадок лінійної нероздільності, дамо змогу алгоритму допускати помилки на навчальних об'єктах, але при цьому будемо прагнути, щоб помилок було якнайменше. Уведемо множину додаткових змінних  $\xi \geq 0$ , які характеризують величину помилки на об'єктах  $x_i$ ,  $i = 1, 2, \dots, l$ . Візьмемо за початкову точку задачу (3.45), пом'якшимо в ній обмеження-нерівності, та одночасно введемо у функціонал, що мінімізується штраф за сумарну помилку

$$\begin{cases} \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^l \xi_i \rightarrow \min_{w, w_0, \xi}; \\ y_i (\langle w, x_i \rangle - w_0) \geq 1 - \xi_i, \quad i = 1, \dots, l; \\ \xi_i \geq 0, \quad i = 1, \dots, l. \end{cases} \quad (3.50)$$

До цієї оптимізаційної задачі призводить ще один ланцюжок міркувань. У випадку  $Y = \{-1, +1\}$  відступом (margin) об'єкта  $x_i$  від межі класів називаємо величину  $m_i = y_i (\langle w, x_i \rangle - w_0)$ .

Алгоритм робить помилку на об'єкті  $x_i$  тоді і лише тоді, коли відступ  $m_i$  від'ємний. Якщо  $m_i \in (-1, +1)$ , то об'єкт  $x_i$  є всередині розділяючої смуги. Якщо  $m_i > 1$ , то об'єкт  $x_i$  класифікується правильно та знаходиться на певній відстані від розділяючої смуги.

Запишемо функціонал кількості помилок алгоритму  $a$  на вибірці  $X^l$  у термінах відступів

$$Q(a, X^l) = \sum_{i=1}^l [m_i < 0].$$

Замінімо в цьому функціоналі порогову функцію втрат на кусково-лінійну верхню оцінку  $[m_i < 0] \leq (1 - m_i)_+$  (див. рис. 3.59). Зміст такої заміни полягає в тому, щоб зробити функцію втрат чутливою до величини похибки та ввести штраф за наближення об'єкта до межі класів.

Крім того, додамо до функціонала  $Q$  штрафний доданок  $\tau \|w\|^2$ . У відповідності до принципу регуляризації некоректно поставлених задач за А. Н. Тихоновим такий доданок означає, що серед усіх векторів  $w$ , які мінімізують функціонал  $Q$ , краще брати вектори з мінімальною нормою. Регуляризацію часто застосовують для налагодження лінійних моделей класифікації та регресії.

У разі наявності шумових та/або залежних ознак вона підвищує стійкість алгоритму по відношенню до складу вибірки та його узагальнюючу властивість.

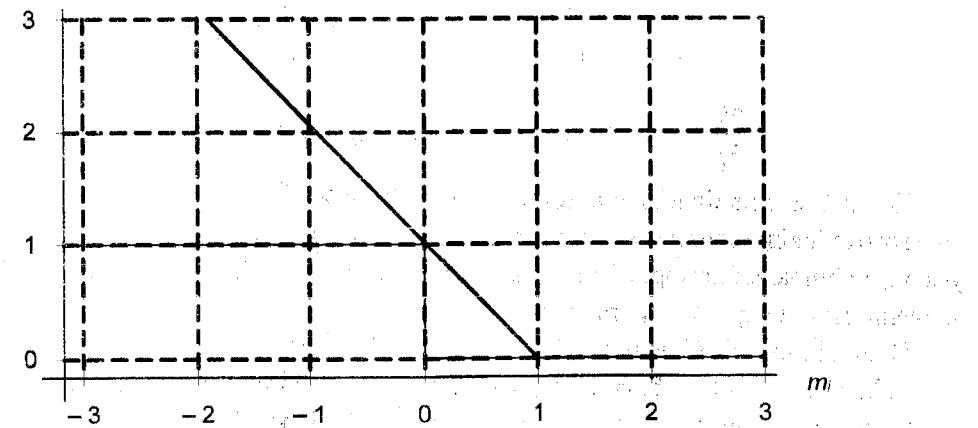


Рис. 3.59

З урахуванням обох модифікацій функціонал якості набуває вигляду

$$Q(a, X^l) = \sum_{i=1}^l (1 - m_i)_+ + \tau \|w\|^2 \rightarrow \min_{w, w_0}.$$

Задача мінімізації цього функціонала еквівалентна оптимізаційній задачі з обмеженнями (3.50), якщо взяти параметр регуляризації  $\tau = 1/2C$ . Отже, принцип оптимальної розділяючої гіперплощини (або максимізації ширини розділяючої смуги) співпадає із принципом регуляризації за Тихоновим.

Додатна стала  $C$  (або  $\tau$ ) є керуючим параметром методу та дозволяє знаходити компроміс між максимізацією роздільної смуги та мінімізацією сумарної похибки.

Повернімось до задачі (3.50) та запишемо її функцію Лагранжа

$$L(w, w_0, \xi; \lambda, \eta) = \frac{1}{2} \langle w, w \rangle - \sum_{i=1}^l \lambda_i (y_i (\langle w, x_i \rangle - w_0) - 1) - \sum_{i=1}^l \xi_i (\lambda_i + \eta_i - C),$$

де  $\eta = (\eta_1, \dots, \eta_l)$  – вектор змінних, двоїстих до змінних  $\xi = (\xi_1, \dots, \xi_l)$ . Як і раніше, умови Куна–Таккера зводять задачу до пошуку сідлової точки функції Лагранжа

$$\begin{cases} L(w, w_0, \xi; \lambda, \eta) \rightarrow \min_{w, w_0, \xi} \max_{\lambda, \eta}; \\ \xi_i \geq 0, \quad \lambda_i \geq 0, \quad \eta_i \geq 0, \quad i = 1, \dots, l; \\ \lambda_i = 0, \quad \text{або} \quad y_i (\langle w, x_i \rangle - w_0) = 1 - \xi_i, \quad i = 1, \dots, l; \\ \eta_i = 0, \quad \text{або} \quad \xi_i = 0, \quad i = 1, \dots, l; \end{cases}$$

У двох останніх рядках записано умови доповнюючої нежорсткості. Необхідною умовою сідлової точки є рівність нулю похідних Лагранжіана. Звідси отримуємо три корисних співвідношення

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^l \lambda_i y_i x_i = 0 \quad \Rightarrow \quad w = \sum_{i=1}^l \lambda_i y_i x_i; \quad (3.51)$$

$$\frac{\partial L}{\partial w_0} = -\sum_{i=1}^l \lambda_i y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^l \lambda_i y_i = 0; \quad (3.52)$$

$$\frac{\partial L}{\partial \xi} = -\lambda - \eta + C = 0 \quad \Rightarrow \quad \eta_i + \lambda_i = C, \quad i = 1, \dots, l. \quad (3.53)$$

Перші два співвідношення точно такі самі, що й у лінійно роздільному випадку. Із третього співвідношення та нерівності  $\eta_i \geq 0$  випливає обмеження  $\lambda_i \leq C$ . Звідси та з умов доповнюючої нежорсткості випливає, що можливі лише три допустимі комбінації значень змінних  $\xi_i$ ,  $\lambda_i$ ,  $\eta_i$  та відступів  $m_i$ .

Відповідно, усі об'єкти  $x_i$ ,  $i = 1, 2, \dots, l$  поділяють на такі три типи.

1.  $\lambda_i = 0$ ;  $\eta_i = C$ ;  $\xi_i = 0$ ;  $m_i > 1$ . Об'єкт  $x_i$  класифікується правильно та знаходиться недалеко від розділяючої смуги. Такі об'єкти називають *периферійними*.

2.  $0 < \lambda_i < C$ ;  $0 < \eta_i < C$ ;  $\xi_i = 0$ ;  $m_i = 1$ . Об'єкт  $x_i$  класифікується правильно та лежить точно на межі розділяючої смуги. Такі об'єкти, як і раніше, називаємо *опорними*.

3.  $\lambda_i = C$ ;  $\eta_i = 0$ ;  $\xi_i > 0$ ;  $m_i < 1$ . Об'єкт  $x_i$  лежить або всередині роздільної смуги, але класифікується правильно ( $0 < \xi_i < 1$ ,  $0 < m_i < 1$ ), або попадає на межу

класів ( $\xi_i = 1$ ,  $m_i = 0$ ), або взагалі відноситься до іншого класу ( $\xi_i > 1$ ,  $m_i < 0$ ). У всіх цих трьох випадках об'єкт  $x_i$  називають *порушником*.

У силу співвідношення (3.53) у Лагранжіані обнулюють усі члени, які містять змінні  $\xi_i$  та  $\eta_i$ , а він набуває того самого вигляду, що і у випадку лінійної роздільності. Параметри роздільної поверхні  $w$  та  $w_0$ , у відповідності до (3.51) та (3.52) також виражаються лише через двоїсті змінні  $\lambda_i$ . Єдина відмінність від лінійно роздільного випадку полягає в появі обмеження зверху  $\lambda_i \leq C$ .

$$\begin{cases} -L(\lambda) = -\sum_{i=1}^l \lambda_i + \frac{1}{2} \sum_{i=1}^l \sum_{j=1}^l \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle \rightarrow \min_{\lambda}; \\ 0 \leq \lambda_i \leq C, \quad i = 1, \dots, l; \\ \sum_{i=1}^l \lambda_i y_i = 0. \end{cases} \quad (3.54)$$

На практиці для побудови SVM розв'язують саме цю задачу, а не (3.48), оскільки гарантувати лінійну роздільність вибірки в загальному випадку неможливо. Цей варіант алгоритму називають *SVM із м'яким зазором (soft-margin SVM)*, тоді як у лінійно роздільному випадку говорять про *SVM із жорстким зазором (hard-margin SVM)*.

Для алгоритму класифікації зберігається формула (3.49), з тією лише відмінністю, що тепер ненульовими  $\lambda_i$  володіють не лише опорні вектори, але й об'єкти-порушники. У певному розумінні це являє собою недолік SVM, оскільки порушниками часто виявляються шумові викиди, а побудоване на їхній основі вирішуюче правило, за своєю суттю, спирається на шум.

Сталу  $C$  зазвичай обирають за критерієм ковзаючого контролю. Це трудомісткий процес, оскільки задачу доводиться розв'язувати для кожного значення  $C$ .

Якщо є підстави припускати, що вибірка майже лінійно роздільна, а лише об'єкти-викиди класифікуються неправильно, то можна застосувати фільтрацію викидів. Спочатку задачу розв'язують для деякого  $C$ , а з вибірки вилучають невелику частину об'єктів, які мають найбільше значення похибки  $\xi_i$ . Після цього задачу розв'язують наново з обрізаною вибіркою. Можливо, доведеться виконати декілька таких ітерацій, поки об'єкти, що залишаються, стануть лінійно розділними.

Двоїста задача (3.54) – це задача квадратичного програмування. Загальні методи розв'язування таких задач відомі, але доволі трудомісткі як у сенсі реалізації, так і за часом виконання. Тому для навчання SVM застосовують спеціалізовані алгоритми, які враховують специфічні особливості SVM. Специфіка полягає в тому, що кількість опорних векторів  $h$ , як правило, невелика ( $h \ll l$ ), і ці вектори знаходяться близько від межі класів. Саме ця особливість і дає змогу прискорити пошук опорних об'єктів. Спеціалізовані алгоритми налаштування SVM успішно дають раду з вибірками з десятками тисяч об'єктів.

**Приклад 3.57.** Використаємо метод опорних векторів для розділення класів

$$\omega_1 = \{x_1\} \text{ та } \omega_2 = \{x_2, x_3\}, \text{ якщо } x_1 = (1, 1)^T, \quad x_2 = (1, 2)^T, \quad x_3 = (2, 3)^T.$$

Візьмемо  $y_1 = 1, y_2 = -1, y_3 = -1$ . Тоді функція  $L(\lambda)$  виглядатиме так

$$L(\lambda) = \sum_{i=1}^3 \lambda_i - 0.5 \sum_{i,j=1}^3 \lambda_i \lambda_j y_i y_j (x_i, x_j) =$$

$$= \lambda_1 + \lambda_2 + \lambda_3 - 0.5(2\lambda_1^2 + 5\lambda_2^2 + 13\lambda_3^2 - 6\lambda_1\lambda_2 - 10\lambda_1\lambda_3 + 16\lambda_2\lambda_3),$$

причому  $\lambda_1 - \lambda_2 - \lambda_3 = 0$ .

Звідси  $\lambda_3 = \lambda_1 - \lambda_2$ , тоді  $L(\lambda_1, \lambda_2) = 2\lambda_1 - 2.5\lambda_1^2 - \lambda_2^2 + 3\lambda_1\lambda_2$ . Складемо й розв'яжемо нормальну систему для функції  $L(\lambda_1, \lambda_2)$ :

$$\begin{cases} \frac{\partial L}{\partial \lambda_1} = 0, \\ \frac{\partial L}{\partial \lambda_2} = 0 \end{cases} \Leftrightarrow \begin{cases} 2 - 5\lambda_1 + 3\lambda_2 = 0, \\ -2\lambda_2 + \lambda_1 = 0 \end{cases} \Leftrightarrow \begin{cases} \lambda_1 = 4 \\ \lambda_2 = 6. \end{cases}$$

Отже,  $\lambda_1 = 4, \lambda_2 = 6, \lambda_3 = -2$ . Оскільки  $\lambda_3 < 0$ , то дослідимо функцію  $L(\lambda)$  на межі області  $\lambda_i \geq 0$  ( $i=1,2,3$ ) за умови  $\lambda_3 = \lambda_1 - \lambda_2$ . Якщо  $\lambda_1 = 0$ , то  $\lambda_3 = -\lambda_2$ , звідки випливає, що  $\lambda_i^{(1)} = 0$  для  $i=1,2,3$ , і далі  $F(\lambda^{(1)}) = 0$ . Нехай  $\lambda_2 = 0$ , тоді  $\lambda_1 = \lambda_3$  і  $L(\lambda) = 2\lambda_3 - 2.5\lambda_3^2$ ;  $L'(\lambda) = 0$  при  $\lambda_3^{(2)} = 2/5$ . Отже,  $\lambda_1^{(2)} = \lambda_3^{(2)} = 2/5, \lambda_2^{(2)} = 0$  і  $L(\lambda^{(2)}) = 2/5$ . Якщо ж  $\lambda_3 = 0$ , то  $\lambda_1 = \lambda_2$  і  $L(\lambda) = 2\lambda_2 - 0.5\lambda_2^2, L'(\lambda) = 0$  при  $\lambda_2^{(3)} = 2$ . Отже,  $\lambda_1^{(3)} = \lambda_2^{(3)} = 2, \lambda_3^{(3)} = 0$  і  $L(\lambda^{(3)}) = 2$ .

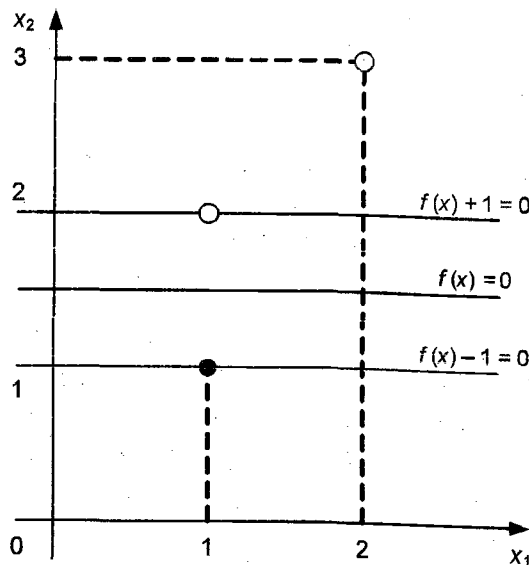


Рис. 3.60

Звідси доходимо висновку, що максимальне значення функції  $L(\lambda)$  в області  $\lambda_i \geq 0$  ( $i=1,2,3$ ) за умови  $\lambda_3 = \lambda_1 - \lambda_2$  досягається в точці  $\lambda^{(3)} = (2, 2, 0)^T$ . У цьому випадку

$$\begin{cases} w = \sum_{i=1}^3 \lambda_i y_i x_i = 2x_1 - 2x_2 = 2 \begin{pmatrix} 1 \\ 1 \end{pmatrix} - 2 \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ -2 \end{pmatrix}, \\ b = \frac{1}{y_1} - (w, x_1) = 1 - (0 - 2) = 3. \end{cases}$$

Отже,  $f(x) = (w, x) + b = -2x_2 + 3$  і  $f(x) = 0 \Leftrightarrow x_2 = 1.5$ . Розділяюча смуга завширшки  $2/\|w\|$ , а прямі  $f(x) + 1 = 0 \Leftrightarrow x_2 = 2$  і  $f(x) - 1 = 0 \Leftrightarrow x_2 = 1$  – це її межі (див. рис. 3.60).

### 3.17. МЕРЕЖІ, ЩО САМООРГАНІЗУЮТЬСЯ

Серед мереж, що самоорганізуються, важливий клас утворюють мережі, основа яких – змагання між нейронами.

#### 3.17.1. Опис мереж, що самоорганізуються

Це одношарові мережі, у яких кожен нейрон з'єднаний з усіма компонентами  $N$ -вимірного вхідного вектора  $x$ , як це показано на рис. 3.61 при  $N=2$

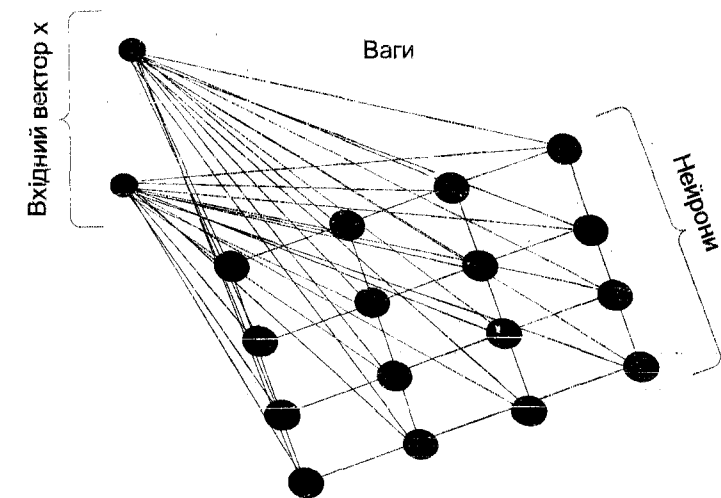


Рис. 3.61

Ваги семантичних зв'язків утворюють вектор  $W_i = [W_1^{(i)}, W_2^{(i)}, \dots, W_N^{(i)}]^T$ . При введеній нормалізації вхідних векторів, перемагає нейрон, ваги якого найменше відрізняються від відповідних складових вектора  $x$ . Для переможця ( $w$ -го нейрона) виконується співвідношення

$$d(x, W_w) = \min_{1 \leq i \leq n} d(x, W_i),$$

де  $d(x, W)$  означає відстань у сенсі вибраної метрики між вектором  $x$  та вектором  $W$ , де  $n$  – кількість нейронів. Навколо нейрона-переможця визначене топологічне сусідство  $S_w(k)$  із заданим радіусом, що зменшується в процесі навчання мережі.

Нейрон-переможець та всі нейрони в околі сусідства підлягають адаптації, змінюючи свої вектори ваг по відношенню до вектора  $x(k)$  у відповідності до правила Кохонена:

$$W_i(k+1) = W_i(k) + \eta_i(k)(x - W_i(k)) \quad (3.55)$$

для  $i \in S_w(k)$ , де  $\eta_i(k)$  – кроки навчання  $i$ -го нейрона з околу  $S_w(k)$  в момент  $k$ . Величина  $\eta(k)$  зменшується із збільшенням відстані нейрона від переможця. Ваги векторів за межами  $S_w(k)$  не підлягають зміні.

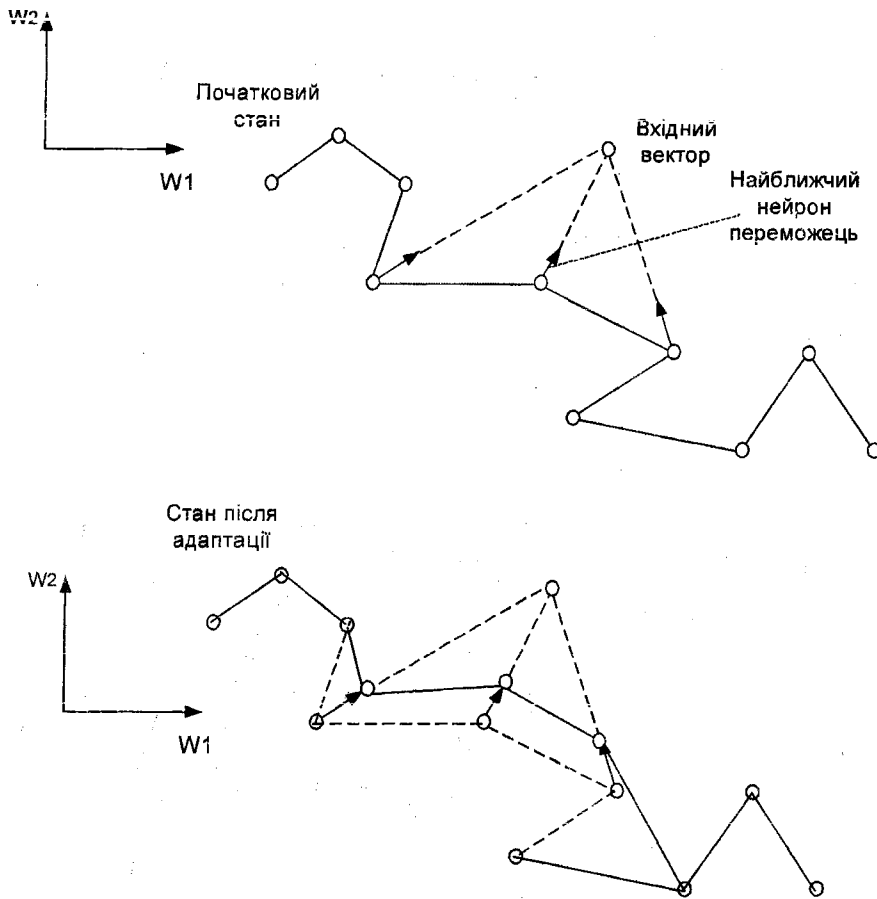


Рис. 3.62

Можна показати, що формула (3.55) є аналогом градієнтного методу навчання з мінімізацією функції мети

$$E(W) = \frac{1}{2} \sum_{i,j,k} S_i(x(k)) [x_j(k) - W_{ij}(k)]^2.$$

При цьому  $S_i(x(k))$  – функція сусідства, яка змінюється в процесі навчання. Величина сусідства та величина множників навчання окремих нейронів є функціями, що зменшуються в процесі навчання мережі. На рис. 3.62 графічно показано процес адаптації ваг нейрона-переможця та ваг нейронів, що знаходяться в його околі.

У разі подання двох різних векторів  $x_1$  та  $x_2$ , активізуються два нейрони мережі, кожний із яких презентує найближчу вагу. Це ваги  $W_1$  та  $W_2$ , які можуть бути інтерпретовані як дві точки простору. Наближення до себе векторів  $x_1$  та  $x_2$  призводить до зміни положень векторів  $W_1$  та  $W_2$ , у граничному випадку  $W_1 = W_2$  тоді й тільки тоді, коли  $x_1$  та  $x_2$  рівні або близькі між собою. Мережу, для якої виконуються ці умови, називають *топографічною картою* (або *картою Кохонена*).

### 3.17.2. Міри відстані між векторами

Істотною проблемою в процесі самоорганізації є вибір метрик, у яких вимірюється відстань між векторами  $x$  та  $W_i$ . Найчастіше використовують такі метрики:

◆ евклідова

$$d(x, W_i) = \|x - W_i\| = \sqrt{\sum_{j=1}^N (x_j - W_j^{(i)})^2};$$

◆ скалярний добуток

$$d(x, W_i) = x \cdot W_i = \|x\| \cdot \|W_i\| \cos(x, W_i);$$

◆ метрика згідно норми  $L_1$  (мангетенська)

$$d(x, W_i) = \sqrt{\sum_{j=1}^N |x_j - W_j^{(i)}|};$$

◆ метрика згідно норми  $L_\infty$

$$d(x, W_i) = \max_j |x_j - W_j^{(i)}|.$$

При використанні евклідової метрики поділ простору на області домінування нейронів є подібний до *мозаїки Вороного* (Voronoi), у якій простір навколо центральних пунктів є областю переважання даного нейрона. Застосування іншої метрики при самоорганізації формує інший поділ сфер впливу. Зокрема, застосування скалярного добутку без нормалізації векторів може привести до поділу простору, при якому в один окіл попадають кілька нейронів, а інший – не міститиме жодного.

### 3.17.3. Проблема нормалізації векторів

Доведено, що процес самоорганізації завжди призводить до узгодженого поділу простору даних, якщо хоча б один із векторів,  $x$  чи  $W$ , є нормалізованим. Для нормалізованих навчальних векторів  $x$  вектори ваг, пов'язані з ними, автоматично

лізуються. З іншого боку, нормалізація векторів ваг приводить до того, що коли  $\|W_i\| = \text{const}$ , то для всіх нейронів добуток  $\|x\| \cdot \|W_i\|$  також сталий для заданого  $x$ .

активізацію нейрона приймає рішення  $\cos(x, w_i)$ , який є спільною мірою для всієї мережі. Слід зазначити, що при нормалізації вектора ваг метрики як евклідова, так і ларнний добуток рівнозначні, бо  $\|x - W_i\|^2 = \|x\|^2 + \|W_i\|^2 - 2x \cdot W_i$ . Звідси

$$\min \|x - W_i\|^2 = \max (x \cdot W_i),$$

якщо  $\|W_i\| = \text{const}$ .

Експериментальні дослідження показали необхідність нормалізації векторів при малих значеннях розмірності простору ( $N = 2, N = 3$ ). При збільшенні розмірності вхідного вектора ефект нормалізації стає все менш помітним, і при великих розмірах мережі ( $N > 200$ ) нормалізація не відіграє ролі при самоорганізації.

Нормалізація може бути виконана двома способами:

- ◆ перевизначенням складових вхідного вектора за формулою

$$x_i := \frac{x_i}{\sqrt{\sum_{j=1}^N x_j^2}};$$

- ◆ збільшенням розміру простору на одиницю,  $R^N \rightarrow R^{N+1}$ , з таким вибором  $(N+1)$ -ї складової вектора, що

$$\sum_{i=1}^{N+1} x_i^2 = 1. \quad (3.56)$$

При такому способі нормалізації виникає необхідність попереднього переобчислення складових вектора  $x$  у просторі  $R^N$ , яка дозволить виконати (3.56).

### 3.17.4. Міра організації мережі

Важливою проблемою навчання мережі, що самоорганізується, є кількісна оцінка ступеня організації її нейронів. Таку оцінку можна дати лише при двовимірному векторі, рівномірному розташуванню нейронів по площині та регулярності вузлів сітки, яка вимірюється як відстань між сусідніми точками, що відповідають векторам ваг окремих нейронів.

При великих розмірностях вхідного вектора (більших ніж 2) вводиться міра нерівномірності організації нейронів на статистичному розподілі евклідових мір різниць між векторами ваг окремих нейронів. Міра впорядкованості нейронів визначається за формулою:

$$\Theta = \frac{\sigma}{\mu}, \quad (3.57)$$

де

$$\mu = \frac{\sum_{i=1}^k \sum_{j=1}^{g-1} \Delta h_{ij} + \sum_{i=1}^{h-1} \sum_{j=1}^g \Delta v_{ij}}{2gh - g - h},$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^h \sum_{j=1}^{g-1} (\Delta h_{ij} - \mu)^2 + \sum_{i=1}^{h-1} \sum_{j=1}^g (\Delta v_{ij} - \mu)^2}{2gh - g - h}}, \quad (3.59)$$

$$\Delta h_{ij} = \|W_{ij} - W_{i,j+1}\|, \Delta v_{ij} = \|W_{ij} - W_{i+1,j}\|.$$

У формулах (3.57)–(3.59)  $W_{ij}$  означає вектор ваг  $(i, j)$ -го нейрона в прямокутній сітці, яка має  $h$  нейронів по горизонтальній осі та  $g$  – по вертикальній. Величина  $\mu$  визначає середню відстань між векторами ваг, а  $\sigma$  – стандартне відхилення цієї відстані між двома сусідніми нейронами мережі. При ідеальній організації нейронів  $\sigma \rightarrow 0$  (аналогічно і  $\Theta \rightarrow 0$ ).

### 3.17.5. Механізм стомлення нейронів

Кращі результати самоорганізації спостерігаються, коли навчальний алгоритм урахує кількість перемог окремих нейронів та організує процес навчання так, щоб дати можливість перемагати менш активним нейронам. Таке припущення про організацію навчання враховує поведінку біологічних нейронів, коли нейрон, що виграв, деякий час робить паузу, відпочиваючи до наступного акту. Такий спосіб урахування активності нейрона називають механізмом стомлення. Для практичного застосування цього принципу в мережах, що самоорганізуються, вводять потенціал  $p_i$  кожного нейрона, який модифікують після кожного  $k$ -го подання значення вхідного вектора  $x$  у відповідності до співвідношення (тут ураховано перемогу  $w$ -го нейрона):

$$p_i(k+1) = \begin{cases} p_i(k) + \frac{1}{n}, & \text{для } i \neq w \\ p_i(k) - p_{\min}, & \text{для } i = w \end{cases}$$

Доданок  $p_{\min}$  означає мінімальний потенціал, який уповноважує брати участь у змаганні. Якщо актуальне значення  $p_i$  опиниться меншим від  $p_{\min}$ , то  $i$ -й нейрон „відпочиває”, а переможця шукають серед нейронів, для яких виконується рівність

$$d(x, W_n) = \min \{d(x, W_i)\}$$

для  $1 \leq i \leq N$  та  $p_i \geq p_{\min}$ . Максимальне значення потенціалу обмежується значенням 1.

Вибір величини  $p_{\min}$  впливає на готовність нейрона до змагання. Якщо  $p_{\min} = 0$ , то не відбувається стоми нейронів, і кожний із них одразу після виграшу готовий прийняти участь у змаганні (стандартний алгоритм Кохонена). Якщо  $p_{\min} = 1$ , то маємо іншу крайність, за якої нейрони виграють по черзі, оскільки в кожний момент тільки один із них готовий до змагання. Практично хороший результат отримується коли  $p_{\min} \approx 0.75$ .

### 3.17.6. Методи навчання мереж, що самоорганізуються

Метою навчання мережі, що самоорганізується, є таке впорядкування нейронів (вибір вартості ваг), яке мінімізує величину очікуваного спотворення та вимірюється як похибка апроксимації вхідного вектора  $x$  величинами ваг нейрона, який переміг у змаганні. У разі  $p$  вхідних векторів  $x$  та застосуванні евклідової норми, цю похибку визначають за формулою

$$E = \frac{1}{p} \sum_{i=1}^p \|x_i - W_{w(i)}\|$$

де  $W_{w(i)}$  – вага нейрона, який переміг при презентації вектора  $x_i$ .

