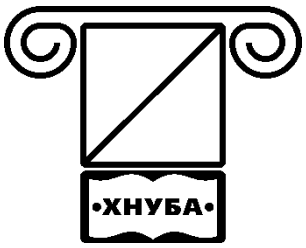


Міністерство освіти і науки України

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА ТА АРХІТЕКТУРИ

Г.В. Солодовник

МЕТОДИ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Навчальний посібник

Харків 2021

Міністерство освіти і науки України

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА ТА АРХІТЕКТУРИ

Г.В. Солодовник

МЕТОДИ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Рекомендовано
Вченою радою університету як навчальний посібник
для студентів економічних і технічних спеціальностей

Харків 2021

УДК 004.89 519.863
С 60

Рецензенти:

Гриф надано Вченою радою Харківського національного університету будівництва та архітектури (протокол № 11 від 24.12 2021 року)

Г.В.Солодовник
С – 60 Методи та системи штучного інтелекту. – Х.: ТОВ «ДІСА ПЛЮС», 2021. -177 с.
ISBN 978-617-7927-99-9

Системи штучного інтелекту давно вже увійшли в повсякденне життя людини: перекладачі, помічники, розумні будинки. Тому для сучасних фахівців необхідними є знання про методи розробки таких систем.

Навчальний посібник присвячено питанням логіки мислення, методам подання знань, передумовам виникнення та розвитку систем штучного інтелекту, принципам розробки інтелектуальних систем.

Новизна посібника полягає в системному підході до висвітлення питань розробки систем штучного інтелекту починаючи основами логіки та закінчуючи інструментальними засобами створення експертних систем. У посібнику наведено велику кількість прикладів до наведених понять, а також прикладів розв'язання задач. Самостійне навчання здобувачів забезпечуються наявністю питань та завдань для самостійної перевірки знань в кожному розділі.

Призначено для самостійного вивчення методів та систем штучного інтелекту здобувачами вищої освіти всіх спеціальностей.

Іл.: 33; табл.: 26; бібл.: 16 назв.

ISBN 978-617-7927-99-9

ВСТУП

Предметом вивчення навчальної дисципліни „Методи та системи штучного інтелекту” є методологія та інструментальні засоби логічного програмування, принципи, моделі, методи, технології проектування і розроблення програмних продуктів різного призначення.

З урахуванням вищевикладеного, мета дисципліни „Методи та системи штучного інтелекту” полягає у формуванні систематизованих знань про основні моделі, методи, засоби та мови, які використовуються під час розробки систем штучного інтелекту. Ознайомлення здобувачів освіти з основними методами пошуку рішень, які використовуються в системах штучного інтелекту, дозволять їм робити обґрунтований вибір методів, засобів та мов під час вирішення практичних завдань, а також розробляти та впроваджувати експертні системи раціональним шляхом.

Під час вивчення дисципліни студенти набувають таких компетентностей: здатність розв’язувати складні спеціалізовані задачі та практичні проблеми у галузі комп’ютерних наук або у процесі навчання, що передбачає застосування теорій та методів комп’ютерних наук, інформаційних технологій і характеризується комплексністю та невизначеністю умов; здатність до інтелектуального аналізу даних на основі методів обчислювального інтелекту включно з великими та погано структурованими даними, їхньої оперативної обробки та візуалізації результатів аналізу в процесі розв’язування прикладних задач.

Основні завдання курсу полягають:

– у розширенні й поглибленні знань про системи, які володіють можливостями розв’язання інтелектуальних задач;

– у вивченні загальних принципів побудови та функціонування інтелектуальних систем, математичних методів моделювання актів інтелектуальної діяльності;

– у оволодінні практичними навичками розробки, відлагодження та використання програмних засобів з елементами штучного інтелекту;

– у засвоїнні прийомів роботи з розробки та впровадження експертних систем засобами PROLOG.

Посібник " Методи та системи штучного інтелекту " складається з семи основних розділів.

Розділ 1 "Поняття штучного інтелекту" – це розділ, в якому розглядаються основні поняття та означення, які використовуються у сфері розробки систем штучного інтелекту, інструментальні засоби розробки

таких систем та методи подання знань. Цей розділ присвячено також питанням виникнення інтелектуальних систем, розглянуто основні напрямки їх розвитку.

Розділ 2 "Подання знань засобами математичної логіки" – це розділ, в якому розглядається логіка як наука, форми мислення, основні поняття та операції числення висловлювань. Розділ містить опис виконання практичних робіт з побудови таблиці істинності та процедури перетворення формул в досконалу нормальну форму.

Розділ 3 "Логіка предикатів першого порядку" – це розділ, в якому розглядається основні поняття логіки предикатів першого порядку, процедури інтерпретації формул у логіці предикатів першого порядку та побудови предвареної нормальної форми таких формул. У розділі наведено також принцип резолюції як метод логічного виведення фактів та продукційна модель представлення знань. Розділ містить опис виконання практичних робіт з процедур переведення формули у предварену нормальну форму, логічного виведення методом резолюцій та побудови прямого та зворотного ланцюгів міркувань.

Розділ 4 "Подання знань в вигляді семантичних мереж та фреймів". – це розділ, в якому висвітлюються питання, пов'язані з представлення знань у вигляді семантичних мереж та фреймів.

Розділ 5 "Експертні системи" – це розділ, в якому наводяться структура, класифікація та основні принципи побудови експертних систем, а також опис інструментальних засобів розробки та логіка їх побудови.

Розділ 6 "Мова логічного програмування PROLOG" – це розділ, в якому розглядаються концепція логічного програмування, компоненти та основні розділи програм складених мовою PROLOG. Розділ містить описи складових об’єктів у мові PROLOG та предикати управління процедурою виведення висновків.

Розділ 7 "Карти, що самоорганізуються" – це розділ, в якому розглядається основні поняття штучних нейронних мереж, процедури навчання та використання карт Кохонена, а також мережи Хопфілда, як різновид штучних нейронних мереж.

До складу кожного розділу навчального посібника входять: основний теоретичний матеріал, що стосується відповідної теми, приклади розв’язання типових задач за темами посібника, контрольні запитання для поточного контролю знань, а також задачі для самостійного розв’язання.

РОЗДІЛ 1
ПОНЯТТЯ ШТУЧОГО ІНТЕЛЕКТУ

1.1 Основні поняття та означення

Існує багато визначень поняття «штучний інтелект» надаймо деякі з них.

Штучний інтелект (Artificial Intelligence (AI)) – це галузь інформатики, предметом якої є розробка комп'ютерних систем, що володіють можливостями, традиційно пов'язаними зі здібностями природного інтелекту.

Штучний інтелект – властивість інтелектуальних систем виконувати творчі функції, які традиційно вважаються прерогативою людини; наука і технологія створення інтелектуальних машин, особливо інтелектуальних комп’ютерних програм.

Штучний інтелект – здатність системи правильно інтерпретувати зовнішні дані, отримувати уроки з цих даних та використовувати отримані знання для досягнення конкретних цілей та розв’язання задач за допомогою гнучкої адоптації.

Інтелект (від лат. *intelectus* – пізнання, розуміння) – це цілісна сукупність функцій, проявів діяльності високоорганізованої матерії – людського мозку (мислення, емоцій, уяви, тощо), що спрямована на пізнання та перетворення природи, суспільства та самого себе.

Штучний інтелект має два різновиди:

Сильний штучний інтелект (Strong AI) – програмне забезпечення, завдяки якому комп'ютер може думати подібно до того, як думає людина.

Слабкий штучний інтелект (Weak AI) – функції, які можна додавати до вже існуючих систем, щоб надавати їм «розумних» властивостей.

Системи штучного інтелекту оперують ні даними, а знаннями. Наведемо різницю між цими поняттями.

Дані – це окремі факти, що характеризують об’єкти, процеси і явища предметної області, а також їх властивості.

Знання – це закономірності певної предметної області (принципи, зв’язки, закони), які були отримані в результаті практичної діяльності та

професійного досвіду, які дозволяють фахівцям формулювати та розв’язувати задачі в цій предметнїй області.

Кажуть також, що знання – це добре структуровані дані, або дані про дані, або метадані.

Знання повинні мати наступні особливості:

1 *внутрішню інтерпритуємість*: кожна інформаційна одиниця повинна мати унікальне ім’я, за яким система має її знаходити, за переходу до представлення знань в пам’яті комп’ютера до них додається певна інформація про деяку протоструктуру інформаційної одиниці (наприклад, спеціальне машинне слово, що визначає, в яких розрядах сберігаються відомості про прізвища, роки народження, тощо);

2 *структурованість*: інформаційні одиниці повинні мати гнучку структуру та рекурсивну вкладеність одних інформаційних одиниць в інші; така особливість надає можливість встановлювати між інформаційними одиницями ієрархічні відношення типу «частина-ціле», «род-вид», «елемент-клас»;

3 *зв’язаність*: повинна існувати можливість встановлення між інформаційними одиницями зв’язків, що характеризують відношення між ними;

4 *семантичну метрику*: повинна існувати можливість встановлення на множині інформаційних одиниць відносини, що завдають силу асоціативних зв’язків, для пошуку знань, що є близькими до вже знайдених;

5 *активність*: знання є чинником актуалізацій дій в системі, тобто виконання програм у інтелектуальних системах ініціюється станом інформаційної бази.

Наведемо визначення понять «предметна область» та «предметна область» для висвітлення різниці між цими поняттями.

Предметна область – частина реального світу, що розглядається з точки зору певного дослідження або це область людської діяльності, яка виокремлена об’єктно-орієнтованим чином.

Проблемна область – це певна сукупність задач та питань, що розв’язуються в рамках окремого дослідження.

1.2 Проблеми штучного інтелекту

1 Подання знань. Наявність адекватно поданих знань та здатність їх ефективно використовувати визначають "вміння", які повинні мати системи штучного інтелекту. Для цього необхідно знайти спосіб вираження загальних закономірностей реального світу моделями подання знань.

2 Розв'язання задач – це відшукування шляху з деякої початкової точки до цільової точки. Людина здійснює це за допомогою дедуктивного логічного виведення (міркувань), процедурального аналізу, аналогії, індукції та власного досвіду. Комп'ютери поки що розв'язують задачі лише з використанням дедуктивного логічного висновку й процедурального аналізу. Задачами, які зводяться до процедурального аналізу і найкраще розв'язуються на комп'ютері, є наступні: облікові задачі, ведення рахунків, аналіз надходження готівки.

Дедукція – спосіб міркування, за якого нове положення (факт) виводиться суто логічним шляхом від загальних положень до конкретних висновків.

Іншим способом міркування є індукція. Різницю між дедуктивним та індукційним способами міркування можна подати схемою (рис. 1.1)

Рисунок 1.1 – Різниця між дедуктивним та індукційним способами міркування

Наведемо приклади виведення висновків за допомогою дедуктивного та індуктивного способів мислення.

Дедуктивне виведення висновків (від загального до окремого) базується на аналізі логічної структури припущень та наслідків.

Приклад 1.1

Факт 1: «якщо чотирикутник є квадратом, то його діагоналі рівні».

Факт 2: «чотирикутник ABCD є квадратом».

Висновок: «AD = CB».

Приклад 1.2

Факт 1: «якщо число є кратним 6, то воно є парним».

Факт 2: «18 є кратним 6».

Висновок: «18 є парним числом».

Індуктивне виведення висновків (від окремого до загального) базується на аналізі вмісту припущень та наслідків.

Приклад 1.3

Факт 1: «дуб – листяне дерево».

Факт 2: «береза – листяне дерево».

Факт 3: «липа – листяне дерево».

Висновок: «всі дерева листяні».

Приклад 1.4

Факт 1: «Земля – планета, що обертається навколо Сонця».

Факт 2: «Меркурій – планета, що обертається навколо Сонця».

Факт 3: «Юпітер – планета, що обертається навколо Сонця».

Висновок: «всі планети обертаються навколо Сонця».

Приклад 1.5

Факт 1: «Земля – планета Сонячної системи, що обертається навколо Сонця».

Факт 2: «Меркурій – планета Сонячної системи, що обертається навколо Сонця».

Факт 3: «Юпітер – планета Сонячної системи, що обертається навколо Сонця».

Висновок: «всі планети Сонячної системи обертаються навколо Сонця».

З аналізу наведених прикладів можна зробити висновок, що формування множини початкових припущень для подальшого виведення висновків має велике значення для істинності висновків.

3 Експертні системи – це програми, які здатні накопичувати знання, що містяться в різних джерелах, та моделювати процес експертизи. ЕС орієнтовані на розв’язання задач, що зазвичай вимагають проведення експертизи людиною-експертом. Експертні системи справляються з відсутністю структурованості задачі шляхом залучення евристик, тобто правил експерта, що може бути корисним у тих ситуаціях, коли брак знань або часу виключає можливість проведення повного аналізу.

4 Засоби спілкування з комп’ютером природною мовою. Мову можна визначити як набір символів, що використовуються для подання знань (семантика), правил, призначених для обробки таких символів (синтаксис) і для розв’язання задач.

В рамках цієї проблеми розглядають чотири ключові питання:

1) машинний переклад – використання комп'ютера для перекладу тексту з однієї мови на іншу;

2) інформаційний пошук – забезпечення за допомогою комп'ютера доступу до інформації з конкретної тематики, що зберігається в базі даних;

3) генерація документів – використання комп'ютера для перетворення документів, що мають певну форму або заданих спеціалізованою мовою, в еквівалентний документ в іншій формі або іншою мовою, наприклад, керівництво для інженера на керівництво, яке зрозуміле лікареві;

4) взаємодія з комп'ютером – організація діалогу між непідготовленими користувачами й комп'ютером.

5 Когнітивне моделювання – це галузь науки, яка розробляє теорію і моделі людського мислення та його функцій. Метою когнітивного моделювання є розробка теоретичної концепції і моделей людського мислення та його функцій. Когнетивне моделювання повинно допомогти не тільки діагностувати та лікувати психічні захворювання, але й виявляти процеси, що протікають у свідомості людини під час розв’язання задач.

6 Робототехніка та обробка візуальної інформації. Такі дослідження проводяться в трьох напрямках:

1) розробка елементів, які сприймають інформацію, та розпізнання інформації, що надходить від системи сприйняття;

2) створення маніпуляторів та систем управління ними;

3) виявлення евристик для розв’язання задач переміщення в просторі та маніпулювання об’єктами (планування діяльності).

1.3 Виникнення та розвиток штучного інтелекту

1.3.1 Наукові передумови створення штучного інтелекту

Виникнення *філософських передумов* створення штучного інтелекту можна віднести до часів древньої Греції, а саме до спроб розуміння процесу мислення та формулювання Аристотелем силогізмів, що стали підґрунтям процедур доведення гіпотез.

Технічні передумови створення штучного інтелекту беруть початок зі створення Вільгельмом Шикардом першої механічної цифрової обчислювальної машини у 1623 році.

Початком історії виникнення штучного інтелекту прийнято вважати середину XX століття, коли обчислювальні можливості комп’ютерів значно перевищили можливості людини, через що у науковців виникло питання стосовно границь можливостей комп’ютерів. Саме тоді в 1950 році було запропоновано тест Тбюринга, про який піде мова в наступному пункті. К цьому часу передумови створення штучного інтелекту виникли в багатьох галузях людської діяльності: філософи намагалися визначити сутність процесу пізнання світу людиною, психологи та нейрофізіологи розробляли теорії стосовно роботи людського мозку та мислення, економісти та математики вивчали питання оптимальних розрахунків та формалізації представлення знань про навколишній світ.

Наведемо перші результати, що були отримані за деякими проблемами штучного інтелекту, які було наведено в попередньому пункті.

Експертні системи. Прикладом першої експертної системи може слугувати Dendral – система для ідентифікації органічних сполук за допомогою аналізу мас-спектрограм, яка була розроблена в 1965 році. На підставі даних спектрометрії про речовину система надавала висновок щодо її хімічної структури. Dendral містила програми, які надавали можливість приймати або відкидати гіпотези на підставі знань про зв’язки показань мас-спектрометра зі структурою молекул сполуки.