Такий підхід також відомий як *квантифікація вектора (Vector Quantization, VQ)*. Номери нейронів-переможців при чергових поданнях векторів  $x$  утворюють так звану *кодovu книгу*. У класичному кодуванні використовують алгоритм *K-середніх (K-means)*.

Методу квантифікації вектора в застосуванні до нейромережі відповідає метод WTA (*Winner Takes All*). У цьому методі при поданні вектора  $x$  обчислюють активність кожного нейрона. Переможцем є нейрон із найбільшим вихідним значенням, а саме той, скалярний добуток якого  $xW$  найбільший. У разі застосування нормалізованих векторів це відповідає найменшій евклідовій відстані між вхідним вектором та вектором ваг нейронів. Переможець має привілей по адаптації своїх ваг у напрямку вектора  $x$  за правилом  $W_w := W_w + \eta(x - W_w)$ , а решта нейронів не підлягають адаптації. Методи WTA, у яких тільки один нейрон може адаптуватись на кожній ітерації, погано збігаються, особливо при великій кількості нейронів. Практично застосовують метод *WTM (Winner Takes Most)*, у якому крім переможця активізують свої ваги також нейрони з його околу, причому що далі вони знаходяться від переможця, то менше коректуються їхні ваги. Цей процес описується такою узагальненою залежністю

$$W_i := W_i + \eta_i G(i, x)(x - W_i)$$

для всіх нейронів із номерами  $i$ , які перебувають в околі  $S_w$  переможця. У цій формулі крок навчання  $\eta_i$  кожного нейрона доповнений функцією  $G(i, x)$ , визначеною співвідношенням

$$G(i, x) = \begin{cases} 1, & \text{для } i = w, \\ 0, & \text{для } i \neq w, \end{cases}$$

де  $w$  – номер переможця, отриманий у класичному методі WTA. Існує багато методів WTM, у яких по-різному вибирається вигляд функції  $G(i, x)$ .

**Метод Кохонена.** У топологічних картах Кохонена, що самоорганізуються, у першу чергу, визначається переможець за допомогою евклідової відстані, а потім знаходиться величина множника адаптації нейронів з околу переможця. У класичному методі Кохонена функцію  $G(i, x)$  визначено так:

$$G(i, x) = \begin{cases} 1, & \text{для } d(i, w) \leq \lambda, \\ 0, & \text{для всіх інших.} \end{cases}$$

Тут  $d(i, w)$  – евклідова відстань між позицією  $i$ -го нейрона з топологічного сусідства нейрона-переможця  $w$  та нейроном  $w$ . Тут  $\lambda$  – радіус сусідства, який зменшується в процесі навчання. Сусідство такого типу називають *прямокутним сусідством*.

Інший тип сусідства, що застосовують у картах Кохонена, – *гаусівське сусідство*, де  $G(i, x)$  визначено за формулою

$$G(i, x) = \exp\left(-\frac{d^2(i, w)}{2\lambda^2}\right)$$

Про ступінь адаптації нейронів з околу сусідства переможця приймається рішення не тільки на основі евклідової відстані  $i$ -го нейрона від переможця ( $w$ -го нейрона), але за значенням радіусу  $\lambda$ .

Якщо при прямокутному сусідстві всі нейрони адаптуються однаково, то при гаусівському величина адаптації залежить від значень функції Гауса, і для різних нейронів має різні значення. Гаусівське навчання дає кращі результати ніж прямокутне.

**Метод стохастичної релаксації.** У цьому методі всі нейрони підлягають адаптації з імовірністю, яка визначається розподілом Гіббса (Gibbs)

$$P(i) = \frac{\exp\left(-\frac{\|x - W_i\|^2}{T}\right)}{\sum_{j=1}^n \exp\left(-\frac{\|x - W_j\|^2}{T}\right)}. \quad (3.60)$$

Тут  $T$  – параметр, який називають *температурою*. При високій температурі всі нейрони змінюють свої ваги з імовірністю  $\lim_{T \rightarrow \infty} P(i) = \frac{1}{N}$ . У крайньому випадку,



$P(i) \rightarrow 1$  при  $T \rightarrow 0$  тільки для переможця ( $i = w$ ), і  $P(i) \rightarrow 0$  для інших нейронів ( $i \neq w$ ). У процесі навчання температура зменшується від максимального значення до нуля, тому в граничному випадку метод прямує до класичного WTA.

В методі стохастичної релаксації  $G(i, x)$  визначають за формулою

$$G(i, x) = \begin{cases} 1, & \text{для } P(i) > P, \\ 0, & \text{у всіх інших випадках,} \end{cases}$$

де  $P$  означає випадкове число з проміжку  $[0, 1]$ . У разі відповідної організації процесу навчання та довільному зменшенні температури метод дає змогу обійти пастки локальних мінімумів та отримати хороші результати організації мережі. Плата за це – уповільнення процесу навчання.

**Метод SCS.** Для збільшення швидкості навчання існує модифікація методу стохастичної релаксації у вигляді методу *м'якої конкуренції* (*Soft Competition Scheme* – SCS). У цій модифікації двійкове значення функції  $G(i, x)$  замінюють на поточне значення функції ймовірностей  $P(i)$ , тобто  $G(i, x) = P(i)$ , причому  $P(i)$  обчислене за формулою (3.60). Для врахування ступеня активності нейронів, які опікують малоактивні елементи, вводиться відмінна для кожного нейрона величина множника навчання  $\eta_i(k)$ , яка залежить від  $k-1$  попередніх значень функції  $G(i, x)$  згідно співвідношення

$$\eta_i(k) = \frac{1}{1 + \sum_{j=1}^{k-1} G_j(i, x)}.$$

При цьому  $j$  представляє всі попередні кроки ( $j = 1, 2, \dots, k-1$ ), а  $G(i, x)$  – значення функції сусідства на  $j$ -му кроці навчання. Метод SCS використовує міру Евкліда й на практиці є дещо результативнішим за метод стохастичної релаксації, хоча швидкість його збіжності до розв'язку залишається малою.

**Метод нейронного газу.** Значне покращення самоорганізації й збільшення швидкості збіжності може отримати з допомогою методу *нейронного газу*.

На кожній ітерації всі нейрони сортуються за зростанням у залежності від їхньої відстані від вектора  $x$  таким способом

$$d_0 < d_1 < \dots < d_{n-1},$$

де  $d_m = \|x - W_{m(i)}\|$  – відстань  $i$ -го нейрона, який займає у відсортованій послідовності позицію  $m$  за нейроном-переможцем. Величину функції сусідства  $i$ -го нейрона визначають за формулою

$$G(i, x) = e^{-\frac{m(i)}{\lambda}}$$

де  $m(i)$  – номер, отриманий у результаті сортування ( $m(i) = 1, 2, \dots, n-1$ ), а  $\lambda$  – параметр, аналогічний радіусу сусідства в методі Кохонена, який зменшується в процесі навчання мережі. При  $\lambda = 0$  тільки переможець піддається зміні, і метод стає звичайним WTA. При  $\lambda > 0$  адаптується багато нейронів залежно від значення функції  $G(i, x)$ . Щоб отримати хороші результати самоорганізації, процес навчання треба починати з великих значень  $\lambda$ , а під час навчання значення цього параметра має зменшуватись. Функція  $G(i, x)$  при  $\lambda \rightarrow 0$  перетворюється з унімодальною на багатоекстремальну, тобто у таку, що має багато локальних екстремумів. Завдяки повільному зменшенню параметра  $\lambda$  вдається уникнути пасток локальних екстремумів.

Параметр  $\lambda$  визначають за формулою

$$\lambda(k) = \lambda_{\max} \left( \frac{\lambda_{\min}}{\lambda_{\max}} \right)^{k/k_{\max}},$$

де  $\lambda(k)$  – величина  $\lambda$  на  $k$ -ій ітерації, а  $\lambda_{\min}$  та  $\lambda_{\max}$  – задані відповідно мінімальне та максимальне значення  $\lambda$ ;  $k_{\max}$  – максимально допустима кількість ітерацій.

Крок навчання  $i$ -го нейрона змінюється в подібній залежності:

$$\eta_i(k) = \eta_i(0) \left( \frac{\eta_{\min}}{\eta_i(0)} \right)^{k/k_{\max}},$$

де  $\eta_i(0)$  – початкове значення кроку навчання, а  $\eta_{\min}$  – задане заздалегідь його мінімальне значення, яке відповідає  $k = k_{\max}$ .



## ЗАДАЧІ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

1. У яких із наведених нижче задач маємо або, принаймні, можемо мати справу з навчанням комп'ютерною програмою, яка навчається в розумінні сформульованого в цьому розділі означення:

- уведення інформації до бази даних;
  - пошук залежностей між атрибутами бази даних;
  - відповіді на запити до бази даних;
  - передбачення значень атрибутів у разі неповних даних (частина значень атрибутів відсутня);
  - компіляція програми;
  - автоматичне генерування фрагментів програми.
- Відповідь обґрунтувати.

2. Охарактеризуйте такі види навчання людей і тварин за допомогою поданих у цьому розділі критеріїв класифікації машинного навчання:

- а) навчання розв'язуванню задач геометрії;
- б) навчання іноземної мови;
- в) навчання керування авто;
- г) навчання розпізнавати види грибів;
- д) навчання собаки приносити речі;
- е) навчання папути наслідувати мову.

3. Розміркуйте про можливе застосування навчальних систем у таких галузях:

- а) інвестиція капіталу;
- б) торгівля;
- в) маркетинг;
- г) безпека;
- д) соціологічне дослідження;
- е) навчання;
- ж) телекомунікація;
- з) громадський транспорт;
- и) розваги.

4. Яке значення може мати машинне навчання для розвитку штучного інтелекту в його „слабкому” і „сильному” розумінні, вважаючи, що цілі сильного штучного інтелекту можливо реалізувати в доволі віддаленому майбутньому.

5. Поясніть, чому потужність простору гіпотез у задачі навчання *НасСпортом* дорівнює 973. Як зростає кількість можливих прикладів і можливих гіпотез із введенням атрибуту *ДжерелоВоди*, який прийматиме значення „Слабке”, „Помірне” чи „Сильне”? Як збільшить кількість можливих прикладів і гіпотез додавання нового атрибуту *A*, який набуває *k* можливих значень?

6. Наведіть послідовність межових множин *S* та *G*, які будуть обчислені за алгоритмом СЕ, якщо йому надати послідовність навчальних прикладів із таблиці 3.1, але в зворотному порядку. Хоча кінцевий простір версій буде тим самим, що й у початковому випадку, та не залежатиме від послідовності прикладів, множини *S* та *G*, які обчислені на проміжних стадіях, залежатимуть від цієї послідовності. Як можна впорядкувати навчальні приклади, щоб мінімізувати суму потужностей цих проміжних множин *S* та *G* з простору гіпотез *H*, використаного в прикладі *НасСпортом*?

7. Іще раз розглянемо задачу навчання *НасСпортом* і простір гіпотез *H*. Уведемо новий простір гіпотез *H'*, який складається із усіх пар гіпотез у *H*, з'єднаних диз'юнкцією. Приклад типової гіпотези в *H'*:

$\langle ?, \text{Холодна, Висока, ?, ?, ?} \rangle \wedge \langle \text{Сонячна, ?, Тепла, ?, ?, Той самий} \rangle$ .

Необхідно простежити виконання алгоритму СЕ (навести послідовності межових множин *S* та *G*) у просторі гіпотез *H'* на послідовності навчальних прикладів із табл. 3.1.

8. Розглянемо множину прикладів, що складається з точок у площині *xOy*, які мають цілочисельні значення, та множину гіпотез *H*, яка складається із прямокутників. Точніше, гіпотези мають вигляд  $a \leq x \leq b$ ,  $c \leq y \leq d$ , де *a*, *b*, *c* та *d* – довільні цілі числа.

а) Розгляньте простір версій для множин позитивних (+) та негативних (–) навчальних прикладів, показаних на рисунку до цієї задачі. Якою є межа *S* простору версій у цьому випадку? Випишіть гіпотези та покажіть їх на діаграмі.

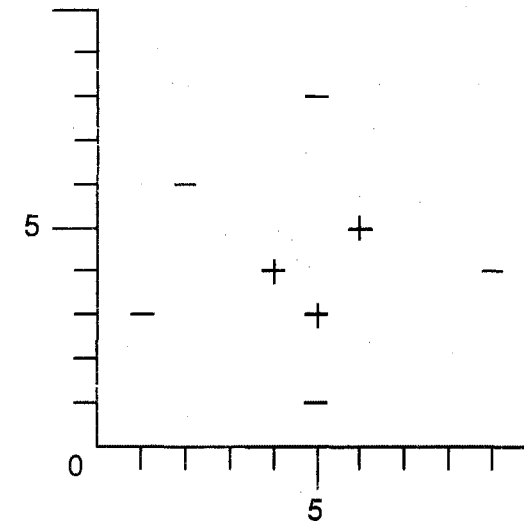


Рис. до задачі 8

б) Яка межа *G* простору версій у цьому випадку? Випишіть гіпотези та покажіть їх на діаграмі.

в) Нехай на площині *xOy* пропонується новий приклад та необхідно його класифікувати. Як сформулювати цей приклад, щоб гарантувати зменшення потужності простору версій незалежно від класифікації цього прикладу. Запропонуйте інший приклад, який цього не гарантуватиме.

г) Нехай потрібно навчити специфічне цільове поняття, наприклад,  $3 \leq x \leq 5$ ,  $2 \leq y \leq 9$ . Яку найменшу кількість навчальних прикладів можна запропонувати, щоб алгоритм СЕ повністю навчився цільовому поняттю?

9. Розглянемо послідовність позитивних і негативних навчальних прикладів, які описують поняття „пари осіб, що мешкають в одному будинку”. Кожний навчальний приклад описує впорядковану пару людей, у якій кожна людина описана своєю статтю, кольором волосся (шатен, брюнет або блондин), ростом (високий, середній або низький) та національністю (американець, француз, німець, ірландець, індус, японець або португалець).

+ <<чоловік, шатен, високий, американець><жінка, брюнетка, невисока, американка>>

+ <<чоловік, шатен, невисокий, француз><жінка, брюнетка, невисока, американка>>

– <<жінка, шатенка, висока, німкеня><жінка, брюнетка, невисока, індуска>>

+ <<чоловік, шатен, високий, ірландець><жінка, шатенка, невисока, ірландка>>

Розгляньте простір гіпотез, визначений цими прикладами, у якому кожна гіпотеза подана парою записів з чотирьох атрибутів, і де кожним обмеженням атрибута може бути або конкретне значення, або „?”, або „Ø”, так само, як подання гіпотез у *НасСпортом*. Наприклад, гіпотеза

+ <<чоловік, ?, високий, ?>> <жінка, ?, ?, японка>>

задає множину всіх пар людей, де перша людина – високий чоловік (будь-якої національності та кольору волосся), а друга – японка (будь-якого кольору волосся та зросту).

а) Проаналізуйте роботу алгоритму СЕ, що навчається на наведених навчальних прикладах, на мові гіпотез. Зокрема, покажіть межі загальності та конкретності простору версій після того, як він опрацює перший навчальний приклад, потім – другий навчальний приклад тощо.

б) Скільки різних гіпотез у визначеному просторі гіпотез сумісні з таким позитивним навчальним прикладом

+ <<чоловік, брюнет, невисокий, португалець>> <жінка, блондинка, висока, індуска>>

в) Нехай розглянуто лише позитивний приклад із пункту б) і тепер можна згенерувати будь-який приклад, який потрібно класифікувати. Сформулюйте характерну послідовність прикладів, за якою можна знайти єдину правильну гіпотезу незалежно від того, якою вона буде (припускається, що цільове поняття можна описати мовою гіпотез). Знайдіть таку найкоротшу послідовність прикладів. Як довжина цієї послідовності співвідноситься з відповіддю на запитання б)?

г) Зазначимо, що мова гіпотез не може висловити всі поняття, які можна визначити на прикладах. Тобто, можна визначити множини позитивних і негативних прикладів, для яких немає жодної відповідної гіпотези, описаної цією мовою гіпотез. Якщо розширити мову так, щоб вона могла виразити всі поняття, котрі можуть бути визначені на мові прикладів, то як зміниться відповідь на запитання в)?

10. Розглянемо проблему навчання поняття, у якій кожний приклад – дійсне число, а кожна гіпотеза – інтервал на вісі дійсних чисел, тобто, кожна гіпотеза в просторі гіпотез  $H$  має вигляд  $a \leq x \leq b$ , де  $a$  та  $b$  – довільні дійсні числа, а  $x$  – приклад. Наприклад, гіпотеза  $4.5 \leq x \leq 6.1$  класифікує приклади між 4.5 та 6.1 як позитивні, а решту прикладів – як негативні. Поясніть, чому не існує найконкретнішої гіпотези, що узгоджується з довільною множиною позитивних навчальних прикладів. Запропонуйте таку модифікацію подання гіпотез, щоб зазначена гіпотеза існувала.

11. Доведіть, що для заданого неупередженого простору гіпотез кожному не розглянутому прикладу відповідатиме точно половина елементів простору версій, незалежно від того, які навчальні приклади розглядалися. Тобто, нехай є довільний простір прикладів  $X$ , довільна множина навчальних прикладів  $D$  та довільний приклад  $x \in X$ , який не належить  $D$ ,  $H$  – булеан множини  $X$ . Доведіть, що перша половина гіпотез у  $VS_{H,D}$  класифікує приклад  $x$  як позитивний, а друга половина – як негативний.

12. Розглянемо проблему навчання, у якій кожен приклад описано кон'юнкцією  $n$  булевих атрибутів  $a_1, a_2, a_n$ . Отже, типовий приклад має вигляд

$$(a_1 = \text{true}) \wedge (a_2 = \text{false}) \wedge \dots \wedge (a_n = \text{true}).$$

Тепер розглянемо простір гіпотез  $H$  у якому кожна гіпотеза є диз'юнкцією обмежень за цими атрибутами. Наприклад, типова гіпотеза має вигляд

$$(a_1 = \text{true}) \vee (a_3 = \text{false}) \vee (a_7 = \text{true}).$$

Запропонуйте алгоритм, який набуває послідовність навчальних прикладів і виводить узгоджену гіпотезу, якщо така існує. Час виконання цього алгоритму має поліноміально залежати від  $n$  і кількості навчальних прикладів.

13. Скільки потрібно зважувань на балансових терезах для знаходження серед чотирьох монет фальшивої, якщо вона легша або важча від інших? Розв'язок задачі подати у вигляді дерева рішень. Описати алгоритм знаходження фальшивої монети з використанням цієї кількості зважувань.

14. Скільки потрібно зважувань на балансових терезах для знаходження серед восьми монет фальшивої, якщо вона легша або важча від інших? Розв'язок задачі подати у вигляді дерева рішень. Описати алгоритм знаходження фальшивої монети з використанням цієї кількості зважувань.

15. На рисунку до цієї задачі наведено дерево рішень для прогнозування погоди залежно від тиску й температури. На основі цього дерева побудувати правила ЯКЩО – ТО.

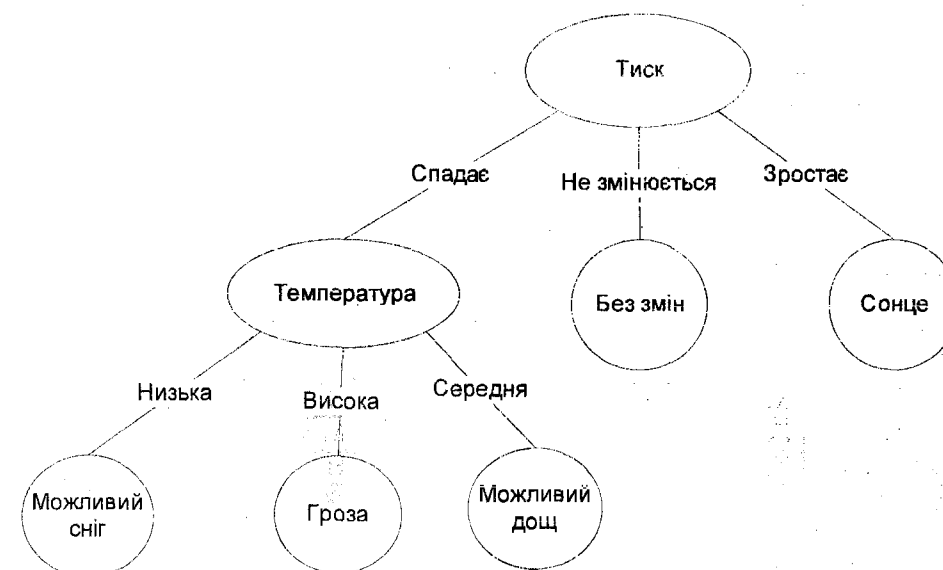


Рис. до задачі 15

16. У таблиці до цієї задачі наведено дані про оцінку кредитного ризику на підставі доходу, кредитної історії, поточного боргу та наявності поруки. Використовуючи алгоритм ID3, побудувати дерево рішень. На основі побудованого дерева рішень побудувати правила ЯКЩО – ТО.

Таблиця до задачі 16

| Варіант даних | Доход    | Кредитна історія | Борг    | Порука    | Рішення (ступінь ризику) |
|---------------|----------|------------------|---------|-----------|--------------------------|
| 1             | Низький  | Погана           | Високий | Немає     | Високий                  |
| 2             | Середній | Невідома         | Високий | Немає     | Високий                  |
| 3             | Середній | Невідома         | Низький | Немає     | Середній                 |
| 4             | Низький  | Невідома         | Низький | Немає     | Високий                  |
| 5             | Високий  | Невідома         | Низький | Немає     | Низький                  |
| 6             | Високий  | Невідома         | Високий | Адекватна | Низький                  |
| 7             | Низький  | Погана           | Низький | Немає     | Високий                  |
| 8             | Високий  | Погана           | Низький | Адекватна | Середній                 |
| 9             | Високий  | Добра            | Низький | Немає     | Низький                  |
| 10            | Високий  | Добра            | Високий | Адекватна | Низький                  |
| 11            | Низький  | Добра            | Високий | Немає     | Високий                  |
| 12            | Середній | Добра            | Високий | Немає     | Середній                 |
| 13            | Високий  | Добра            | Високий | Немає     | Низький                  |
| 14            | Середній | Погана           | Високий | Немає     | Високий                  |

17. У таблиці до цієї задачі наведено дані обстеження зору 24 пацієнтів із метою рекомендації використання контактних лінз певного типу. Ураховано вік, зір, наявність астигматизму, розширення зіниці. Побудувати дерево рішень за цими даними. На основі побудованого дерева рішень побудувати правила ЯКЩО – ТО.

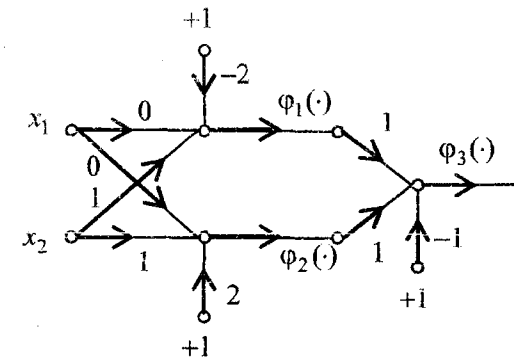
Таблиця до задачі 17

| Пацієнт | Вік      | Зір            | Астигматизм | Розширення зіниці | Рішення (які лінзи) |
|---------|----------|----------------|-------------|-------------------|---------------------|
| 1       | Молодший | Короткозорість | Так         | Так               | Не рекомендовано    |
| 2       | Молодший | Короткозорість | Так         | Ні                | М'які               |
| 3       | Молодший | Короткозорість | Ні          | Так               | Не рекомендовано    |
| 4       | Молодший | Короткозорість | Ні          | Ні                | Жорсткі             |
| 5       | Молодший | Далекозорість  | Так         | Так               | Не рекомендовано    |
| 6       | Молодший | Далекозорість  | Так         | Ні                | М'які               |
| 7       | Молодший | Далекозорість  | Ні          | Так               | Не рекомендовано    |
| 8       | Молодший | Далекозорість  | Ні          | Ні                | Жорсткі             |
| 9       | Середній | Короткозорість | Так         | Так               | Не рекомендовано    |
| 10      | Середній | Короткозорість | Так         | Ні                | М'які               |
| 11      | Середній | Короткозорість | Ні          | Так               | Не рекомендовано    |
| 12      | Середній | Короткозорість | Ні          | Ні                | Жорсткі             |
| 13      | Середній | Далекозорість  | Так         | Так               | Не рекомендовано    |
| 14      | Середній | Далекозорість  | Так         | Ні                | М'які               |

Продовження таблиці до задачі 17

| Пацієнт | Вік      | Зір            | Астигматизм | Розширення зіниці | Рішення (які лінзи) |
|---------|----------|----------------|-------------|-------------------|---------------------|
| 15      | Середній | Далекозорість  | Ні          | Так               | Не рекомендовано    |
| 16      | Середній | Далекозорість  | Ні          | Ні                | Жорсткі             |
| 17      | Старший  | Короткозорість | Так         | Так               | Не рекомендовано    |
| 18      | Старший  | Короткозорість | Так         | Ні                | М'які               |
| 19      | Старший  | Короткозорість | Ні          | Так               | Не рекомендовано    |
| 20      | Старший  | Короткозорість | Ні          | Ні                | Жорсткі             |
| 21      | Старший  | Далекозорість  | Так         | Так               | Не рекомендовано    |
| 22      | Старший  | Далекозорість  | Так         | Ні                | М'які               |
| 23      | Старший  | Далекозорість  | Ні          | Так               | Не рекомендовано    |
| 24      | Старший  | Далекозорість  | Ні          | Ні                | Жорсткі             |

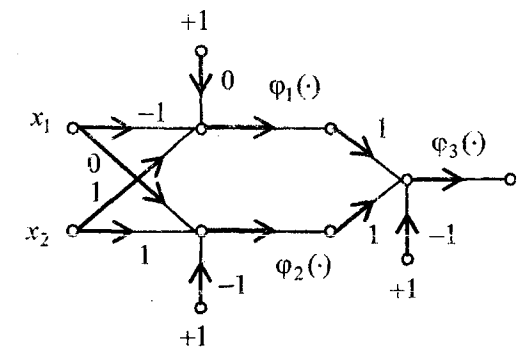
18. Сконструювати штучний нейрон для моделювання прямої, що проходить через точки  $x_1 = (-2, 1)$  та  $x_2 = (7, 3)$ . Обчислити реакцію сконструйованого нейрона, якщо його активаційна функція кусково-лінійна, а вхідним значенням є  $x = 0.5$ .



$$\phi_1(\cdot) = \begin{cases} \cdot \leq 0 \Rightarrow 1 \\ \cdot > 0 \Rightarrow 0 \end{cases} \quad \phi_2(\cdot) = \begin{cases} \cdot \geq 0 \Rightarrow 1 \\ \cdot < 0 \Rightarrow 0 \end{cases}$$

$$\phi_3(\cdot) = \begin{cases} \cdot > 0 \Rightarrow 1 \\ \cdot \leq 0 \Rightarrow 0 \end{cases}$$

а



$$\phi_1(\cdot) = \begin{cases} \cdot \leq 0 \Rightarrow 1 \\ \cdot > 0 \Rightarrow 0 \end{cases} \quad \phi_2(\cdot) = \begin{cases} \cdot \geq 0 \Rightarrow 1 \\ \cdot < 0 \Rightarrow 0 \end{cases}$$

$$\phi_3(\cdot) = \begin{cases} \cdot > 0 \Rightarrow 1 \\ \cdot \leq 0 \Rightarrow 0 \end{cases}$$

б

Рис. до задачі 22

19. Аналогічно до прикладу 3.48 обчислити реакцію нейронмережі із прикладу 3.50.

20. Для наведеної на рис. 3.45 нейронмережі обчислити вагові коефіцієнти так, щоб вона реалізовувала функцію AND. **Вказівка:** для обчислення вагових коефіцієнтів можна або скористатись дельта правилом, або визначити їхнє значення з рівняння прямої, що проходить через дві точки.

21. Якою має бути активаційна функція в моделі нейрона, наведений на рис. 3.43, для реалізації нею функції OR?
22. Використовуючи міркування, наведені в прикладі 3.55, побудувати область прийняття рішень для нейромереж, зображених на рисунку до цієї задачі.
23. Сконструювати нейромережі, які б відповідали областям прийняття рішень, зображеним на рисунку до цієї задачі.

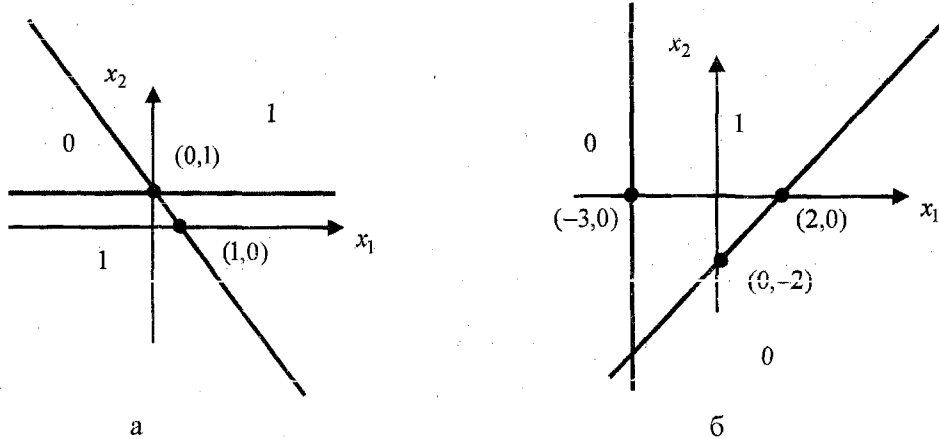


Рис. до задачі 23

24. Виконати один крок алгоритму дельта правила для нейромереж та умов, які наведено на рисунку до цієї задачі.



- 1) Активаційна функція  $\phi(v) = v$ .
- 2) Очікуване вихідне значення  $d = 0.5$ .
- 3) Крок  $\eta = 0.5$ .

Результатом має бути:

- 1) модифікація ваг;
- 2) обчислення помилки після коректування ваг для тих самих вхідних та очікуваного значень.

а

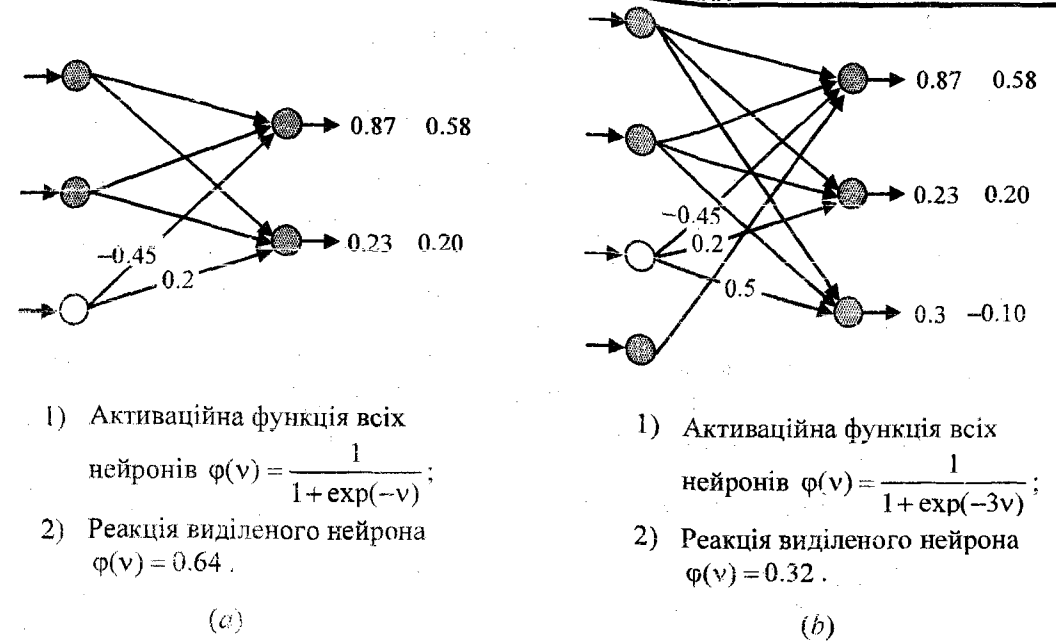
Результатом має бути:

- 1) модифікація ваг;
- 2) обчислення помилки після коректування ваг для тих самих вхідних та очікуваного значень.

б

Рис. до задачі 24

25. Обчислити значення помилки для виділеного у фрагменті нейромережі нейрона при заданих умовах (див. рис. до цієї задачі).



- 1) Активаційна функція всіх нейронів  $\phi(v) = \frac{1}{1 + \exp(-v)}$ ;
- 2) Реакція виділеного нейрона  $\phi(v) = 0.64$ .

(а)

- 1) Активаційна функція всіх нейронів  $\phi(v) = \frac{1}{1 + \exp(-3v)}$ ;
- 2) Реакція виділеного нейрона  $\phi(v) = 0.32$ .

(б)

Рис. до задачі 25

26. Запропонувати та обґрунтувати алгоритм для методу навчання зі змаганням.

27. Виконати один крок алгоритму навчання зі змаганням для вхідного вектора  $x = (0, -1)$ . Вагові вектори вихідних нейронів такі:  $w_1 = (1, 0)$ ,  $w_2 = (0, -1)$  та  $w_3 = (-1, 0)$ ; крок навчання  $\eta = 0.3$  (див. рис. до цієї задачі).

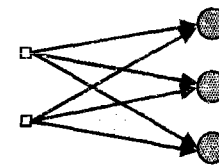


Рис. до задачі 27

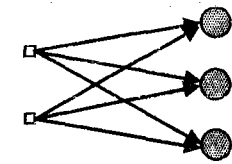


Рис. до задачі 28

28. Обчисліть якість апроксимації нейромережею навчання зі змаганням векторів, які гіпотетично входять до одного кластеру  $x_1 = (-2, 3)$ ,  $x_2 = (-2.5, 2.2)$ . Вагові вектори вихідних нейронів є  $w_1 = (1, 0)$ ,  $w_2 = (0, -1)$  та  $w_3 = (-1, 1)$  (див. рис. до цієї задачі).



## КОМП'ЮТЕРНІ ПРОЕКТИ

Скласти програми із зазначеними вхідними даними та результатами

1. Реалізуйте програмно алгоритм FIND-S. Спочатку перевірте, чи успішно він розв'язує приклад *НасСпортом*. Використайте цю програму для визначення кількості випадкових навчальних прикладів, необхідних для точного навчання цільового поняття. Реалізуйте генератор навчальних прикладів, що спочатку генерує випадкові приклади, а потім класифікує їх відповідно до цільового поняття  $\langle \text{Сонячно, Тепла, ?, ?, ?} \rangle$ . Розгляньте процес навчання вашої програми FIND-S на випадково генерованих прикладах і визначте кількість прикладів, потрібних для того, щоб знайдена програмою гіпотеза була ідентичною цільовому поняттю. Чи можна передбачити середню кількість необхідних прикладів? Виконайте експеримент принаймні 20 разів, і отримайте середню кількість необхідних прикладів. Як очікувана кількість прикладів залежить від кількості „?” у цільовому понятті? Як відрізняється ця кількість від кількості атрибутів, які описують приклади та гіпотези?

2. Задано таблицю зі значеннями властивостей об'єктів та мітками класів. Використовуючи алгоритм ID3, побудувати дерево рішень; на його основі виписати правила ЯКІЩО – ТО.

3. Задано простір станів у вигляді орієнтованого ациклічного графа. Знайти цільову вершину пошуком ушир з ітераційним заглибленням. Передбачити різні варіанти обмеження глибини пошуку.

4. Задано задачу про комівояжера. Реалізувати евристичний пошук, використовуючи „стратегію найближчого сусіда”.

5. Задано задачу про комівояжера. Реалізувати евристичний пошук, використовуючи евристику, відмінну від „стратегії найближчого сусіда”.

6. Задача лінійної регресії. Велику кількість практичних проблем аналізу даних (часових рядів тощо) можна звести до задачі про найменші квадрати. Найпростіший, але водночас найбільш поширений її варіант – задача лінійної регресії (див. приклад 3.52).

Потрібно написати програму-нейросимулятор, яка б розв'язувала задачу лінійної регресії відповідно до таких вимог.

◆ Реалізувати одношарову нейромережу прямого поширення з одними обчислювальним елементом відповідно до моделі лінійного нейрона, наведених на рис. 3.43. Ця реалізація повинна приймати вектор вхідних значень та обчислювати реакцію нейромережі. Розмірність вхідного вектора 1 (відповідно до моделі, наведених на рис. 3.43). Передбачити можливість ініціалізації вагових коефіцієнтів довільними значеннями.

◆ Реалізувати процес навчання нейромережі, ґрунтуючись на дельта правилі. Потрібно реалізувати почергове, випадкове та епохальне (див. метод зворотного поширення помилки, де було введено поняття епохи) подання вхідних значень. Передбачити три критерії зупинки: наперед визначена кількість ітерацій; досягнення функцією енергії наперед визначеного значення; досягнення модуля різниці значень функції енергії для різних епох наперед визначеного значення.

◆ Передбачити вивід значення функції енергії та значень вагових коефіцієнтів після завершення навчання.

◆ Реалізувати засіб зчитування даних із текстового файлу, де кожна навчальна пара розташована на новому рядку, а значення самої пари розділені „;”.

◆ Забезпечити можливість побудови графіка протікання процесу навчання (бажано динамічно), у якому на вісі абсцис показано номери ітерацій, а на вісі ординат – значення функції енергії.

◆ Забезпечити динамічну візуалізацію процесу навчання, де було б зображено дані з навчальної множини та пряму, яку моделює нейромережа протягом навчання.

7. Планування проведення тенісних матчів. Потрібно сконструювати та навчити нейромережу, яку можна було б використати для визначення, чи можливо проводити змагання з тенісу залежно від погодних умов. Погодні умови та відповідний атрибут прийняття рішень подано в таблиці до цього проекту. Атрибут *гра* є атрибутом прийняття рішень.

Таблиця до проекту 7

| Погода | Температура | Вологість | Вітер   | Гра |
|--------|-------------|-----------|---------|-----|
| Сонце  | Спека       | Висока    | Слабкий | Ні  |
| Сонце  | Спека       | Висока    | Сильний | Ні  |
| Хмари  | Спека       | Висока    | Слабкий | Так |
| Дощ    | Помірно     | Висока    | Слабкий | Так |
| Дощ    | Холод       | Норма     | Слабкий | Так |
| Дощ    | Холод       | Норма     | Сильний | Ні  |
| Хмари  | Холод       | Норма     | Сильний | Так |
| Сонце  | Помірно     | Висока    | Слабкий | Ні  |
| Сонце  | Холод       | Норма     | Слабкий | Так |
| Дощ    | Помірно     | Норма     | Слабкий | Так |
| Сонце  | Помірно     | Норма     | Сильний | Так |
| Хмари  | Помірно     | Висока    | Сильний | Так |
| Хмари  | Спека       | Норма     | Слабкий | Так |
| Дощ    | Помірно     | Висока    | Сильний | Ні  |

**Вказівки.** Сформульовану задачу можна розглядати як задачу класифікації, де дані потрібно розділити на два класи – *погодні умови сприятливі для гри*, та *погодні умови несприятливі для гри*. Очевидно, що нейромережа має містити чотири елементи у вхідному прошарку, що відповідає чотирьом „вхідним” атрибутам (Погода, Температура, Вологість та Вітер), від яких залежить рішення, та один елемент у вихідному прошарку. Реакція нейрона у вихідному прошарку, а отже, і нейромережі, може набувати лише двох значень – „ні” і „так”.

Залежність між вхідними атрибутами та атрибутом прийняття рішень нелінійна, отже, виникає потреба використання багатошарової нейромережі із прихованими прошарками. Оптимальну кількість прихованих прошарків та елементів у них пропонується знайти експериментальним шляхом. Для навчання мережі доцільно використати алгоритм зворотного поширення помилок.

Оскільки нейромережа працює з числовими даними, то потрібно замінити словесні вхідні/вихідні дані відповідними числовими. Запропонуйте різні числові відповідники словесним значенням. Проаналізуйте ефективність вибору.



8. Обчислення кредитного ризику для забезпечення інвестицій. Одне з можливих застосувань нейромережі – визначення ризику надання кредитів банком. Потрібно сконструювати та навчити нейромережу для оцінювання *кредитного ризику*. Розглянемо оцінку ризику неповернення платежу за позицією банку на основі таких ознак про клієнта: *кредитоспроможність*, *поточний борг*, *застава* та *прибуток*. У таблиці до цього проекту наведено перелік осіб із відомим ризиком неплатежу за позицією. *Ризик* – атрибут прийняття рішення.

Таблиця до проекту 8

| Кредитоспроможність | Борг    | Застава   | Прибуток | Ризик    |
|---------------------|---------|-----------|----------|----------|
| Погана              | Високий | Немає     | 0 – 15   | Високий  |
| Невідома            | Високий | Немає     | 15 – 35  | Високий  |
| Невідома            | Низький | Немає     | 15 – 35  | Помірний |
| Невідома            | Низький | Немає     | 0 – 15   | Високий  |
| Невідома            | Низький | Немає     | Вище 35  | Низький  |
| Невідома            | Низький | Адекватна | Вище 35  | Низький  |
| Погана              | Низький | Немає     | 0 – 15   | Високий  |
| Погана              | Низький | Адекватна | Вище 35  | Помірний |
| Хороша              | Низький | Немає     | Вище 35  | Низький  |
| Хороша              | Високий | Адекватна | Вище 35  | Низький  |
| Хороша              | Високий | Немає     | 0 – 15   | Високий  |
| Хороша              | Високий | Немає     | 15 – 35  | Високий  |
| Хороша              | Високий | Немає     | Вище 35  | Помірний |
| Погана              | Високий | Немає     | 15 – 35  | Високий  |

## РОЗДІЛ 4



## МІРКУВАННЯ В УМОВАХ НЕВИЗНАЧЕНОСТІ

- ◆ Нечіткі множини
- ◆ Міркування з використанням нечітких множин
- ◆ Стохастичний підхід до подання невизначеності

## 4.1. НЕЧІТКІ МНОЖИНИ

Традиційна формальна логіка ґрунтується на двох припущеннях. Перше пов'язане з виявленням приналежності – для будь-якого елемента й множини, яка є підмножиною деякого універсуму, елемент належить або множині, або її доповненню. Друге припущення полягає в тому, що елемент не може належати одночасно й множині, і її доповненню. Обидва ці припущення порушуються в *теорії нечітких множин* (fuzzy set theory) Лофті Заде (L. Zadeh). З погляду цієї теорії множини й закони міркувань у рамках традиційної логіки називають *чіткими*.

## 4.1.1. Означення нечіткої множини

У 1965 р. у журналі „Information and Control” з'явилась відома праця Л. Заде „Fuzzy sets”. Побудову теорії нечітких множин спричинила необхідність опису понять і явищ, котрі мають багатозначний та неточний характер. Математичні методи, які використовують класичні теорію множин і логіку, не дають змоги розв'язувати проблеми такого типу.

За допомогою нечітких множин можна формально визначити такі неточні або багатозначні поняття як „висока температура”, „молода людина”, „середній зріст”, „велике місто” тощо. Перед означенням нечіткої множини вводять *область розв'язків*, на якій визначають нечітку множину. Область розв'язків ще називають *простором*, і він не є нечіткою множиною.

*Нечіткою* (або *розмитою*, *туманною*, *пушистою*) множиною  $X$  із деякого простору  $A$  ( $A \neq \emptyset$ ) називають множину пар  $X = \{(a, \mu_X(a)) \mid a \in A\}$ , що записують як  $X \subset A$ . Функцію  $\mu_X: A \rightarrow [0, 1]$  називають *функцією належності нечіткої множини*  $X$ . Ця функція приписує кожному елементу  $a \in A$  ступінь його належності нечіткій множині  $X$ . Розглядають три випадки:

- ◆  $\mu_X(a) = 1$  у разі повної належності елемента  $a$  нечіткій множині  $X$ , ( $a \in X$ );
- ◆  $\mu_X(a) = 0$  у разі відсутності належності елемента  $a$  нечіткій множині  $X$ , ( $a \notin X$ );
- ◆  $0 < \mu_X(a) < 1$  у разі часткової належності елемента  $a$  нечіткій множині  $X$ .

Для нечітких множин використовують спеціальний запис: якщо  $A = \{a_1, a_2, \dots, a_n\}$  – область розв'язків із скінченною кількістю елементів, то нечітку множину  $X \subset A$  записують так:

$$X = \frac{\mu_X(a_1)}{a_1} + \frac{\mu_X(a_2)}{a_2} + \dots + \frac{\mu_X(a_n)}{a_n} = \sum_{i=1}^n \frac{\mu_X(a_i)}{a_i}.$$

Зображення елементів нечіткої множини у вигляді простого дробу, у якому під ризкою записують елемент множини, а над ризкою – ступінь його належності, задає відношення між елементами  $a_1, \dots, a_n$  та відповідними ступенями належності  $\mu_X(a_1), \dots, \mu_X(a_n)$ . Іншими словами, запис  $\frac{\mu_X(a_i)}{a_i}$  ( $i=1, 2, \dots, n$ ) означає елемент

відношення  $(a_i, \mu_X(a_i))$ . Знак „+” тут використано як позначення для множини елементів. Якщо ж область розв’язків  $A$  – нескінченна множина, то нечітку множину  $X \subset A$  записують так:

$$X = \int_a \frac{\mu_X(a)}{a}.$$

**Приклад 4.1.** Нехай  $A = N$  – множина натуральних чисел. Уведемо поняття натуральних чисел, „близьких до числа 7”. Це можна зробити за допомогою нечіткої множини  $X \subset N$ , записаної у вигляді  $X = \frac{0.2}{4} + \frac{0.5}{5} + \frac{0.8}{6} + \frac{1}{7} + \frac{0.8}{8} + \frac{0.5}{9} + \frac{0.2}{10}$ .

**Приклад 4.2.** Якщо  $A = R$ , де  $R$  – множина дійсних чисел, то числа, „близькі до числа 7” визначимо за допомогою функції належності  $\mu_X(a) = \frac{1}{1+(a-7)^2}$ , а нечітку множину дійсних чисел „близьких до числа 7” запишемо так:  $X = \int_a \frac{(1+(a-7)^2)^{-1}}{a}$ .

Множину нечітких натуральних та дійсних чисел, які „близькі до числа 7”, можна записати різними способами. Наприклад, функцію  $\mu_X(a)$  ще можна записати так:

$$\mu_X(a) = \begin{cases} 1 - \sqrt{\frac{|a-7|}{3}}, & \text{якщо } 4 \leq a \leq 10 \\ 0 & \text{у всіх інших випадках.} \end{cases}$$

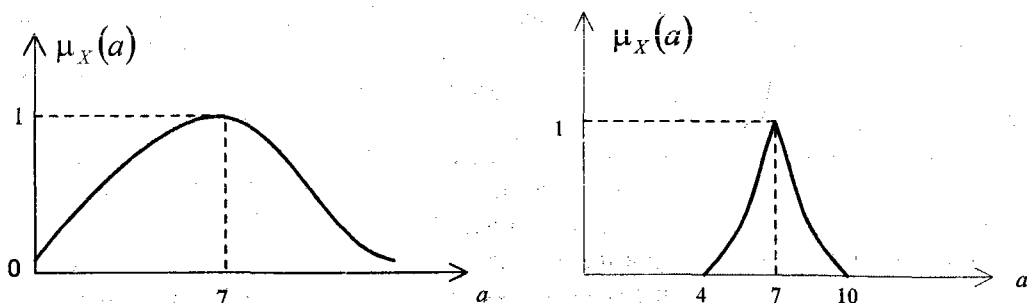


Рис. 4.1

Рис. 4.2

На рис. 4.1 та 4.2 наведено функції належності нечіткої множини  $X$  для введеного поняття дійсних чисел, які „близькі до числа 7”.

**Приклад 4.3.** Сформулюємо нечітке означення „приємна температура води в морі”. Задамо область розв’язків множиною  $A = \{15^\circ, \dots, 25^\circ\}$ . Тодіприємна температура води в  $21^\circ$  за оцінкою одного відпочиваючого може бути визначена нечіткою множиною

$$X = \frac{0.1}{16} + \frac{0.3}{17} + \frac{0.5}{18} + \frac{0.8}{19} + \frac{0.95}{20} + \frac{1}{21} + \frac{0.9}{22} + \frac{0.8}{23} + \frac{0.75}{24} + \frac{0.7}{25},$$

а за оцінкою іншогоприємною є температура від  $24^\circ$  до  $26^\circ$ , що можна подати нечіткою множиною

$$Y = \frac{0.1}{19} + \frac{0.2}{20} + \frac{0.4}{21} + \frac{0.7}{22} + \frac{0.9}{23} + \frac{1}{24} + \frac{1}{25} + \frac{1}{26} + \frac{0.8}{27} + \frac{0.75}{28} + \frac{0.7}{29}.$$

Розглянемо типові функції належності.

1. Функція належності типу  $s$ :

$$s(x; a, b, c) = \begin{cases} 0, & x \leq a, \\ 2\left(\frac{x-a}{c-a}\right)^2, & a < x \leq b, \\ 1 - 2\left(\frac{x-c}{c-a}\right)^2, & b < x \leq c, \\ 1, & x > c. \end{cases}$$

де  $b = (a+c)/2$ . Графік функції типу  $s$  наведено на рис. 4.3. Форму кривої визначає набір параметрів  $a, b, c$ . При  $x = b = (a+c)/2$  отримуємо  $s(b) = 0.5$ .

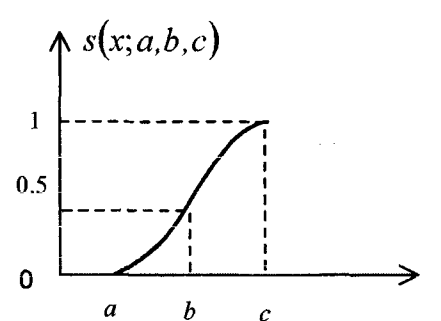


Рис. 4.3

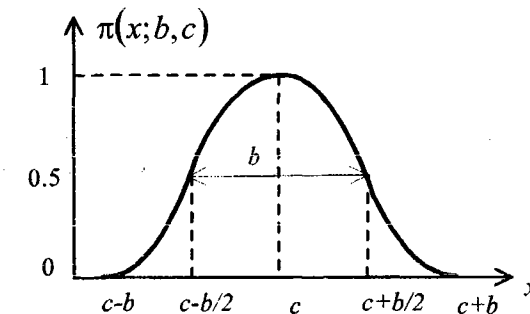


Рис. 4.4

2. Функція належності типу  $\pi$ :

$$\pi(x; b, c) = \begin{cases} s(x; c-b, c-b/2, c), & x \leq c, \\ 1 - s(x; c, c+b/2, c+b), & x > c. \end{cases}$$

Форму кривої визначають параметри  $b$  та  $c$ . Ця функція дорівнює нулю для  $x \geq c+b$  та  $x \leq c-b$ , а при  $x=c+b/2$  вона набуває значення 0,5. Графік функції типу  $\pi$  наведено на рис. 4.4.

### 3. Функція належності типу $\gamma$

$$\gamma(x; a, b) = \begin{cases} 0, & x \leq a; \\ \frac{x-a}{b-a}, & a < x \leq b; \\ 1, & x > b. \end{cases}$$

Форму кривої визначають параметри  $a$  та  $b$ . Графік функції типу  $\gamma$  наведено на рис. 4.5.

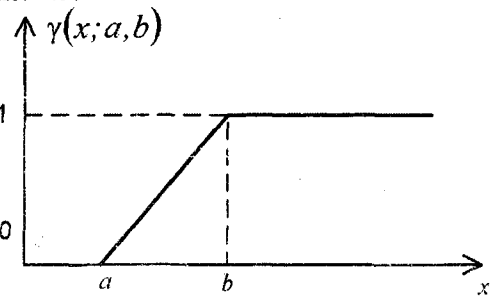


Рис. 4.5

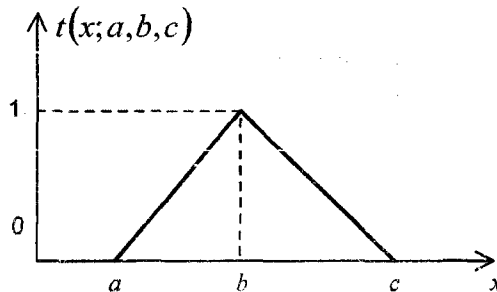


Рис. 4.6

### 4. Функція належності типу $t$ :

$$t(x; a, b, c) = \begin{cases} 0, & x \leq a; \\ \frac{x-a}{b-a}, & a < x \leq b; \\ \frac{c-x}{c-b}, & b < x \leq c; \\ 0, & x > c. \end{cases}$$

Форму кривої визначає набір параметрів  $a, b, c$ . Графік функції типу  $t$  наведено на рис. 4.6. Функція типу  $t$  є альтернативою функції типу  $\pi$ .

### 5. Функція належності типу $L$ :

$$L(x; a, b) = \begin{cases} 1, & x \leq a; \\ \frac{b-x}{b-a}, & a < x \leq b; \\ 0, & x > b. \end{cases}$$

Форму кривої визначають параметри  $a$  та  $b$ . Графік функції типу  $L$  наведено на рис. 4.7.

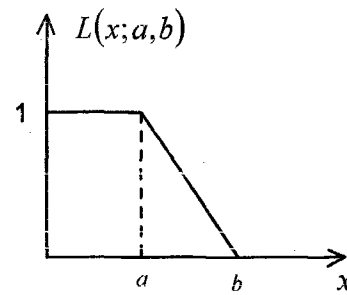


Рис. 4.7

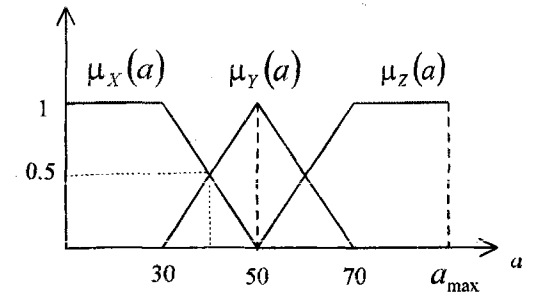


Рис. 4.8

**Приклад 4.4.** Уведемо означення нечітких множин  $X$  = „мала швидкість автомобіля”,  $Y$  = „середня швидкість автомобіля”,  $Z$  = „велика швидкість автомобіля”. Як область розв’язків візьмемо відрізок  $A = [0, a_{max}]$ , де  $a_{max}$  – можлива максимальна швидкість автомобіля. На рис. 4.8 наведено графіки функцій належності, що визначають зазначені нечіткі множини. Множину  $X$  задано функцією типу  $L$ , множину  $Y$  – функцією типу  $t$ , а множину  $Z$  – функцією типу  $\gamma$ . У точці  $a=40$  км/год функція належності нечіткої множини  $X$  набуває значення 0,5, бо  $\mu_X(40)=0.5$ . Таке ж значення набуває функція належності множини  $Y$ , тобто  $\mu_Y(40)=0.5$ , тоді як для множини  $Z$  матимемо  $\mu_Z(40)=0$ .

**Приклад 4.5.** На рис. 4.9 наведено нечітку множину  $N$  = „багато грошей”. Цю множину задано функцією типу  $s$ , де  $A = [0, 1000000]$  грн.,  $a=1000$  грн.,  $c=10000$  грн. Якщо вважати, що суми до 1000 грн. невеликі, то значення функції належності для таких сум дорівнює нулю. Якщо суми, більші від 10000 грн. уважати великими, то значення функції належності для таких сум дорівнює одиниці. Зрозуміло, що таке означення поняття „багато грошей” є суб’єктивним і кожен може самостійно означити ці поняття підбором параметрів  $a$  та  $c$  відповідної функції класу  $s$ .

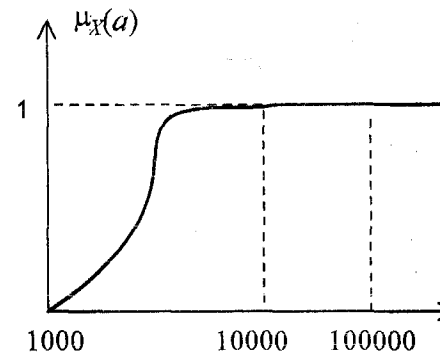


Рис. 4.9

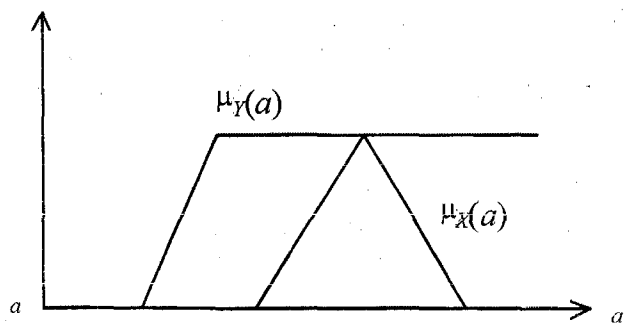


Рис. 4.10

Множину елементів області розв’язків  $A$ , для яких  $\mu_X(a) > 0$ , називають *носієм* нечіткої множини  $X$  та позначають як  $\text{supp } X = \{a \in A \mid \mu_X(a) > 0\}$ . Висотою нечіткої множини  $X$  називають число  $h(X) = \max_{a \in A} \mu_X(a)$ .

**Приклад 4.6.** Якщо множина  $A=\{1, 2, 3, 4, 5\}$  є областю розв'язків нечіткої

множини  $X = \frac{0.2}{1} + \frac{0.4}{2} + \frac{0.7}{4}$ , то  $\text{supp } X = \{1, 2, 4\}$ , а  $h(X)=0.7$ .

Нечітку множину  $X$  називають *нормальною*, якщо  $h(X) = 1$ . Якщо нечітка множина  $X$  не є нормальною, то нормалізацією множини  $X$  можна побудувати множину  $X_N$  за

правилом  $\mu_{X_N}(a) = \frac{\mu_X(a)}{h(X)}$ , де  $h(X)$  – висота множини  $X$ .

**Приклад 4.7.** Нечітку множину  $X = \frac{0.1}{2} + \frac{0.5}{4} + \frac{0.3}{6}$  з висотою  $h(X) = 0.5$

нормалізація перетворює на множину  $X_N = \frac{0.2}{2} + \frac{1}{4} + \frac{0.6}{6}$ .

Нечітку множину  $X$  називають *порожньою* та записують  $X=\emptyset$ , якщо  $\mu_X(a)=0$  для всіх  $a \in A$ . Нечітку множину  $X$  називають *підмножиною* нечіткої множини  $Y$ , якщо для кожного  $a \in A$  виконується нерівність  $\mu_X(a) \leq \mu_Y(a)$ . Тобто нечітка множина  $X$  включена в нечітку множину  $Y$ , що записують як  $X \subset Y$ . На рис. 4.10 наведено включення множини  $X$  у множину  $Y$ . Для нечітких множин  $X$  та  $Y$ , наведених на рис. 4.11, не виконується нерівність  $\mu_X(a) \leq \mu_Y(a)$  для кожного  $a \in A$ , отож уведенне означення підмножини не застосовне. Тому використовують поняття *ступеня включення* нечітких множин. Нечітка множина  $X$  є підмножиною нечіткої множини  $Y$  у ступені  $I$ , де  $I(X \subset Y) = \min_{a \in T} \mu_Y(a)$ ,  $T = \{a \in A \mid \mu_X(a) < \mu_Y(a), \mu_X(a) > 0\}$ . Це означення ілюструється прикладом на рис. 4.11. Для множин, наведених на рис. 4.10, ступінь включення дорівнює 1.

Нечіткі множини  $X$  та  $Y$  *рівні*, якщо  $\mu_X(a) = \mu_Y(a)$  для всіх  $a \in A$ . Це записують як  $X=Y$ . Наведене означення не універсальне, бо не враховує ситуації, коли значення  $\mu_X(a)$  та  $\mu_Y(a)$  мало відрізняються між собою. Тому вводять поняття *ступеня рівності* нечітких множин  $X$  та  $Y$ , яке визначають співвідношенням

$$E(X=Y) = 1 - \max_{a \in T} |\mu_X(a) - \mu_Y(a)|,$$

де  $T = \{a \in A \mid \mu_X(a) \neq \mu_Y(a)\}$ .

Множину  $X_\alpha = \{a \in A \mid \mu_X(a) \geq \alpha\}$ ,  $\alpha \in [0, 1]$  називають  $\alpha$ -перекриттям нечіткої множини  $X \subset A$ , її можна задати характеристичною функцією

$$\chi_{X_\alpha}(a) = \begin{cases} 1, & \mu_X(a) \geq \alpha; \\ 0, & \mu_X(a) < \alpha. \end{cases}$$

Множина  $X_\alpha$  – не є нечіткою. На рис. 4.12 для різних значень  $\alpha$  наведено множину  $X_\alpha$ . Зазначимо, що коли  $\alpha_2 < \alpha_1$ , то  $X_{\alpha_1} \subset X_{\alpha_2}$ .