Інший приклад першої експертної системи це MYCIN – система для діагностування бактерій, що викликають важкі захворювання, а також визначення кількості необхідних антибіотиків з урахуванням ваги пацієнта. Систему було розроблено в Стенфордському університеті у 70-х роках XX століття (розпочато роботу над системою було в 1972 році).

Засоби спілкування з комп'ютером природною мовою. Обробка природньої мови або комп'ютерна лінгвістика (Neural language processing) як напрям розвитку штучного інтелекту бере початок у 1954 році з Джорджтаунського експерименту, в якому було продемонстровано переклад понад 60 простих речень з однієї мови на іншу. Проте згодом з'ясувалися труднощі у перекладі складніших речень, які були подолані лише десятю роками пізніше.

Робототехніка та обробка візуальної інформації. В 1969-1971 роках в Единбургському університеті було створено першу версію робота Freddy (друга версія датується 1973-1976 роками), який мав зорову, маніпулятивну та інтелектуальну системи. Шляхом співставлення даних про розташування та вид деталей, що надходили з його камери, і з відповідними моделями деталей у пам'яті. Freddy міг збирати прості іграшки з дерев'яних блоків.

1.3.2 Тест Тьюринга

Тест Тьюринга – емпіричний тест, ідея якого була запропонована Аланом Тьюрингом у статті «Обчислювальні машини та розум». Статтю було опубліковано в 1950 році у філософському журналі Mind. Тьюринг поставив собі за мету визначити, чи може машина мислити.

Стандартна інтерпретація цього тесту полягає в наступному: «Людина взаємодіє з одним комп'ютером та з однією людиною. На підставі відповідей на питання вона повинна визначити, з ким розмовляє: з людиною або з комп'ютерною програмою. Завдання комп'ютерної програми – ввести людину в оману, змусивши зробити невірний вибір».

На проведення тесту накладаються наступні умови:

– спілкування ведеться у форматі «лише текст», для того щоб виключити завдання розпізнавання комп'ютером людської мови;

– питання та відповіді надсилаються через певні проміжки часу, тому що комп’ютер формував відповідь довше за людину за часів Тьюрінга або навпаки, швидше за людину у наш час.

Різновидом проведення тесту Тьюрінга є введення в оману ні однієї людини, а журі, що складається з кількох людей. Тест вважається пройденим, якщо помилилося більше за половину членів журі. Як і за проведення групових експертних процедур, таке тестування підвищує вірогідність отриманих результатів.

У 1966 році американським вченим Джозефом Вейценбаумом було розроблено програму Еліза, яка імітувала поведінку психотерапевта, що працює за клієнт-орієнтованою методикою. Основний принцип роботи програми полягав в аналізі введених користувачем коментарів та пошуку ключових слів, далі:

– якщо ключове слово було знайдено, то спрацьовувало правило, за яким коментар перетворювався у речення-результат;

– якщо ключове слово не було знайдено, то або надавалася загальна відповідь, або одне з попередніх речень-результатів.

Робота цієї програми переконувала деяких людей в тому, що вони спілкуються зі справжнім психотерапевтом, більше того їх було складно переконати в тому, що вони спілкуються з комп’ютерною програмою. Однак результат проходження тесту Тьюрінга в цьому випадку не вважається доведеним, оскільки люди були попереджені про те, що вони будуть спілкуватися з психотерапевтом і вони, під час спілкування, не мали за мету дізнатися з людиною чи з комп’ютером вони мають справу.

У 1972 році психіатр Кеннет Колбі розробив програму PARRY призначену для лікування шизофренії. Роботу цієї програми тестували модифікованим тестом Тьюрінга, який полягав в наступному: команда з досвідчених психіатрів аналізувала групу, яка складалася зі справжніх пацієнтів та комп’ютерів, що працювали за управління програми PARRY, а потім, журі, яке складалося з 33 інших психіатрів аналізувало записи бесід. Обидві команди мали за мету визначити хто з пацієнтів є людиною, а хто комп’ютером. Психіатри надали правильні відповіді в 48% випадків.

Висновок про проходження тесту Тьюрінга в цьому випадку також було поставлено під сумнів, оскільки проведення тесту передбачає інтерактивне спілкування, а не аналіз записів бесід.

Проходження тесту Тьюрінга не є ціллю розробок інтелектуальних систем. Для його проходження система повинна бути не надто розумною, адже якщо вона відповість, наприклад, на обчислювальну задачу, що є занадто складною для людини вона не зможе переконати журі в тому що є людиною. З іншого боку людина навчилася літати, але це зовсім не схоже на політ птаха, тому для того, щоб штучний інтелект вирішував складні слабоструктуровані або навіть творчі завдання він не повинен імітувати процес мислення людини.

1.3.3 Штучний інтелект, психологія та процес мислення

Виокремлюють наступні психічні явища:

– психологічні процеси;

– психологічний стан;

– властивості особистості.

До *психологічних процесів* належать: мислення, уявлення, пам’ять, уява, відчуття, сприйняття, увага, тощо.

Наведемо пояснення деяким з наведених психологічних процесів.

Відчуття – це відображення у мозку предметів та явищ, які безпосередньо діють на органи чуття.

Сприйняття – це цілісне відображення предмету, що безпосередньо впливають на органи чуття (результатом процесу сприйняття є образ предмету).

Уявлення – це чуттєвий образ предмету, який не сприймається в даний час, але був сприйнятий в минулому.

Психологічні стани є більш довгостроковими та відбивають психіку в цілому (наприклад, настрій, емоції, втома, функціональний стан).

Властивості особистості описують характер та темперамент людини.

Властивості особистості впливають на психологічний стан, а той в свою чергу впливає на психологічні процеси.

Можна навести кілька визначень поняття «мислення».

Мислення – найбільш узагальнена та опосередкована форма психологічного відображення, яка встановлює зв'язки та відношення між об'єктами, що пізнаються.

Мислення – процес пізнавальної діяльності індивіда, яка характеризується узагальненим та опосередкованим відображенням дійсності.

Мислення дозволяє отримати знання про такі об'єкти, властивості та відношення реального світу, які не можуть бути безпосередньо сприйнятті на чуттєвому рівні пізнання.

Існують наступні *форми абстрактного мислення*:

Розуміння – форма абстрактного мислення, в якій відбиваються суттєві ознаки предмету або класу об'єктів.

Судження – форма абстрактного мислення, в якій стверджується або заперечується щось про предмети або явища. Судження може бути істинним або хибним, індуктивним або дедуктивним.

Умовиведення – форма абстрактного мислення, за допомогою якої з двох суджень отримують третє судження.

Наведемо основні прийоми формулювання понять.

Аналіз – мисленне розбиття предмету або явища на складові частини.

Синтез – мисленне з'єднання окремих частин предмету або явища, на які вони були розбиті під час аналізу.

Порівняння – мисленне встановлення схожості або відмінності між предметами або явищами.

Абстрагування – мисленне виокремлення одних ознак та відкидання інших.

Узагальнення – мисленне об'єднання окремих предметів у певному понятті.

Існують наступні види мислення, що розрізняються за формою.

Наглядно-дійове мислення – це один з видів мислення, що характеризується реальним фізичним перетворенням ситуації та випробування властивостей об'єктів під час пізнавальної діяльності.

Наглядно-образне мислення – це один з видів мислення, що передбачає уявлення ситуації та змін у неї, припускає створення

непередбачуваних поєднань властивостей та об'єктів, і з цієї точки зору є майже тим самим, що й уява.

Словесно-логічне (абстрактне) мислення – це один з видів мислення, що характеризується використанням понять та логічних конструкцій, передбачає використання мовних засобів.

Наведені види мислення можуть бути розглянуті як етапи розвитку та становлення процесу мислення індивіда.

За типом задач, що розв'язуються в процесі мислення.

Теоретичне мислення – це один з видів мислення, що полягає в впізнанні законів та правил шляхом виокремлення та аналізу основного початкового протиріччя ситуації, що досліджується. Пошук засобів розв'язання протиріччя призводить до формування послідовності дій, яка дозволяє розв'язувати цілі класи задач. Використовується наприклад для розв'язання математичних задач.

Практичне мислення – це один з видів мислення, що полягає у фізичному перетворенні дійсності та пов'язано з постановкою цілей та розробкою планів дій з досягнення цих цілей, як правило використовується в умовах дефіциту часу. Використовується для розв'язання практичних задач.

За ступенем новизни продукту, який отримує індивід в процесі мислення та за ступенем оригінальності мислення поділяється на такі види.

Продуктивне (творче) мислення – це один з видів мислення, що характеризується створенням суб'єктивно нового продукту та новоутвореннями в самій пізнавальній діяльності щодо створення цього продукту.

Репродуктивне (відтворювальне) мислення – це один з видів мислення, який передбачає розв'язання задач за допомогою вже відомого способу дій.

За ступенем розгорнутості та впорядкованості мислення поділяється на наступні види.

Дискусивне мислення – це один з видів мислення, що передбачає розв'язання задач за допомогою послідовного логічного міркування, в якому кожний наступний крок обумовлений попереднім. Існують дедуктивний та індуктивний типи дискусивного мислення.

Інтуїтивне мислення – це один з видів мислення, для якого характерно сприйняття даних про ситуацію вцілому та «стрибок» до нових знань з розривом у виведенні фактів та відступлення від раніше встановленої логіки. Такий «стрибок» прийнято визначати терміном «інсайт», який походить від англійського «insight», що перекладається як проникливість, проникнення в сутність, осяяння, здогадка.

Окремим видом мислення є пралогічне мислення.

Пралогічне мислення – це один з видів мислення, що характеризується містифікацією причинно-наслідкових зв'язків та встановлення цих зв'язків навіть у тих випадках, коли відбувається просто співпадіння явищ у часі.

З наведеної класифікації процесу мислення можна зробити висновок, що для сучасних інтелектуальних систем найбільш притаманними є словесно-логічне теоретичне репродуктивне дискусивне дедуктивне мислення. Продуктивність процесу мислення може бути описана наступними показниками: швидкість, творчість, широта, гнучкість, глибина, критичність.

Здатність до певного роду діяльності, втому числі знатність до мислення, може бути визначена як індивідуальна психологічна властивість, що проявляється у діяльності та є запорукою її успішного здійснення. Здатність передбачає два рівні: перший – репродуктивний рівень (вміння засвоювати знання), другий – творчій (вміння створювати нові знання або продукувати оригінальні ідеї).

Аналіз та моделювання творчого процесу мислення слід починати з поняття «проблемна ситуація».

Проблемна ситуація – ситуація, в якій індивід (або група індивідів) повинен знайти та дослідити нові для себе засоби та способи діяльності.

Проблемна ситуація передбачає наявність наступних умов:

– реальне існування протиріччя;

– розуміння сутності протиріччя;

– наявність необхідності зняття даного протиріччя;

– наявність часу та засобів (в тому числі інтелектуальних) зняття даного протиріччя.

З метою моделювання та підвищення ефективності інтелектуальної творчої діяльності було розроблено декілька методів.

1 Метод «проб та помилок». Цей метод використовується під час розв'язання відносно нескладних задач, для пошуку розв'язку яких достатньо до кількох десятків тисяч спроб.

Недоліками методу є значні витрати часу та неможливість використання знайденого шляху розв'язання для розв'язання іншої подібної задачі.

2 Метод «мозкової атаки». Цей метод полягає у генерації ідей розв'язання задачі без критики, з подальшим відкидання непродуктивних ідей та впровадженням продуктивних. Метод має багато різновидів.

Недоліком методу є велика кількість тривіальних та непродуктивних ідей.

3 Метод синектики. Цей метод передбачає використання аналогій та застосування для рішення нових завдань, що виникли, раніше розроблених рішень. Ґрунтується на тому, що індивід за певних особливих умов висуває несподівані аналогії та асоціації щодо проблемної ситуації.

Недоліками є те, що ідей не завжди є оригінальними, та цей метод не гарантує абсолютного успіху.

4 Метод аналогій. Цей метод є різновидом попереднього методу, полягає у застосуванні емпатії, тобто уявлення себе на місті предмету вивчення.

5 Метод АРВЗ (алгоритм розв'язання винахідницьких задач). Цей метод передбачає з'ясування питань: які існують протиріччя в проблемній ситуації та як їх можна розв'язати. Застосування методу передбачає три стадії, які в свою чергу розбиті на окремі кроки. Стадіями методу АРВЗ є: аналітична, оперативна, синтетична.

1.4 Поняття інтелектуальної системи (ІС) та інтелектуальної задачі (ІЗ)

Системами штучного інтелекту або інтелектуальними системами (ІС) називаються системи, що виконують функції, які прийнято розглядати як інтелектуальні.

Інтелектуальна діяльність людини пов’язана з пошуком розв’язань (дій, закономірностей) в нових, нестандартних ситуаціях. Задача є інтелектуальною, якщо точний (алгоритмізований) метод її розв’язання є заздалегідь невідомим. Розв’язання задачі – це будь-яка діяльність (людини або машини), пов’язана з розробкою планів та дій, необхідних для:

– досягнення визначеної мети;

– виведення нових закономірностей;

– формування фраз комп’ютером, звичайною мовою або мовою, близькою до звичайної.

В процесі інтелектуальної діяльності людина використовує знання про певну частину реального світу, в сфері якої формулюються та вирішуються завдання. Сукупність взаємопов’язаних відомостей, необхідних та достатніх для розв’язання певної задачі або сукупності задач – предметна або проблемна область. Знання про предметну область вміщують опис об’єктів, явищ, фактів, а також взаємозв’язків між ними.

Особливістю ІС є наявність в них блоку подання знань, пов’язаного з «зовнішнім» світом двома блоками перетворювачів, які перетворюють знання про предметну область з зовнішнього подання на внутрішнє і, навпаки, з внутрішнього – на зовнішнє, яке є зрозумілим для користувача.

Інформаційна модель блоку подання знань в загальному випадку складається з наступних блоків:

1 блоку інтерпретацій;

2 блоку виведення розв’язків;

3 блоку навчання;

4 інтелектуального банку даних.

В свою чергу інтелектуальний банк даних підрозділяється на:

– базу знань, яка містить відомості, що відбивають закономірності даної предметної області та дозволяють прогнозувати й виводити нові факти, яких не містить база даних;

– базу даних, яка містить фактографічні та кількісні дані, які характеризують предметну область.

У загальному вигляді знання в комп’ютері подаються деякою знаковою (семіотичною) системою, в якій виокремлюють три аспекти, описи яких наведено у таблиці 1.1.

Таблиця 1.1 – Опис аспектів семіотичної системи

Номер аспекта	Назва аспекта	Роль аспекта	Тип відповідних знань
1	Синтаксичний	Описує внутрішню будову знакової системи, тобто *правила побудови* та *перетворення* складних знакових виразів	Синтаксичні знання характеризують синтаксичну структуру об'єкта або явища, що описуються, і не залежать від сенсу та вмісту понять, що при цьому використовуються
2	Семантичний	Визначає відношення між знаками та їх властивостями, тобто задає *сенс* або значення конкретним знакам	Семантичні знання містять інформацію, безпосередньо пов'язану зі значенням та сенсом явищ і об'єктів, які описуються
3	Прагматичний	Визначає знаки з точки зору конкретної сфери його використання або з *точки зору суб'єкта*, який використовує дану знакову систему	Прагматичні знання описують об'єкти та явища з точки зору задачі, що вирішується, з урахуванням специфічних факторів, які діють в даній задачі

Трьом типам знань відповідають три типи моделей: синтаксичні, семантичні та прагматичні. Саме наявність семантичних та прагматичних моделей є найбільш суттєвою ознакою, яка відрізняє ІС від інших програмно-апаратних засобів.

Широке застосування ШІ у всіх сферах людської діяльності, зокрема у повсякденні пов'язане зі змінами не тільки і, навіть, не стільки в інформаційних технологіях скільки у свідомості людей, законодавстві та побутових звичках. Основними чинниками, які обумовлюють ефективність впровадження ШІ є: по-перше, можливість автоматизації процесів, для яких до того необхідна була участь людини, по-друге, обробка та аналіз надвеликої кількості інформації. Наведемо приклади застосування ІС у різних сферах людської діяльності.