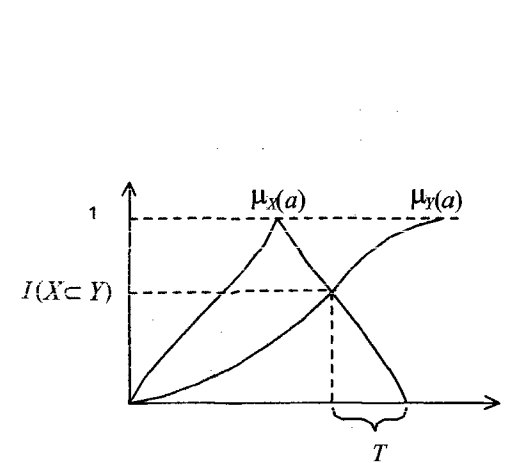


Рис. 4.11

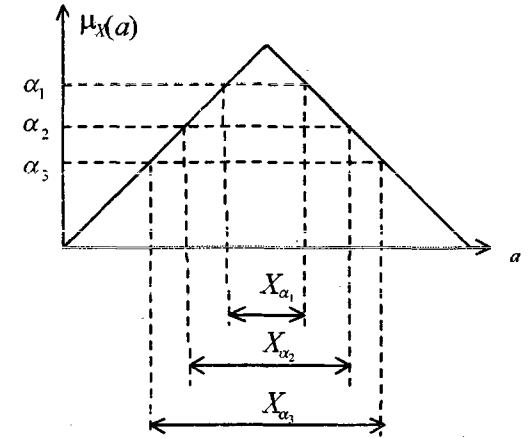


Рис. 4.12

**Приклад 4.8.** Нехай нечітка множина  $X = \frac{0.1}{2} + \frac{0.3}{4} + \frac{0.7}{5} + \frac{0.8}{8} + \frac{1}{10}$ ,  $X \subset A$  для  $A = \{1, \dots, 10\}$ . Тоді множини  $X_0 = A = \{1, \dots, 10\}$ ,  $X_{0.1} = \{2, 4, 5, 8, 10\}$ ,  $X_{0.3} = \{4, 5, 8, 10\}$ ,  $X_{0.7} = \{5, 8, 10\}$ ,  $X_{0.8} = \{8, 10\}$ ,  $X_1 = \{10\}$ .

Нечітку множину  $X \subset R$  називають *опуклою*, якщо для довільних  $a_1, a_2 \in R$  та  $\lambda \in [0, 1]$  виконується нерівність  $\mu_X(\lambda a_1 + (1-\lambda)a_2) \geq \min(\mu_X(a_1), \mu_X(a_2))$ . Приклад функції належності опуклої нечіткої множини наведено на рис. 4.13.

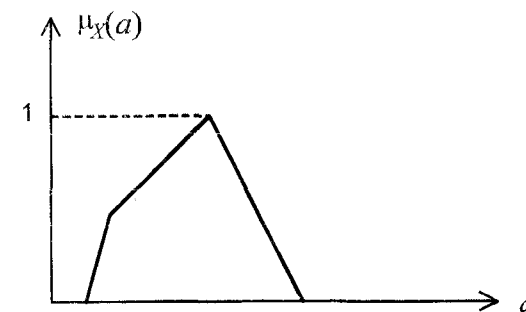


Рис. 4.14

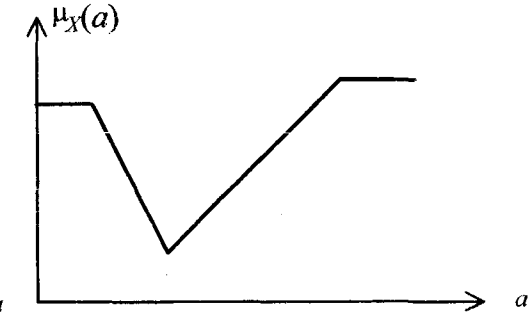


Рис. 4.13

Нечітку множину  $X \subset R$  називають *вгнутою*, якщо для довільних  $a_1, a_2 \in R$  та  $\lambda \in [0, 1]$  виконується нерівність  $\mu_X(\lambda a_1 + (1-\lambda)a_2) \leq \max(\mu_X(a_1), \mu_X(a_2))$ . Приклад функції належності вгнутої нечіткої множини наведено на рис. 4.14.

#### 4.1.2. Операції на нечітких множинах

Розглянемо операції на нечітких множинах.

*Перетин* нечітких множин  $X \subset A$  та  $Y \subset A$  – це нечітка множина  $X \cap Y$  із функцією належності  $\mu_{X \cap Y}(a) = \min(\mu_X(a), \mu_Y(a))$ , визначеною для всіх  $a \in A$ .

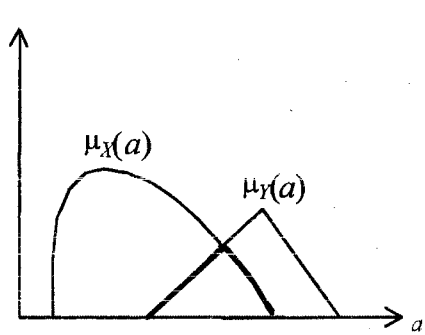


Рис. 4.15

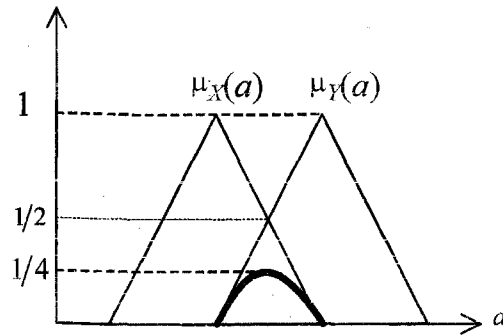


Рис. 4.16

На рис. 4.15 функцію належності перетину нечітких множин показано потовщеною лінією.

Перетин нечітких множин  $X_1, X_2, \dots, X_n$  задають функцією належності  $\mu_{X_1 \cap X_2 \cap \dots \cap X_n}(a) = \min(\mu_{X_1}(a), \mu_{X_2}(a), \dots, \mu_{X_n}(a))$ , визначеною для всіх  $a \in A$ .

Алгебраїчним добутком нечітких множин  $X$  та  $Y$  називають нечітку множину  $Z = X \cdot Y$ , яку визначають як  $Z = \{(a, \mu_X(a)\mu_Y(a)) \mid a \in A\}$ . Функцію належності алгебраїчного добутку нечітких множин  $X$  та  $Y$  показано потовщеною лінією на рис. 4.16.

Об'єднанням нечітких множин  $X$  та  $Y$  називають нечітку множину  $X \cup Y$  із функцією належності  $\mu_{X \cup Y}(a) = \max(\mu_X(a), \mu_Y(a))$ , визначеною для всіх  $a \in A$ . Результат виконання цієї операції зображено потовщеною лінією на рис. 4.17.

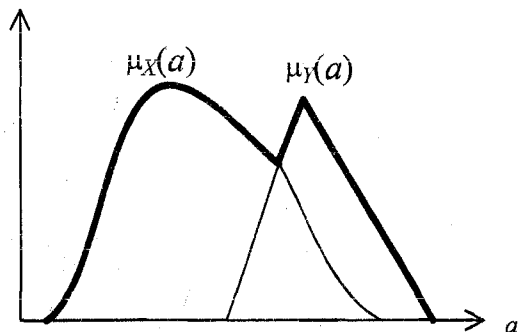


Рис. 4.17

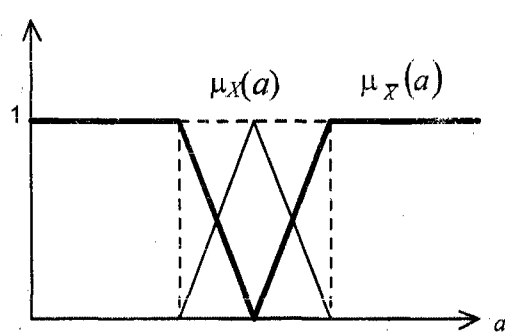


Рис. 4.18

Об'єднання нечітких множин  $X_1, X_2, \dots, X_n$  задають функцією належності  $\mu_{X_1 \cup X_2 \cup \dots \cup X_n}(a) = \max(\mu_{X_1}(a), \mu_{X_2}(a), \dots, \mu_{X_n}(a))$ , визначеною для всіх  $a \in A$ .

**Приклад 4.9.** Нехай  $A = \{1, 2, 3, 4, 5, 6, 7\}$ ,  $X = \frac{0.9}{3} + \frac{1}{4} + \frac{0.6}{6}$  та  $Y = \frac{0.7}{3} + \frac{1}{5} + \frac{0.4}{6}$ . За означенням перетину нечітких множин матимемо

$$X \cap Y = \frac{0.7}{3} + \frac{0.4}{6}, \text{ а за означенням об'єднання } - X \cup Y = \frac{0.9}{3} + \frac{1}{4} + \frac{1}{5} + \frac{0.6}{6}.$$

$$\text{Алгебраїчний добуток множини } X \text{ та } Y - \text{ нечітка множина } X \cdot Y = \frac{0.63}{3} + \frac{0.24}{6}.$$

Для довільної нечіткої множини  $X \subset A$  можна виконати її декомпозицію записом у вигляді  $X = \bigcup_{\alpha \in [0,1]} \alpha X_\alpha$ , де  $\alpha X_\alpha$  – нечіткі множини, елементи яких мають функції належності

$$\mu_{\alpha X_\alpha}(a) = \begin{cases} \alpha, & a \in X_\alpha \\ 0, & a \notin X_\alpha \end{cases}.$$

**Приклад 4.10.** Виконаємо декомпозицію нечіткої множини

$$X = \frac{0.1}{2} + \frac{0.3}{4} + \frac{0.7}{5} + \frac{0.8}{8} + \frac{1}{10},$$

де  $A = \{1, \dots, 10\}$ .

Одержимо

$$\begin{aligned} X &= 0.1X_{0.1} \cup 0.3X_{0.3} \cup 0.7X_{0.7} \cup 0.8X_{0.8} \cup 1X_{1.0} = \\ &= \left( \frac{0.1}{2} + \frac{0.1}{4} + \frac{0.1}{5} + \frac{0.1}{8} + \frac{0.1}{10} \right) \cup \left( \frac{0.3}{4} + \frac{0.3}{5} + \frac{0.3}{8} + \frac{0.3}{10} \right) \cup \\ &\cup \left( \frac{0.7}{5} + \frac{0.7}{8} + \frac{0.7}{10} \right) \cup \left( \frac{0.8}{8} + \frac{0.8}{10} \right) \cup \left( \frac{1}{10} \right). \end{aligned}$$

Доповненням нечіткої множини  $X \subset A$  називають нечітку множину  $\bar{X}$  із функцією належності  $\mu_{\bar{X}}(a) = 1 - \mu_X(a)$ . На рис. 4.18 потовщеною лінією наведено функцію належності  $\mu_{\bar{X}}(a)$  доповнення нечіткої множини  $A$ .

**Приклад 4.11.** Нехай  $A = \{1, 2, 3, 4, 5, 6\}$  та  $X = \frac{0.3}{2} + \frac{1}{3} + \frac{0.7}{5} + \frac{0.9}{6}$ .

За означенням одержимо нечітку множину  $\bar{X} = \frac{1}{1} + \frac{0.7}{2} + \frac{1}{4} + \frac{0.3}{5} + \frac{0.1}{6}$ .

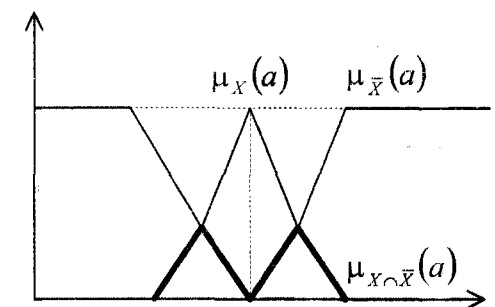


Рис. 4.19

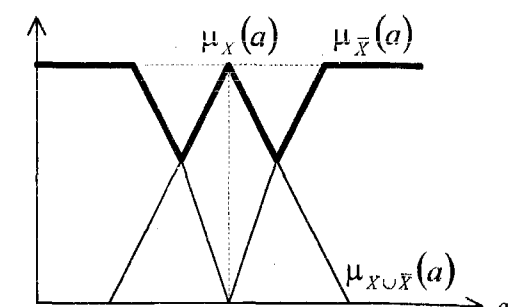


Рис. 4.20

Декартовим добутком  $X \times Y$  двох нечітких множин  $X \subset A$  та  $Y \subset B$  називають нечітку множину, визначену на парах елементів областей розв'язків  $A$  та  $B$  із функцією належності  $\mu_{X \times Y}(a, b) = \min(\mu_X(a), \mu_Y(b))$ , визначеною для всіх  $a \in A$  та  $b \in B$ . За альтернативним означенням декартовим добутком  $X \times Y$  називають нечітку множину з функцією належності  $\mu_{X \times Y}(a, b) = \mu_X(a) \mu_Y(b)$ , визначеною для всіх  $a \in A$  та  $b \in B$ .

**Приклад 4.12.** Нехай  $A = \{2, 4\}$  та  $B = \{2, 4, 6\}$  та  $X = \frac{0.5}{2} + \frac{0.9}{4}$ ,  $Y = \frac{0.3}{2} + \frac{0.7}{4} + \frac{0.1}{6}$ .

Тоді

$$X \times Y = \frac{0.3}{(2,2)} + \frac{0.5}{(2,4)} + \frac{0.1}{(2,6)} + \frac{0.3}{(4,2)} + \frac{0.7}{(4,4)} + \frac{0.1}{(4,6)},$$

а за альтернативним означенням

$$X \times Y = \frac{0.15}{(2,2)} + \frac{0.35}{(2,4)} + \frac{0.05}{(2,6)} + \frac{0.27}{(4,2)} + \frac{0.63}{(4,4)} + \frac{0.09}{(4,6)}.$$

Концентрацію нечіткої множини  $X \subset A$  (позначають  $CON(X)$ ) задають функцією належності  $\mu_{CON(X)} = (\mu_X(a))^2$ , визначеною для всіх  $a \in A$ .

Розтягування нечіткої множини  $X \subset A$  (позначають  $DIL(X)$ ) задають функцією належності  $\mu_{DIL(X)} = (\mu_X(a))^{1/2}$ , визначеною для всіх  $x \in X$ .

На рис. 4.21 наведено приклад функцій належності концентрації та розтягування.

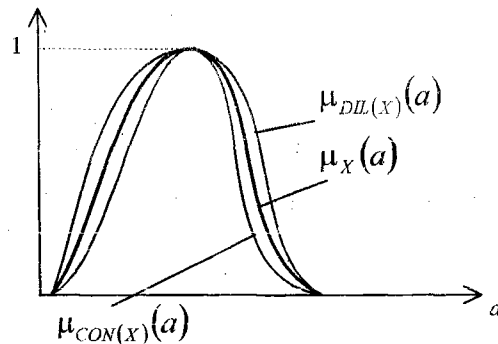


Рис. 4.21

**Приклад 4.13.** Нехай  $A = \{1, 2, 3, 4\}$  та  $X = \frac{0.4}{2} + \frac{0.7}{3} + \frac{1}{4}$ . Тоді

$$CON(X) = \frac{0.16}{2} + \frac{0.49}{3} + \frac{1}{4}, \quad DIL(X) = \frac{0.63}{2} + \frac{0.84}{3} + \frac{1}{4}.$$

### 4.1.3. Нечіткі числа

Нечітке (розмите) число — це нечітка множина  $X \subset R$ , визначена на множині дійсних чисел, причому виконуються такі умови:

- ◆ множина  $X$  нормальна, тобто  $\max \mu_X(a) = 1$ ;
- ◆ множина  $X$  опукла, тобто  $\mu_X(\lambda a_1 + (1-\lambda) a_2) \geq \min(\mu_X(a_1), \mu_X(a_2))$ ;
- ◆ функція належності  $\mu_X(a)$  множини  $X$  неперервна.

Нечітке число  $X \subset R$  додатне, якщо  $\mu_X(a) = 0$  для всіх  $a < 0$  та від'ємне, якщо  $\mu_X(a) = 0$  для всіх  $a > 0$ . На рис. 4.22 наведено функції належності від'ємного ( $a$ ), додатного ( $b$ ) нечітких чисел і нечітке число, яке не є ні від'ємним, ні додатним ( $\bar{b}$ ).

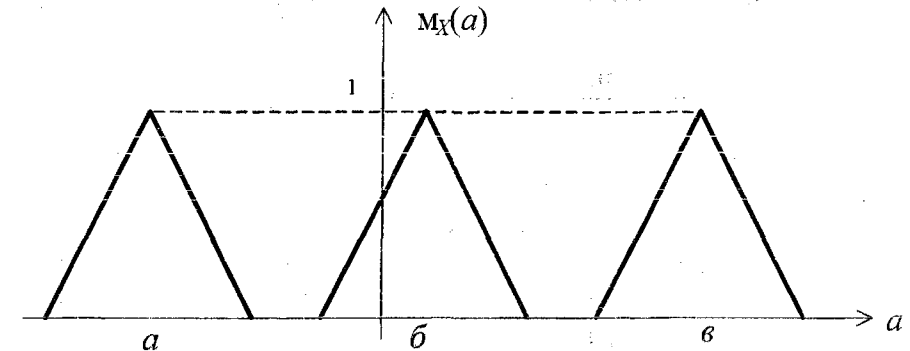


Рис. 4.22

Означимо головні операції над нечіткими числами  $X_1 \subset R$  та  $X_2 \subset R$ .

Сумою двох нечітких чисел  $X_1$  та  $X_2$  називають нечітке число  $Y = X_1 + X_2$  із функцією належності  $\mu_Y(b) = \max_{\substack{a_1, a_2 \\ b = a_1 + a_2}} \min(\mu_{X_1}(a_1), \mu_{X_2}(a_2))$ .

Різницею двох нечітких чисел  $X_1$  та  $X_2$  називають нечітке число  $Y = X_1 - X_2$  із функцією належності  $\mu_Y(b) = \max_{\substack{a_1, a_2 \\ b = a_1 - a_2}} \min(\mu_{X_1}(a_1), \mu_{X_2}(a_2))$ .

Добутком двох нечітких чисел  $X_1$  та  $X_2$  називають нечітке число  $Y = X_1 * X_2$  із функцією належності  $\mu_Y(b) = \max_{\substack{a_1, a_2 \\ b = a_1 * a_2}} \min(\mu_{X_1}(a_1), \mu_{X_2}(a_2))$ .

Часткою двох нечітких чисел  $X_1$  та  $X_2$  називають нечітке число  $Y = X_1 / X_2$  із функцією належності  $\mu_Y(b) = \max_{\substack{a_1, a_2 \\ b = a_1 / a_2}} \min(\mu_{X_1}(a_1), \mu_{X_2}(a_2))$ .

Для практичних застосувань цікавими є нечіткі числа з неперервною функцією належності. Для ілюстрації покажемо їхні дискретні аналоги.

**Приклад 4.14.** Розглянемо нечіткі числа  $X_1 = \frac{0.7}{2} + \frac{1}{3} + \frac{0.6}{4}$  та  $X_2 = \frac{0.8}{3} + \frac{1}{4} + \frac{0.5}{6}$ .

За означенням суми нечітких чисел одержимо

$$X_1 + X_2 = \frac{\min\{0.7, 0.8\}}{5} + \frac{\max\{\min\{0.7, 1\}, \min\{1, 0.8\}\}}{6} + \frac{\max\{\min\{1, 1\}, \min\{0.6, 0.8\}\}}{7} + \frac{\max\{\min\{0.7, 0.5\}, \min\{0.6, 1\}\}}{8} + \frac{\min\{1, 0.5\}}{9} + \frac{\min\{0.6, 0.5\}}{10} = \frac{0.7}{5} + \frac{0.8}{6} + \frac{1}{7} + \frac{0.6}{8} + \frac{0.5}{9} + \frac{0.7}{10}.$$

За означенням добутку нечітких чисел маємо



$$X_1 * X_2 = \frac{\min\{0.7, 0.8\}}{6} + \frac{\min\{0.7, 1\}}{8} + \frac{\min\{1, 0.8\}}{9} +$$

$$+ \frac{\max\{\min\{0.7, 0.5\}, \min\{1, 1\}, \min\{0.6, 0.8\}\}}{12} +$$

$$+ \frac{\min\{0.6, 1\}}{16} + \frac{\min\{1, 0.5\}}{18} + \frac{\min\{0.6, 0.5\}}{24} = \frac{0.7}{6} + \frac{0.7}{8} + \frac{0.8}{9} + \frac{1}{12} + \frac{0.6}{16} + \frac{0.5}{18} + \frac{0.5}{24}.$$

Зазначимо, що результат операції над нечіткими числами – не завжди нечітке число. Наприклад, добуток  $X_2 * X_2 = \frac{0.7}{3} + \frac{0.7}{4} + \frac{0.7}{5} + \frac{0.8}{6} + \frac{1}{8} + \frac{0.8}{9} + \frac{0.9}{10} + \frac{0.8}{12} + \frac{0.8}{15}$  нечітких чисел  $X_1 = \frac{0.7}{1} + \frac{1}{2} + \frac{0.8}{3}$  та  $X_2 = \frac{0.8}{3} + \frac{1}{4} + \frac{0.9}{5}$  не є нечітким числом, бо для нього не виконується умова опуклості.

#### 4.1.4. Нечіткі відношення та їхні властивості

Одним із головних понять теорії нечітких множин є поняття нечіткого відношення. Такі відношення дають змогу побудувати неточні формулювання типу „ $x$  майже дорівнює  $y$ ” або „ $x$  значно більше ніж  $y$ ”.

Нечітким відношенням  $R$  із непорожньої множини  $A$  у непорожню множину  $B$  називають нечітку множину  $R \subset A \times B = \{(a, b) | a \in A, b \in B\}$ . Зазначимо, що множини  $A$  та  $B$  не є нечіткими. Отож, нечітке відношення – це множина пар  $R = \{((a, b), \mu_R(a, b))\}$ , де  $\mu_R: A \times B \rightarrow [0, 1]$  – функція належності цього відношення, яка приписує кожній парі  $(a, b)$  ступінь її належності  $\mu_R(a, b)$  до множини  $R$ . Функцію  $\mu_R(a, b)$  інтерпретують як величину зв'язку між елементами  $a \in A$  та  $b \in B$ . Згідно із прийнятою системою позначень

$$\text{нечітке відношення можна записати як } R = \sum_{a \times b} \frac{\mu_R(a, b)}{(a, b)} \text{ або } R = \int_{a \times b} \frac{\mu_R(a, b)}{(a, b)}.$$

**Приклад 4.15.** Використаємо означення нечіткого відношення для формалізації неточного твердження. Нехай  $A = \{3, 4, 5\}$  та  $B = \{4, 5, 6\}$ . Тоді речення „ $a$  близьке до  $b$ ” за допомогою нечіткого відношення можна задати так:

$$R = \frac{1}{(4, 4)} + \frac{1}{(5, 5)} + \frac{0.8}{(3, 4)} + \frac{0.8}{(4, 5)} + \frac{0.8}{(5, 4)} + \frac{0.8}{(5, 6)} + \frac{0.6}{(3, 5)} + \frac{0.6}{(4, 6)} + \frac{0.4}{(3, 6)}.$$

Отож, функція належності  $\mu_R(a, b)$  відношення  $R$  має вигляд

$$\mu_R = \begin{cases} 1, & \text{якщо } x = y, \\ 0.8, & \text{якщо } |x - y| = 1, \\ 0.6, & \text{якщо } |x - y| = 2, \\ 0.4, & \text{якщо } |x - y| = 3. \end{cases}$$

Уведене відношення також можна задати за допомогою матриці, рядки та стовпці якої позначено елементами множин  $A$  та  $B$ , відповідно:

|       | $b_1$ | $b_2$ | $b_3$ |
|-------|-------|-------|-------|
| $a_1$ | 0.8   | 0.6   | 0.4   |
| $a_2$ | 1     | 0.8   | 0.6   |
| $a_3$ | 0.8   | 1     | 0.8   |

Тут  $a_1=3, a_2=4, a_3=5, b_1=4, b_2=5, b_3=6$ .

**Приклад 4.16.** Нехай елементи множин  $A$  та  $B$  мають значення тривалості життя людини в роках, а саме  $A=B=\{0, 1, \dots, 120\}$ . Тоді відношення  $R$  із функцією належності

$$\mu_R(a, b) = \begin{cases} 0, & \text{якщо } a - b \leq 0, \\ \frac{a-b}{30}, & \text{якщо } 0 < a - b < 30, \\ 1, & \text{якщо } a - b \geq 30 \end{cases}$$

задає відношення „особа  $a$  значно старша особи  $b$ ”.

Зазначимо, що нечітке відношення  $R$  є нечіткою множиною. Тому залишаються правильними означення перетину, об'єднання та доповнення відношень:

$$\mu_{R \cap S}(a, b) = \min\{\mu_R(a, b), \mu_S(a, b)\},$$

$$\mu_{R \cup S}(a, b) = \max\{\mu_R(a, b), \mu_S(a, b)\},$$

$$\mu_{\bar{R}}(a, b) = 1 - \mu_R(a, b).$$

Для практичних застосувань важливе значення має поняття композиції двох нечітких відношень. Нехай маємо три множини  $A, B, C$  та два нечітких відношення з функціями належності  $\mu_R(a, b)$  та  $\mu_S(b, c)$ , відповідно. Множини  $A, B, C$  не є нечіткими.

Композицією нечіткого відношення  $R$  із множини  $A$  в множину  $B$  та нечіткого відношення  $S$  із множини  $B$  у множину  $C$  називають нечітке відношення із множини  $A$  в множину  $C$  (позначають  $S \circ R$ ) із функцією належності

$$\mu_{S \circ R}(a, c) = \sup_{b \in B} \min(\mu_R(a, b), \mu_S(b, c)).$$

**Приклад 4.17.** Нехай відношення  $R \subset A \times B$  та  $S \subset B \times C$ , де  $A = \{a_1, a_2\}$ ,  $B = \{b_1, b_2\}$ ,

$$C = \{c_1, c_2, c_3\}, \text{ задані матрицями } R = \begin{pmatrix} 0.2 & 0.5 \\ 0.6 & 1 \end{pmatrix} \text{ та } S = \begin{pmatrix} 0.3 & 0.6 & 0.8 \\ 0.7 & 0.9 & 0.4 \end{pmatrix}.$$

Композиція відношень  $R$  та  $S$  має вигляд

$$Q = S \circ R = \begin{pmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \end{pmatrix}.$$

Обчислимо елемент  $q_{11}$  матриці  $Q$ :

$$q_{11} = q(a_1, c_1) = \max\{\min\{\mu(a_1, b_1), \mu(b_1, c_1)\}, \min\{\mu(a_1, b_2), \mu(b_2, c_1)\}\} =$$

$$= \max\{\min\{0.2, 0.3\}, \min\{0.5, 0.7\}\} = \max\{0.2, 0.5\} = 0.5.$$

Аналогічно обчислюють решту елементів матриці  $Q$ :

$$\begin{aligned} q_{12} &= \max(\min(0.2, 0.6), \min(0.5, 0.9)) = 0.5; \\ q_{13} &= \max(\min(0.2, 0.8), \min(0.5, 0.4)) = 0.4; \\ q_{21} &= \max(\min(0.6, 0.3), \min(1, 0.7)) = 0.7; \\ q_{22} &= \max(\min(0.6, 0.6), \min(1, 0.9)) = 0.9; \\ q_{23} &= \max(\min(0.6, 0.8), \min(1, 0.4)) = 0.6. \end{aligned}$$

$$\text{Отже, } Q = \begin{pmatrix} 0.5 & 0.5 & 0.4 \\ 0.7 & 0.9 & 0.6 \end{pmatrix}.$$

У різних застосуваннях важливою є композиція нечіткої множини та нечіткого відношення. Нехай задано нечітку множину  $X \subset A$  та нечітке відношення  $R \subset A \times B$  із функціями належності, відповідно,  $\mu_X(a)$  та  $\mu_R(a, b)$ . Композицією нечіткої множини  $X$  та нечіткого відношення  $R$  є нечітка множина  $Y \subset B$  із функцією належності

$$\mu_Y(b) = \sup_{a \in A} \min(\mu_X(a), \mu_R(a, b)),$$

яку визначають за правилом  $Y = R \circ X$ .

**Приклад 4.18.** Нехай задано множини  $A = \{a_1, a_2, a_3\}$ ,  $B = \{b_1, b_2\}$  та нечітку множину

$$X = \frac{0.4}{a_1} + \frac{1}{a_2} + \frac{0.6}{a_3},$$

а нечітке відношення  $R$  задано матрицею

$$R = \begin{pmatrix} 0.5 & 0.7 \\ 0.2 & 1 \\ 0.9 & 0.3 \end{pmatrix}.$$

Композиція  $Y = R \circ X$  має вигляд

$$Y = \frac{\mu_Y(b_1)}{b_1} + \frac{\mu_Y(b_2)}{b_2},$$

де

$$\begin{aligned} \mu_Y(b_1) &= \max(\min(0.4; 0.5), \min(1; 0.2), \min(0.6; 0.9)) = 0.6, \\ \mu_Y(b_2) &= \max(\min(0.4; 0.7), \min(1; 1), \min(0.6; 0.3)) = 1. \end{aligned}$$

$$\text{Отже, } Y = \frac{0.6}{b_1} + \frac{1}{b_2}.$$

#### 4.1.5. Трикутні норми

Розглянемо перетин нечітких множин  $X$  та  $Y$ , функцію належності якого раніше було означено як  $\mu_{X \cap Y}(a) = \min(\mu_X(a), \mu_Y(a))$ . Зазначимо, що це не єдиний спосіб означення цієї операції. Її можна узагальнити, якщо в означенні функції належності замість  $\min$  узяти деяку двомісну функцію  $T$ , яку називають  $T$ -нормою. Тоді  $\mu_{X \cap Y}(a) = T(\mu_X(a), \mu_Y(a))$ .

Аналогічно, об'єднання нечітких множин, функцію належності якого було означено як  $\mu_{X \cup Y}(a) = \max(\mu_X(a), \mu_Y(a))$ , узагальнюють введенням двомісної функції  $S$ :  $\mu_{X \cup Y}(a) = S(\mu_X(a), \mu_Y(a))$ .

Функцію двох змінних  $T: [0, 1] \times [0, 1] \rightarrow [0, 1]$  називають  $T$ -нормою, якщо:

♦ функція  $T$  є незростаючою функцією обох аргументів:  $T(x, y) \leq T(u, v)$  для  $x \leq u$ ,  $y \leq v$ ;

♦ для функції  $T$  виконується умова комутативності  $T(x, y) = T(y, x)$ ;

♦ для функції  $T$  виконується умова зв'язності  $T(T(x, y), u) = T(x, T(y, u))$ ;

♦ функція  $T$  задовольняє граничні умови  $T(x, 0) = 0$ ;  $T(x, 1) = x$ .

Тут всюди  $x, y, u, v \in [0, 1]$ .

Довільна  $T$ -норма задовольняє обмеження  $T_w(x, y) \leq T(x, y) \leq \min(x, y)$ , де

$$T_w(x, y) = \begin{cases} x, & \text{якщо } y = 1, \\ y, & \text{якщо } x = 1, \\ 0, & \text{якщо } x \neq 1 \text{ та } y \neq 1. \end{cases}$$

$T$ -норму як функцію аргументів  $x$  та  $y$  позначають  $T(x, y) = x \overset{T}{*} y$ . Відповідно до введеного позначення функцію належності перетину нечітких множин  $A$  та  $B$  записують у вигляді

$$\mu_{A \cap B}(a) = T(\mu_X(a), \mu_Y(a)) = \mu_X(a) \overset{T}{*} \mu_Y(a).$$

Функцію двох змінних  $S: [0, 1] \times [0, 1] \rightarrow [0, 1]$  називають  $S$ -нормою, якщо вона є незростаючою функцією обох аргументів, задовольняє умову комутативності, умову зв'язності та граничні умови  $S(x, 0) = x$ ;  $S(x, 1) = 1$ . Функцію  $S$  називають також *конормою* або *нормою*, *дуальною* до  $T$ -норми. Довільна  $S$ -норма задовольняє обмеження  $\max(x, y) \leq S(x, y) \leq S_w(x, y)$ , де

$$S_w(x, y) = \begin{cases} x, & \text{якщо } y = 0, \\ y, & \text{якщо } x = 0, \\ 1, & \text{якщо } x \neq 0 \text{ та } y \neq 0. \end{cases}$$

$S$ -норму як функцію аргументів  $x$  та  $y$  позначають  $S(x, y) = x \overset{S}{*} y$ , отже, функцію належності об'єднання нечітких множин  $A$  та  $B$  можна записати у вигляді

$$\mu_{A \cup B}(a) = S(\mu_X(a), \mu_Y(a)) = \mu_X(a) \overset{S}{*} \mu_Y(a).$$

Кожній  $S$ -нормі відповідає  $T$ -норма за формулою

$$x \overset{T}{*} y = 1 - \left( (1-x) \overset{S}{*} (1-y) \right).$$

$T$  та  $S$  норми називають *трикутними нормами*. У табл. 4.1 наведено різні трикутних норм.

Таблиця 4.1

| № | $T(x, y)$   | $S(x, y)$   | Параметри           |
|---|---|---|---------------------|
| 1 | $\min(x, y)$  | $\max(x, y)$  |                     |
| 2 | $xy$  | $x+y-xy$  |                     |
| 3 | $\max(x+y-1, 0)$  | $\min(x+y, 1)$  |                     |
| 4 | $\begin{cases} x, \text{ якщо } y=1, \\ y, \text{ якщо } x=1, \\ 0, \text{ якщо } x \neq 1 \text{ та } y \neq 1. \end{cases}$ | $\begin{cases} x, \text{ якщо } y=0, \\ y, \text{ якщо } x=0, \\ 1, \text{ якщо } x \neq 0 \text{ та } y \neq 0. \end{cases}$ |                     |
| 5 | $\frac{xy}{\gamma + (1-\gamma)(x+y-xy)}$  | $\frac{x+y-(2-\gamma)xy}{1-(1-\gamma)xy}$   | $\gamma > 0$        |
| 6 | $\frac{xy}{\max(x, y, \alpha)}$   | $\frac{x+y-xy-\min(x, y, 1-\alpha)}{\max(1-\alpha, 1-x, \alpha)}$   | $\alpha \in [0, 1]$ |
| 7 | $1 - \sqrt[p]{(1-x)^p + (1-y)^p} + (1-x)^p(1-y)^p$  | $\sqrt[p]{x^p + y^p - x^p y^p}$   | $p > 0$             |
| 8 | $\max\left(1 - \sqrt[p]{(1-x)^p + (1-y)^p}, 0\right)$   | $\min\left(\sqrt[p]{x^p + y^p}, 1\right)$   | $p \geq 1$          |

$T$ -норми дають змогу узагальнити поняття композиції відношень. Композицією нечіткого відношення  $R$  із множини  $A$  у множини  $B$  та нечіткого відношення  $S$  із множини  $B$  у множини  $C$  називають нечітке відношення  $S \circ R$  із множини  $A$  у множини  $C$  із функцією

належності  $\mu_{S \circ R}(a, c) = \sup_{b \in B} \left( \mu_R(a, b) * \mu_S(b, c) \right)$ . Вигляд функції належності  $\mu_{S \circ R}(a, b)$

залежить від прийнятої  $T$ -норми. Наприклад, для  $T$ -норми  $T(x, y) = xy$  функція належності композиції відношень така  $\mu_{S \circ R}(a, c) = \sup_{b \in B} (\mu_R(a, b) \cdot \mu_S(b, c))$ .

Аналогічно, можна узагальнити поняття композиції нечіткої множини  $X$  та нечіткого відношення  $R$  за допомогою функції належності  $\mu_Y(b) = \sup_{a \in A} \left( \mu_X(a) * \mu_R(a, b) \right)$ .

## 4.2. МІРКУВАННЯ З ВИКОРИСТАННЯМ НЕЧІТКИХ МНОЖИН

У цьому підрозділі розглянуто способи побудови міркувань із використанням нечітких множин.

### 4.2.1. Логічне виведення з використанням апарату нечітких множин

Лінгвістичною називають змінну, яка має своїми значеннями слова природної мови. Розглянемо речення: „мала швидкість”, „помірна температура”, „молода людина”. Значеннями лінгвістичних змінних у цих реченнях є слова „мала”, „помірна” та „молода”. Такі речення можна формалізувати за допомогою відповідних нечітких множин.

## Розділ 4. Міркування в умовах невизначеності

Головним правилом виведення в численні висловлювань є правило *modus ponens*, яке задають схемою логічного виведення  $p, p \rightarrow q \vdash q$ .

Відповідне нечітке правило виведення – аналог правила *modus ponens* – задають такою схемою логічного виведення:

**факт:**  $a$  це  $X'$ ;

**правило:** якщо  $a$  це  $X$ , то  $b$  це  $Y$ ;

**висновок:**  $b$  це  $Y'$ ;

де  $a$  та  $b$  – лінгвістичні змінні, а  $X'$ ,  $X$ ,  $Y$  та  $Y'$  ( $X \subset A$ ,  $X' \subset A'$ ,  $Y \subset B$ ,  $Y' \subset B'$ ) – нечіткі множини.

**Приклад 4.19.** Задано схему нечіткого логічного виведення:

**факт:** велика швидкість автомобіля;

**правило:** якщо швидкість автомобіля дуже велика, то рівень шуму високий;

**висновок:** рівень шуму не дуже високий.

У цій схемі факт, правило та висновок – речення з лінгвістичними змінними. Розглянемо лінгвістичну змінну  $a$  „швидкість автомобіля” із множиною значень  $T_1 = \{„мала”, „середня”, „велика”, „дуже велика”\}$  та лінгвістичну змінну  $b$  „рівень шуму” із множиною значень  $T_2 = \{„малий”, „середній”, „не дуже високий”, „високий”\}$ . Кожному елементу множин  $T_1$  та  $T_2$  можна поставити у відповідність нечітку множину. Задана схема логічного виведення містить такі неточні речення як „дуже велика швидкість автомобіля” із нечіткою множиною  $X$ , „велика швидкість автомобіля” із нечіткою множиною  $X'$ , „високий рівень шуму” із нечіткою множиною  $Y$  та „не дуже високий рівень шуму” із нечіткою множиною  $Y'$ . Тут речення „дуже велика швидкість автомобіля”, яке міститься в правилі, та речення „велика швидкість автомобіля”, яке міститься у факті, та відповідні нечіткі множини – різні.

Якщо в схемі логічного виведення застосовують нечітку множину  $X$  із функцією належності  $\mu_X(a)$  та нечітку множину  $X'$  із функцією належності  $\mu_{X'}(a)$ , то можна встановити такий зв'язок між ними:

- ◆  $X' = X$  із функцією належності  $\mu_{X'}(a) = \mu_X(a)$ ;
- ◆  $X' = „дуже X”$  із функцією належності  $\mu_{X'}(a) = \mu_X^2(a)$ ;
- ◆  $X' = „майже X”$  із функцією належності  $\mu_{X'}(a) = \mu_X^{1/2}(a)$ ;
- ◆  $X' = „не X”$  із функцією належності  $\mu_{X'}(a) = 1 - \mu_X(a)$ .

Тоді між нечіткими множинами  $X'$  та  $Y'$  існують інтуїтивні зв'язки, які наведено в табл. 4.2. Співвідношення 1 є звичайним правилом *modus ponens*, 3 та 5 означають, що немає безпосереднього зв'язку між  $X'$  та  $Y'$ , співвідношення 6 означає, що за заданим фактом та за допомогою нечіткого правила *modus ponens* не можна отримати висновку.

Таблиця 4.2

| Співвідношення | Факт: $a$ це $X'$   | Висновок: $b$ це $Y'$ |
|----------------|---------------------|-----------------------|
| 1              | $a$ це $X$          | $b$ це $Y$            |
| 2              | $a$ це „дуже $X$ ”  | $b$ це „дуже $Y$ ”    |
| 3              | $a$ це „дуже $X$ ”  | $b$ це $Y$            |
| 4              | $a$ це „майже $X$ ” | $b$ це „майже $Y$ ”   |
| 5              | $a$ це „майже $X$ ” | $b$ це $Y$            |
| 6              | $A$ це „не $X$ ”    | $b$ не визначено      |
| 7              | $A$ це „не $X$ ”    | $b$ це „не $Y$ ”      |

Правило в схемі логічного виведення з нечіткими множинами називають *нечіткою імплікацією*  $X \rightarrow Y$ , яку визначають як нечітке відношення  $R \subset A \times B$  із функцією належності  $\mu_R(a, b) = \mu_{X \rightarrow Y}(a, b)$ . Якщо  $X$  та  $Y$  ( $X \subset A$ ,  $Y \subset B$ ) – нечіткі множини з функціями належності  $\mu_X(a)$  та  $\mu_Y(b)$ , то функцію належності  $\mu_{X \rightarrow Y}(a, b)$  нечіткої імплікації  $X \rightarrow Y$  задають одним із правил, наведених у табл. 4.3.

Таблиця 4.3

| №  | Назва правила              | Функція належності $\mu_{X \rightarrow Y}(a, b)$ нечіткої імплікації   |
|----|----------------------------|--|
| 1  | Правило мінімуму (Мамдані) | $\mu_{X \rightarrow Y}(a, b) = \min(\mu_X(a), \mu_Y(b))$   |
| 2  | Правило добутку (Ларсена)  | $\mu_{X \rightarrow Y}(a, b) = \mu_X(a) \cdot \mu_Y(b)$  |
| 3  | Правило Лукасевича         | $\mu_{X \rightarrow Y}(a, b) = \min(1, 1 - \mu_X(a) + \mu_Y(b))$   |
| 4  | Правило max-min (Заде)     | $\mu_{X \rightarrow Y}(a, b) = \max(\min(\mu_X(a), \mu_Y(b)), 1 - \mu_X(a))$   |
| 5  | Бінарне правило            | $\mu_{X \rightarrow Y}(a, b) = \max(1 - \mu_X(a), \mu_Y(b))$   |
| 6  | Правило Гогена             | $\mu_{X \rightarrow Y}(a, b) = \min\left(1, \frac{\mu_Y(b)}{\mu_X(a)}\right)$  |
| 7  | Правило Шарпа              | $\mu_{X \rightarrow Y}(a, b) = \begin{cases} 1, & \text{якщо } \mu_X(a) \leq \mu_Y(b) \\ 0, & \text{якщо } \mu_X(a) > \mu_Y(b) \end{cases}$        |
| 8  | Правило Гьоделя            | $\mu_{X \rightarrow Y}(a, b) = \begin{cases} 1, & \text{якщо } \mu_X(a) \leq \mu_Y(b) \\ \mu_Y(b), & \text{якщо } \mu_X(a) > \mu_Y(b) \end{cases}$ |
| 9  | Імовірнісне правило        | $\mu_{X \rightarrow Y}(a, b) = \min(1, 1 - \mu_X(a) + \mu_X(a) \cdot \mu_Y(b))$  |
| 10 | Правило обмеженої суми     | $\mu_{X \rightarrow Y}(a, b) = \min(1, \mu_X(a) + \mu_Y(b))$   |

**Висновок нечіткого правила modus ponens** – це нечітка множина  $Y'$ . Ця множина являє собою композицію нечіткої множини  $X'$  та нечіткої імплікації  $X \rightarrow Y$ :

$$Y' = (X \rightarrow Y) \circ X'.$$

Функцію належності нечіткої множини  $Y'$  можна обчислити за формулою

$$\mu_{Y'}(b) = \sup_{a \in A} (\min(\mu_{X'}(a), \mu_{X \rightarrow Y}(a, b))).$$

Якщо задати  $\mu_{X \rightarrow Y}(a, b) = \min(\mu_X(a), \mu_Y(b))$ , то функцію належності нечіткої множини  $Y'$  можна визначити за формулою

$$\mu_{Y'}(b) = \sup_{a \in A} (\min(\mu_{X'}(a), \min(\mu_X(a), \mu_Y(b)))).$$

Зазначимо, що коли  $X = X'$  та  $Y = Y'$ , то нечітке та чітке правила modus ponens співпадають.

**Приклад 4.20.** Задамо функцію належності нечіткої множини  $Y' = (X \rightarrow Y) \circ X'$  формулою  $\mu_{Y'}(b) = \sup_{a \in A} (\min(\mu_{X'}(a), \mu_{X \rightarrow Y}(a, b)))$ . Цю функцію належності розглядатимемо для різних способів подання нечіткої множини  $X'$ . З'ясуємо, які зі співвідношень,

наведених у табл. 4.2, справджуються, якщо нечітку імплікацію визначено за правилом Ларсена. Додатково припускаємо, що  $\sup_{a \in A} (\mu_X(a)) = 1$ .

1. Покладемо  $X' = X$ . У разі такого визначення множини  $X'$  нечітка множина  $Y'$  має функцію належності

$$\mu_{Y'}(b) = \sup_{a \in A} (\min(\mu_X(a), \mu_{X \rightarrow Y}(a, b))) = \sup_{a \in A} (\min(\mu_X(a), \mu_X(a) \mu_Y(b))) = \mu_Y(b).$$

Цей результат співпадає зі співвідношенням 1 із табл. 4.2.

2. Покладемо  $X' =$  „дуже  $X$ ” та  $\mu_{X'}(a) = \mu_X^2(a)$ . У разі такого визначення множини  $X'$  нечітка множина  $Y'$  має функцію належності

$$\mu_{Y'}(b) = \sup_{a \in A} (\min(\mu_X^2(a), \mu_{X \rightarrow Y}(a, b))) = \sup_{a \in A} (\min(\mu_X^2(a), \mu_X(a) \mu_Y(b))) = \mu_Y(b).$$

Цей результат збігається зі співвідношенням 3 із табл. 4.2.

3. Покладемо  $X' =$  „майже  $X$ ” та  $\mu_{X'}(a) = \mu_X^{1/2}(a)$ . У разі такого визначення множини  $X'$  нечітка множина  $Y'$  має функцію належності

$$\mu_{Y'}(b) = \sup_{a \in A} (\min(\mu_X^{1/2}(a), \mu_{X \rightarrow Y}(a, b))) = \sup_{a \in A} (\min(\mu_X^{1/2}(a), \mu_X(a) \mu_Y(b))) = \mu_Y(b).$$

Цей результат збігається зі співвідношенням 5 із табл. 4.2.

**Приклад 4.21.** В умовах прикладу 4.20 знайдемо нечітку імплікацію за правилом Шарпа. Аналогічно як і в попередньому прикладі одержимо функції належності нечіткої множини  $Y'$ . Результати розв'язування наведено в табл. 4.4

Таблиця 4.4

| Нечітка множина $X'$  | Функція належності нечіткої множини $X'$ | Функція належності нечіткої множини $Y'$ | Правило з табл. 4.2 |
|-----------------------|--|--|---------------------|
| $X' = X$              | $\mu_{X'}(a) = \mu_X(a)$                 | $\mu_{Y'}(b) = \mu_Y(b)$                 | 1                   |
| $X' =$ „дуже $X$ ”    | $\mu_{X'}(a) = \mu_X^2(a)$               | $\mu_{Y'}(b) = \mu_Y^2(b)$               | 2                   |
| $X' =$ „не дуже $X$ ” | $\mu_{X'}(a) = \mu_X^{1/2}(a)$           | $\mu_{Y'}(b) = \mu_Y^{1/2}(b)$           | 4                   |
| $X' =$ „не $X$ ”      | $\mu_{X'}(a) = 1 - \mu_X(a)$             | $\mu_{Y'}(b) = 1$                        | 6                   |

#### 4.2.2. Застосування теорії нечітких множин до побудови пристроїв керування

Під час конструювання системи керування технологічним процесом важливою задачею є побудова його моделі, що вимагає знання фізики цього процесу та врахування різноманітних обмежень. Застосування теорії нечітких множин до побудови таких систем не вимагає побудови моделей, а потребує лише опису за допомогою правил вигляду „ЯКЩО – ТО” процедури керування процесом. На рис. 4.23 наведено типову схему нечіткого пристрою керування. Розглянемо функції кожного з елементів пристрою керування.

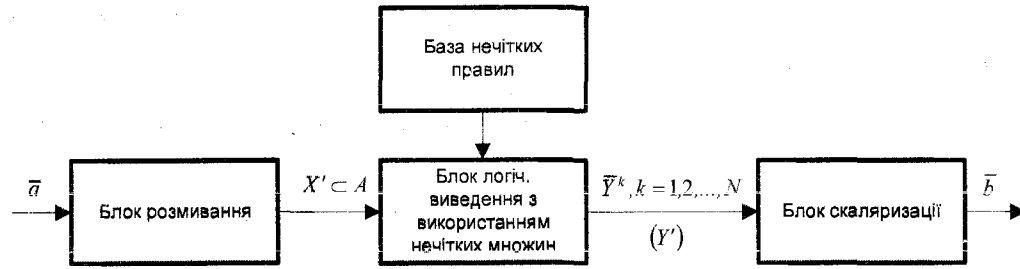


Рис. 4.23

Розглянемо  $N$  нечітких правил  $R^{(k)}$  ( $k=1, 2, \dots, N$ ) вигляду:

$R^{(k)} : \text{ЯКЩО } (a_1 \text{ це } X_1^k \text{ AND } a_2 \text{ це } X_2^k \text{ AND } \dots \text{ AND } a_n \text{ це } X_n^k) \text{ ТО } (b \text{ це } Y^k),$

де  $X_i^k, (X_i^k \subset A_i)$  ( $i=1, 2, \dots, n$ ),  $Y^k (Y^k \subset B)$  – нечіткі множини,

$a = (a_1, a_2, \dots, a_n)^T$  – вхідні змінні,  $(a_1, a_2, \dots, a_n) \in A = A_1 \times A_2 \times \dots \times A_n$ ,  
 $b$  – вихідна змінна ( $b \in B$ ).

Припускаємо, що всі правила пов'язані між собою логічною зв'язкою „або”.

Змінні  $a = (a_1, a_2, \dots, a_n)^T$  та  $b$  можуть бути лінгвістичними або звичайними змінними. Кожне правило  $R^{(k)}$  розглядають як нечітку імплікацію  $X^k \circledast Y^k$  ( $k=1, 2, \dots, N$ ), або нечітке відношення  $R^{(k)} \text{ MAI } B$  з функцією належності  $\mu_{R^{(k)}}(a, b) = \mu_{X^k \rightarrow Y^k}(a, b)$ . Множину нечітких правил називають базою правил або лінгвістичною моделлю.

Система керування, яку ми розглядатимемо, виконує перетворення нечітких множин. Для цього конкретне значення сигналу  $\bar{a} = (\bar{a}_1, \bar{a}_2, \dots, \bar{a}_n)^T$  на вході блоку нечіткого логічного виведення піддають розмиванню, що перетворює цей сигнал на нечітку множину  $X'$  MA. Тут і далі у цьому підрозділі  $\bar{a}$  використано для позначення вектора-стовпця з компонентами  $\bar{a}_1, \bar{a}_2, \dots, \bar{a}_n$ ; це позначення не має відношення до логічного заперечення.

Розмивання вхідного сигналу можна виконати двома способами. Перший спосіб розмивання вхідного сигналу полягає в застосуванні дельта-функції, яка перетворює

його в нечітку множину  $X'$  із функцією належності  $\mu_{X'}(a) = \delta(a - \bar{a}) = \begin{cases} 1, & \text{якщо } a = \bar{a} \\ 0, & \text{якщо } a \neq \bar{a} \end{cases}$ .

Другий спосіб розмивання вхідного сигналу та побудови нечіткої множини  $X'$

полягає в побудові функції належності за формулою  $\mu_{X'}(a) = \exp\left(-\frac{(a - \bar{a})^T (a - \bar{a})}{\delta^2}\right)$ ,

де  $\delta > 0$ . У цій формулі запис  $(a - \bar{a})^T (a - \bar{a})$  означає скалярний квадрат вектора  $(a - \bar{a})$ .

Нехай на вхід блоку нечіткого логічного виведення подано нечітку множину  $X'$  MA, а на виході блоку нечіткого логічного виведення отримано  $N$  нечітких множин  $\bar{Y}^k \subset B$  за правилами:

**факт:**  $a = (a_1, a_2, \dots, a_n)^T \text{ це } X'$ ,  $X' = X'_1 \times X'_2 \times \dots \times X'_n$ ;

**правила:**  $R^{(k)} : X^k \circledast Y^k$ ,  $X^k = X_1^k \times X_2^k \times \dots \times X_n^k$ ,  $k=1, 2, \dots, N$ ;

**ВИСНОВОК:**  $b \text{ це } \bar{Y}^k, k=1, 2, \dots, N$ .

Звернемо увагу на те, як отримано нечітку множину  $\bar{Y}^k$ . Кожне правило  $R^{(k)}$  є нечіткою імплікацією. Нечітку імплікацію визначено як нечітке відношення з відповідною функцією належності (див. підрозділ 4.2.1). Саму ж множину  $\bar{Y}^k$  визначають як композицію нечіткої множини  $X'$  та правила  $R^{(k)}$  за формулою (див. підрозділ 4.1.4)

$$\bar{Y}^k = (X^k \rightarrow Y^k) \circ X' \quad (k=1, 2, \dots, N).$$

Функцію належності нечіткої множини  $\bar{Y}^k$  обчислюють за формулою

$$\mu_{\bar{Y}^k}(b) = \sup_{a \in A} \left( \mu_{X'}(a)^T \mu_{X^k \rightarrow Y^k}(a, b) \right), \quad (4.1)$$

вигляд якої залежить від визначення  $T$ -норми, нечіткої імплікації  $R^{(k)}$  та способу обчислення декартового добутку нечітких множин.

Зазначимо, що в разі розмивання вхідного сигналу за допомогою дельта-функції, функція належності нечіткої множини  $\bar{Y}^k$  набуває вигляду  $\mu_{\bar{Y}^k}(b) = \mu_{X^k \rightarrow Y^k}(\bar{a}, b)$ .

Розглянемо приклади, у яких побудовано функції належності вихідного сигналу для різних способів визначення  $T$ -норми, нечіткої імплікації  $R^{(k)}$  та декартового добутку нечітких множин. Зазначимо, що у всіх прикладах  $a = (a_1, a_2, \dots, a_n)^T$ .

**Приклад 4.22.** Візьмемо  $n = 2$  та введемо:

- ♦  $T$ -норму аргументів  $x$  та  $y$  за правилом  $\min(x, y)$ ;
- ♦ нечітку імплікацію за правилом  $\mu_{X \rightarrow Y}(a, b) = \min(\mu_X(a), \mu_Y(b))$ ;
- ♦ декартів добуток нечітких множин за правилом  $\mu_{X \times Y}(a, b) = \min(\mu_X(a), \mu_Y(b))$ .

Функція належності нечіткого вихідного сигналу з блоку нечіткого логічного виведення набуде вигляду

$$\begin{aligned} \mu_{\bar{Y}^k}(b) &= \sup_{a \in A} \left( \min(\mu_{X'}(a), \mu_{X^k \rightarrow Y^k}(a, b)) \right) = \sup_{a \in A} \left( \min(\mu_{X'}(a), \min(\mu_{X^k}(a), \mu_{Y^k}(b))) \right) = \\ &= \sup_{(a_1, a_2) \in A} \left( \min(\mu_{X'_1}(a_1), \mu_{X'_2}(a_2), \mu_{X_1^k}(a_1), \mu_{X_2^k}(a_2), \mu_{Y^k}(b)) \right). \end{aligned}$$

Останню рівність отримано з урахуванням того, що

$$\mu_{X^k}(a) = \mu_{X_1^k \times X_2^k}(a_1, a_2) = \min(\mu_{X_1^k}(a_1), \mu_{X_2^k}(a_2)),$$

$$\mu_{X'}(a) = \mu_{X'_1 \times X'_2}(a_1, a_2) = \min(\mu_{X'_1}(a_1), \mu_{X'_2}(a_2)).$$

**Приклад 4.23.** Візьмемо  $n = 2$  та введемо:

- ♦  $T$ -норму аргументів  $x$  та  $y$  за правилом добутку  $xy$ ;
- ♦ нечітку імплікацію за правилом добутку  $\mu_{X \rightarrow Y}(a, b) = \mu_X(a) \mu_Y(b)$ ;
- ♦ декартів добуток нечітких множин за правилом  $\mu_{X \times Y}(a, b) = \mu_X(a) \mu_Y(b)$ .

Функція належності нечіткого вихідного сигналу з блоку нечіткого логічного виведення набуде вигляду

$$\begin{aligned}\mu_{\bar{Y}^k}(b) &= \sup_{a \in A} (\mu_{X'}(a) \cdot \mu_{X^k \rightarrow Y^k}(a, b)) = \sup_{a \in A} (\mu_{X'}(a) \cdot \mu_{X^k}(a) \cdot \mu_{Y^k}(b)) = \\ &= \sup_{(a_1, a_2) \in A} (\mu_{X_1^k}(a_1) \cdot \mu_{X_2^k}(a_2) \cdot \mu_{X_1^k}(a_1) \cdot \mu_{X_2^k}(a_2) \cdot \mu_{Y^k}(b)).\end{aligned}$$

**Приклад 4.24.** Візьмемо  $n = 2$  та виберемо:

- ◆  $T$ -норму аргументів  $x$  та  $y$  за правилом  $\min(x, y)$ ;
- ◆ нечітку імплікацію за правилом добутку  $\mu_{X \rightarrow Y}(a, b) = \mu_X(a) \mu_Y(b)$ ;
- ◆ декартів добуток нечітких множин за правилом  $\mu_{X \times Y}(a, b) = \mu_X(a) \mu_Y(b)$ .

Функція належності нечіткого вихідного сигналу із блоку нечіткого логічного виведення набуде вигляду

$$\begin{aligned}\mu_{\bar{Y}^k}(b) &= \sup_{a \in A} (\min(\mu_{X'}(a), \mu_{X^k \rightarrow Y^k}(a, b))) = \sup_{a \in A} (\min(\mu_{X'}(a), \mu_{X^k}(a) \cdot \mu_{Y^k}(b))) = \\ &= \sup_{(a_1, a_2) \in A} (\min(\mu_{X_1^k}(a_1) \cdot \mu_{X_2^k}(a_2), \mu_{X_1^k}(a_1) \cdot \mu_{X_2^k}(a_2) \cdot \mu_{Y^k}(b))).\end{aligned}$$

На виході блоку нечіткого логічного виведення отримують або  $N$  нечітких множин

$\bar{Y}^k \subset B$ , або одну нечітку множину  $Y'$  за формулою  $Y' = \left( \bigcup_{k=1}^N R^{(k)} \right) \circ X'$ . За означеннями суми нечітких множин і композиції нечіткої множини та нечіткого відношення одержимо

$$\mu_{Y'}(b) = \sup_{a \in A} \left[ \mu_{X'}(a) * \max_{1 \leq k \leq N} \mu_{R^{(k)}}(a, b) \right].$$

Якщо правила нечіткої імплікації визначено через  $T$ -норму типу  $\min$  або добуток, то функцію належності нечіткої множини  $Y'$  одержимо [34] у вигляді  $\mu_{Y'}(b) = \max_{1 \leq k \leq N} \mu_{\bar{Y}^k}(b)$ , причому функцію належності  $\mu_{\bar{Y}^k}(b)$  задано виразом (4.1).

**Приклад 4.25.** Нехай на вхід нечіткого пристрою керування подано сигнал  $\bar{a} = (\bar{a}_1, \bar{a}_2)$ , а правила блоку логічного виведення визначено так:

$R^{(1)}$ : ЯКЩО  $(a_1$  це  $X_1^1$  AND  $a_2$  це  $X_2^1$ ) ТО  $(b$  це  $Y^1$ );

$R^{(2)}$ : ЯКЩО  $(a_1$  це  $X_1^2$  AND  $a_2$  це  $X_2^2$ ) ТО  $(b$  це  $Y^2$ ).

Розмиванням вхідного сигналу з допомогою дельта-функції отримаємо нечіткі множини  $X_1'$  та  $X_2'$  з функціями належності  $\mu_{X_1'}(a_1) = \delta(a_1 - \bar{a}_1)$  та  $\mu_{X_2'}(a_2) = \delta(a_2 - \bar{a}_2)$ , відповідно.