В *медицині* ШІ застосовується для:

1 надання порад лікарям, розробки планів терапевтичного лікування, виявлення генетичної схильності до патологій (IBM Watson та DeepMind Health – розробки компанії Google);

2 надання порад пацієнтам та встановлення діагнозів на підставі даних, зібраних з фітнес-браслетів та інших датчиків, а також опитуванню (Your.MD и Ada);

3 розробки та модифікації лікарських препаратів за допомогою моделювання та створення молекулярної структури (розробки компаній Atomwise та Berg Health).

В *промисловості* ШІ застосовується для:

1 автоматичного збирання деталей;

2 ведення бухгалтерських розрахунків;

3 спілкування з клієнтами.

В *освіті* ШІ застосовується для:

1 відстеження оволодіння матеріалом кожного з учнів та надання відповідних даних викладачу або модифікація курсу (платформа Third Space Learning);

2 контролю поведінки учнів під час екзаменів та тестування, зокрема виявлення несанкціонованого використання інформаційних систем, телефонів, тощо, а також відведення погляду від монітору та розмов.

В *сільському господарстві* ШІ застосовується для:

1 проведення аерофотозйомки та підвищення ефективності обприскування сільськогосподарських культур за рахунок безпечної доставки хімікатів за допомогою дронів з радарами и GPS-мониторінгом;

2 збирання вражаю культур, які раніше збиралися виключно вручну (наприклад, полуниці);

3 розпізнавання та знищення бур’янів механічним способом або обприскуванням гербіцидами (Hortibot – розробка университета Aarhus Universitet в Данії).

В *організації дорожнього руху* ШІ застосовується для:

1 контролю та прогнозування трафіку та перемикання світлофорів на підставі аналізу даних про щільність руху, ДТП, метеорологічних та інших умов;

2 надання порад водіям щодо уникнення заторів, прокладання найкоротшого маршруту, виклику евакуатора, тощо;

3 автоматичного керування транспортом.

У *побуті* ШІ застосовується для:

1 ефективного використання ресурсів та зручного контролю за побутовими приладами завдяки системам типу «розумний будинок»;

2 автоматичного перекладу з однієї мови на іншу за допомогою сприйняття та аналізу не окремих слів, а цілих речень (Google Translate);

3 розваг та спілкування.

1.5 Моделі подання знань, їх переваги та недоліки

З позицій способів подання інтелектуальних знань та механізмів їх логічного виведення, можна дати наступне визначення: «знання – це формалізована інформація, на яку посилаються або використовують у процесі логічного виведення». Знання поділяються на:

– факти (фактичні знання);

– правила (знання для прийняття рішення);

– метазнання (знання про знання).

З позицій системного аналізу знання можна поділити на:

1 структурні (знання про структуру системи або про будову об’єктів предметної області);

2 каузальні (знання про причинно-наслідкові залежності в системі, тобто про поведінку об’єктів предметної області);

3 функціональні (знання про зв’язок системи з зовнішнім середовищем тобто метасистемою).

З метою подальшої програмної обробки знання мають бути представлені за допомогою формальних моделей подання. Виокремлюють чотири типові моделі подання знань:

1 модель семантичної мережі;

2 фреймова модель;

3 логічна модель;

4 продукційна модель.

Класифікацію моделей подання знань можна подати у вигляді схеми, наведеної на рисунку 1.2.

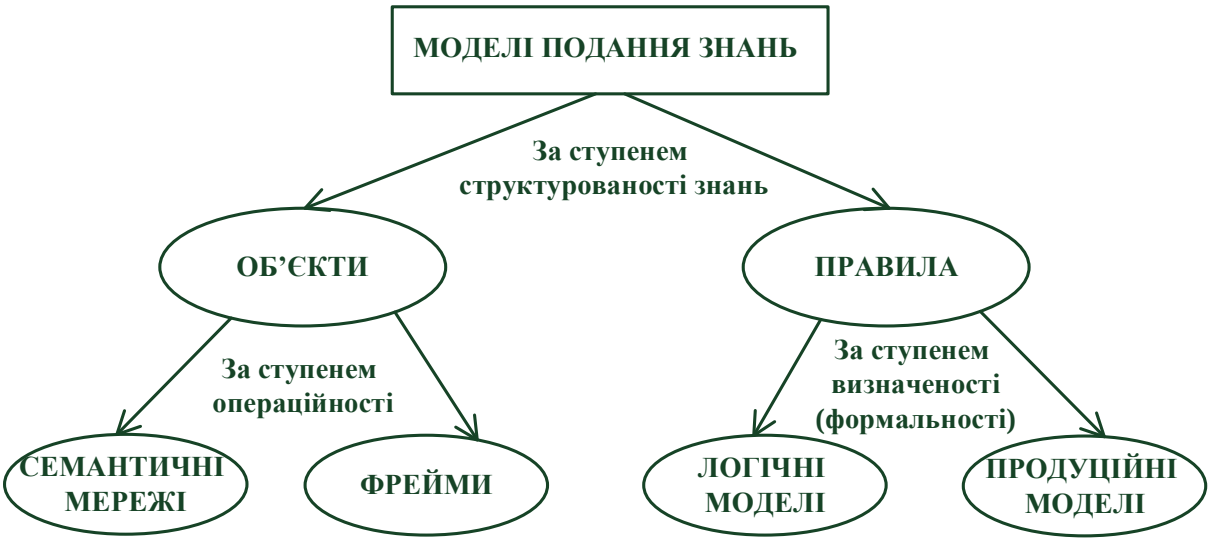


Рисунок 1.2 – Класифікація моделей подання знань

Семантична мережа (або смислова мережа) – це модель предметної області, яка має вигляд графа, вершинами якого є поняття, а дугами (ребрами) – відносини між цими поняттями. Відносини можуть бути наступних типів: «Рід-Вид», «Ціле-Частина», «Причина-Наслідок», «Засіб-Мета», «Аргумент-Функція», «Ситуація-Дія».

Фрейм є різновидом семантичної мережі – це модель представлення абстрактних образів, з мінімально можливим описом сутності певного об'єкта, явища, події, ситуації, процеса. Всі відносини у фреймах зібрані в одну структуру, що має зазвичай ієрархічний вигляд.

Логічна модель – це модель подання знань, що передбачає опис предметної області логічною мовою за допомогою логіки предікатів. За використання цієї моделі властивості об'єктів предметної області та зв'язки між ними подаються у вигляді функцій (предикатів), які можуть приймати тільки два значення: ІСТИНА або ХИБНО.

Продукційна модель – це модель подання знань, за якої знання описуються за допомогою правил «якщо-то» (або явище → реакція). Знання подаються у вигляді:

ЯКЩО умова (антецедент)

ТО дія (консеквент)

В даному випадку умова – це деяке речення-зразок, за яким відбувається пошук в базі знань, а дія – набір операцій, що виконуються за успішного результату пошуку.

Переваги та недоліки наведених способів подання знань представлені в таблиці 1.2.

Таблиця 1.2 – Переваги та недоліки різних способів подання знань

Переваги	Недоліки
Семантична мережа	
1 Наочність системи знань завдяки графічному представленню. 2 Подібність структури мережі семантичній структурі фраз природньої мови. 3 Відповідність сучасним представленням довгострокової пам'яті людини.	1 Низька ефективність представлення знань (лише апаратом семантичної мережі). 2 Складність організації процедури пошуку потрібного знання як фрагменту мережі.
Фрейм	
1 Можливість збереження родової ієрархії понять в базі знань в явній формі, що дозволяє ефективно використовувати пам'ять та аналізувати ситуації за відсутності певних деталей. 2 Універсальність стосовно виду зв'язків між поняттями завдяки наявності фреймів-структур, фреймів-ролей, фреймів-сценаріїв, фреймів-ситуацій, тощо. 3 Універсальність стосовно механізму управління виведенням висновку завдяки приєднаним процедурам.	1 Складність системи, що призводить до зниження швидкості виведення висновків та збільшення трудомісткості внесення змін в родову ієрархію. 2 Ускладнена процедура обробки виключень. 3 Відсутність спеціального механізму виведення висновків вимагає застосування приєднаних процедур. 4 Мова опису знань, що застосовується у фреймових системах далека від природньої.
Логічні моделі	
1 Наявність формальної процедури виведення висновків. 2 Можливість програмування механізму виведення синтаксично вірних висловлювань.	1 Складність побудови доведень евристик специфічних предметних областей.

	2 Складність аналізу великої кількості окремих фактів через відсутність структуризації. 3 Неприпустимість протиріч та труднощі аналізу неточних даних.
Продукційна модель	
1 Модульність моделі, яка забезпечує відносну незмінність продукцій у базі знань за додавання та видалення елементів. 2 Можливість реалізації різних алгоритмів, що надає можливість представлення будь-яких процедурних знань, які можуть бути реалізовані комп'ютером. 3 Наявність вказівок на сферу застосування продукцій дозволяє скоротити використання ресурсів на пошук потрібних знань у базі.	1 Складність перевірки несуперечливості системи продукцій за великої кількості елементів (така перевірка відбувається за кожного додавання нової продукції). 2 Складність перевірки правильності роботи системи через неоднозначність вибору продукції для виконання.

1.6 Інструментальні засоби розробки штучного інтелекту

Виокремлюють два основні підходи до розробки штучного інтелекту:

– спадний (Top-Down AI) або семіотичний – використовується для створення експертних систем, баз знань та систем логічного виведення, що імітують психічні процеси високого рівня: мислення, міркування, мова, емоції, творчість, тощо;

– висхідний (Bottom-Up AI) або біологічний – використовується для вивчення нейронних мереж та еволюційних обчислень, що моделюють інтелектуальну поведінку на підставі біологічних елементів, а також для створення відповідних обчислювальних систем, таких як нейрокомп'ютер або біокомп'ютер.

Процес створення засобів автоматизації програмування інтелектуальних систем, що орієнтовані на знання (зокрема експертних систем) рухається в трьох напрямках:

1 розробка систем представлення знань з використанням мов програмування обробки символьної інформації та мов програмування загального призначення;

2 розширення мов штучного інтелекту до систем представлення знань шляхом інтегрування спеціальних бібліотек та пакетів прикладних програм;

3 створення мов представлення знань, що орієнтовані на підтримку певних способів формалізації знань.

Сьогодні для розробки інтелектуальних систем використовують наступні інструментальні засоби:

1 *Спеціальні мови програмування*, орієнтовані на розробку символьної інформації (Smalltalk), та мови програмування загального призначення: Форт, LISP.

Smalltalk – об'єктно-орієнтована мова програмування з динамічною типізацією (не передбачено вказання типу змінних в програмі), яку було розроблено фірмою Xerox PARC в 1970-х роках. Перша версія мови Smalltalk-80 Version 1 була призначена для використання університетами та невеликим колом компаній, а друга, починаючи з 1983 року, стала загально доступною.

2 *Мови програмування для задач штучного інтелекту*: INTERLISP, PROLOG.

INTERLISP – це середовище програмування, яке було створено на основі мови програмування LISP. Розробка INTERLISP почалась в 1966 році. INTERLISP є інструментом розробки розповсюдженим серед дослідників штучного інтелекту в Стенфордському університеті. Особливостями середовища INTERLISP є інтегрування інтерактивних інструментів розробки: засобів відлагодження, засобів виправлення помилок та засобів аналізу, які зібрані в єдиному середовищі.

3 *Мови подання знань*.

Мови подання знань на базі фреймів: KRL, FRL, SRL, KEE, HP-RL, а також вдосконалені засоби мови програмування LISP для роботи з фреймами: LOOPS, FLAVORS. Характерними рисами яких є: дворівнева модель представлення даних (абстрактна модель предметної області у вигляді ієрархії множин понять та конкретна модель ситуації у вигляді сукупності взаємопов'язаних екземплярів цих понять).

KRL – це мова представлення знань, розроблена фірмою Xerox PARC (Стенфордський університет), яка базується на фреймовій моделі

представлення знань. Метою створення мови KRL була спроба створення засобу написання систем з обробки даних подібно до людської пам’яті, але недоліком мови стало переобтяження задачами розробки.

Мова подання знань на базі продукційних моделей – OPS 5 (Official Production System, version 5), особливостями якої є: гнучкість, виведення рішень, що управляється цілями, наявність засобів визначення стратегії управління даними.

4 *Інтегровані програмні середовища*, що містять арсенал інструментів для створення систем штучного інтелекту (ARTS, GURU, G2, KEE).

ARTS – інтегроване середовище, що підтримує технологію проектування систем, яка заснована на правилах. ARTS підтримує два основних метода формалізації представлення знань: правила для процедурного представлення знань та фреймові структури для декларативного представлення знань. Ефективність введення правил забезпечується механізмом Viewpoint, що заснований на іт істинності припущень. Продукційні знання в цьому середовищі описуються правилами п’яти видів: правила виведення, продукційні правила, гіпотетичні правила, правила обмежень та правила полаганія.

5 *Оболонки для розробки експертних систем* (BUILD, EMYCIN, EXSYS Prof, CLIPS), які дозволяють створювати прикладні експертні системи.

CLIPS, (C Language Integrated Production System) – інтегроване програмне середовище для розробки експертних систем. Початок розробок перших версій CLIPS датуються 1984 роком, розробки проводилися в Космічному центрі Джонсона NASA. CLIPS є продукційною системою, реалізація виведення якої базується на алгоритмі Rete – алгоритм співставлення зі зразком для продукційних систем, експертних систем та баз знань.

Задачі для самостійного розв’язання

Задача 1 Провести порівняльний аналіз штучного та людського інтелекту. Відповідь подати у табличному вигляді.

Задача 2 Провести порівняльний аналіз понять «дані» та «знання». Відповідь подати у табличному вигляді.

Задача 3 Провести аналіз останніх розробок в кожній з шості наведених вище проблем штучного інтелекту. Визначити інтелектуальні задачі, які вони вирішують.

Питання для самоконтролю

1 Дати визначення поняття «штучний інтелект» як галузі інформатики.

2 Дати визначення поняття «штучний інтелект» як певної властивості інтелектуальних систем.

3 Дати визначення поняття «штучний інтелект» як певної здатності інтелектуальних систем.

4 Дати визначення поняття «інтелект».

5 Які різновиди систем штучного інтелекту Вам відомі?

6 В чому на Вашу думку полягає різниця між поняттями «дані» та «знання»?

7 Які особливості повинні мати знання?

8 Дати визначення понять «предметна область» та «проблемна область».

9 Які проблеми, що виникають перед розробниками систем штучного інтелекту Вам відомі?

10 В чому полягає різниця між дедуктивним та індукційним способами міркування? Наведіть приклади таких міркувань.

11 Які наукові передумови створення штучного інтелекту Вам відомі?

12 В чому полягає сутність тесту Тьюринга? Які різновиди процедури проведення цього тесту Вам відомі?

13 Які психічні явища Вам відомі?

14 Дати визначення поняття «мислення».

15 Які існують форми абстрактного мислення?

16 Дати опис основних прийомів формулювання понять.

17 Які існують наступні види мислення, що розрізняються за формою?

18 Навести класифікацію видів мислення за типом задач, що розв’язуються.

19 Навести класифікацію видів мислення за ступенем новизни продукту, який отримує індивід в процесі мислення та за ступенем оригінальності мислення.

20 Навести класифікацію видів мислення за ступенем розгорнутості та впорядкованості.

21 Дати визначення поняття «проблемна ситуація». За наявності яких умов складається проблемна ситуація?

22 Які методи моделювання та підвищення ефективності інтелектуальної творчої діяльності Вам відомі? Навести переваги та недоліки цих методів.

23 В чому на Вашу думку полягає інтелектуальна діяльність людини?

24 Навести особливості інтелектуальних систем.

25 З яких блоків складається інформаційна модель блоку подання знань?

26 Навести опис аспектів семіотичної системи.

27 Які сфери застосування ШІ Вам відомі?

28 Навести класифікацію знань з позицій системного аналізу.

29 Навести класифікацію моделей подання знань.

30 Дати визначення наступних понять: «семантична мережа», «фрейм», «логічна модель», «продукційна модель».

31 Навести переваги та недоліки різних способів подання знань.

32 Які існують основні підходи до розробки ШІ?