Побудуємо функції належності нечітких множин  $\bar{Y}^k$  ( $k = 1, \dots, N$ ) на виході блоку логічного виведення. Для цього скористаємося формулою

$$\mu_{\bar{Y}^k}(b) = \sup_{a_1, a_2} (\min(\mu_{X_1' \times X_2'}(a_1, a_2), \mu_{R^{(k)}}(a_1, a_2, b))).$$

Виберемо:

- 1)  $T$ -норму аргументів  $x$  та  $y$  за правилом  $\min(x, y)$ ;

2) нечітку імплікацію за формулою  $\mu_{X_1^k \times X_2^k \rightarrow Y^k}(\bar{a}_1, \bar{a}_2, b) = \min(\mu_{X_1^k}(\bar{a}_1), \mu_{X_2^k}(\bar{a}_2), \mu_{Y^k}(b))$ ;

3) декартів добуток нечітких множин за правилами:

$$\mu_{X_1^k \times X_2^k}(\bar{a}_1, \bar{a}_2) = \min(\mu_{X_1^k}(\bar{a}_1), \mu_{X_2^k}(\bar{a}_2)),$$

$$\mu_{X_1^k \times X_2^k}(a_1, a_2) = \min(\mu_{X_1^k}(a_1), \mu_{X_2^k}(a_2)) = \min(\delta(a_1 - \bar{a}_1), \delta(a_2 - \bar{a}_2)).$$

Тоді з 1) та 3) дістанемо

$$\mu_{\bar{Y}^k}(b) = \sup_{a_1, a_2} (\min(\delta(a_1 - \bar{a}_1), \delta(a_2 - \bar{a}_2), \mu_{R^{(k)}}(a_1, a_2, b))) = \mu_{R^{(k)}}(\bar{a}_1, \bar{a}_2, b),$$

і, нарешті, скориставшись 2), матимемо

$$\mu_{Y'}(b) = \min(\min(\mu_{X_1^k}(\bar{a}_1), \mu_{X_2^k}(\bar{a}_2)), \mu_{Y^k}(b)) = \min(\mu_{X_1^k}(\bar{a}_1), \mu_{X_2^k}(\bar{a}_2), \mu_{Y^k}(b)).$$

Якщо на виході блоку логічного виведення треба отримати одну множину  $Y'$ , то

$$\mu_{Y'}(b) = \max_{k=1,2} (\min(\mu_{X_1^k}(\bar{a}_1), \mu_{X_2^k}(\bar{a}_2), \mu_{Y^k}(b))).$$

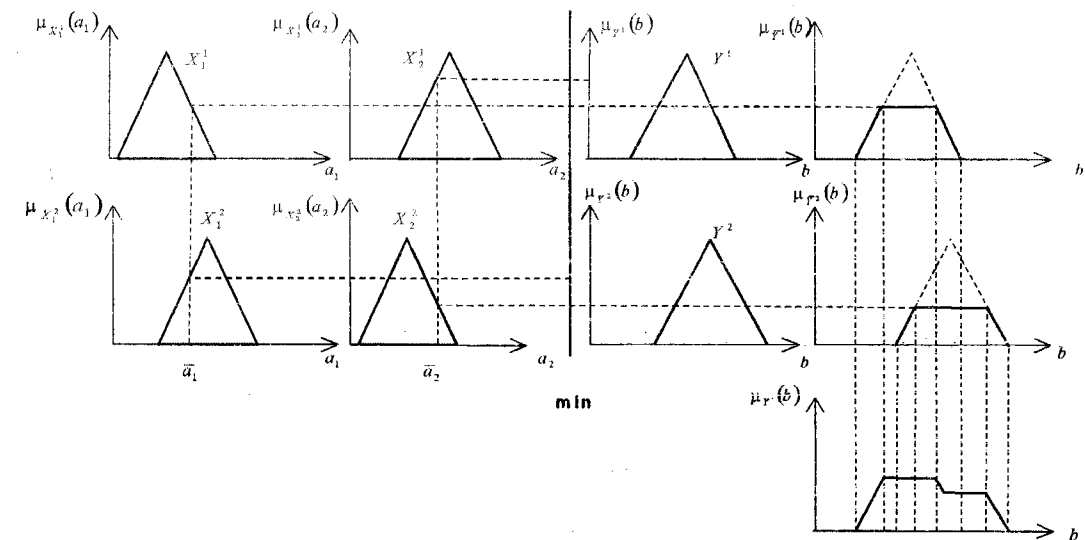


Рис. 4.24

На рис. 4.24 наведено графічну інтерпретацію нечіткого виведення, виконаного в цьому прикладі.

Отже, на виході блоку логічного виведення отримано  $N$  нечітких множин  $Y^k$  із функціями належності  $\mu_{Y^k}(b)$  ( $k = 1, 2, \dots, N$ ) або нечітку множину  $Y'$  із функцією належності  $\mu_{Y'}(b)$ . Необхідно перетворити ці нечіткі множини в одну скалярну величину  $\bar{b} \in B$ , яка є вихідним сигналом із нечіткого пристрою керування. Таке перетворення називають *скаляризацией* та виконують у блоці *скаляризації*.

Якщо на виході блоку логічного виведення отримано  $N$  нечітких множин  $\bar{Y}^k$ , то значення  $\bar{b} \in B$  можна обчислити методами середніх центрів або методом суми центрів.



**Метод середніх центрів.** Значення  $\bar{b}$  знаходять за формулою

$$\bar{b} = \frac{\sum_{k=1}^N \mu_{Y^k}(\bar{b}^k) \bar{b}^k}{\sum_{k=1}^N \mu_{Y^k}(\bar{b}^k)},$$

де  $\bar{b}^k$  – точка, у якій функція  $\mu_{Y^k}(b)$  набуває максимальне значення  $\mu_{Y^k}(\bar{b}^k) = \max_b \mu_{Y^k}(b)$ . Точку  $\bar{b}^k$  називають *центром* нечіткої множини  $Y^k$ . Рис. 4.25 ілюструє метод середніх центрів для  $N = 2$ . Зазначимо, що значення  $\bar{b}$  не залежить від форми та носія функцій належності  $\mu_{Y^k}(b)$ .

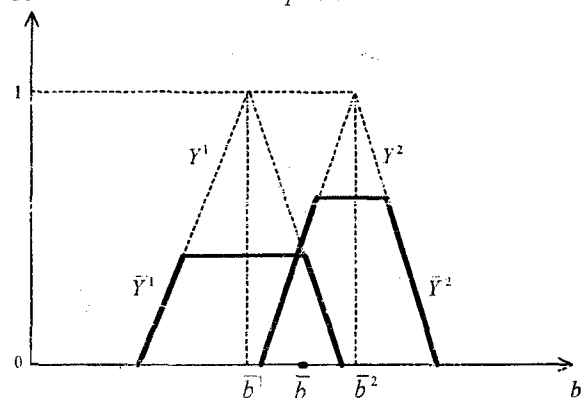


Рис. 4.25

**Метод суми центрів.** Значення  $\bar{b}$  обчислюють за формулою

$$\bar{b} = \frac{\int_B b \sum_{k=1}^N \mu_{Y^k}(b) db}{\int_B \sum_{k=1}^N \mu_{Y^k}(b) db}.$$

Якщо на виході блоку логічного виведення є лише одна нечітка множина  $Y$ , то значення  $\bar{b}$  можна знайти методом центру ваги або методом максимуму функції належності.

**Метод центру ваги.** Значення  $\bar{b}$  обчислюють як центр ваги фігури, обмеженої функцією належності  $\mu_{Y'}(b)$ . Обчислення виконують за формулою

$$\bar{b} = \frac{\int_B b \mu_{Y'}(b) db}{\int_B \mu_{Y'}(b) db} = \frac{\int_B b \max_k \mu_{Y^k}(b) db}{\int_B \max_k \mu_{Y^k}(b) db}$$

за умови, що інтеграли в цій формулі існують. Рис. 4.26 ілюструє спосіб обчислення  $\bar{b}$  методом центру ваги.

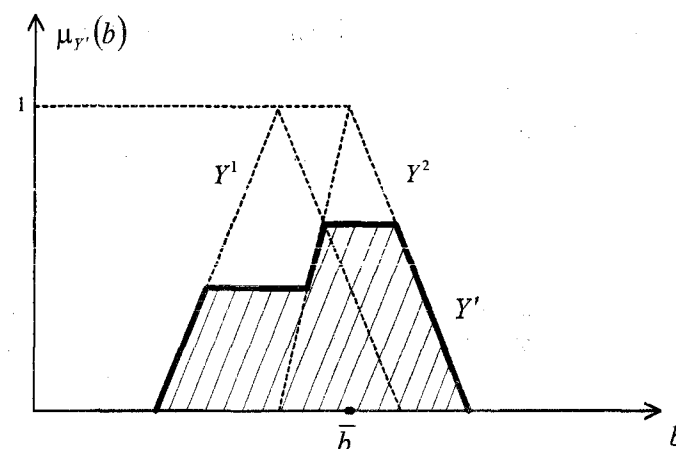


Рис. 4.26

**Метод максимуму функції належності.** Значення  $\bar{b}$  обчислюють за формулою  $\mu_{Y'}(\bar{b}) = \sup_{b \in B} \mu_{Y'}(b)$ . Результат застосування цього методу не залежить від форми

функції належності. Рис. 4.27 ілюструє знаходження  $\bar{b}$  методом максимуму функції належності.

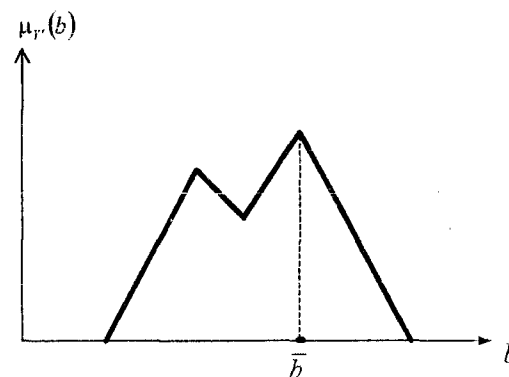


Рис. 4.27

#### 4.2.3. Приклад застосування нечітких множин до побудови пристрою керування

Застосування нечітких множин обіймає велику гаму задач, у яких проектується пристрій керування – від простого обладнання домашнього вжитку (пральні машини, холодильники, пылесоси тощо) до більш складних систем, таких, як вентиляція тунелів метрополітену. Як приклад розглянемо побудову системи керування кондиціонером.

На рис. 4.28 наведено схему системи керування, яка на основі значень температури та вологості визначає величину сигналу керування, який задає швидкість обертання вентилятора кондиціонера. Задача проектування пристрою нечіткого керування кондиціонером полягає в знаходженні швидкості  $\bar{b}$  обертання вентилятора кондиціонера

для заданих значень вхідних сигналів  $\bar{a} = (\bar{a}_1, \bar{a}_2)$ , де  $\bar{a}_1$  – температура,  $\bar{a}_2$  – вологість. Змінна  $\bar{b}$  – скалярна величина.

На схемі системи керування (рис. 12.12) позначено:

$\bar{a}_1$  – значення температури, яку подають на пристрій нечіткого керування з датчика температури;

$\bar{a}_2$  – значення вологості, яку подають на пристрій нечіткого керування з датчика вологості;

$\bar{b}$  – керуючий сигнал, який задає швидкість обертання вентилятора та поступає із пристрою нечіткого керування на кондиціонер.

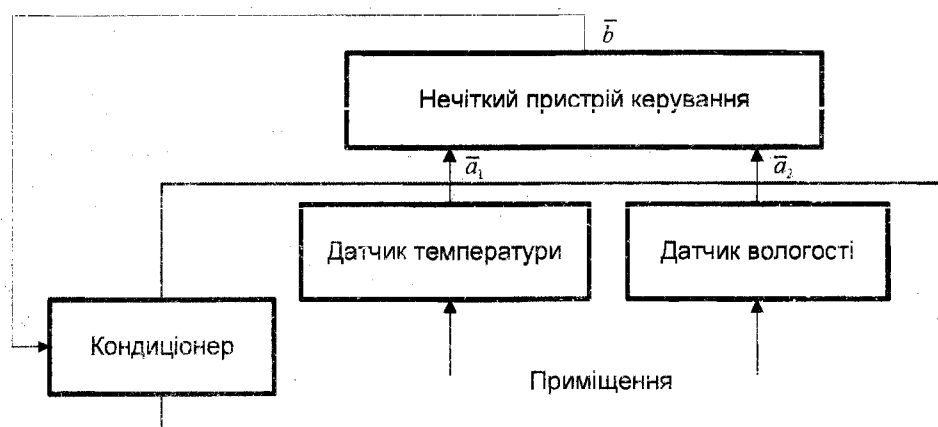


Рис. 4.28

#### Алгоритм побудови нечіткого пристрою керування

Крок 1. Сформулювати нечіткі правила функціонування нечіткого пристрою керування.

Крок 2. Визначити лінгвістичні змінні, простори змінних, нечіткі множини та їхні функції належності.

Крок 3. Визначити:

- ◆ нечітку імплікацію для кожного з нечітких правил;
- ◆ нечіткий декартів добуток нечітких множин, яким належать вхідні сигнали;
- ◆ T-норму.

Крок 4. Виконати розмивання вхідного сигналу  $\bar{a} = (\bar{a}_1, \bar{a}_2)^T$ .

Крок 5. Для заданих значень вхідних сигналів та введених правил обчислення функцій належності нечіткої імплікації, декартового добутку та композиції побудувати нечіткі множини, яким належать вихідні сигнали.

Крок 6. Побудувати нечіткий вихідний сигнал.

Крок 7. Скаляризацією нечіткого вихідного сигналу отримати шукане значення швидкості обертання валу вентилятора.

**Приклад 4.26.** Спроекуємо пристрій нечіткого керування швидкістю обертання вентилятора кондиціонера за наведеним алгоритмом.

*Крок 1.* Сформулюємо правила функціонування нечіткого пристрою керування:

„ЯКЩО температура висока ТА вологість висока, ТО швидкість обертання вентилятора в кондиціонері велика”;

„ЯКЩО температура нормальна ТА вологість нормальна, ТО швидкість обертання вентилятора в кондиціонері нормальна”.

*Крок 2.* Визначимо лінгвістичні змінні та простори значень цих змінних.

◆  $a_1$  = „температура” із простором значень  $A_1 = [15, 30]$ ;

◆  $a_2$  = „вологість” із простором значень  $A_2 = [10, 100]$ ;

◆  $b$  = „швидкість обертання” із простором значень  $B = [500, 1500]$ .

Уведемо нечіткі множини, яким належать введені лінгвістичні змінні:

$X_1^1$  = „висока температура”,  $X_2^2$  = „нормальна вологість”,

$X_2^1$  = „висока вологість”,  $Y^1$  = „велика швидкість обертання”,

$X_1^2$  = „нормальна температура”,  $Y^2$  = „нормальна швидкість обертання”,

Побудуємо функції належності кожної з нечітких множин.

1. Функція належності нечіткої множини  $X_1^1$  = „висока температура”, має тип  $\gamma$  з параметрами  $a = 20$ ,  $b = 25$ . На рис. 4.29 наведено графік функції належності нечіткої множини  $X_1^1$ .

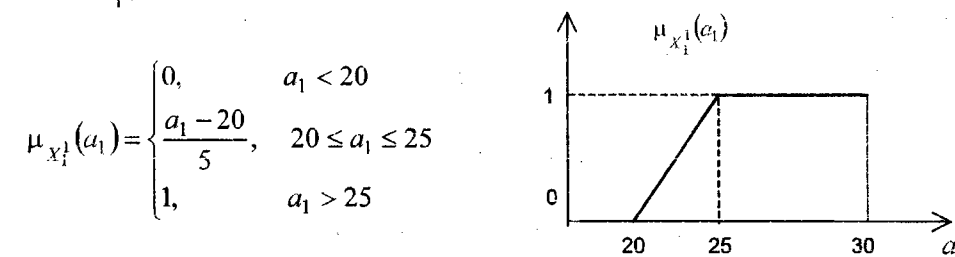


Рис. 4.29

2. Функція належності нечіткої множини  $X_2^1$  = „висока вологість” є функцією типу  $g$  з параметрами  $a = 50$ ,  $b = 75$ . На рис. 4.30 наведено графік функції належності нечіткої множини  $X_2^1$ .

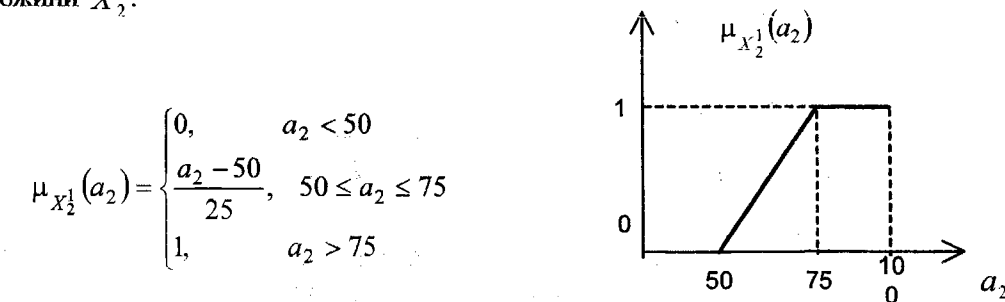


Рис. 4.30

3. Функція належності множини  $Y^1$  „велика швидкість обертання” є функцією типу  $\gamma$  з параметрами  $a = 1000$ ,  $b = 1250$ . На рис. 4.31 наведено графік функції належності нечіткої множини  $Y^1$ .

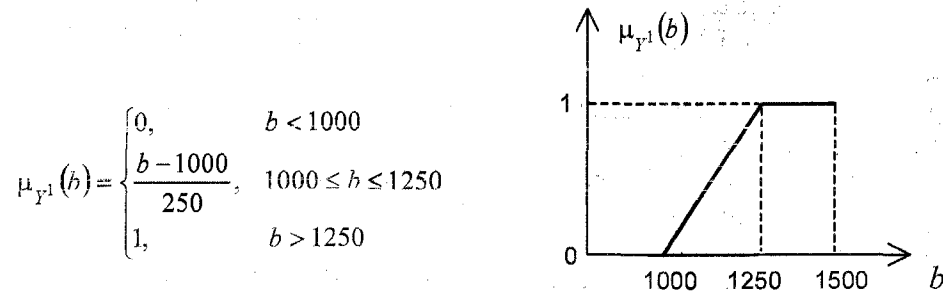


Рис. 4.31

4. Функція належності множини  $X_1^2$  „нормальна температура” є функцією типу  $t$  з параметрами  $a = 15$ ,  $b = 20$ ,  $c = 25$ . На рис. 4.32 наведено графік функції належності нечіткої множини  $X_1^2$ .

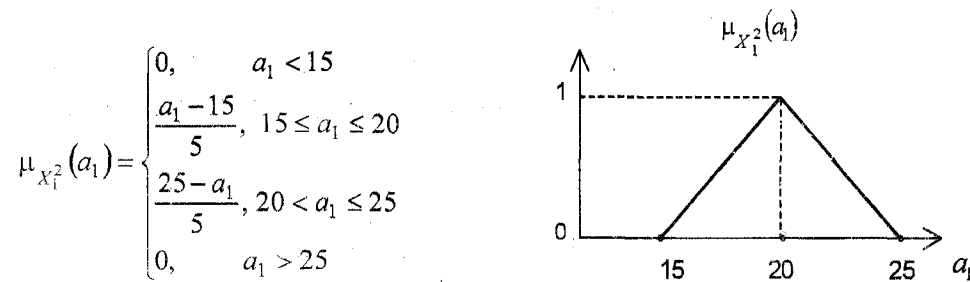


Рис. 4.32

5. Функція належності множини  $X_2^2$  „нормальна вологість” є функцією типу  $t$  з параметрами  $a = 25$ ,  $b = 50$ ,  $c = 75$ . На рис. 4.33 наведено графік функції належності нечіткої множини  $X_2^2$ .

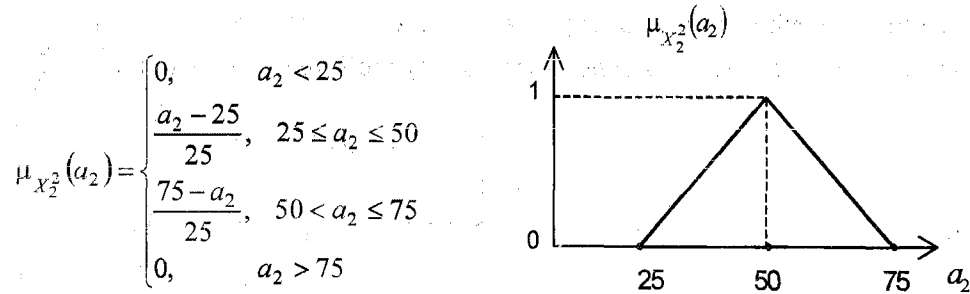


Рис. 4.33

6. Функція належності множини  $Y^2$  „нормальна швидкість обертання” є функцією типу  $t$  з параметрами  $a = 750$ ,  $b = 1000$ ,  $c = 1250$ . На рис. 4.34 наведено графік функції належності нечіткої множини  $Y^2$ .

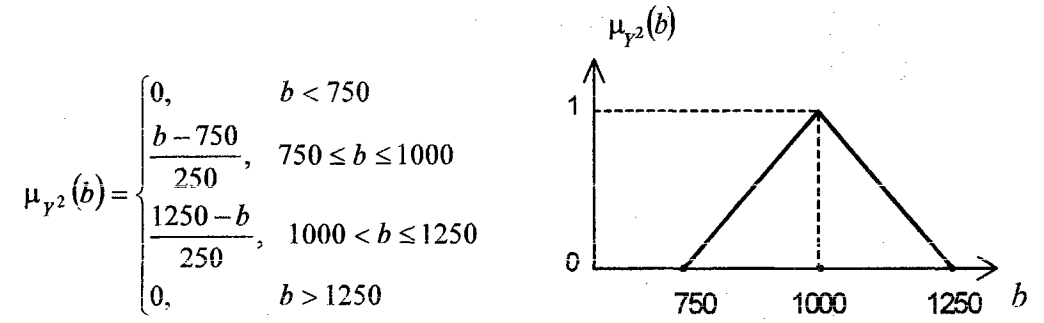


Рис. 4.34

Крок 3. Виберемо формули для обчислення:

◆ нечіткої імплікації за правилом Мамдані (мінімуму):

$$\mu_{X^1 \rightarrow Y^1}(a, b) = \min(\mu_{X^1}(a), \mu_{Y^1}(b)), \quad \mu_{X^2 \rightarrow Y^2}(a, b) = \min(\mu_{X^2}(a), \mu_{Y^2}(b)).$$

◆ нечіткого декартового добутку нечітких множин  $X^1 = X_1^1 \times X_2^1$ ,  $X^2 = X_1^2 \times X_2^2$ , яким належать вхідні сигнали, за правилом мінімуму:

$$\mu_{X^1}(a) = \mu_{X_1^1 \times X_2^1}(a_1, a_2) = \min(\mu_{X_1^1}(a_1), \mu_{X_2^1}(a_2)),$$

$$\mu_{X^2}(a) = \mu_{X_1^2 \times X_2^2}(a_1, a_2) = \min(\mu_{X_1^2}(a_1), \mu_{X_2^2}(a_2))$$

◆ композиції нечіткої множини та нечіткої імплікації з  $T$ -нормою, визначеною за правилом мінімуму:

$$\mu_{X^1}(a)^T \mu_{X^1 \rightarrow Y^1}(a, b) = \min(\mu_{X^1}(a), \mu_{X^1 \rightarrow Y^1}(a, b)),$$

$$\mu_{X^1}(a)^T \mu_{X^2 \rightarrow Y^2}(a, b) = \min(\mu_{X^1}(a), \mu_{X^2 \rightarrow Y^2}(a, b))$$

Крок 4. Виконаємо операцію розмивання вхідного сигналу  $\bar{a} = (\bar{a}_1, \bar{a}_2)^T$ . Для цього кожний із сигналів  $\bar{a}_1$  та  $\bar{a}_2$  подамо відповідними функціями належності  $\mu_{X_1^1}(a_1) = \delta(a_1 - \bar{a}_1)$  та  $\mu_{X_2^1}(a_2) = \delta(a_2 - \bar{a}_2)$ , побудованими з допомогою дельта-функції.

Крок 5. Нечіткі множини  $\bar{Y}^1$  та  $\bar{Y}^2$  є композиціями нечіткої множини  $X^1$  та правил  $R^{(1)}$  та  $R^{(2)}$ :

$$\bar{Y}^1 = (X^1 \rightarrow Y^1) \circ X^1, \quad \bar{Y}^2 = (X^2 \rightarrow Y^2) \circ X^1.$$

Тут  $a = (a_1, a_2)^T$ ,  $(a_1, a_2) \in X^1$ ,  $X^1 = X_1^1 \times X_2^1$ ,  $X^k = X_1^k \times X_2^k$  ( $k=1, 2$ ),  $b \in Y^1$ .

Побудуємо композицію нечіткої множини та нечіткої імплікації та визначимо функцію належності нечітких множин  $Y^1$  та  $Y^2$  за правилом

$$\mu_{\bar{Y}^k}(b) = \sup_{a_1, a_2} \left( \mu_{X^1}(a)^T \mu_{X^k \rightarrow Y^k}(a, b) \right), \quad k=1, 2.$$

Як наведено в прикладі 4.25, остання формула набуде вигляду

$$\mu_{\bar{Y}^k}(b) = \mu_{X^k \rightarrow Y^k}(\bar{a}, b), \quad k=1,2.$$

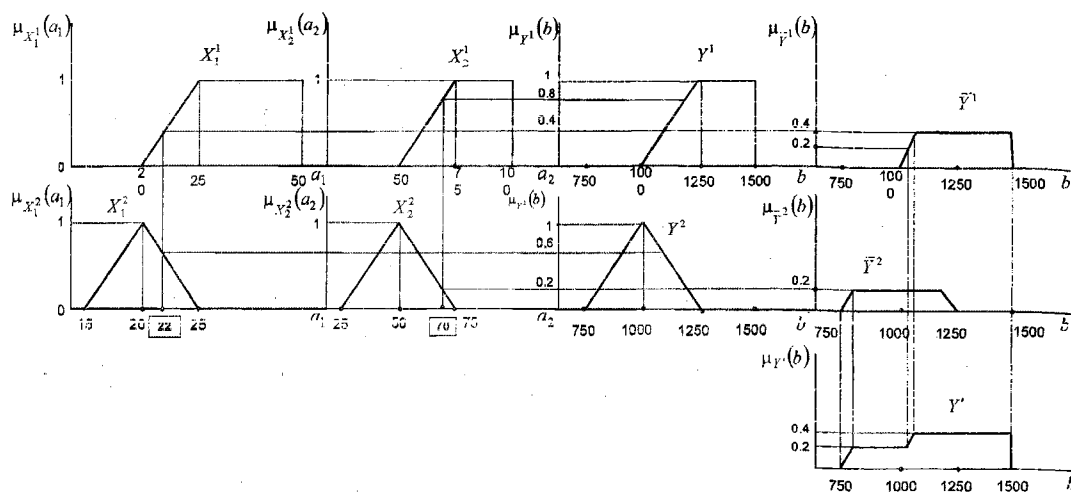


Рис. 4.35

Крок 6. Для побудови функції належності нечіткої множини  $Y'$  ( $b \in Y'$ ) скористаємось формулою  $\mu_{Y'}(b) = \max_{1 \leq k \leq N} \mu_{Y^k}(b)$ .

Крок 7. Виконаємо скаляризацію вихідної нечіткої множини  $Y'$ . Для цього скористаємось методом центру ваги

$$\bar{b} = \frac{\int_B \mu_{Y'}(b) db}{\int_B \mu_{Y'}(b) db} = \frac{\int_B \max_{k=1,2} \mu_{Y^k}(b) db}{\int_B \max_{k=1,2} \mu_{Y^k}(b) db}.$$

Обчислимо вихідний сигнал  $\bar{b}$  для температури  $\bar{a}_1 = 22^\circ$  та вологості  $\bar{a}_2 = 70\%$ . Для цих вхідних сигналів функції належності мають такі значення:

$$\mu_{X_1^1}(22) = 0.4; \mu_{X_2^1}(70) = 0.8; \mu_{X_1^2}(22) = 0.6; \mu_{X_2^2}(70) = 0.2;$$

$$\mu_{\bar{Y}^1}(b) = \min(\min(\mu_{X_1^1}(22), \mu_{X_2^1}(70)), \mu_{Y^1}(b)) = \min(\min(0.4, 0.8), \mu_{Y^1}(b)) = \min(0.4, \mu_{Y^1}(b));$$

$$\mu_{\bar{Y}^2}(b) = \min(\min(\mu_{X_1^2}(22), \mu_{X_2^2}(70)), \mu_{Y^2}(b)) = \min(\min(0.6, 0.2), \mu_{Y^2}(b)) = \min(0.2, \mu_{Y^2}(b)).$$

## Розділ 4. Міркування в умовах невизначеності

Функцію належності швидкості обертання валу вентилятора обчислюємо за формулою

$$\mu_{Y'}(b) = \max_{k=1,2} \mu_{\bar{Y}^k}(b) = \max_{k=1,2} (\min(0.4, \mu_{Y^1}(b)), \min(0.2, \mu_{Y^2}(b)))$$

Графічний розв'язок цієї задачі наведено на рис. 4.35.

Обчислимо скаляризоване значення швидкості обертання вентилятора за

формулою суми центрів  $\bar{b} = \frac{I_1 + I_2 + I_3 + I_4}{J_1 + J_2 + J_3 + J_4}$ , де

$$I_1 = \int_{750}^{800} x \cdot \frac{x-750}{250} dx = 3916.7, \quad I_2 = \int_{800}^{1050} x \cdot 0.2 dx = 46250,$$

$$I_3 = \int_{1050}^{1100} x \cdot \frac{x-1000}{250} dx = 16166.7, \quad I_4 = \int_{1100}^{1500} x \cdot 0.4 dx = 208000,$$

$$J_1 = \int_{750}^{800} \frac{x-750}{250} dx = 5, \quad J_2 = \int_{800}^{1050} 0.2 dx = 50,$$

$$J_3 = \int_{1050}^{1100} \frac{x-1000}{250} dx = 15, \quad J_4 = \int_{1100}^{1500} 0.4 dx = 160.$$

Границі інтегрування обчислені як значення аргументів відповідних функцій належності. Остаточного отримуємо швидкість обертання валу вентилятора

$$\bar{b} = \frac{I_1 + I_2 + I_3 + I_4}{J_1 + J_2 + J_3 + J_4} = \frac{3916.7 + 46250 + 16166.7 + 208000}{5 + 50 + 15 + 160} = 1192.7.$$

## 4.3. СТОХАСТИЧНИЙ ПІДХІД ДО ПОДАННЯ НЕВИЗНАЧЕНОСТІ

Під час розв'язування задач методами, які ґрунтуються на знаннях, часто доводиться проводити міркування з обмеженими знаннями та неповною інформацією. У цьому підрозділі спочатку викладено повний байєсівський аналіз, а після цього обмежена форма байєсівського виведення, відома як мережі довіри.

### 4.3.1. Байєсівські міркування

Байєсівські міркування ґрунтуються на теорії ймовірностей; їх інтенсивно використовують у деяких сучасних напрямках досліджень, наприклад, у розпізнаванні образів і класифікації. Теорія Байєса дає змогу обчислювати складні ймовірності на основі випадкової вибірки подій.

У теорії ймовірностей окремі ймовірності обчислюють або комбінаторними методами, або емпірично. Якщо відомо, що  $A$  та  $B$  незалежні, то ймовірність їхньої комбінації обчислюють за таким правилом:

$$\text{імовірність}(A \wedge B) = \text{імовірність}(A) \cdot \text{імовірність}(B).$$

*Апріорна ймовірність*, часто називана *безумовною ймовірністю* події, – це ймовірність, події при проведенні стохастичного експерименту. Апріорну ймовірність події позначають як  $P(\text{подія})$ .

*Апостеріорна ймовірність*, часто називана *умовною ймовірністю*, – це ймовірність появи події за умови виконання деякої іншої події. Апостеріорну ймовірність події позначають як  $P(\text{подія} | \text{інша подія})$ .

Апріорна ймовірність випадіння двійки чи трійки під час кидання гральної кістки – це сума двох можливостей, поділена на загальну кількість можливих альтернатив, тобто  $2/6$ . Апріорна ймовірність хвороби окремої людини дорівнює кількості людей із цією хворобою, поділеній на кількість людей, що перебувають під наглядом.

Апостеріорна ймовірність захворювання  $d$  у людини з симптомом  $s$  дорівнює

$$P(d | s) = \frac{|d \cap s|}{|s|},$$

тут прямі дужки позначають *кількість елементів множини*. Праву частину цього рівняння потрібно розуміти так: кількість людей, що мають одночасно й захворювання  $d$  і синдром  $s$ , поділено на кількість людей із синдромом  $s$ . Узагальнимо це рівняння

$$P(d | s) = \frac{P(d \cap s)}{P(s)}$$

і одержимо аналогічне співвідношення для  $P(s | d)$  і  $P(d \cap s)$ :

$$P(s | d) = \frac{P(d \cap s)}{P(d)},$$

$$P(d \cap s) = P(s | d) \cdot P(d). \quad (4.2)$$

Підстановка цього результату в співвідношення для  $P(d | s)$  дасть частковий варіант теореми Байєса для нашого прикладу (одне захворювання та один симптом):

$$P(d | s) = \frac{P(s | d) \cdot P(d)}{P(s)}.$$

Теорема Байєса дає змогу „переставити причину  $s$  й наслідок  $d$ ”: за відомим фактом події обчислити ймовірність того, що вона була зумовлена даною причиною. Події, які відображають дію „причин”, у даному випадку зазвичай називають *гіпотезами*, бо вони – *передбачувані* події, які викликали дану подію. Апріорна ймовірність гіпотези показує, наскільки ймовірна причина загалом, тоді як апостеріорна показує, наскільки ймовірною причина виявилась з урахуванням даних про подію. Важливою особливістю теореми Байєса є те, що значення в правій частині рівняння одержати легше, ніж значення в його лівій частині. Наприклад, унаслідок меншої популяції значно легше знайти кількість хворих на менінгіт, які страждають головними болями, ніж процент хворих менінгітом із загальної кількості страждаючих через

головний біль. Більш важливою особливістю теореми Байєса є те, що для простого випадку однієї хвороби та одного симптому в обчисленнях використовується мало чисел. Труднощі починаються, коли розглядають комплексні захворювання  $d_m$  із області захворювань  $D$  і комплексні симптоми  $s_n$  із множини можливих симптомів  $S$ . Під час розгляду кожної хвороби із  $D$  і кожного симптому із  $S$  окремо необхідно зібрати і об'єднати  $m \cdot n$  одиниць інформації. Точніше,  $m \cdot n$  апостеріорних ймовірностей плюс  $(m + n)$  апріорних ймовірностей.

На жаль, аналіз при цьому значно ускладнюється. Досі ми розглядали симптоми окремо. Насправді окремі симптоми зустрічаються рідко. Наприклад, коли лікар оглядає пацієнта, він має враховувати багато різних комбінацій симптомів. Цю ситуацію описує форма правила Байєса з багатьма симптомами:

$$P(d | s_1 \wedge s_2 \wedge \dots \wedge s_n) = \frac{P(d) \cdot P(s_1 \wedge s_2 \wedge \dots \wedge s_n | d)}{P(s_1 \wedge s_2 \wedge \dots \wedge s_n)},$$

або, в інших позначеннях

$$P(d | s_1, s_2, \dots, s_n) = \frac{P(d) \cdot P(s_1, s_2, \dots, s_n | d)}{P(s_1, s_2, \dots, s_n)} \quad (4.3)$$

Для обробки одного захворювання з одним симптомом треба лише  $m \cdot n$  одиниць інформації. Для кожної пари симптомів  $s_i$  та  $s_j$  і хвороби  $d$  необхідно знати як  $P(s_i \wedge s_j | d)$ , так і  $P(s_i \wedge s_j)$ . Якщо  $S$  містить  $n$  симптомів, то таких пар буде  $n \cdot (n - 1)$ , або приблизно  $n^2$ . Якщо ми захочемо використати правило Байєса, то доведеться обчислити близько  $(m \cdot n^2$  умовних ймовірностей) +  $(n^2$  ймовірностей симптомів) +  $(m$  ймовірностей хвороб), тобто зібрати  $m \cdot n^2 + n^2 + m$  одиниць інформації. У реальній медичній системі з 200 захворюваннями та 2000 симптомами це значення перевищує 800000000 [20].

Проте можна надіятись, що багато із цих пар незалежні, тобто  $P(s_i | s_j) = P(s_i)$ . Незалежність означає, що ймовірність  $s_i$  не залежить від  $s_j$ . У медицині більшість симптомів не пов'язані, наприклад, облісіння з кашлем. Але навіть якщо десять відсотків симптомів залежні, необхідно розглянути близько 80000000 одиниць інформації [20].

У багатьох задачах діагностики доводиться використовувати від'ємну інформацію, коли симптом у пацієнта відсутній (наприклад, низький тиск крові). В обох випадках необхідно щоб

$$P(\text{not } s) = 1 - P(s) \text{ та } P(\text{not } d | s) = 1 - P(d | s).$$

Зазначимо, що  $P(s | d)$  і  $P(d | s)$  – це не одне й те саме, у загальному випадку ці величини матимуть різні значення. Ці співвідношення та спроба уникнути циклічних міркувань є важливими при розробці байєсівських мереж довіри.

Розглянемо тепер один із найважливіших результатів теорії ймовірностей – теорему Байєса в загальній формі. Вона забезпечує спосіб обчислення ймовірності гіпотези  $H_i$ , що впливає із окремої події, якщо задані лише ймовірності цієї події, яка впливає з реальних причин (гіпотез):

$$P(H_i|E) = \frac{P(E|H_i) \cdot P(H_i)}{\sum_{k=1}^n P(E|H_k) \cdot P(H_k)},$$

де

$P(H_i|E)$  – імовірність події-гіпотези  $H_i$  за умови появи події  $E$ ;

$P(H_i)$  – імовірність події-гіпотези  $H_i$  загалом;

$P(E|H_i)$  – імовірність появи події  $E$  за умови виконання гіпотези  $H_i$ ;

$n$  – кількість можливих гіпотез.

**Приклад 4.27.** Припустимо, необхідно визначити ймовірність виявлення міді на основі проби ґрунту. Для цього потрібно знати наперед імовірність виявлення кожного із множини мінералів і ймовірність певного ґрунту в разі виявлення кожного окремого мінералу. Тоді можна використати теорему Байєса для визначення ймовірності наявності міді на основі проби ґрунту. Такий підхід використовують у практиці геологічної розвідки [20].

Для використання теореми Байєса потрібно обчислити всі взаємозв'язки між подією й гіпотезами, тобто  $P(E|H_k)$ . Загалом і особливо в таких галузях, як медицина, припущення незалежності не може бути апріорі обґрунтованим.

Проблема, яка утруднює використання складних байєсівських систем, полягає в перевизначенні таблиці ймовірностей при виявленні нових взаємозв'язків між гіпотезами і подіями. У таких активних дослідницьких напрямках, як медицина, нові відкриття здійснюються неперервно. У багатьох галузях такий великий збір даних і верифікація неможливі, а якщо й можливі, то коштують дуже дорого. Там, де такий збір даних можливий, байєсівський підхід забезпечує добре обумовлене управління невизначеністю. Більшість областей експертних систем не задовольняють ці вимоги й повинні спиратись на евристичний підхід. Більше того, через складність задач навіть дуже потужні комп'ютери не можуть використовувати байєсівські методи для успішного розв'язування задач у реальному часі.

Розглянемо приклад, який показує, як повний байєсівський підхід можна використати для організації взаємозв'язку гіпотеза-подія.

**Приклад 4.28.** Розглянемо можливі пояснення факту зниження швидкості автомобіля на трасі через велике скупчення транспорту. Можливо, це пов'язано з ремонтом дороги (подія  $C$ )? Була аварія (подія  $A$ )? Надалі вважатимемо, що події  $A$  та  $C$  несумісні. Якщо через декілька хвилин ми проїдемо повз групи дорожніх робітників в оранжевих жилетах, які працюють посеред дороги, то в цей момент найліпшим поясненням зниження швидкості є ремонт дороги. У такому разі альтернативна гіпотеза аварії відкидається. Якби ми побачили попереду „мигалку” автомобіля автоінспекції або швидкої допомоги, то найліпшим поясненням цієї ситуації стала б транспортна аварія, що дало б змогу відкинути версію про ремонт дороги. Зазначимо, що відмова від гіпотези зовсім не означає, що вона взагалі неможлива. Просто в контексті нової події вона має набагато меншу ймовірність.

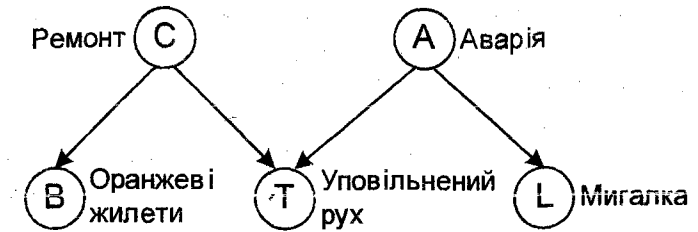


Рис. 4.36

|                  | C     | T     | P   |                  |
|------------------|-------|-------|-----|------------------|
| C – істина = 0,5 | true  | true  | 0,3 | T – істина = 0,4 |
|                  | true  | false | 0,2 |                  |
|                  | false | true  | 0,1 |                  |
|                  | false | false | 0,4 |                  |

Рис. 4.37

На рис. 4.36 наведено байєсівську інтерпретацію цієї ситуації. Ремонт дороги пов'язаний з оранжевими жилетами й уповільненим рухом. Аналогічно, аварія пов'язана з мигалкою й уповільненим рухом. Далі, відповідно до рис. 4.36 будемо сумісний імовірнісний розподіл для відношення ремонт дороги і сповільнений рух. Для спрощення припустимо, що ці змінні можуть набувати значення *true* або *false*. Імовірнісні розподіли для цього випадку наведено на рис. 4.37. Зазначимо, що коли ремонт має значення *false*, уповільнений рух неможливий, а при значенні *true* – можливий.

Зазначимо, що в цьому прикладі ймовірність ремонту дороги  $C = \text{true}$  дорівнює 0.5, а ймовірність сповільненого руху  $T = \text{true}$  дорівнює 0.4. Тепер розглянемо ймовірність ремонту дороги за умови уповільненого руху –  $P(C|T)$ , або  $P(C = \text{true} | T = \text{true})$ . За формулою (4.2) можемо записати

$$P(C = \text{true} \wedge T = \text{true}) = P(C = \text{true} | T = \text{true}) \cdot P(T = \text{true}),$$

звідки

$$P(C|T) = \frac{P(C = \text{true}, T = \text{true})}{P(T = \text{true})} = \frac{P(C = \text{true}, T = \text{true})}{P(C = \text{true}, T = \text{true}) + P(C = \text{false}, T = \text{true})} = \frac{0.3}{0.3 + 0.1} = 0.75.$$



Отже, імовірність *ремонт дороги* за умови *сповільненого руху* зростає з 0.5 до 0.75. Аналогічно ця ймовірність зростає ще більше за умови наявності *оранжевих жилетів*, що дає змогу відкинути гіпотезу *аварії*.

Окрім необхідності знати значення будь-якого з параметрів у кожному стані, потрібно враховувати питання складності. Розглянемо обчислення сумісної ймовірності для всіх параметрів, наведених на рис. 4.36 (використовуючи топологічне сортування змінних):

$$P(C, A, B, T, L) = P(C) \cdot P(A|C) \cdot P(B|C, A) \cdot P(T|C, A, B) \cdot P(L|C, A, B, T). \quad (4.4)$$

Це загальна декомпозиція ймовірностей, яка завжди правильна. Вартість генерування таблиці сумісної ймовірності експоненціально зростає зі збільшенням кількості параметрів. У нашому прикладі потрібна таблиця розміром  $2^5=32$ . Ми розглядаємо ігравкову проблему з п'ятьма параметрами. Для більш цікавої ситуації, скажімо, із тридцятьма або більше параметрами потрібна таблиця сумісної ймовірності з більйоном елементів. Байєсівські мережі довіри дають змогу вирішити проблеми подання та складності обчислень.

### 4.3.2. Наївний байєсівський класифікатор

Це простий імовірнісний класифікатор, який ґрунтується на теоремі Байєса з „наївними” припущеннями про незалежність. У багатьох практичних застосуваннях для оцінки параметрів для наївних байєсівських моделей використовують метод максимальної правдоподібності. Перевагою наївного байєсівського класифікатора є мала кількість даних для навчання, необхідних для знаходження параметрів, що потрібні для класифікації.

Імовірнісна модель класифікатора – це умовна модель

$$P(C|F_1, F_2, \dots, F_n),$$

де  $C$  – змінна класу з невеликою кількістю значень (*класів*), залежна від декількох змінних  $F_1, F_2, \dots, F_n$ . Суть проблеми в тому, що коли кількість властивостей  $n$  дуже велика або коли властивість може приймати велику кількість значень, тоді будувати таку модель на імовірнісних таблицях стає неможливим. Тому ми переформулюємо модель, щоб спростити обчислення.

Використовуючи правило Байєса (4.3), запишемо

$$P(C|F_1, F_2, \dots, F_n) = \frac{P(C) \cdot P(F_1, F_2, \dots, F_n|C)}{P(F_1, F_2, \dots, F_n)}.$$

На практиці важливий лише чисельник цього дробу, бо знаменник не залежить від  $C$ , а значення властивостей  $F_i$  задані, отож знаменник – константа. Чисельник дорівнює сумісній імовірнісній моделі

$$P(C, F_1, F_2, \dots, F_n),$$

яку можна переписати, як це наведено нижче, багатократно використавши означення умовної ймовірності:

$$P(C, F_1, F_2, \dots, F_n) = P(C) \cdot P(F_1, F_2, \dots, F_n|C) = P(C) \cdot P(F_1|C) \cdot P(F_2, \dots, F_n|C, F_1) =$$

$$= P(C) \cdot P(F_1|C) \cdot P(F_2|C, F_1) \cdot P(F_3, \dots, F_n|C, F_1, F_2) = \\ = P(C) \cdot P(F_1|C) \cdot P(F_2|C, F_1) \cdot P(F_3|C, F_1, F_2) \cdot P(F_4, \dots, F_n|C, F_1, F_2, F_3)$$

і т. д. Тепер використаємо „наївні” припущення для умовної ймовірності: уважатимемо, що кожна властивість  $F_i$  умовно незалежна від будь-якої властивості  $F_j$  при  $j \neq i$ .

Це означає, що  $P(F_i|C, F_j) = P(F_i|C)$ . Отож, сумісну модель можна записати так:

$$P(C, F_1, F_2, \dots, F_n) = P(C) \cdot P(F_1|C) \cdot P(F_2|C) \cdot \dots \cdot P(F_n|C) = P(C) \prod_{i=1}^n P(F_i|C)$$

Це означає, що за припущення про незалежність властивостей умовний розподіл для класової змінної  $C$  можна виразити як

$$P(C|F_1, \dots, F_n) = \frac{1}{Z} P(C) \prod_{i=1}^n P(F_i|C),$$

де  $Z$  залежить лише від  $F_1, \dots, F_n$ , тобто константа, якщо значення властивостей відомі.

Усі параметри моделі можна апроксимувати відносними частотами з набору даних навчання. Це оцінки максимальної правдоподібності ймовірностей. Не дискретні властивості спочатку потрібно дискретизувати.

Коли якийсь клас і значення властивості ніколи не виявляються разом у наборі для навчання, тоді оцінка, яка ґрунтується на ймовірностях, дорівнюватиме нулю. Це проблема, бо при множенні нульова оцінка призведе до втрати інформації про інші ймовірності. Тому варто вводити невеликі поправки в усі оцінки ймовірностей так, щоб жодна ймовірність не була строго рівною нулю.

Наївний байєсівський класифікатор об'єднує модель із правилом розв'язування. Одне загальне правило має вибрати найімовірнішу гіпотезу; воно відоме як *апостеріорне правило прийняття рішення* (MAP, Maximum A Posteriori). Відповідний класифікатор – це функція *classify*, яку означають так:

$$\text{classify}(f_1, \dots, f_n) = \arg \max_c P(C=c) \prod P(F_i=f_i|C=c)$$

**Приклад 4.29 (фільтрація спаму).** Розглянемо приклад застосування наївного байєсівського класифікатора до задачі класифікації документів за їхнім змістом, а саме, до класифікації електронних листів на два класи – спам ( $S$ ) і не-спам ( $\bar{S}$ ). Уважатимемо, що документи вибрані з декількох класів документів, котрі можна подати множиною слів з (незалежною) ймовірністю того, що  $i$ -те слово  $w_i$  даного документа зустрічається в документі класу  $C$ :

$$P(w_i|C). \quad (4.5)$$

Для цієї задачі припустимо, що ймовірність появи слова в документі незалежна від довжини документа. Тоді ймовірність для даного документа  $D$  і класу  $C$  становить

$$P(D|C) = \prod_i P(w_i|C).$$

Ми шукаємо відповідь на питання: „Яка ймовірність того, що даний документ  $D$  належить класу  $C$ ?” Іншими словами, чому дорівнює  $P(C|D)$ ? За правилом Байєса запишемо

$$P(C|D) = \frac{P(C) \cdot P(D|C)}{P(D)}. \quad (4.6)$$

Ми маємо лише два класи:  $S$  і  $\bar{S}$  (тут це спам і не-спам). Використовуючи (4.5), можемо записати

$$P(D|S) = \prod_i P(w_i|S); \quad P(D|\bar{S}) = \prod_i P(w_i|\bar{S}).$$

Використовуючи (4.6), запишемо

$$P(S|D) = \frac{P(S)}{P(D)} \prod_i P(w_i|S); \quad P(\bar{S}|D) = \frac{P(\bar{S})}{P(D)} \prod_i P(w_i|\bar{S}).$$

звідки

$$\frac{P(S|D)}{P(\bar{S}|D)} = \frac{P(S)}{P(\bar{S})} \prod_i \frac{P(w_i|S)}{P(w_i|\bar{S})}.$$

Узявши від обох частин логарифм, одержимо

$$\ln \frac{P(S|D)}{P(\bar{S}|D)} = \ln \frac{P(S)}{P(\bar{S})} + \sum_i \ln \frac{P(w_i|S)}{P(w_i|\bar{S})}.$$

Нарешті, документ можна класифікувати так. Це спам, якщо  $\ln \frac{P(S|D)}{P(\bar{S}|D)} > 0$ , інакше це не спам.

### 4.3.3. Байєсівські мережі довіри

Наївний байєсівський класифікатор часто добре працює. Але він ґрунтується на дуже серйозному припущенні, а саме на умовній незалежності атрибутів за умови даного цільового значення.

Теорія Байєса забезпечує математичне підґрунтя для міркувань в умовах невизначеності, але складність, яка виникає під час її застосувань до реальних предметних областей, може виявитись недопустимою. На щастя, ми можемо зменшити цю складність, зосередивши пошук на меншій множині найбільш адекватних подій і спостережень. Підхід, називаний *байєсівськими мережами довіри* (*Bayesian belief network*), пропонує обчислювальну модель міркування з найкращою

інтерпретацією множини даних у контексті очікуваних причинних зв'язків у предметній області.

Байєсівська мережа довіри (або байєсівська мережа) – це імовірнісна модель, яка являє собою множину змінних і їхніх імовірнісних залежностей. Наприклад, її можна використати для обчислення ймовірності того, на що хворий пацієнт за наявності або відсутності сукупності симптомів, ґрунтуючись на даних про залежність між симптомами й хворобами.

Формально, *байєсівська мережа* – це орієнтований ациклічний граф, вершини якого відповідають змінним, а дуги – умовним залежностям між змінними.

Байєсівські мережі довіри послаблюють багато обмежень повної байєсівської моделі й показують, як дані із предметної області (або навіть відсутні дані) дають змогу розгалужувати й концентрувати міркування. Набутий досвід свідчить, що модульність предметної області часто дає змогу послабити багато обмежень залежності/незалежності, які вимагаються для правила Байєса. У переважній більшості ситуацій не потрібно будувати велику таблицю сумісної ймовірності, котра містить імовірності всіх можливих комбінацій подій і спостережень. Людина-експерт вибирає локальні явища, котрі напевно пов'язані одне з одним, і одержує ймовірності, або міри впливу, які відображають лише ці кластери подій. Експерти припускають, що інші події або умовно незалежні, або їхні кореляції настільки малі, що ними можна знехтувати.

**Приклад 4.30.** Продовжимо розгляд задачі про причини зменшення швидкості руху на швидкісній трасі із прикладу 4.28. Цю задачу наведено на рис. 4.36. Якщо припустити, що параметри залежать тільки від імовірностей їхніх батьків, тобто що за наявності знань про батьків, вершини не залежать від інших попередників, то обчислення  $P(C, A, B, T, L)$  виконується так:

$$P(C, A, B, T, L) = P(C) \cdot P(A) \cdot P(B|C) \cdot P(T|C, A) \cdot P(L|A). \quad (4.7)$$

Щоб краще зрозуміти зроблене нами спрощення, розглянуте в останньому прикладі, розглянемо ймовірність  $P(B|C, A)$  із співвідношення (4.4). У співвідношенні (4.7) ми послабили її до  $P(B|C)$ . Це ґрунтується на припущенні, що аварія не впливає на ремонт дороги. Аналогічно, на „оранжеві жилети” не впливає сповільнений рух, але „ремонт” і „аварія” враховуються в  $P(T|C, A)$ , а не в  $P(T|C, A, B)$ . Нарешті,  $P(L|C, A, B, T)$  послабляється до  $P(L|A)$ . Імовірнісний розподіл для  $P(C, A, B, T, L)$  тепер має всього 20 параметрів, а не 32. При вивченні до більш реальної задачі, нехай з 30 змінними, де кожний стан має максимум двох батьків, у розподілі буде не більше 240 елементів. Якщо кожний стан має трьох батьків, у розподілі буде максимум 480 елементів – це значно менше ніж білйон, потрібний для повної байєсівської моделі.

Необхідно обґрунтувати залежність вершини мережі довіри від батьківських вершин. Зв'язки між вершинами мережі довіри – це умовні ймовірності причинного впливу. У міркуванні експерта, який використовує причинно-наслідкове виведення, неявно припускається, що ці впливи спрямовані, тобто реалізація якоїсь події викликає інші події в мережі. Окрім того, міркування причинного впливу не циклічні, отож, вплив

не може повернутись назад, щоб викликати себе. Із цієї причини байєсівські мережі довіри можна природно подати ациклічним орієнтованим графом, де логічно послідовні міркування відображаються як шляхи, які проходять через дуги причина-симптом.

У прикладі 4.28 ми маємо навіть більш стійку ситуацію – там немає навіть неорієнтованих циклів. Це зможливилося дуже просто обчислювати ймовірнісний розподіл у кожній вершині. Розподіли вершин, котрі не мають батьків, знаходять безпосередньо. Значення вершин-синів обчислюють на основі ймовірнісного розподілу кожного з батьків за допомогою відповідних обчислень за таблицями умовних ймовірностей. Це можливо тому, що ми не піклуємось про кореляцію між батьками будь-якої вершини (оскільки мережу подають як орієнтований ациклічний граф). Це забезпечує природне абдукційне відділення, при якому „аварія” зовсім не корелює з наявністю „оранжевих жилетів” (див. рис. 4.36).

**Зауваження.** Абдукція – це пізнавальна процедура прийняття гіпотез. Уперше явно виділена Ч. С. Пірсом, котрий розглядав абдукцію (абдукційне виведення) поряд з індукцією й дедукцією. Абдукція дає змогу вибрати серед незорої множини гіпотез найбільш істотні.

Далі розглядатимемо припущення, яке неявно використовується в міркуваннях багатьох експертів: наявність або відсутність даних про область може розділяти й концентрувати пошук пояснень. Це має важливі комплексні наслідки для простору пошуку. Наведемо декілька прикладів і концепцію *d*-відокремлення, яка підтримує ці інтуїтивні міркування.

**Приклад 4.31.** Розглянемо задачу діагностики наявності масла в автомобілі: припустимо, що старі поршневі кільця викликають надмірний розхід масла, що, у свою чергу, призводить до низького рівня масла. Цю ситуацію наведено на рис. 4.38а, де *A* – старі поршневі кільця, *V* – надмірний розхід масла і *B* – низький рівень масла. Нічого не знаючи про надмірний розхід масла, ми одержуємо причинне відношення між старими поршневими кільцями та низьким рівнем масла. Проте, якщо перевірка свідчить, що змінна *V*, яка характеризує надмірне споживання масла, має значення *true* чи *false*, то змінні, які описують старі поршневі кільця та низький рівень масла, не залежать одна від одної.

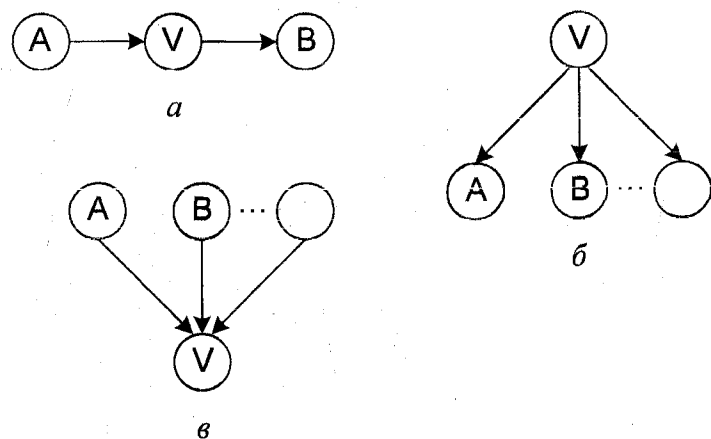


Рис. 4.38

**Приклад 4.32.** Старі поршневі кільця можуть викликати як сильний вихлоп, так і низький рівень масла. Цю ситуацію наведено на рис. 4.38б, де *V* – старі поршневі кільця, *A* – сильний вихлоп і *B* – низький рівень масла. Якщо невідомо, яке значення має змінна *V* – *true* чи *false*, – ми не знаємо чи корельовані змінні *A* (сильний вихлоп) і *B* (низький рівень масла). Наявність інформації про змінну *V* (старі поршневі кільця) означає, що ці змінні не корельовані.

**Приклад 4.33.** Якщо низький рівень масла викликаний або надмірним його розходом, або виток, то при наявності даних про низький рівень масла ці дві можливі причини корельовані. На рис. 4.38в змінна *A* означає надмірний розхід масла, *B* – витік і *V* – низький рівень масла. Якщо змінна *V* істинна, то витік масла пояснює його надмірний розхід. Якщо ж значення змінної *V* невідомо, то ці дві можливі причини незалежні. У кожному випадку інформація про низький рівень масла є ключовим елементом у процесі міркування.

Узагальнюючи ситуації, зображені на рис. 4.38, можна виділити такі зв'язки вершин. Послідовний зв'язок вершин (а), у якому вплив розповсюджується від *A* до *B* доти, доки значення *V* визначене. Розбіжний зв'язок (б), де інформація розповсюджується до спадкоємців *V* доти, доки значення *V* визначене. Збіжний зв'язок (в): якщо нічого не відомо про значення *V*, то його батьки незалежні, у протилежному випадку між батьками є кореляція.

Уточнимо інтуїтивні уявлення, розглянуті в прикладах 4.31 – 4.33, увівши *d*-відокремлення вершин у мережі довіри.

Дві вершини *A* і *B* в ациклічному орієнтованому графі називають *d*-відокремленими, якщо всі шляхи між ними заблоковані. Тут шлях – це довільна неперервна послідовність дуг у графі без урахування орієнтації, наприклад, на рис. 4.38б – шлях від *A* до *B*. Шлях називають заблокованим, якщо в ньому є внутрішня вершина *V*, яка має одну з наведених нижче властивостей:

- 1) зв'язок є послідовним або розбіжним, і значення *V* відоме;
- 2) зв'язок є збіжним, і ані *V*, ані жоден із спадкоємців *V* значення не одержав.

У протилежному випадку вершини називають *d*-зв'язаними.

*D*-відокремлені вершини незалежні.

Тут, по суті, міститься відповідь на питання, які вершини (події) у байєсівській мережі є залежними? Зрозуміло, що ті, котрі з'єднані дугою, але не лише... Свідчення – це твердження вигляду „подія у вершині *V* відбулася”. Розглянемо випадок послідовного зв'язку між вершинами мережі (рис. 4.38а). У цьому випадку *A* впливає на *V*, а *V*, у свою чергу, впливає на *B*, і вершини *A* і *B* виявляються зв'язаними. Проте, якщо у *V* поступило свідчення, то зв'язок між *A* і *B* порушується. Нехай маємо розбіжний зв'язок (рис. 4.38б). Інформація про одного з нащадків вершини *V* може вплинути на ймовірність іншого її нащадка. Це пов'язано з тим, що не тільки інформація про причину, яка відбулася, збільшує ймовірність наслідку, але й наслідок, який відбувся, збільшує ймовірність причини. Якщо спільний батько вже отримав значення, то зв'язок між *A* і *B* порушується. І, нарешті, розглянемо збіжний зв'язок (рис. 4.38в). Зв'язку між *A* і *B* немає: якщо подія *A* відбулася, то це вплине на ймовірність події *V*, але ймовірність *B* не повинна змінитись. Проте ситуація буде іншою, якщо свідчення *V* уже одержано: коли ми знаємо, що одна з причин відбулася, то це має зменшити ймовірність іншої причини, адже наслідок вже пояснено.

Зміст  $d$ -відокремлення полягає в тому, що за наявності певної інформації можна проігнорувати при обчисленні імовірнісних розподілів частину мережі довіри.

На рис. 4.39 наведено приклад байесівської мережі довіри, де імовірнісні залежності показані поруч із кожною вершиною. На цьому рисунку можна побачити додаткові приклади послідовних, розбіжних і збіжних відношень між вершинами, а також як  $d$ -відокремлення впливає на побудову шляхів.

Перед тим як завершити розгляд графів на рис. 4.38, покажемо, як припущення байесівської мережі довіри спрощують обчислення умовних імовірностей. За законом Байєса будь-який сумісний розподіл імовірностей може розглядатись як добуток умовних імовірностей. На рис. 4.38а умовну ймовірність для шляху від  $A$  до  $V$  і від  $V$  до  $B$  обчислюють так:

$$P(A, V, B) = P(A) \cdot P(V|A) \cdot P(B|A, V).$$

Використаємо припущення байесівської мережі довіри, що умовна ймовірність змінної при заданих значеннях імовірностей усіх її предків дорівнює умовній імовірності при заданих значеннях лише для батьків. Як результат у наведеній вище рівності  $P(B|A, V)$  заміниться на  $P(B|V)$ , бо  $V$  – батько  $B$ , а  $A$  – хоча і предок, але не батько. Сумісні ймовірності для трьох мереж, наведених на рис. 4.38, обчислюють так:

$$а) P(A, V, B) = P(A) \cdot P(V|A) \cdot P(B|V);$$

$$б) P(V, A, B) = P(V) \cdot P(A|V) \cdot P(B|V);$$

$$в) P(A, B, V) = P(A) \cdot P(B) \cdot P(V|A, B).$$

Як можна побачити із прикладу 4.28, у більших байесівських мережах довіри багато змінних умовних імовірностей можуть бути вилучені (рис. 4.36). Це робить мережі довіри значно простішими для реалізації ніж повний байесівський аналіз.