33 Які інструментальні засоби використовують для розробки інтелектуальних систем?

РОЗДІЛ 2
ПОДАННЯ ЗНАНЬ ЗАСОБАМИ МАТЕМАТИЧНОЇ ЛОГІКИ

2.1 Логіка як модель мислення

В попередньому розділі було описано процес мислення (п. 1.3.3 Штучний інтелект, психологія та процес мислення). Далі розглянемо логіку як основу мислення та виведення одних суджень з інших.

Логіка – це наука про форми та закони правильного мислення. Предметом дослідження цієї науки є не вміст а форма мислення, тому ця наука також має назву *формальної логіки*.

Наприклад, висловлювання «всі люди є ссавцями» та «всі комп'ютери є електроприладами» різні за вмістом, але однакові за формою. Так само і висловлювання «якщо не підготуватися до екзамену, то екзамен можна не скласти» та «якщо визирне сонце, то калюжі висохнуть». Наведемо інший приклад, висловлювання «всі люди є ссавцями» та «якщо ти є людиною, то ти є ссавцем» мають різну форму але єдиний зміст.

З наведених прикладів можна зробити висновок, що мислення може мати нескінченні варіації вмісту, але певний набір форм. Саме ці форми мислення вивчає логіка або формальна логіка.

Форма мислення – це спосіб, яким людина висловлює свої думки, або схема, за якою ці думки будуються.

Існує три форми мислення:

1 Поняття – це форма мислення, яка позначає певний об'єкт або ознаку об'єкта (наприклад, екзамен, людина, радість, відвага).

Кожне поняття характеризується *вмістом* поняття – сукупність суттєвих ознак предмету (або класу однорідних предметів), та *обсягом* поняття – клас предметів, що узагальнені в цьому понятті. Чим більшим є обсяг поняття, тим вужчим є його вміст, та навпаки.

На рисунку 2.1 наведено класифікацію понять за вмістом.

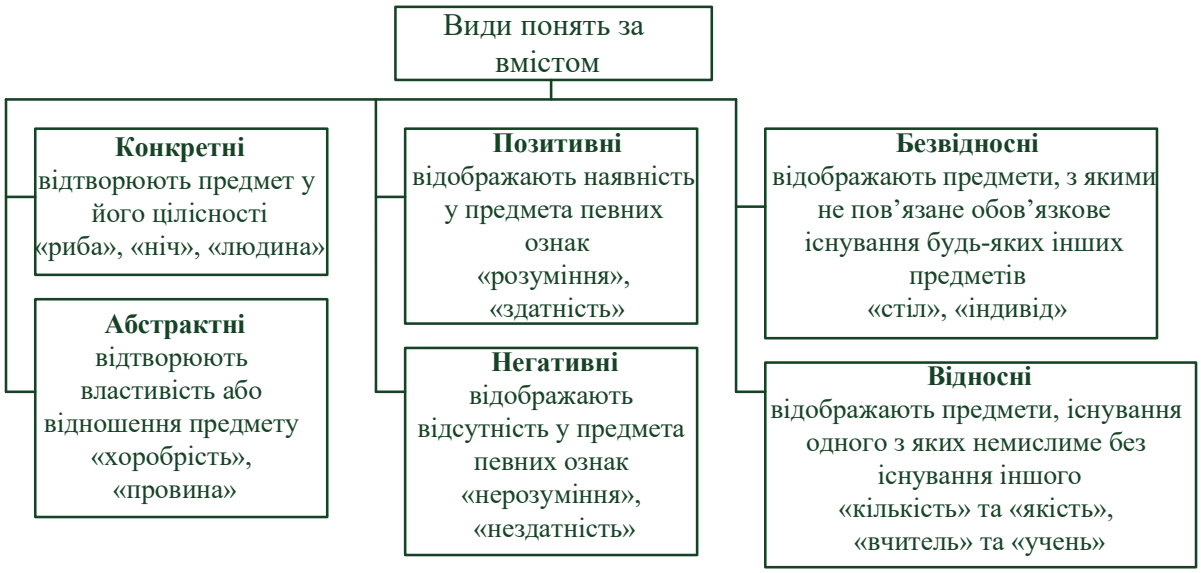


Рисунок 2.1 – Класифікація понять за вмістом

На рисунку 2.2 наведено класифікацію понять за обсягом.

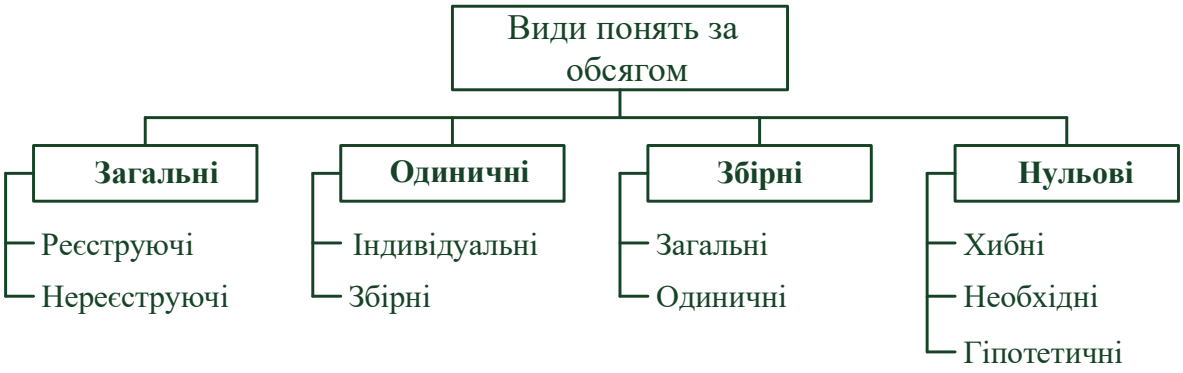


Рисунок 2.2 – Класифікація понять за обсягом

Загальні реєструючі поняття відображають обмежену кількість предметів (наприклад, «студент першого року навчання ХНУБА», «планета Сонячної системи»). Загальні реєструючі поняття мають скінченний обсяг.

Загальні нереєструючі поняття відображають ознаки невизначеної кількості предметів (наприклад, "людина", "закон", "студент"). Загальні нереєструючі поняття мають відносно нескінченний обсяг.

Одиничні індивідуальні поняття відображають один єдиний предмет (наприклад, «найстаріша людина у світі», «місто Харків»).

Одиничні збірні відображають ознаки однієї множини предметів (наприклад, «український народ», «італійський ресторан»).

Збірні загальні поняття відображають ознаки множини сукупності предметів (наприклад, «колектив», «планета», «баскетбольна збірна»).

Збірні одиничні відображають ознаки однієї єдиної множини предметів (наприклад, «колектив ХУБА», «планета Сонячної системи», «баскетбольна збірна України»).

Нульові хибні відображають ознаки міфічних істот (наприклад, «мавка», «чебурашка»).

Нульові необхідні відображають наукові абстракції (наприклад, «абсолютний нуль», «абсолютно тверде тіло»).

Нульові гіпотетичні відображають ознаки предметів, які ще недостатньо вивчені сучасною наукою (наприклад «паралельний світ», «снігова людина»).

Визначення (діфініція, від латинського definitio – межа) поняття – це логічна операція, яка розкриває вміст поняття або встановлює значення терміну. Під час операції визначення певна множина предметів, які відображаються даним поняттям, відділяються від інших предметів.

2 Судження – це форма мислення, яка складається з понять, які пов’язані між собою, вона або щось стверджує або щось заперечує (наприклад, «всі планети Сонячної системи обертаються навколо Сонця», «жодна рептилія не є ссавцем»).

Суб’єкт судження *S* – поняття про предмет судження (визначає про що йдеться у судженні та є аналогом підмета).

Предикат судження *P* – поняття про ознаку предмету судження (визначає що саме йдеться у судженні та є аналогом присудка).

Судження поділяються на:

– прості – це судження, до складу яких входить один суб’єкт та один предикат (існують загальні, частинні та одиничні прості судження);

– складні – це судження, що складаються з кількох простих суджень, пов'язаних між собою логічними зв’язками (існують умовні, еквівалентні, єднальні та розподільні складні судження).

Існують категоричні судження – це судження, що мають зв’язку «є» (наприклад, «всі ящірки є рептиліями»).

Виокремлюють чотири види суджень (рис. 2.3):

1 загальностверджувальні позначаються літерою А (наприклад, «всі студенти є людьми»);

2 частковостверджувльні позначаються літерою І (наприклад, «деякі студенти є спортсменами»);

3 загальнозаперечувальні позначаються літерою Е (наприклад, «всі студенти не є прибульцями»);

4 частковозаперечувальні позначаються літерою О (наприклад, «деякі студенти не є відмінниками»).

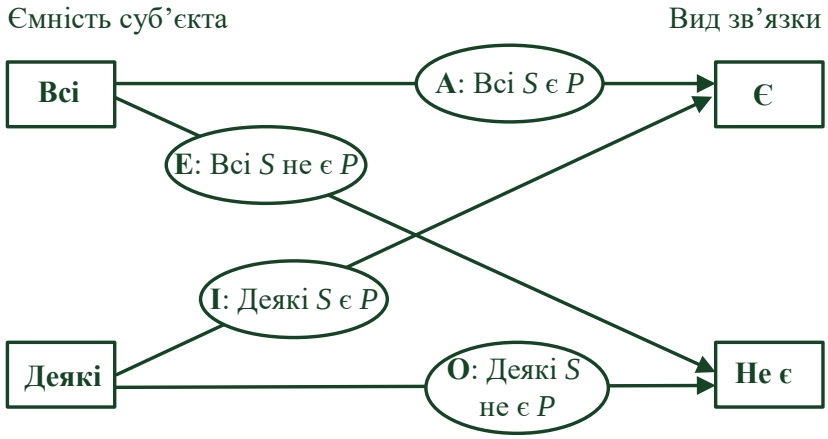


Рисунок 2.3 – Види суджень

В судженнях терміни S та P можуть бути розподілені (якщо термін судження повністю включається до обсягу іншого терміна або повністю виключається з нього) або нерозподілені (якщо термін судження частково включається до обсягу іншого терміна або частково виключається з нього).

Для загальностверджувальних суджень є два випадки розподілення:

1 «всі люди є ссавцями»: суб'єкт є розподіленим, предикат є нерозподіленим (рис. 2.4 а));

2 «всі квадрати є рівносторонніми прямокутниками»: суб'єкт є розподіленим, предикат є розподіленим (рис. 2.4 б)).

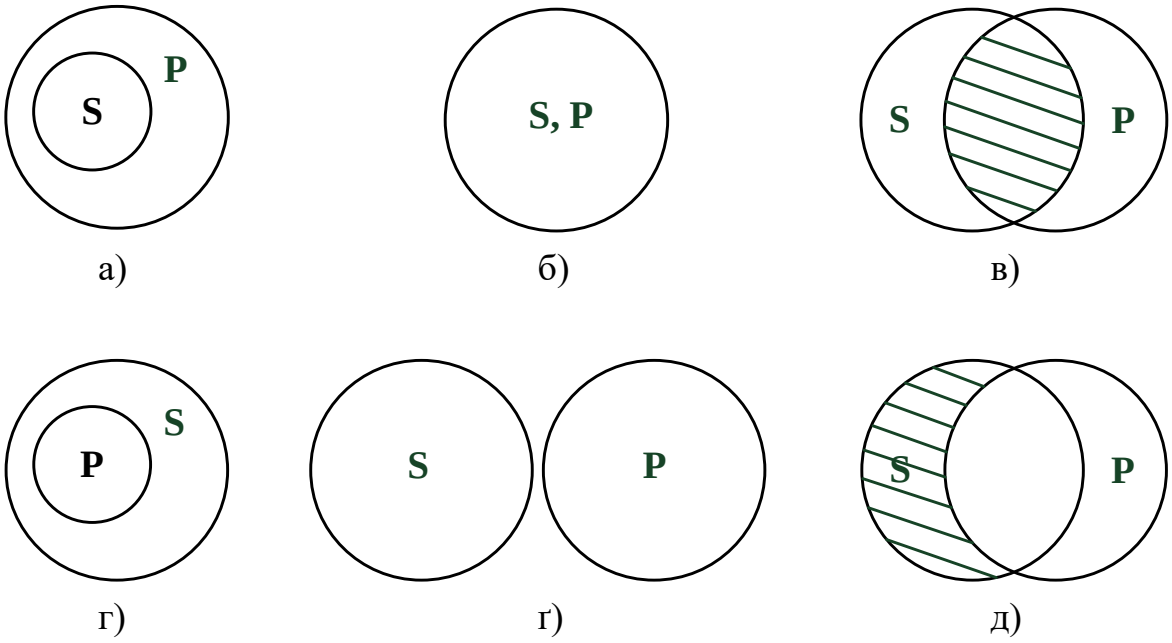
Для частковостверджувльних суджень є два випадки розподілення:

1 «деякі студенти є спортсменами»: суб'єкт є не розподіленим, предикат є не розподіленим (рис. 2.4 в));

2 «деякі письменники є драматургами»: суб'єкт є не розподіленим відносно предиката, предикат є розподіленим відносно суб'єкта (рис. 2.4 г)).

Наведемо приклад розподілення у загальнозаперечувальних судженнях: «жодний студент не є рептилією» (рис. 2.4 г)).

Наведемо приклад розподілення у частковозаперечувальних судженнях: «деякі студенти не є спортсменами» (рис. 2.4 д)).



а) б) в)

г) ґ) д)

а – розподілення у загальностверджувальних судженнях з нерозподіленим предикатом; б – розподілення у загальностверджувальних судженнях з розподіленим предикатом; в – розподілення у частковостверджувльних судженнях з нерозподіленим предикатом; г – розподілення у частковостверджувльних судженнях з розподіленим предикатом; ґ – розподілення у загальнозаперечувальних судженнях; д – розподілення у частковозаперечувальних судженнях

Рисунок 2.4 – Види розподілення термінів

Судження бувають порівнянні (судження, що мають однаковий суб’єкт або предикат) та непорівнянні (судження, що не мають спільних складових).

Порівнянні судження поділяються на сумісні та несумісні.

Сумісні судження відображають одну й туж саму думку повністю або лише в деякій частині. Сумісні судження можуть бути одночасно істинними.

Існує три види відносин сумісності:

1 еквівалентність (судження є одночасно істинними або одночасно хибними);

2 часткова сумісність (судження є одночасно істинними, але не є одночасно хибними);

3 логічне підпорядкування (за істинності судження, що підпорядковує, підлегле судження завжди буде істинним).

Існує два види відносин несумісності:

1 протилежність (судження не можуть бути одночасно істинними, але можуть бути одночасно хибними);

2 протиріччя (судження не можуть бути одночасно істинними та не можуть бути одночасно хибними).

3 Умовивід – це форма мислення, за якої з одного або кількох початкових суджень випливає нове судження або висновок, яке обов'язково або певним ступенем ймовірності випливає з цих суджень (наприклад, із суджень «всі ящірки це рептилії» та «жодна рептилія не є ссавцем» випливає нове судження «жодна ящірка не є ссавцем»).

Дедуктивний умовивід, в якому з двох істинних категоричних суджень, що пов'язані середнім терміном, за дотримання правил обов'язково випливає висновок – *категоричний силогізм*.

Два засновки (початкові судження) силогізму містять три терміни, що позначаються латинськими літерами: S – менший термін (менше за обсягом поняття, що відбиває суб'єкт висновку), P – більший термін (більше за обсягом, але вужче за змістом поняття, що відбиває предикат висновку), M – середній, зв'язуючий термін.

Силогізм можна подати у наступному вигляді:

Всі **M** є **P**	більший засновок	Всі метали є електропровідниками
Певне **S** є **M**	менший засновок	Мідь є металом
Певне **S** є **P**	висновок	Мідь є електропровідником

В наведеному прикладі поняття «мідь» є меншим терміном, яке є меншим за обсягом за поняття «електропровідники», що є більшим терміном даного силогізму, поняття «метал» є середнім терміном, що пов'язує більший та менший терміни.

За розташуванням середнього терміну виокремлюють чотири види силогізмів, які ще мають назву фігур силогізмів.