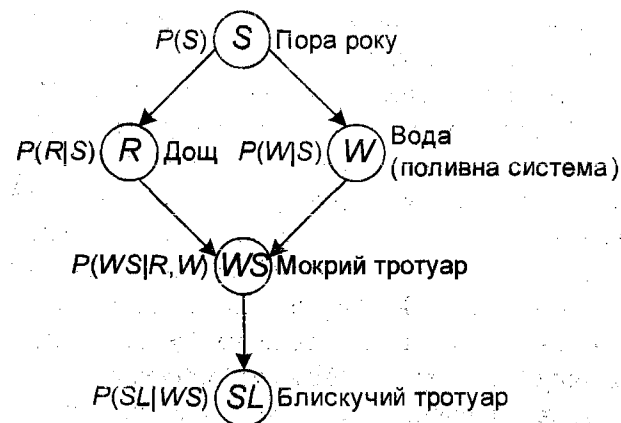


Рис. 4.39

**Приклад 4.34.** Розглянемо байесівську мережу, наведену на рис. 4.39. Змінна  $S$  (пора року) визначає ймовірність  $R$  (падає дощ), а також ймовірність  $S$  (вода поступає

з поливної системи). Змінна  $WS$  (мокрый тротуар) корелюватиме з дощем або водою з поливної системи. Нарешті, тротуар блищатиме (змінна  $SL$ ) залежно від того, мокрий він чи ні. На рисунку наведено імовірнісні відношення для цієї ситуації.

| $R$   | $W$   | $P$ |
|-------|-------|-----|
| true  | true  | x   |
| true  | false |     |
| false | true  |     |
| false | false |     |

Рис. 4.40

Як можна описати ймовірність мокрого тротуару  $P(WS)$ ? Цього не можна зробити так, як було описано вище, тобто  $P(W) = P(W|S) \cdot P(S)$  або  $P(R) = P(R|S) \cdot P(S)$ . Обидві причини  $WS$  взаємно незалежні, наприклад, якщо пора року – літо, то можна використовувати як  $P(W)$  так і  $P(R)$ . Отож, потрібно обчислити повні кореляції двох змінних, а також їхні кореляції з  $S$ . У нашому прикладі це можливо, але складність таких обчислень експоненціально залежить від кількості можливих причин  $WS$ .

Результати обчислень наведено на рис. 4.40. Тут обчислено один елемент  $x$  при істинних значеннях  $R$  та  $W$  ( $R = \text{true}$ ,  $W = \text{true}$ ); змінна  $S$  (пора року) набуває два значення: *hot* (тепло) та *cold* (холодно):

$$\begin{aligned}
 x &= P(R = \text{true}, W = \text{true}) = \\
 &= P(S = \text{hot}) \cdot P(R = \text{true} | S = \text{hot}) \cdot P(W = \text{true} | S = \text{hot}) + \\
 &+ P(S = \text{cold}) \cdot P(R = \text{true} | S = \text{cold}) \cdot P(W = \text{true} | S = \text{cold}).
 \end{aligned}$$

Аналогічно можна обчислити решту значень таблиці, наведеної на рис. 4.40. При цьому ми одержимо сумісну ймовірність дощу та води з поливної системи:  $P(WS) = P(WS|R, W) \cdot P(R, W)$ . Цю задачу можна розв'язати за прийнятеного обсягу обчислень, але проблема в тому, що цей обсяг експоненціально зростає зі збільшенням кількості батьків вершини. Повернемося до питання про  $d$ -відокремлення. Нагадаємо, що за наявності певної інформації  $d$ -відокремлення дає змогу пропустити при обчисленні імовірнісних розподілів частину мережі довіри.

1. Вершина  $SL$  є  $d$ -відокремленою від вершин  $R$ ,  $S$ ,  $W$ , якщо відоме значення  $WS$ .
  2.  $D$ -відокремлення симетричне, тобто  $S$  також  $d$ -відокремлена (і не являє собою пояснення  $SL$ ), коли значення  $WS$  відоме.
  3.  $R$  і  $W$  є залежними внаслідок  $S$ , але значення  $S$ ,  $R$  і  $W$  є  $d$ -відокремленими.
  4. Якщо значення  $WS$  невідоме, то  $R$  і  $W$   $d$ -відокремлені, якщо відоме – то ні.
  5. Для ланцюжка  $R \rightarrow WS \rightarrow SL$  коли відоме  $WS$ , то  $R$  і  $SL$  є  $d$ -відокремленими.
- Потрібно бути уважним, коли відома інформація про спадкоємців якоїсь вершини. Наприклад, якщо відомо  $SL$ , то  $R$  і  $W$  не є  $d$ -відокремленими, бо  $SL$  корелює з  $WS$ .

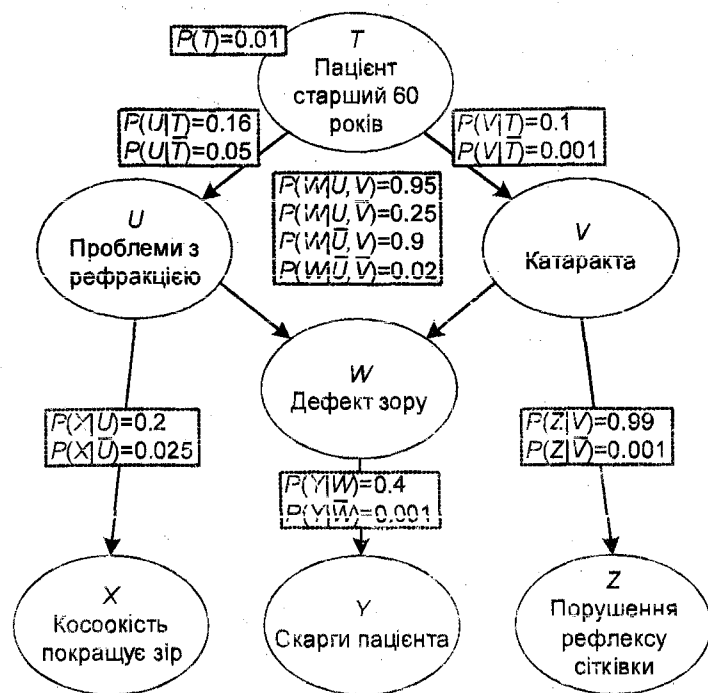


Рис. 4.41

Розгляд теми завершимо прикладом, узятим із [28].

**Приклад 4.35.** Розглянемо конкретний приклад того, як можна задати граф байєсівської мережі. На рис. 4.41 початкові вершини дуг – причини, а кінцеві вершини дуг – наслідки. Коренева вершина – першоджерело, а листки – симптоми (як правило, їх і спостерігають). Спочатку задано умовні ймовірності нападків за умови батьків. Це вже дає змогу визначити сумісну апіорну ймовірність будь-якої комбінації подій у мережі.

Суть міркувань у байєсівській мережі – пропagaція свідчень. Наприклад, на вхід поступають дані типу „Зір пацієнта покращився через косоокість” і „Пацієнт старше 60 років”, а задача – оцінити, як змінилась ймовірність інших вершин (наприклад, того що „У пацієнта порушення рефлексу сітківки”).

Обчислимо сумісні ймовірності ланцюжка  $\tilde{U}, \tilde{V}, \tilde{W}$ . Тут  $\tilde{U}$  означає  $U$  або  $\bar{U}$ .

$$P(U, V, W) = P(W|U, V) \sum_{\tilde{T}} P(U|\tilde{T}) P(V|\tilde{T}) P(\tilde{T}) = 0.000199025,$$

$$P(\bar{U}, V, \bar{W}) = P(\bar{W}|\bar{U}, V) \sum_{\tilde{T}} P(\bar{U}|\tilde{T}) P(V|\tilde{T}) P(\tilde{T}) = 0.000010475,$$

$$P(U, \bar{V}, W) = P(W|U, \bar{V}) \sum_{\tilde{T}} P(U|\tilde{T}) P(\bar{V}|\tilde{T}) P(\tilde{T}) = 0.012722625,$$

$$P(U, \bar{V}, \bar{W}) = P(\bar{W}|\bar{U}, \bar{V}) \sum_{\tilde{T}} P(U|\tilde{T}) P(\bar{V}|\tilde{T}) P(\tilde{T}) = 0.038167875,$$

$$P(\bar{U}, V, W) = P(W|\bar{U}, V) \sum_{\tilde{T}} P(\bar{U}|\tilde{T}) P(V|\tilde{T}) P(\tilde{T}) = 0.00160245,$$

$$P(\bar{U}, V, \bar{W}) = P(\bar{W}|\bar{U}, V) \sum_{\tilde{T}} P(\bar{U}|\tilde{T}) P(V|\tilde{T}) P(\tilde{T}) = 0.00017805,$$

$$P(\bar{U}, \bar{V}, W) = P(W|\bar{U}, \bar{V}) \sum_{\tilde{T}} P(\bar{U}|\tilde{T}) P(\bar{V}|\tilde{T}) P(\tilde{T}) = 0.01894239,$$

$$P(\bar{U}, \bar{V}, \bar{W}) = P(\bar{W}|\bar{U}, \bar{V}) \sum_{\tilde{T}} P(\bar{U}|\tilde{T}) P(\bar{V}|\tilde{T}) P(\tilde{T}) = 0.92817711.$$

Нехай поступило свідчення „Дефект зору”. Обчислимо апостеріорну ймовірність  $P(U|W)$ . Спочатку потрібно прирівняти до нуля несумісні зі свідченням події в таблиці сумісних ймовірностей:

$$P((U, V, W) \wedge W) = 0.000199025, \quad P((U, V, \bar{W}) \wedge W) = 0,$$

$$P((U, \bar{V}, W) \wedge W) = 0.012722625, \quad P((U, \bar{V}, \bar{W}) \wedge W) = 0,$$

$$P((\bar{U}, V, W) \wedge W) = 0.00160245, \quad P((\bar{U}, V, \bar{W}) \wedge W) = 0,$$

$$P((\bar{U}, \bar{V}, W) \wedge W) = 0.01894239, \quad P((\bar{U}, \bar{V}, \bar{W}) \wedge W) = 0.$$

І, нарешті, нормувавши результат, одержимо

$$P(U|W) = \frac{\sum_{\tilde{V}, \tilde{W}} P((U, \tilde{V}, \tilde{W}) \wedge W)}{P(W)} = 0.386107 \dots,$$

$$P(\bar{U}|W) = \frac{\sum_{\tilde{V}, \tilde{W}} P((\bar{U}, \tilde{V}, \tilde{W}) \wedge W)}{P(W)} = 0.61389288 \dots$$



## ЗАДАЧІ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

1. Побудувати функцію належності вихідного сигналу із блоку нечіткого логічного виведення для  $n = 2$  та визначених:

а)  $T$ -норми аргументів  $x$  та  $y$  за правилом  $\min(x, y)$ , функції належності нечіткої імплікації за формулою  $\mu_{x \rightarrow y}(a, b) = \min(\mu_x(a), \mu_y(b))$  та декартового добутку нечітких множин за формулою  $\mu_{x \times y}(a, b) = \min(\mu_x(a), \mu_y(b))$ ;

б)  $T$ -норми аргументів  $x$  та  $y$  за правилом добутку, функції належності нечіткої імплікації за правилом добутку  $\mu_{x \rightarrow y}(a, b) = \mu_x(a) \mu_y(b)$  та декартового добутку нечітких множин за формулою  $\mu_{x \times y}(a, b) = \mu_x(a) \mu_y(b)$ ;

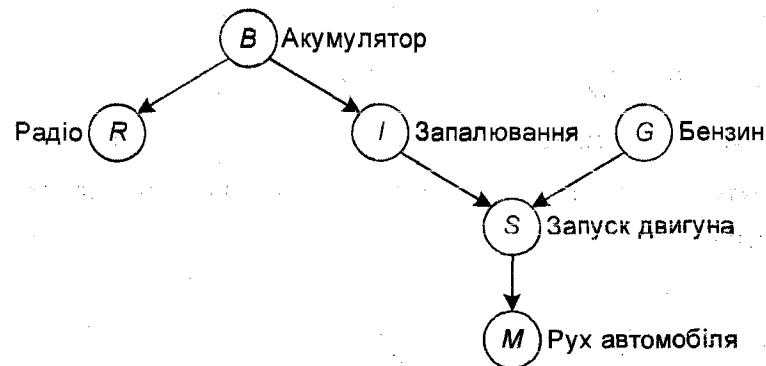
в)  $T$ -норми аргументів  $x$  та  $y$  за правилом  $\min(x, y)$ , функції належності нечіткої імплікації за правилом добутку  $\mu_{x \rightarrow y}(a, b) = \mu_x(a) \mu_y(b)$  та декартового добутку нечітких множин за правилом  $\mu_{x \times y}(a, b) = \mu_x(a) \mu_y(b)$ .

2. Поверніться до прикладу 4.26. Відтворіть не менше двох ітерацій роботи нечіткого пристрою керування швидкістю обертання вентилятора в кондиціонері.

3. Обчисліть решту значень таблиці, наведеної на рис. 4.40.

4. Розгляньте байєсівську мережу діагностики автомобіля, наведену на рисунку. Ця мережа описує певні характеристики електричної системи та двигуна автомобіля.

Кожна змінна – булева, а значення *true* засвідчує, що відповідна підсистема автомобіля перебуває в робочому стані.



Виконайте такі завдання:

- доповніть мережу булевими змінними *W* (Морозна погода) та *SM* (Стартер);
- наведіть прийнятні таблиці умовних ймовірностей для всіх вершин (окрім вершин, які не мають батьків);
- знайдіть, скільки незалежних значень міститься в сумісному розподілі ймовірностей для восьми булевих вершин, якщо припустити, що невідомі будь-які відношення умовної залежності, які б їх зв'язували;
- знайдіть кількість незалежних ймовірнісних значень, що містяться в таблицях нашої мережі.

5. Як у прикладі 4.35 обчисліть апостеріорну ймовірність  $P(V|W)$  у разі, коли поступило свідчення „Дефект зору” (див. рис. 4.41).



## КОМП'ЮТЕРНІ ПРОЕКТИ

### Скласти програми, які реалізують алгоритми

- Складіть програму, яка реалізує нечіткий пристрій керування з підрозділу 4.2.3.
- Скориставшись матеріалами [28], складіть програму алгоритму пропагації в байєсівських мережах довіри за умови, що заборонено не тільки орієнтовані, але й неорієнтовані цикли.
- Скориставшись матеріалами [28], складіть програму алгоритму виведення в байєсівських мережах довіри з неорієнтованими циклами.

## Алфавітний покажчик

Алгоритм 18, 119

- Девіса–Патнема 68
  - зведення формули до випередженої нормальної форми 59
  - інтерпретатора логічної програми 100
  - Куайна 62
  - модифікований 64
  - методу резолюцій 77
  - побудови нечіткого пристрою керування 250
  - пошуку вглиб 27
  - вшир 25
  - по першому найкращому збігу 32
  - уніфікації 85
  - C4.5 побудови дерева рішень 162
  - candidate elimination (CE) 142
  - CART побудови дерева рішень 167
  - Find-S 136
  - ID3 побудови дерева рішень 155
  - list-then-eliminate (LTE) 139
  - SLD-резолюцій 97
- Алфавіт числення предикатів 79
- Атом 45
- логіки першого ступеня 55

База правил 244

Байєсівська мережа довіри 263

Виведення логічне 60

– нечітке 241

Висловлювання 45

– складне 46

Граф орієнтований 21

– кореневий 21

– AND/OR 35, 37

– позначений 22

Дерево 21

– кореневе 22

– пошуку 102

– рішень 153, 155

– семантичне 50

– спростування 78

Диз'юнкт 63

– хорнівський 96

## Ентропія 160

## Закони логіки висловлювань 52

- першого ступеня 57, 58

## Змінна вільна 56, 80

- зв'язана 56, 80
- лінгвістична 240

## Інтерпретація 48

- формули логіки першого ступеня 56

## Інформаційний прибуток 161

## Карта Кохонена 207

- Квантор загальності 56
- існування 56

## Контрарна пара 50

## Літерал 50

- відрізний 89
- негативний 50
- позитивний 50

## Логіка висловлювань 45

- першого ступеня 55
- предикатів 55

## Логічні зв'язки 46

- операції 46

## Логічний наслідок 60, 73

## Метод зворотного поширення помилки 189

- Кохонена 211
- максимуму функції належності 249
- м'якої конкуренції (SCS) 212
- нейронного газу 212
- опорних векторів 197
- резолюцій 75, 89
- середніх центрів 248
- стохастичної релаксації 211
- суми центрів 248
- центру ваги 248
- *SLD*-резолюцій 95

## Множина нечітка 225

- формул числення предикатів 79

## Модель предметної області 16, 17

- алгоритму 119
- лінгвістична 244

## Навчання дедуктивне 118

- з коректуванням помилкою 180
- за прецедентами 117
- індуктивне 117
- машинне 117
- на основі зв'язків 168
- понять 132

## Нейромережа прямого поширення багатошарова 177

- — — одношарова 175

- рекурентна 178

## Нечітка імплікація 242

- множина 225

## Нечітке відношення 236

- число 234

## Нечіткий пристрій керування 243

## Нормальна форма випереджена 59

- диз'юнктивна 53
- клаузальна 87
- кон'юнктивна 53
- Сколема 86

## Підстановка 83

- основна 83

## Правило виведення 70, 73, 80, 81

- — нечітке 241

## Предикат 55

## Предметна область 16

## Приклад 83

- навчальний 134
- основний 83
- спільний 84

## Принцип прямої дедукції 61

- логічного програмування 98

## Простір гіпотез 134

- станів 22

## Резольвента 76, 89

## Розмивання 244

## Семантика 46

## Символи індивідні 55

- предикатні 55
- предметні 55
- функціональні 55

## Синтаксис 46



Система штучного інтелекту 16  
 Скаляризація 247  
 Стан початковий 17  
 – предметної області 17  
 – цільовий 17

Теорема 61, 71  
 Терм 55, 79  
 Топографічна карта 207  
 Трикутна норма 238, 240

Уніфікатор 84  
 – найзагальніший 84  
 Упередженість індукції 150

Формальна теорія 70, 79  
 Формула виконання 48  
 – довідна 71  
 – загальнозначуща 48  
 – невиконання 48  
 – правильно побудована 46  
 – — — логіки першого ступеня 56  
 Функція належності 225

Штучна нейронна мережа 169  
 Штучний інтелект 16  
 – нейрон 169

## Література

1. Берсегян А.А. Методы и модели анализа данных: OLAP и Data Mining / Берсегян А.А., Куприянов М.С., Степаненко В.В., Холод И.И. – СПб.: БХВ-Петербург, 2004. – 336 с.
2. Болотова Л.С. Системы искусственного интеллекта. Теоретические основы СИИ и формальные модели представления знаний: учеб. пособие / Л.С. Болотова, М.А. Комаров, А.А. Смольянинов. – М.: Моск. гос. ин-тут радиотехники, электроники и автоматики (технический университет), 1998. – 108 с.
3. Болотова Л.С. Системы искусственного интеллекта. Неформальные модели представления знаний в системах искусственного интеллекта: учеб. пособие / Л.С. Болотова, А.А. Смольянинов. – М.: Моск. гос. ин-тут радиотехники, электроники и автоматики (технический университет), 1998. – 108 с.
4. Братко И. Алгоритмы искусственного интеллекта на языке Пролог / Братко И. – М.: Издательский дом „Вильямс”, 2004. – 640 с.
5. Воронцов К.В. Машинное обучение: Курс лекций / Воронцов К.В. – Режим доступа: <http://www.machinelearning.ru>
6. Вапник В.Н. Восстановление зависимостей по эмпирическим данным / Вапник В.Н. – М.: Наука, 1979. – 448 с.
7. Гаврилова Т.А. Базы знаний интеллектуальных систем / Гаврилова Т.А., Хорошевский Б.Ф. – СПб: Питер, 2000. – 384 с.
8. Глибовець М.М. Штучний інтелект : Підручник для студ. вищих навч. закладів, що навч. за спец. „Комп’ютерні науки” та „Прикладна математика” / Глибовець М.М., Олецкий О.В. – К. : ВД „КМ Академія”, 2002. – 366 с.
9. Годич О.В. За-стосування штучної нейронної мережі типу SOM для розв’язування задачі діагностування / Годич О.В., Нікольський Ю.В., Щербина Ю.М. // Вісник Національного університету „Львівська політехніка”: [збірник наукових праць]. – 2002. – № 464: Серія: Інформаційні системи та мережі. – С. 31-43.
10. Годич О.В. Дослідження ефективності алгоритмів навчання мереж Кохонена / Годич О.В., Нікольський Ю.В., Щербина Ю.М., Пасічник В.В. // Управляющие системы и машины. – 2006. – №2. – С.63-80.
11. Годыч О.В. Динамическая сегментация изображений для учебного симулятора языка жестов / Годыч О.В., Гушин К.Н., Никольский Ю.В., Пасичник В.В., Щербина Ю.Н. // Управляющие системы и машины. – 2009. – №1. – С.79-85.
12. Грэй П. Логика, алгебра и базы данных / Питер Грэй; пер. Х.И. Килова, Г.Е. Минца. – Москва : Машиностроение, 1989. – 360 с.
13. Джарратано Д. Экспертные системы: принципы разработки и программирование / Джарратано Д., Райли Г. – М.: ООО „И.Д.Вильямс”, 2007. – 1152 с.
14. Джексон П. Введение в экспертные системы / Джексон П. – М.: Издательский дом „Вильямс”, 2001. – 624 с.
15. Журавлев Ю.И. Об алгебраическом подходе к решению задач распознавания или классификации / Журавлев Ю.И. // Проблемы кибернетики. – 1978. – Т.33. – С.5–68.

16. Загоруйко Н. Г. Прикладные методы анализа данных и знаний / Загоруйко Н. Г. – Новосибирск: ИМ СО РАН, 1999. – 270 с.
17. Ивахненко А.Г. Индуктивный метод самоорганизации моделей сложных систем / Ивахненко А.Г. –К.: Наукова думка, 1982. – 296 с.
18. Круглов В. В. Искусственные нейронные сети: Теория и практика / Круглов В. В., Борисов В.В. – М.: Горячая линия – Телеком, 2001. – 382 с.
19. Кузнецов О. П. Дискретная математика для инженера / О.П. Кузнецов, Г.М. Адельсон-Вельский. – Москва: Энергоатомиздат, 1988. – 480 с.
20. Люгер Д. Искусственный интеллект / Люгер Д. – СПб.: „Вильямс”, 2003.
21. Мандель И.Д. Кластерный анализ / И.Д.Мандель. – Москва: Финансы и статистика, 1988. –176 с.
22. Мендельсон Э. Введение в математическую логику / Мендельсон Э. – М.: Наука, 1976. – 320 с.
23. Мошков М.Ю. Деревья решений: Теория и приложения / Мошков М.Ю. – Нижний Новгород: Изд-во Нижегородского ун-та, 1994. – 175 с.
24. Нікольський Ю.В. Деревя прийняття рішень та їхнє застосування для прогнозування діагнозу у медицині / Нікольський Ю.В., Щербина Ю.М., Якимечко Р.Б. // Вісник Львівського університету. Серія: прикладна математика та інформатика. – Вип. 6. – 2003. – С.191-211.
25. Нікольський, Ю. В. Дискретна математика: підручник / Ю.В. Нікольський, В.В. Пасічник, Ю.М. Щербина. –Київ: Видавнича група ВНУ, 2007. –367 с.
26. Нікольський Ю.В. Застосування методів кластерного аналізу при побудові класифікуючих правил в задачі прийняття рішень // Вісник Національного університету “Львівська політехніка”, 2003, № 489: Серія: Інформаційні системи та мережі. – С.213-223.
27. Нікольський Ю.В. Генетичні алгоритми в екстремальних задачах. / Нікольський Ю.В., Щербина Ю.М. // Вісник Львівського університету. Серія прикладна математика та інформатика. – 2000. – Вип.2. – С.191-208.
28. Николенко С.И. Байесовские сети доверия / Сергей Николенко. – Режим доступа: <http://logic.pdmi.ras.ru/~infclub/files/07-bbn.pdf>
29. Новиков П.С. Элементы математической логики / П.С. Новиков. – М.: Наука, 1973.
30. Осовский С. Нейронные сети для обработки информации / Осовский С. – М.: Финансы и статистика, 2004. – 344 стр.
31. Пшеничный Б. Н. Численные методы в экстремальных задачах / Б.Н. Пшеничный, Ю.М. Данилин. – Москва: Наука, 1975. –319 с.
32. Рассел С. Искусственный интеллект: современный подход / Рассел С., Норвиг П. – М.: Издательский дом „Вильямс”, 2006. – 1408 с.
33. Розенблатт Ф. Принципы нейродинамики: Перцептроны и теория механизмов мозга. – М.: Мир, 1965. – 480 с.
34. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Рутковская Д., Пилиньский М., Рутковский Л. – М.: Горячая линия – Телеком, 2004.
35. Стерлинг Л. Искусство программирования на языке Пролог / Л. Стерлинг, Э. Шапиро; пер. С.Ф. Сопрунова, Л.В. Шабанова; под ред. Ю.Г. Дадаева. – Москва: Мир, 1990. – 333 с.

36. Столл Р. Множества. Логика. Аксиоматические теории / Столл Р. –М.: Просвещение, 1968. –235 с.
37. Тейз А. Логический подход к искусственному интеллекту: от классической логики к логическому программированию / Пер.с фр.; Тейз А. и др. – М. : Мир, 1990. – 432 с.
38. Уиллиамс У.Т. Методы иерархической классификации / Уиллиамс У.Т., Ланс Д.Н. // Статистические методы для ЭВМ / под. ред. Б.М. Малютов. – М.: Наука, 1986. – С. 269-301.
39. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика / Уоссермен Ф. – М.: Мир, 1992. – 240 с.
40. Форсайт Д.А. Компьютерное зрение: современный подход / Форсайт Д.А., Понс Ж. – М.: Издательский дом „Вильямс”, 2004. – 928 с.
41. Хайкин С. Нейронные сети: полный курс / Хайкин С. – М.: Издательский дом „Вильямс”, 2006. – 1104 с.
42. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем / Чень Ч., Ли Р. – М.: Наука, 1983.
43. Шлезингер М. И. Десять лекций по статистическому и структурному распознаванию: Монографія / Шлезингер М.И., Главач В. – К.: Наукова думка, 2004. – 545 с.
44. Bishop C. Neural Networks for Pattern Recognition / Bishop C. – Oxford: Oxford University Press, 1995.
45. Charniak E. Bayesian Networks without Tears / Charniak E. // AI magazine. – 1991. – v.12, № 4. – P. 50 – 63.
46. Czichosz P. Systemy uczace sie = [Системы, що навчаються] / Czichosz P. – Warszawa: Wydawnictwa Naukowo-Techniczne, 2000.
47. Duda R.O. Pattern Classification / Richard O. Duda, Peter E. Hart, David G. Stork – New York: John Wiley & Sons, Inc. 2000. – 654 p.
48. Dunhan M. H. Data Mining Introductory and Advanced Topics / Margareth H. Dunhan – Upper Saddle River: Prentice Hall, 2003. – 315 p.
49. Han J. Data Mining: Concepts and Techniques / Han J., Kamber M. – San Francisco: Morgan Kaufmann Publishers, 2001. – 550 p.
50. Hastie T. The elements of statistical learning: data mining, inference, and prediction / Hastie T., Tibshirani R., Friedman J. – New York : Springer, 2001. – 533 p.
51. Kaufman L. Finding Groups in Data: An Introduction to Cluster Analysis /Kaufman L., Rousseeuw P.J.– New York: John Wiley & Sons, 1990.
52. Mitchell T. Machine Learning / Mitchell T. – Columbus: McGraw-Hill Companies, Inc., 1997. – 414 p.
53. Nikolski I. SOM-Based Dynamic Image Segmentation for Sign Language Training Simulator / Kostiantyn Hushchyn, Oles Hodych, Iouri Nikolski, Volodymyr Pasichnyk, Yuri Shcherbyna // Information Systems: Modeling, Development, and Integration: Proceedings of the Third International United Information Systems Conference (UNISCON 2009) / Eds. Jianhua Yang, Athula Ginige, Heinrich C. Mayr, Ralf-D. Kutsche, 21-24 April, 2009, Sydney, Australia. – Berlin; Heidelberg: Springer-Verlag., 2009. – P. 29-40.

54. Nikolski I. Determining cluster boundaries within Self-Organizing Maps = [Визначення границь кластерів всередині карти, що само організується] / O. Hodych, I. Nikolski, V. Pasichnyk, Y. Shcherbuna // Вісник Нац. техн. ун-ту „Харківський політехнічний інститут”: 36. наук. пр. – Харків: НТУ „ХПІ”. – 2007. – №5: Тематичний випуск: Системний аналіз, управління та інформаційні технології. – С.97-109.
55. Pyle D. Data Preparation for Data Mining / Pyle D. – San Francisco: Morgan Kaufmann Publisher, 1999. – 535 p.
56. Quinlan J. C 4.5: Programs for Machine learning / Quinlan J. – San Mateo: Morgan Kaufmann Publishers 1993. – 302 p.
57. Rosen K. H. Discrete Mathematics and Its Applications / Kenneth H. Rosen – Columbus: McGraw-Hill, 2002. – 598 p.
58. Rabiner L.R. A Tutorial On Hidden Markov Models and Selected Applications in Speech Recognition / Rabiner L.R. // Proceeding of the IEEE. – 1989.-Vol.77. – No 2. – P. 257-286.

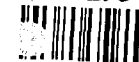
Навчальне видання

*Нікольський Юрій Володимирович*

*Пасічник Володимир Володимирович*

*Щербина Юрій Миколайович*

НБ ПНУС



783870

## Системи штучного інтелекту

НАВЧАЛЬНИЙ ПОСІБНИК

Керівник видавничого проекту *В. М. Піча*

Підписано до друку з оригінал-макета 30.07.2010 р.

Формат 70 × 100/16. Умовн. друк. арк. 17,43.

Гарнітура Таймс Нью Роман

ІПІ “Магнолія 2006”

а/с 431, м. Львів-53, 79060, Україна, тел./факс 240-54-84; 245-63-70

e-mail: magnol@lviv.farlep.net

Свідцтво про внесення суб’єкта видавничої справи до Державного реєстру видавців, виготівників і розповсюджувачів видавничої продукції: серія ДК № 2534 від 21.06.2006 року, видане Державним комітетом інформаційної політики, телебачення та радіомовлення України

Надруковано у друкарні видавництва “Магнолія 2006”  
м. Львів, вул. Зелена, 238 Д.