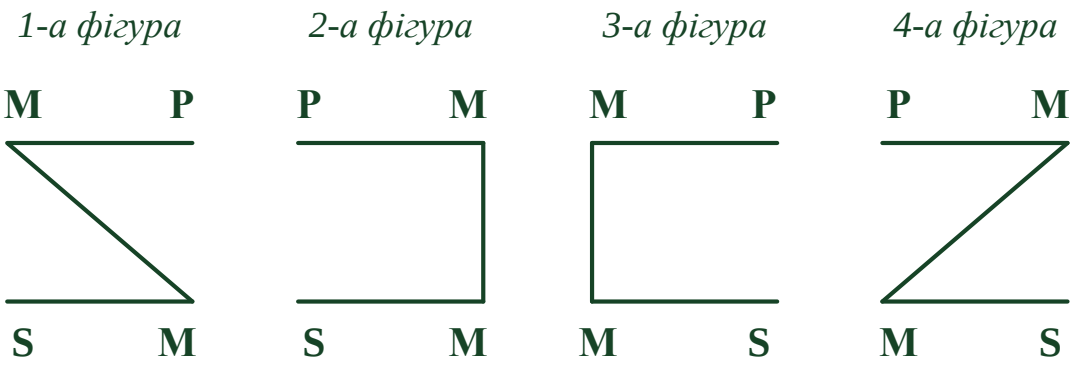


Рисунок 2.5 – Фігури силогізмів

Наведемо приклади силогізмів для наведених фігур:

1-а фігура: «Всі **M** люди повинні **P** правильно мислити», «Всі **S** студенти є **M** людьми», ∴ «Всі **S** студенти повинні **P** правило мислити»;

2-а фігура: «Всі **P** студенти повинні **M** правильно мислити», «**S** Кіт Василій не **M** мислить правильно», ∴ «**S** Кіт Василій не є **P** студентом»;

3-а фігура: «Всі **M** студенти є **P** розумними», «Всі **M** студенти є **S** людьми», ∴ «Всі **S** люди є **P** розумними»;

4-а фігура: «Деякі **P** студенти **M** отримують стипендію», «Всі **M**, хто отримують стипендію **S** добре навчаються», ∴ «Дехто **S**, хто добре навчається є **P** студентом».

Для запобігання помилок під час побудови категоричних силогізмів слід дотримуватися правил фігур, термінів та засновків.

Загальні правила фігур:

– більший засновок повинен бути загальним, менший – стверджувальним;

– більший засновок повинен бути загальним, менший засновок то висновок – заперечувальними;

– менший засновок повинен бути стверджувальним, висновок – частинним;

– якщо більший засновок є стверджувальним, та менший повинен бути загальним;

– якщо один з засновків є заперечувальним, то більший засновок повинен бути загальним.

Правила термінів:

– має бути лише три терміни;

– середній термін М повинен бути розподілений хоча б в одному засновку;

– термін, який є розподілений в засновку не може бути розподілений у висновку.

Правила засновків:

– з двох заперечувальних засновків не можна зробити жодного висновку;

– якщо один з засновків є заперечувальним, то й висновок повинен бути заеречувальним;

– з двох частинних засновків не можна зробити жодного висновку;

– якщо один з засновків є частинним, то й висновок має бути частинним.

В продукційних моделях представлення знань виведення базується на правилі *Modus ponens* (гіпотетичний силогізм), за якого з умовного судження $A \rightarrow B$ та його основи А (антецеденти) випливає висновок В (консеквента), наприклад «Всі метали є електропровідниками», «Мідь є металом», ∴ «Мідь є електропровідником».

Умовне судження – це судження, в якому відбивається залежність того або іншого явища від певних умов. Умова судження складається з основи (антецеденти) та висновку (консеквенти), тобто маємо форму: «ЯКЩО *антецедента*, ТО *консеквента*». Антецедента дає знання про той член відношення, від існування якого залежить існування іншого члена відношення. Зв'язка між членами свідчить про наявність даного відношення між членами. Висновок дає знання про інші члени відношення, тобто маємо форму «ЯКЩО *A* є *B*, ТО *C* є *D*».

Крім форм мислення предметом логіки є також *закони мислення* – це правила, дотримання яких забезпечує істинність висновків, що були виведенні з певних суджень, незалежно від їх вмісту, а також запобігає хибним висновкам, за умови істинності початкових суджень.

Таких законів існує чотири: закон тотожності, закон протиріччя, закон виключного третього, закон достатньої підстави. У випадку недотримання

цих законів виникають логічні помилки (як правило це хибні висновки), якщо закони порушено несвідомо через незнання, то помилки, що при цьому виникають є *паралогізмами*, якщо свідомо, з метою заплутати опонента, то помилки, які при цьому виникають є *софізмами*.

Логіка, як наука з'явилася приблизно в п'ятому сторіччі до нашої ери в Стародавній Греції. Її творцем вважається давньогрецький філософ і вчений Аристотель (384-322 рр. до н. е.). Логіка створена Аристотелем ще носить назву традиційної логіки.

В XIX сторіччі стрімкого розвитку набула сучасна логіка, яка носить назву символічної або математичної логіки. В основі математичної логіки лежать ідей повного переходу до ідеальної, повністю незалежної від вмісту висловлення логічної форми, за допомогою універсальної символьної мови, подібної до мови числення. Ці ідеї запропонував німецький математик та філософ Готфрід Лейбніц ще в XVII сторіччі.

Майже сторіччям пізніше ірландський логік та математик Джордж Буль розглядав умовиведення як результат розв'язання логічних рівнянь, завдяки чому теорія умовиведення набула виду подібного до числення, за винятком наявності чисельних коефіцієнтів та ступенів.

Таким чином головна відмінність традиційної логіки від математичної полягає в тому, що перша для опису правильного мислення використовує природню мову, а друга для того ж правильного мислення використовує специфічну символьну формалізовану мову, яка має назву числення або *числення висловлювань*.

Існує дві мови математичної логіки: числення висловлювань та логіка предикатів.

2.2 Числення висловлювань

Числення висловлювань (або пропозиціональна логіка) — це формальна теорія, основним об'єктом якої є поняття логічного висловлювання. Така логіка є найпростішою і максимально близькою до людського способу неформальних міркувань. Тому її називають логікою нульового порядку.

Висловлювання – це мовний вираз, що має сенс та відносно якого можна стверджувати, що воно є ІСТИННИМ або ХИБНИМ. Таким чином кожному висловлюванню можна приписати *істиностне значення* або І (істина ще позначається 1) або Х (хибно ще позначається 0).

Приклад 2.1 Висловлювання «5 є простим числом» та «8 є парним числом» мають істиностне значення І (або 1), а висловлювання «3 є дільником 7» та «3+5=9» мають істиностне значення Х (або 0). Рівняння «2+x=4» не є висловлюванням, проте надаючи змінній x певні числові значення ми отримуємо різні висловлювання, що можуть мати істиностні значення І або Х.

Символи A, B, C, P, Q, R тощо, які використовуються для позначення висловлювань – *атомарні формули* (атоми) або *пропозиційні змінні*.

Літерал – атомарна формула або її заперечення.

Використовуючи частку заперечення «ні», а також сполучники «та», «або», «якщо …, то…», «тоді і тільки тоді, коли» і тому подібні, можна з одних висловлювань будувати інші, нові висловлювання. Істиностні значення нових висловлювань визначаються істиностними значеннями висловлювань, які входять до їх скаду.

Побудова з наданих висловлювань (або висловлювання) нового висловлювання – *логічна операція*.

Знаки логічних операцій – *логічні зв'язування* або *зв'язування*.

В численні висловлювань логічні операції найчастіше описуються за допомогою *таблиць істинності*. Таблиця істинності для одномісної логічної операції «заперечення» або «інверсія», яка відповідає зв'язуванню «ні» наведено в таблиці 2.1. Заперечення висловлювання A (тобто *ні* A) позначається $\neg A$ або $\sim A$ або $\overline{A}$ та читається як «заперечення A» або «ні A».

Таблиця 2.1 – Таблиця істинності для операції «заперечення»

A	$\neg A$
0	1
1	0

Основні двомісні логічні операції та логічні зв'язування наведено в таблиці 2.2.

Таблиця 2.2 – Основні двомісні логічні операції

Позначення логічної операції	Інші можливі позначення логічної операції	Набір істиносних значень, що відповідають логічній операції	Назва логічної операції та зв'язування	Як читається вираз, наведений в першому стовпці
$A_1 \wedge A_2$	$A_1 \& A_2$ $A_1 \cdot A_2$ $A_1 A_2$ $\min(A_1, A_2)$	0001	кон'юнкція, логічне множення, логічне «та»	A_1 та A_2
$A_1 \vee A_2$	$A_1 + A_2$ $\max(A_1, A_2)$	0111	диз'юнкція, логічне додавання, логічне «або»	A_1 або A_2
$A_1 \rightarrow A_2$	$A_1 \supset A_2$ $A_1 \Rightarrow A_2$	1101	імплікація, логічне слідування	якщо A_1, то A_2 A_1 імпліцирує A_2 з A_1 випливає A_2
$A_1 \leftrightarrow A_2$	$A_1 \equiv A_2$ $A_1 \sim A_2$ $A_1 \Leftrightarrow A_2$	1001	еквіваленція, еквівалентність, рівнозначність, тотожність	A_1 тоді і тільки тоді, коли A_2 A_1 еквівалентно A_2
$A_1 \oplus A_2$	$A_1 + A_2$ $A_1 \dot{\vee} A_2$ $A_1 \Delta A_2$	0110	сума за модулем 2, розділова диз'юнкція, розділове «або»	A_1 плюс A_2 або A_1 або A_2
$A_1 ! A_2$		1110	штрих Шеффера, антикон'юнкція	A_1 штрих Шеффера A_2 невірно, що A_1 та A_2
$A_1 \downarrow A_2$	$A_1 \circ A_2$ $A_1 \bar{\vee} A_2$	1000	стрілка Пірса, антидіз'юнкція, функція Вебба, функція Даггера	ані A_1 ані A_2 A_1 стрілка Пірса A_2

У численні висловлювань вираз, який є висловлюванням, наприклад, P або складене висловлювання $(\neg P \vee Q) \wedge (\neg (P \wedge \neg Q))$, називається *правильно побудованою формулою* або *логічною* (*пропозиційною*) *формулою*. Правильно побудовані формули (або формули) в численні висловлювань визначаються рекурсивно наступним чином:

1 Атом – це формула.

2 Якщо G – це формула, то $(\neg G)$ – також формула.

3 Якщо G і H – це формули, то $(G \vee H)$, $(G \wedge H)$, $(G \to H)$ та $(G \leftrightarrow H)$ – також формули.

4 Жодних формул, крім визначених застосуванням зазначених вище правил, не існує.

Виходячи з цього, такі вирази як $(G \to)$ та $(G \wedge)$ не є формулами.

Наведемо істиностні значення та таблиці істинності висловлювань, які знадобляться надалі.

Нехай G та H – дві формули, тоді:

1 Істіностне значення логічної операції *заперечення* $(\neg G)$ є І (або 1), якщо G є хибною, і є Х (або 0), якщо G є істинною. Таблиця істинності для цієї логічної операції наведена у таблиці 1.1.

2 Істіностне значення логічної операції *кон'юнкції* $(G \wedge H)$ є І (або 1), якщо G та H обидві є істинними, в протилежному випадку є Х (або 0). Таблиця істинності для цієї логічної операції наведена у таблиці 2.3.

Таблиця 2.3 – Таблиця істинності логічної операції кон'юнкції

G	H	$(G \wedge H)$
0	0	0
0	1	0
1	0	0
1	1	1

3 Істіностне значення логічної операції *диз'юнкції* $(G \vee H)$ є І (або 1), якщо хоча б одна з двох формул G або H є істинною, в протилежному випадку є Х (або 0). Таблиця істинності для цієї логічної операції наведена у таблиці 2.4.

Таблиця 2.4 – Таблиця істинності логічної операції диз'юнкції

G	H	$(G \vee H)$
0	0	0
0	1	1
1	0	1
1	1	1

4 Істіностне значення логічної операції *імплікації* $(G \rightarrow H)$ є X (або 0), якщо формула G є істинною, а формула H є хибною, інакше є I (або 1). Таблиця істинності для цієї логічної операції наведена у таблиці 2.5.

Таблиця 2.5 – Таблиця істинності логічної операції імплікації

G	H	$(G \rightarrow H)$
0	0	1
0	1	1
1	0	0
1	1	1

5 Істіностне значення логічної операції *еквівалентності* $(G \leftrightarrow H)$ є I (або 1), тоді і тільки тоді, коли формули G та H мають однакові істиностні значення, інакше є X (або 0). Таблиця істинності для цієї логічної операції наведена у таблиці 2.6.

Таблиця 2.6 – Таблиця істинності логічної операції еквівалентності

G	H	$(G \leftrightarrow H)$
0	0	1
0	1	0
1	0	0
1	1	1

Практична робота 1. Побудова таблиці істинності

Побудувати таблицю істинності будь-якого висловлювання можна за наступним алгоритмом:

Крок 1. Розрахувати кількість рядків таблиці за формулою 2^n+1, де n – кількість логічних змінних у висловлюванні (один рядок додаємо для шапки таблиці).

Крок 2. Розрахувати кількість стовпчиків таблиці за формулою $n+m$, де m – кількість логічних операції у висловлюванні.

Крок 3. Побудувати таблицю з визначеною кількістю рядків та стовпчиків.

Крок 4. Заповнити таблицю:

4 а) виписати набори істинностних значень вхідних змінних з урахуванням того, що їх набір є рядом n-розрядних двійкових чисел від 0 до 2^n-1, для чого потрібно:

– розділити перший стовпчик таблиці на дві половини, першу заповнити значенням 0, а другу – значенням 1;

– розділити другий стовпчик таблиці на чверті, першу заповнити значенням 0, а другу – значенням 1, третю – 0, четверту 1;

– розділити третій стовпчик таблиці на 8 частин, заповнити їх по-черзі 0 або 1, починаючи з 0…

– розділяти та заповнювати стовпчики таблиці, поки кожна з частин не скрадатиметься з однієї позиції;

4 б) виконати логічні операції з урахуванням пріоритетів.

Пріоритет застосування логічних операцій зменшується в наступному порядку: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$.

Задача 2.1 Побудувати таблицю істинності висловлювання $(P\wedge Q)\rightarrow(R\leftrightarrow(\neg S))$.

Розв'язання.

Крок 1. Кількість рядків таблиці $2^4+1=17$.

Крок 2. Кількість стовпчиків $4+4=8$.

Крок 3, 4. Проставимо номера логічних операцій згідно порядку їх виконання за пріоритетом: $(P\wedge^3 Q)\ \rightarrow^4(R\leftrightarrow^2(\neg^1 S))$. Таблицю істинності даного висловлювання наведено в таблиці 2.7. Останні чотири стовпчика заповнюються згідно з таблицями істинності, наведеними в таблицях 2.1 та 2.3-2.6.

Таблиця 2.7 – Таблиця істинності висловлювання $(P \wedge Q) \rightarrow (R \leftrightarrow (\neg S))$

P	Q	R	S	$\neg S$	$R \leftrightarrow (\neg S)$	$P \wedge Q$	$(P \wedge Q) \rightarrow (R \leftrightarrow (\neg S))$
0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	1
0	0	1	0	1	1	0	1
0	0	1	1	0	0	0	1
0	1	0	0	1	0	0	1
0	1	0	1	0	1	0	1
0	1	1	0	1	1	0	1
0	1	1	1	0	0	0	1
1	0	0	0	1	0	0	1
1	0	0	1	0	1	0	1
1	0	1	0	1	1	0	1
1	0	1	1	0	0	0	1
1	1	0	0	1	0	1	0
1	1	0	1	0	1	1	1
1	1	1	0	1	1	1	1
1	1	1	1	0	0	1	0

2.3 Інтерпретація формул у численні висловлювань

Припустимо надана пропозиційна формула G, атомами якої є A_1, A_2, …, A_n. Приписування істинностних значень атомам A_1, A_2, …, A_n , за якого кожному A_i приписано або І (тобто 1) або Х (тобто 0) – *інтерпретація формули G*.

Формула G є *істинною* за певної інтерпретації тоді і тільки тоді, коли G набуває істинностного значення І (тобто 1), інакше G є хибною за цієї інтерпретації.

Формула, яка є істинною за всіх інтерпретацій, називається *загальнозначимою формулою* або *тавтологією*.

Формула, яка є хибною за всіх інтерпретацій, називається *суперечливою формулою* або *протиріччям*.

Дві формули є *рівносильними* (логічно еквівалентними) тоді і тільки тоді, коли істинностні значення формул збігаються. Рівносильність формул позначається знаком ≡ (наприклад $A_1 \equiv A_2$).

2.3.1 Нормальні форми у численні висловлювань

Основна задача числення висловлювань – визначити чи є формула загальнозначимою або суперечливою або такою, що може бути виконана. Розв'язок цієї задачі можна знайти шляхом перетворення формулу в одну з нормальних форм.

Формула G знаходиться в *кон'юнктивній нормальній формі* (КНФ) тоді і тільки тоді, коли G має вигляд

$$G_1 \wedge G_2 \wedge \ldots \wedge G_n, \quad n \geq 1,$$

де $G_1, G_2 \ldots G_n$ – диз'юнкція літер, тобто елементарна диз'юнкція.

Нагадаємо, що *літера* – це атом або заперечення атома.

Приклад 2.2 Наведемо приклад КНФ формули

$G = (G_1 \vee \neg G_2 \vee G_3) \wedge (G_1 \vee \neg G_2 \vee \neg G_3) \wedge (\neg G_1 \vee G_2 \vee G_3) \wedge (\neg G_1 \vee \neg G_2 \vee \neg G_3)$.

Формула G знаходиться в *диз'юнктивній нормальній формі* (ДНФ) тоді і тільки тоді, коли G має вигляд

$$G_1 \vee G_2 \vee \ldots \vee G_n, \quad n \geq 1,$$

де $G_1, G_2 \ldots G_n$ – кон'юнкція літер, тобто елементарна кон'юнкція.

Приклад 2.3 Наведемо приклад ДНФ формули

$G \equiv (\neg G_1 \wedge \neg G_2 \wedge G_3) \vee (\neg G_1 \wedge \neg G_2 \wedge G_3) \vee (G_1 \wedge G_2 \wedge \neg G_3)$.

Теорема 1 Логічна формула є тавтологією тоді і тільки тоді, коли кожна елементарна диз'юнкція, що входить до КНФ цієї формули, містить хоча б одну пропозиційну змінну разом з її запереченням.

Теорема 2 Логічна формула є протиріччям тоді і тільки тоді, коли кожна елементарна кон'юнкція, що входить до ДНФ цієї формули, містить хоча б одну пропозиційну змінну разом з її запереченням.

Для перетворення формули до нормальної форми слід скористатися основними еквівалентностями числення висловлювань.

Припустимо є пропозиційні формули H_1, H_2, H_3, наведемо *основні еквівалентності:*

1 $\neg(\neg H_1) \equiv H_1$ – правило зняття подвійного заперечення;

2 $H_1 \wedge H_1 \equiv H_1$ – ідемпотентність кон'юнкції (ідемпотентність (від лат. *idem* – той самий та *potens* – здатний) – властивість об'єкта або операції під час повторного застосування операції до об'єкта надавати той самий результат, що й перший раз застосування);

3 $H_1 \vee H_1 \equiv H_1$ – ідемпотентність диз'юнкції;

4 $H_1 \wedge H_2 \equiv H_2 \wedge H_1$ – комутативність кон'юнкції (комутативність (від лат. *commutativus* – той що змінюється) – властивість бінарних операцій, яка полягає у можливості заміни місцями операндів зі збереженням результату операції);

5 $H_1 \vee H_2 \equiv H_2 \vee H_1$ – комутативність диз'юнкції;

6 $H_1 \leftrightarrow H_2 \equiv H_2 \leftrightarrow H_1$ – комутативність еквівалентності;

7 $(H_1 \wedge H_2) \wedge H_3 \equiv H_1 \wedge (H_2 \wedge H_3)$ – асоціативність кон'юнкції (асоціативність (від лат. *associatio* – з'єднання) – властивість бінарних операцій, яка полягає у можливості перестановки дужок зі збереженням результату виразу);

8 $(H_1 \vee H_2) \vee H_3 \equiv H_1 \vee (H_2 \vee H_3)$ – асоціативність диз'юнкції;

9 $(H_1 \leftrightarrow H_2) \leftrightarrow H_3 \equiv H_1 \leftrightarrow (H_2 \leftrightarrow H_3)$ – асоціативність еквівалентності;

10 $H_1 \wedge (H_2 \vee H_3) \equiv (H_1 \wedge H_2) \vee (H_1 \wedge H_3)$ – дистрибутивність кон'юнкції відносно диз'юнкції (дистрибутивність (від лат. *distributivus* – розподілений) – властивість узгодженості бінарних операцій);

11 $H_1 \vee (H_2 \wedge H_3) \equiv (H_1 \vee H_2) \wedge (H_1 \vee H_3)$ – дистрибутивність диз'юнкції відносно кон'юнкції;

12 $\neg(H_1 \wedge H_2) \equiv \neg H_1 \wedge \neg H_2$,
$\neg(H_1 \vee H_2) \equiv \neg H_1 \vee \neg H_2$ – правила де Моргана;

13 $H_1 \vee (H_1 \wedge H_2) \equiv H_1$,
$H_1 \wedge (H_1 \vee H_2) \equiv H_1$ – правила поглинання;

14 $H_1 \vee (\neg H_1 \wedge H_2) \equiv H_1 \vee H_2$,
$H_1 \wedge (\neg H_1 \vee H_2) \equiv H_1 \wedge H_2$;

15 $H_1 \rightarrow H_2 \equiv \neg H_1 \vee H_2$;

16 $H_1 \leftrightarrow H_2 \equiv H_1 H_2 \vee \neg H_1 \neg H_2$;

17 $H_1 \wedge \neg H_1 \equiv 0$ – закон протиріччя;

18 $H_1 \vee \neg H_1 \equiv 1$ – закон виключного третього;

19 $H_1 \wedge \mathrm{I} \equiv H_1$;

20 $H_1 \vee \mathrm{I} \equiv \mathrm{I}$;

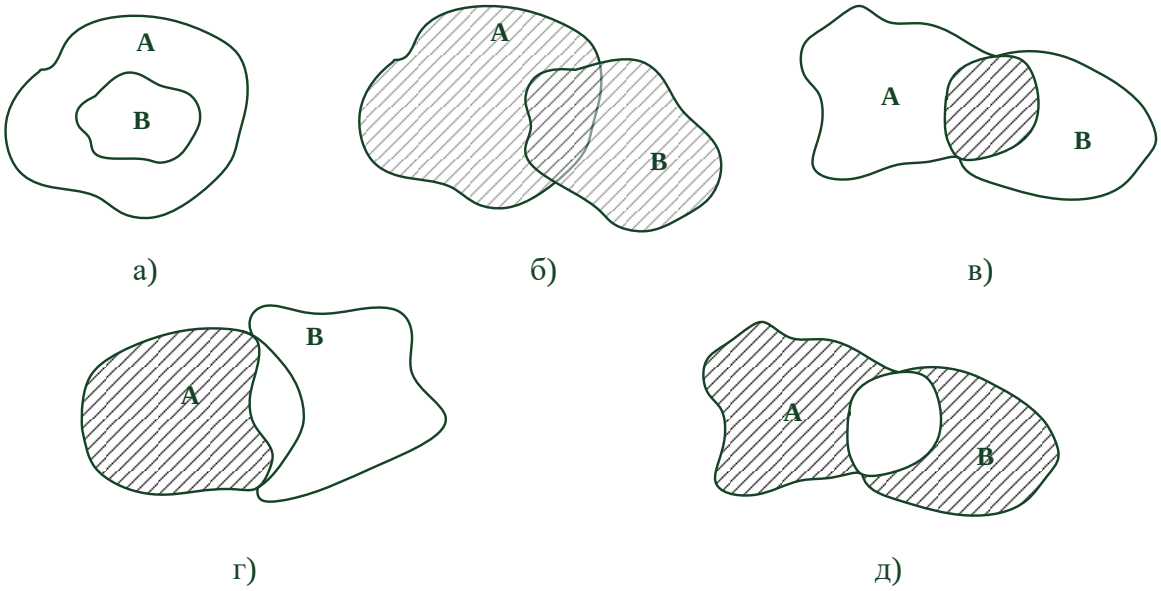
21 $H_1 \wedge \mathrm{X} \equiv \mathrm{X}$;

22 $H_1 \vee \mathrm{X} \equiv H_1$.

Зауваження. Наведена в останніх еквівалентностях літера I (відповідно X) позначає таку пропозиційну формулу, функція істинності якої рівнозначна відповідно 1 або 0.

2.3.2 Інтерпретація логічних операцій діаграмами Ейлера-Венна

Зазвичай діаграми Ейлера-Венна використовуються для пояснення властивовстей операції з множинами. Так, діаграма, наведена на рисунку 2.6 ілюструє відношення включення, перетину, різниці та симетричної різниці.



а) відношення включення $B \subseteq A$ (множина B є підмножиною множини A), б) об'єднання $A \cup B$, в) перетину $A \cap B$, г) різниці $A \setminus B$, д) симетричної різниці $A \Delta B$

Рисунок 2.6 – Діаграми Ейлера-Венна

Припустимо, що A – це множина точок, в яких висловлювання A набуває істинностного значення I (або 1), тоді деякі з наведених основних

логічних операцій можна пояснити за допомогою діаграм Ейлера-Венна графічно, як це показано на рисунку 2.7.

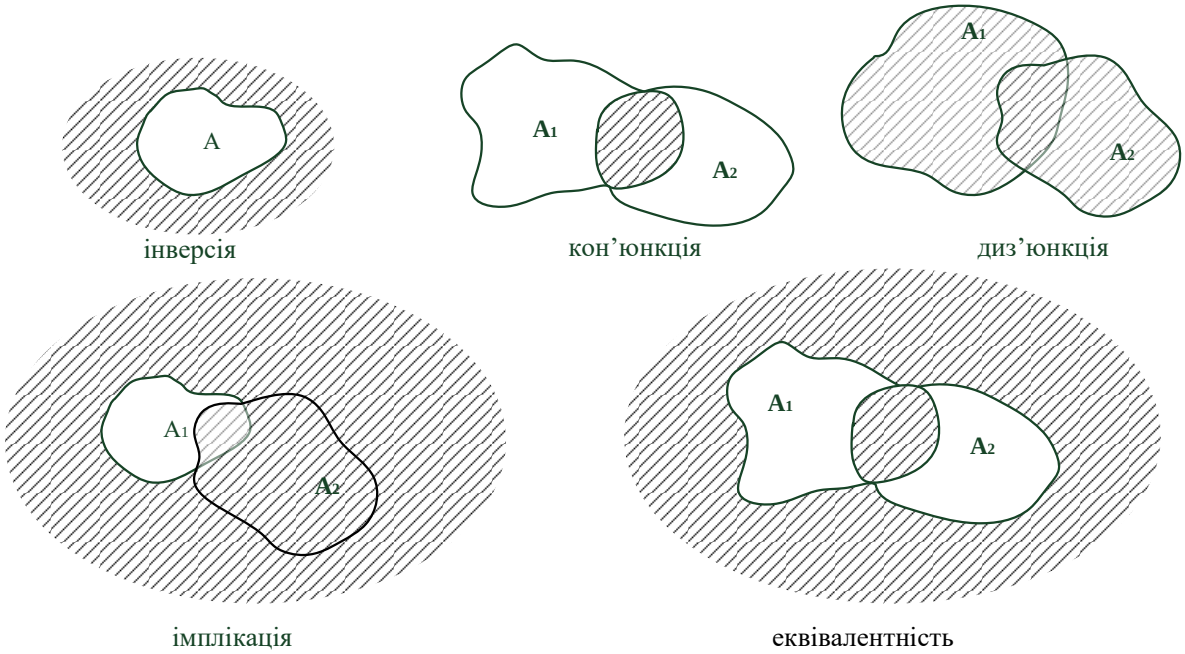


Рисунок 2.7 – Діаграми Ейлера-Венна для інтерпретації основних логічних операцій

Якщо інтерпретувати пропозиційні формули H_1, H_2, H_3, як множини, тобто множина H_1 – множина точок, в яких пропозиційна формула H_1, набуває істинностного значення 1, то, базуючись на діаграмах інтерпретації основних логічних операці, легко надати графічні пояснення деяким еквівалентностям (рисунок 2.8 та 2.9).

$$H_1 \vee (H_2 \wedge H_3) \equiv (H_1 \vee H_2) \wedge (H_1 \vee H_3)$$

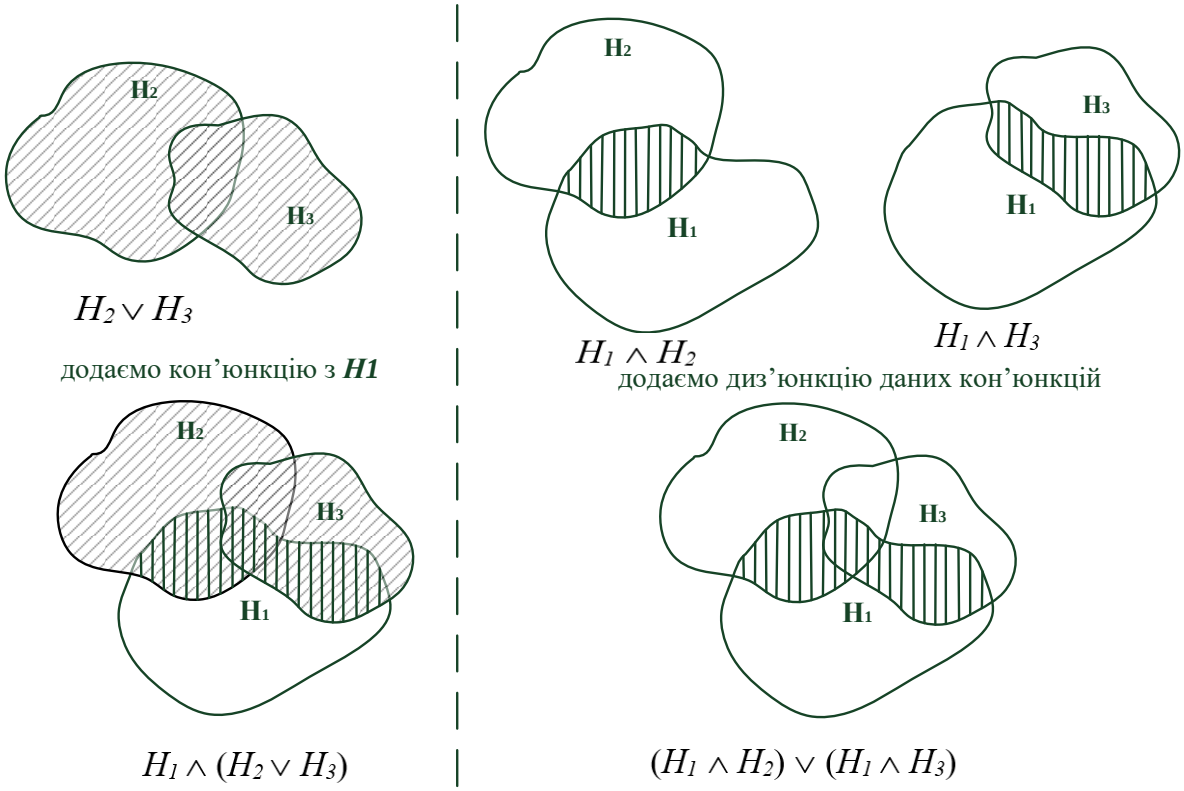


Рисунок 2.8 – Діаграми Ейлера-Венна для інтерпретації еквівалентності $H_1 \wedge (H_2 \vee H_3) \equiv (H_1 \wedge H_2) \vee (H_1 \wedge H_3)$

$$H_1 \leftrightarrow H_2 \equiv H_1 H_2 \vee \neg H_1 \neg H_2$$

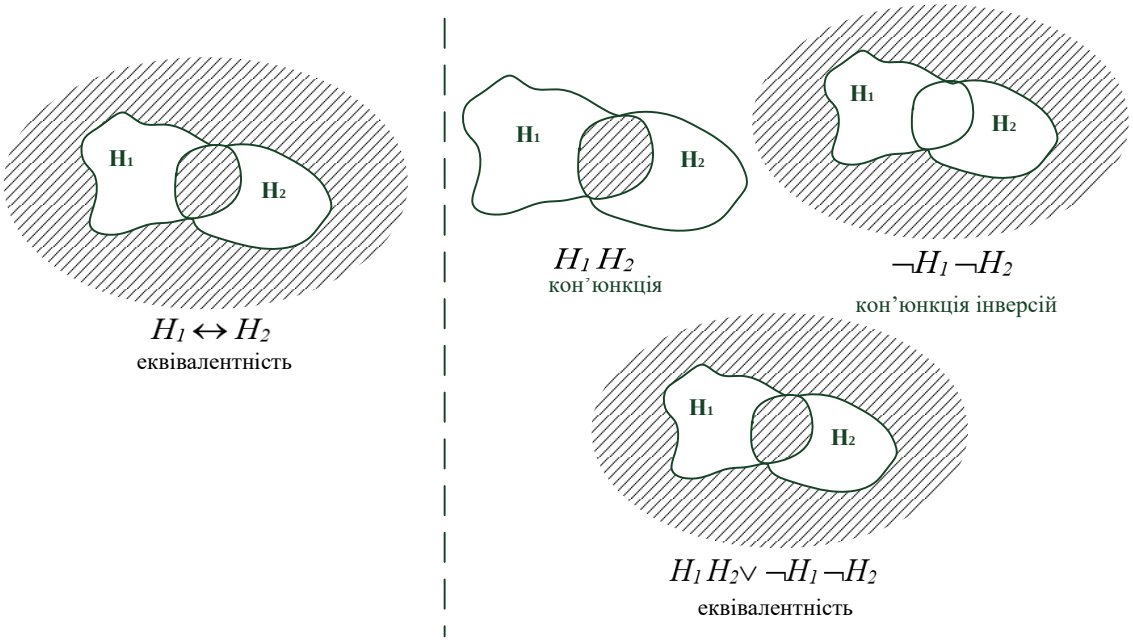


Рисунок 2.9 – Діаграми Ейлера-Венна для інтерпретації еквівалентності $H_1 \leftrightarrow H_2 \equiv H_1 H_2 \vee \neg H_1 \neg H_2$

2.3.3 Процедура перетворення формули в нормальну форму

Наведемо алгоритм процедури перетворення формули в нормальну форму (НФ).

Крок 1. Для усунення логічних зв'язків імлікації $\rightarrow$ та еквівалентності $\leftrightarrow$, використовуються закони (15) та (16):

15 $H_1 \rightarrow H_2 \equiv \neg H_1 \vee H_2$;

16 $H_1 \leftrightarrow H_2 \equiv H_1 H_2 \vee \neg H_1 \neg H_2$;

Крок 2. Для перенесення знака заперечення безпосередньо до атомів декілька разів використовується правило зняття подвійного заперечення (1):

$\neg (\neg H_1) \equiv H_1$

та правила де Моргана (12):

$\neg (H_1 \wedge H_2) \equiv \neg H_1 \wedge \neg H_2$;

$\neg (H_1 \vee H_2) \equiv \neg H_1 \vee \neg H_2$.

Крок 3. Для отримання нормальної форми декілька разів використовуються дистрибутивні закони (10), (11):

$H_1 \wedge (H_2 \vee H_3) \equiv (H_1 \wedge H_2) \vee (H_1 \wedge H_3)$;

$H_1 \vee (H_2 \wedge H_3) \equiv (H_1 \vee H_2) \wedge (H_1 \vee H_3)$

Практична робота 2. Процедури перетворення формул в досконалу нормальну форму

Досконала диз'юнктивна нормальна форма (ДДНФ) формули – рівносильна їй формула, що є диз'юнкцією елементарних кон'юнкцій, яка має наступні властивості:

1) кожен логічний доданок формули містить всі змінні, які входять до початкової формули;
2) всі логічні доданки формули різні;
3) жоден логічний доданок не містить змінну та її заперечення;
4) жоден логічний доданок не містить одну і туж змінну двічі.

Досконала кон'юнктивна нормальна форма (ДКНФ) формули – рівносильна їй формула, що є кон'юнкцією елементарних диз'юнкцій, яка має наведені вище властивості.

Приведемо алгоритм побудови ДДНФ.

Етап I. Побудувати таблицю істинності початкової формули.

Етап II.

Крок 1. У векторі-стовпці істиностних значень формули (останній рядок таблиці істинності) обирати 1, якщо одиниці вичерпано процедура завершується.

Крок 2. За набором значень змінних обраного рядка таблиці істинності сформувати кон'юнкцію всіх аргументів за правилом: якщо *i*-а змінна дорівнює 0, то вона входить до кон'юнкції в ступені 0, тобто з інверсією, інакше в ступені 1.

Крок 3. Отриману кон'юнкцію додати до формули ДДНФ як наступний доданок та перейти до кроку1.

Задача 2.2 Побудувати ДДНФ формули $\neg Y \leftrightarrow X \wedge (\neg Z) \rightarrow Y \wedge Z$.

Розв'язання.

Виконання етапу I. Таблицю істинності початкової формули наведено у таблиці 2.8.

Таблиця 2.8 – Таблиця істинності формули $\neg Y \leftrightarrow X \wedge (\neg Z) \rightarrow Y \wedge Z$

X	Y	Z	$\neg Y$	$\neg Z$	$X \wedge (\neg Z)$	$Y \wedge Z$	$X \wedge (\neg Z) \rightarrow Y \wedge Z$	$\neg Y \leftrightarrow X \wedge (\neg Z) \rightarrow Y \wedge Z$
0	0	0	1	1	0	0	1	1
0	0	1	1	0	0	0	1	1
0	1	0	0	1	0	0	1	0
0	1	1	0	0	0	1	1	0
1	0	0	1	1	1	0	0	0
1	0	1	1	0	0	0	1	1
1	1	0	0	1	1	0	0	1
1	1	1	0	0	0	1	1	0

Виконання етапу II.

Виконання кроку *1*. У векторі-стовпці істиностних значень формули перше значення 1 (перший рядок).

Виконання кроку 2. Набір значень змінних першого рядка таблиці істинності: 0, 0, 0, тому перший доданок ДДНФ бути містити всі змінні з інверсією $\neg X \wedge \neg Y \wedge \neg Z$.

Виконання кроку 3. Отримали доданок $\neg X \wedge \neg Y \wedge \neg Z$ та перейшли до кроку1.

Виконання кроку 1. У векторі-стовпці істиностних значень формули друге значення 1 (другий рядок).

Виконання кроку 2. Набір значень змінних другого рядка таблиці істинності: 0, 0, 1, тому другий доданок ДДНФ бути містити всі змінні крім Z з інверсією $\neg X \wedge \neg Y \wedge Z$.

Виконання кроку 3. Отримали $(\neg X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z)$ та перейшли до кроку1.

Виконання кроку 1. Наступне значення 1 міститься у шостому рядку вектор-стовпці істиностних значень формули.

Виконання кроку 2. Набір значень змінних шостого рядка таблиці істинності: 1, 0, 1, тому третій доданок ДДНФ: $X \wedge \neg Y \wedge Z$.

Виконання кроку 3. Отримали $(\neg X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (X \wedge \neg Y \wedge Z)$ та перейшли до кроку1.

Виконання кроку 1. Наступне значення 1 міститься у сьомому рядку вектор-стовпці істиностних значень формули.

Виконання кроку 2. Набір значень змінних сьомого рядка таблиці істинності: 1, 1, 0, тому третій доданок ДДНФ: $X \wedge Y \wedge \neg Z$.

Виконання кроку 3. Отримали $(\neg X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (X \wedge Y \wedge \neg Z)$ та перейшли до кроку1.

Виконання кроку 1. Більше значень 1 вектор-стовпець істиностних значень формули не містить, тобто процедуру завершено. Шукана ДДНФ формули має вигляд:

$(\neg X \wedge \neg Y \wedge \neg Z) \vee (\neg X \wedge \neg Y \wedge Z) \vee (X \wedge \neg Y \wedge Z) \vee (X \wedge Y \wedge \neg Z)$.

Приведемо алгоритм побудови ДКНФ.

Eman I. Побудувати таблицю істинності початкової формули.

Eman II.

Крок 1. У векторі-стовпці істиностних значень формули (останній рядок таблиці істинності) обирати 0, якщо нулі вичерпано процедура завершується.

Крок 2. За набором значень змінних обраного рядка таблиці істинності сформувати диз'юнкцію всіх аргументів за правилом: якщо *i*-а змінна дорівнює 1, то вона входить до кон'юнкції в ступені 0, тобто з інверсією, інакше в ступені 1.

Крок 3. Отриману кон'юнкцію додати до формули ДКНФ як наступний доданок та перейти до кроку1.

Задача 2.3 Побудувати ДКНФ формули $\neg Y \leftrightarrow X \wedge (\neg Z) \rightarrow Y \wedge Z$.

Розв'язання. Процедура побудови ДКНФ подібна до процедури побудови ДДНФ, яка докладно розглянута у розв'язанні задачі 2.2, тому наведемо лише остаточну відповідь:

$(X \vee \neg Y \vee Z) \wedge (X \vee \neg Y \vee \neg Z) \wedge (\neg X \vee Y \vee Z) \wedge (\neg X \vee \neg Y \vee \neg Z)$.

Задачі для самостійного розв'язання

Задача 1 Побудувати таблицю істинності висловлювання $(P \wedge Q) \rightarrow (R \leftrightarrow (\neg S))$.

Задача 2 Визначити набори змінних, за яких висловлювання $A \wedge (B \vee (\neg B) \wedge (\neg C))$ буде істинним.

Задача 3 Визначити істинність висловлювання $(Z \wedge (\neg Q)) \rightarrow (Q \leftrightarrow R)$.

Задача 4 Побудувати таблицю істинності висловлювання $(A \vee B \rightarrow D) \vee (A \leftrightarrow (\neg B))$.

Задача 5 Визначити набори змінних, за яких висловлювання $(H \wedge (F \rightarrow G)) \rightarrow Q$ буде істинним.

Задача 6 Визначити істинність висловлювання $((A \rightarrow ((\neg B) \vee (C \wedge D))) \wedge A \wedge (\neg D)) \rightarrow (\neg B)$.

Задача 7 Довести тотожність $\neg(S \rightarrow R) \equiv S \wedge (\neg R)$, тобто рівносильність відповідних формул

Задача 8 Довести, що формула є тавтологією $((P \rightarrow Q) \wedge (\neg Q)) \rightarrow (\neg P))$.

Задача 9 Довести, що формула є протиріччям $(H \rightarrow F) \wedge (\neg F) \wedge H$.

Задача 10 Інтерпретувати основні еквівалентності (11)-(15) за допомогою діаграм Ейлера-Венна.

Задача 11 Побудувати ДДНФ та ДКНФ формули $(P \wedge Q) \rightarrow (R \leftrightarrow (\neg S))$.

Задача 12 Побудувати ДДНФ та ДКНФ формули $(H \wedge (F \rightarrow G)) \rightarrow Q$.

Задача 13 Провести синтаксичний аналіз виразу $(P \wedge Q) \rightarrow (R \leftrightarrow (\neg S))$. Чи є він формулою логіки висловлювань?

Задача 14 Написати програму, яка для завданого виразу надає всі можливі формули, що можна отримати за різного розташування дужок (з урахуванням пріоритетів логічних зв'язувань). Наприклад, для виразу $X \wedge Y \rightarrow \neg\, Y \wedge Z$ на вході в програму можна отримати на виході наступні формули, які різняться за порядком виконання логічних зв'язувань: $X \wedge (Y \rightarrow \neg\, Y) \wedge Z$, $(X \wedge Y \rightarrow \neg\, Y) \wedge Z$, $X \wedge Y \rightarrow \neg\, (Y \wedge Z)$,...

Задача 15 Розробити алгоритм синтаксичного аналізу виразу, який розв'язує наступну задачу: на підставі тексту надає відповідь на питання, чи є цей текст формулами логіки висловлювань? З'ясувати, чи можна розв'язати цю задачу за один перегляд тексту?

Задача 16 Скласти приклади висловлювань сумісних та несумісних суджень. Визначити вид зв'язків у судженнях за зразком на рисунку 2.3.

Питання для самоконтролю

1 Дати визначення поняття «логіка».

2 Які існують форми мислення?

3 Дати визначення поняття «поняття».

4 Навести класифікацію понять за вмістом та приклади понять для кожного пункту.

5 Навести класифікацію понять за обсягом та приклади понять для кожного пункту.

6 Дати визначення поняття «судження».

7 З яких лементів складаються судження? Які є різновиди суджень?

8 Навести чотири види суджень з графічним поясненням.

9 Які випадки розподілення суджень Вам відомі? Навести графічні пояснення.

10 Дати визначення понять «умовивід» та «категоричний силогізм».

11 Навести схеми фігур силогізмів та приклади до кожної з фігур.

12 Навести приклад гіпотетичного силогізму.

13 Дати визначення поняття «умовне судження». З яких частин складається умова судження?

14 Дати визначення понять «числення висловлювань», «висловлювання», «логічна операція».

15 Навести основні логічні операції.

16 Дати визначення поняття «правильно побудована формула». Навести приклад.

17 Навести таблиці істинності основних логічних операцій.

18 Навести алгоритм побудови таблиці істинності висловлювання.

19 За яких умов формули є загалнозначимими, суперечливими, рівносильними?

20 Навести основні еквівалентності перетворення формул.

21 Навести графічну інтерпретацію логічних операцій діаграмами Ейлера-Венна.

22 З яких кроків складається процедура перетворення формули в нормальну форму?

23 Які властивості повинні мати досконала диз'юнктивна та кон'юнктивна нормальні форми?

24 Навести алгоритм побудови досконалої диз'юнктивної нормальні форми.

25 В чому полягає різниця між процедурами побудови досконалої диз'юнктивної нормальні форми та досконалої кон'юнктивної нормальні форми?

РОЗДІЛ 3
ЛОГІКА ПРЕДИКАТІВ ПЕРШОГО ПОРЯДКУ

3.1 Основні поняття логіки предикатів

Атоми, з яких складаються формули у численні висловлювань, розглядаються як єдине ціле, структура та склад якого не аналізуються. Такого підходу замало для опису предметної області, оскільки велика кількість думок не може бути адекватно відображена таким чином. Наприклад, мовою числення висловлювань неможливо відобразити той факт, що з твердження «щонайменше один учень розв'язав всі задачі» випливає твердження «кожну задачу розв'язав щонайменше один учень», тому що цим твердженням відповідають елементарні формули числення висловлювань, імплікація яких не є тавтологією. Тому слід розглянути інший спосіб подання знань – мову логіки предикатів.

Мова логіки предикатів – це одна з формальних мов, що є найбільш наближеними до людської мови.

Логіка предикатів першого порядку порівняно з численням висловлювань, крім чотирьох типів символів: предикатних символів, констант, символів предметних змінних, функціональних символів, має ще й три логічних поняття: терми, предикати і квантори.

Предикатний символ – функція, що має певне число аргументів. Якщо функціональний символ f має n аргументів, то f називається n-місним функціональним символом.

Константа може розглядатися як функціональний символ без аргументів.

Наведемо приклади предикатів.

Приклад 3.1 Одномісний предикат $P(x)$, де P означає «бути простим числом». Тоді якщо $x = 5$ отримуємо висловлювання «вірно, що 5 це просте число» або $P(5) = \text{I}$ (тобто істино або дорівнює 1). Якщо $x = 4$ отримуємо висловлювання «невірно, що 4 це просте число» або $P(4) = \text{X}$ (тобто хибно або дорівнює 0).

Приклад 3.2 Двомісний предикат $Z(x, y)$, де Z означає «студент x знаходиться в аудиторії y». Тоді для x = «Коваленко» та y = «аудиторія

№ 604» маємо висловлювання «студент Коваленко знаходиться в аудиторії № 604», або Z(Коваленко, аудиторія № 604) = I (тобто істино або дорівнює 1) за умови, що цей студент дійсно знаходиться в аудиторії № 604, інакше Z(Коваленко, аудиторія № 604) = *X*(тобто хибно або дорівнює 0).

Приклад 3.3 Тримісний предикат $Q(x, y, z)$, де Q означає «z дорівнює сумі чисел x та y». На підставі даного предиката можна побудувати функцію висловлювання для $x = 5$ «існують такі y та z, що z дорівнює сумі 5 та y», така функція висловлювання записується наступною формулою $\exists y\, \exists z\, Q(5, y, z)$, де $\exists$ – квантор існування. Для $x = 5$ та $y = 2$ маємо функцію висловлювання «існує таке z, що дорівнює сумі 5 та 2» або $\exists z\, Q(5, 2, z)$. Для $x = 5$, $y = 2$ та $z = 7$ маємо висловлювання $Q(5, 2, 7)$, яке означає «7 дорівнює сумі чисел 5 та 2» та є істинним. Для наведеного прикладу x, y, z є предметними змінними, а 5, 2, та 7 – предметні константи.

З прикладу 3.3 випливає, що *висловлювання* – предикат, всі предметні змінні якого замінені на предметні константи. Після заміни всіх предметних змінних на предметні константи можна визначити істинним є дане висловлювання чи хибним (наприклад, висловлювання «7 дорівнює сумі чисел 5 та 2» є істинним). Таким чином, можна заначити, що висловлювання є нульмісною висловлювальною формою або *нульмісним предикатом*.

Функціональний символ – це символ, що вказує на завдання певного відношення між змінними та (або) константами, якщо на відповідних елементах предметної області завдана структура, або між ними існують функціональні відношення.

Наприклад, дату в PROLOG можна розглядати як структуру на змінних «число», «місяць», «рік», якщо позначити функціональним символом *f* дату, то маємо функцію *f*(число, місяць, рік), де число, місяць, рік – аргументи.

Іншим прикладом може бути структура PROLOG, яка описує трикутник: «*g* (вершина (координати_*x*, *y*), вершина (координати_*x*, *y*), вершина (координати_*x*, *y*))», тут *g* – функціональний символ, що позначає відношення «трикутник».

Після підстановки констант замість змінних у структурі, її можна додати до предикату та визначити чи є отримане висловлювання істинним або хибним.

Наприклад, якщо x, y, z – натуральні числа можна записати елементарну (атомарну) формулу $f^+(x, y)$, яка означає «скласти числа x та y». Тримісний предикат $Q(x, y, z)$, де Q означає «z дорівнює сумі чисел x та y» може бути записаний у вигляді $Q(z, f^+(x, y))$.

3.1.1 Терми

Введемо поняття сигнатури та терма.

Сигнатурою є множина $\Sigma = \Sigma_F \cup \Sigma_P \cup \Sigma_C$, де $\Sigma_F = \{f_i\}_{i \in I}$ – множина функціональних символів; $\Sigma_p = \{P_j\}_{j \in J}$ – множина предикатних символів, $\Sigma_C = \{c_k\}_{k \in K}$ – множина предметних констант.

Термом називається вираз, який утворено зі змінних і констант, можливо, із застосуванням функцій.

Терми визначаються рекурсивно наступним чином:

1 Константа є терм.

2 Змінна є терм.

3 Якщо f є n-місним функціональним символом і $t_1, \ldots, t_n$ – терми, то $f(t_1, \ldots, t_n)$– терм.

4 Жодних термів, крім визначених застосуванням зазначених вище правил не існує.

Якщо P є n-місним предикатним символом і $t_1, \ldots, t_n$ – терми, то $P(t_1, \ldots, t_n)$ – атом.

Терми призначені для завдання об’єктів предметної множини.

Приклад 3.4 Припустимо є множини Σ_F = {сестра(x), мати(y), любить(x, y), поважає(y, x), брат(z)} та Σ_C = {Семен, Ігор, Наталка, Катерина, Андрій}, які складають сигнатуру. Тоді вирази сестра(Ігор), мати(Катерина), поважає(Семен, Катерина) є термами даної сигнатури, а вирази сестра(Ігор, Катерина), поважає(Семен), батько(Ігор) не є термами даної сигнатури.

3.1.2 Предикати

Предикатом називається функція, що набуває лише два значення: істино або хибно, і яка призначається для вираження властивостей об'єктів або зв’язків між ними.

Термін «предикат» походить від латинського *praedicatum* – згадане, сказане або присудок. Традиційна логіка виокремлює в елементарному висловлювані суб'єкт та предикат. Суб'єкт (логічний підмет) – те, про що йде мова у висловлюванні. Предикат (логічний присудок) – те, що саме стверджується про суб'єкт.

Приклад 3.5 Висловлювання «кішка має чотири ноги», в даному випадку «кішка» – суб'єкт, «має чотири ноги» – предикат. Це висловлювання є істинним, і воно залишиться істинним, якщо замінити суб'єкт «кішка» на суб'єкт «пес». Якщо ж суб'єкт «кішка» замінити на суб'єкт «курка», то висловлювання набуває істинносного значення ХИБНО (або 0). Якщо суб'єкт «кішка» на x, отримаємо форму висловлювання «x має чотири ноги» – це предикат, що завданий такою формою.

Нехай $x_1, x_2 \ldots x_n$ – символи предметних змінних. Нехай набори змінних $\{x_1,x_2 \ldots x_n\}$ належать до множини D, яку будемо називати предметною областю. Предикатом або n-місним предикатом, визначеним на предметній області D, називають відображення D в множину тверджень.

Областю визначення предиката є декартів добуток множин значень предметних змінних. Пояснимо це твердження на прикладі розв'язання наступної задачі.

Задача 3.1 Завдано двомісну форму висловлювання «$x<y$»; припустимо x приймає значення з множини $M_x = \{1;2;3\}$, а y – з множини $M_y = \{2;4\}$. Така форма завдає два предиката P_1 та P_2 відповідно на множинах $M_x \times M_y$ та $M_y \times M_x$, тобто декартових добутках множин M_x та M_y.

Визначити множини істинності предикатів P_1 та P_2.

Розв'язання Наведемо істинностні значення цих предикатів (табл. 3.1).

Таблиця 3.1 – Істинностні значення предикатів P_1 та P_2

$(x;y)$	$x<y$	P_1 $[(x;y)]$	$(y;x)$	$y<x$	P_2 $[(y;x)]$
(1;2)	1<2	1	(2;1)	2<1	0
(1;4)	1<4	1	(2;2)	2<2	0
(2;2)	2<2	0	(2;3)	2<3	1
(2;4)	2<4	1	(4;1)	4<1	0

(3;2)	3<2	0	(4;2)	4<2	0
(3;4)	3<4	1	(4;3)	4<3	0

Таким чином, двомісна форма висловлювання визначає, в загальному випадку, два різних предиката.

Для однозначного завдання предиката за допомогою форми висловлювання з більш ніж однією змінною, достатньо обрати та вказати для змінних певний порядок. Якщо для змінних форми висловлювання $P(x_1, x_2,\ldots,x_n)$ обрано порядок $(x_{i1}, x_{i2},\ldots,x_{in})$ та $M_{i1}, M_{i2},\ldots, M_{in}$ – відповідно множини значень змінних $x_{i1}, x_{i2},\ldots,x_{in}$, то така форма завдає предикат на множині $M_{i1}\times M_{i2}\times\ldots\times M_{in}$.

Виходячи з розглянутого прикладу слід надати наступні визначення.

Впорядкована n-ка – сукупність n не обов'язково різних об'єктів разом з вказаним порядком їх розташування (n=2 маємо пару, n=3 – трійку, n=4 – четвірку тощо).

Компоненти – об'єкти, що складають n-ку.

Дві впорядковані n-ки вважають рівними, якщо компоненти та порядок їх розташування співпадають.

Декартовий добуток $M_1\times M_2\times\ldots\times M_n$ непустих множин $M_1, M_2,\ldots,M_n$ – множина всіх можливих впорядкованих n-ок перша компонента кожної з яких належить M_1, друга – M_2 і т.д.

M^n – декартовий добуток n рівних множин M.

Наприклад, для опису гри в шахи кожній клітинці шахової дошки ставиться у відповідність пара, перша компонента якої є елементом множини M_1 ={a; b; c; d; e; f; g; h}, а дуга – елементом множини M_2 ={1; 2; 3; 4; 5; 6; 7; 8}. Посилання на конкретну клітинку є наприклад парою f8. Декартовим добутком $M_1\times M_2$ в цьому випадку є множина, що містить посилання на всі клітинки шахової дошки.

Множина істинності предиката P, який завданий на множині M – підмножина множини M, що складається тільки з тих елементів, яким відповідає значення I (1) предиката P, позначається $\overline{P}$. Тобто $\overline{P}\subset M$.

Якщо множина істинності предиката співпадає з множиною, на якій він завданий, такий предикат є *тотожно істинним*.

Якщо множина істинності предиката є порожньою множиною, такий предикат є *тотожно хибним*.

Для таблиці 3.1 наведеної вище: $\overline{P}_1$={(1; 2); (1; 4); (2; 4); (3; 4)} – множина істинності предиката P_1 та $\overline{P}_2$={(2; 3)} – множина істинності предиката P_2.

Існують такі *способи завдання предикатів*:

1 Якщо область визначення предиката M має скінченну кількість елементів, тобто є скінченною множиною, то предикат можна завдати перерахуванням n-ок, що належать цьому предикату або побудовою таблиці істинності (див. табл. 3.1).

2 Якщо область визначення предиката M є нескінченною множиною, його можна завдати висловлюванням (формою висловлювання) або формулою.

Приклад 3.6 Припустимо предикат завдано формою висловлювання «x та y – взаємно прості числа», де $x \in$N та $y \in$N, N – нескінченний ряд натуральних чисел. Або предикат завдано формою висловлювання «x кохає y», де x та y – люди, а їх дуже багато і народжуються все нові. Або предикат завдано формою висловлювання «об'єкт x притягує об'єкт y» (закон всесвітнього тяжіння), об'єктів у всесвіті нескінченна множина, оскільки всесвіт є нескінченним. Якщо предикат завдано формулою вона може мати наступний вигляд: $\forall x \forall y((\exists z\ (P(x, z) \wedge P(y, z) \rightarrow \exists u\ Q(x, y, u))$.

Сутність і способи завдання предикатів можна подати схематично (рис. 3.1).

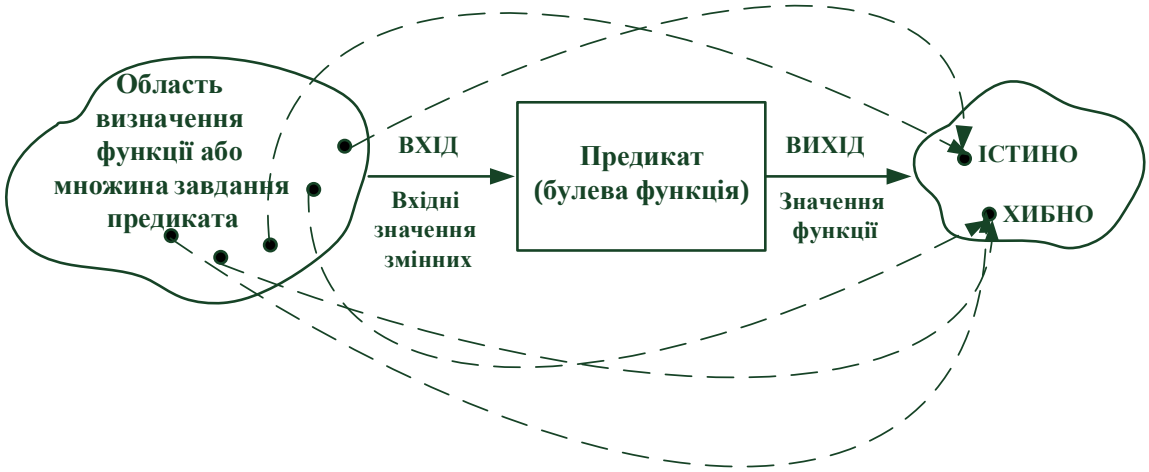


Рисунок 3.1 – Сутність і способи завдання предикатів

Виокремлюють предикати-властивості та предикати-відношення.

Предикат-властивість – це одномісний предикат, який відображає певну властивість, а саме наявність або відсутність властивості у певної множини суб'єктів висловлювання.

Обсяг властивості – це нова множина суб'єктів, які мають дану властивість. Наприклад, розглянемо предикат-властивість «*x* має чотири ноги» та множину суб'єктів {«кішка», «пес», «курка»}, тоді обсяг властивості складає множина {«кішка», «пес»}.

Предикат-відношення – це *n*-місний предикат (де $n > 1$), який встановлює наявність певного відношення між суб'єктами висловлювання з однієї або різних множин. Наприклад, розглянемо предикат-відношення «студент *x* дружить зі студентом *y*» область визначення цього предиката є декартів добуток множини студентів на саму себе, а множину істинності предикату складають двійки студентів, які дружать між собою. Інший предикат-відношення «студент *x* вивчає дисципліну *y*» область визначення цього предиката є декартів добуток множини студентів та множину дисциплін, що викладаються у виші. Множину істинності предикату складають двійки студентів та дисциплін, але тільки тих студентів, які вивчають дисципліну з тієї ж двійки.

3.1.3 Квантори

Для побудови формул поряд з логічними зв'язками використовуються квантори: квантор загальності ($\forall$) та квантор існування ($\exists$).

Якщо *x* – змінна, то ($\forall x$) читається як: «для всіх *x*», або як «для кожного *x*», а ($\exists x$) – «існує *x*», або як «для деяких *x*».

Входження змінної *x* до формули називається зв'язаним тоді і тільки тоді, коли воно збігається із входженням до кванторного комплексу ($\forall x$) або комплексу ($\exists x$) або знаходиться в області дії такого комплексу. Змінну, до якої відноситься квантор, називають *зв'язаною*, решту змінних – вільними.

Змінна є *вільною* у формулі, якщо хоча б одне її входження до даної формули є вільним.

Наприклад, у формулі $(\exists x)Q(x, y) \to \neg(P(f(x, y) \vee (\forall z)P(z))$ перше входження змінної x є зв'язаним, а друге – вільним. Всі входження змінної y є вільними. Входження змінної z є зв'язаним.

Правильно побудовані формули або формули логіки першого порядку рекурсивно визначаються наступним чином:

1) атом є формулою;

2) якщо F і G – є формулами, то $\neg F$, $(F \wedge G)$, $(F \vee G)$, $(F \to G)$, $(F \leftrightarrow G)$ також є формулами;

3) якщо F – є формулою, а x є вільною змінною в F ,то $(\forall x)F$ і $(\exists x)F$ є формулами;

4) формули утворюються тільки за кінцевим числом застосувань правил 1), 2), 3).

Приклад 3.7 Нехай $P(x, y)$ – двумісний предикат «x любить y». Розглянемо як читаються наведені формули:

$\forall x \exists y$ – «для будь-якої людини x існує людина y, яку вона любить»;

$\forall y \exists x$ – «для будь-якої людини y існує людина x, яка її любить»;

$\exists y \forall x$ – «існує людина y, яку люблять всі»;

$\exists x \forall y$ – «існує людина, яка любить всіх»;

$\forall x \forall y$ – «всі люди люблять всіх людей»;

$\exists y \exists x$ – «існує людина, яка когось любить».

3.2 Інтерпретація формул у логіці предикатів першого порядку

Інтерпретація формули F логіки першого порядку складається з непорожньої (предметної) області D і вказаної «оцінки» (значення) всіх констант, функціональних символів та предикатних символів, що в ній зустрічаються. Цього можна досягти виконанням наступних процедур:

1 Кожній константі ставиться у відповідність деякий елемент з області D.

2 Кожному n-місному функціональному символу ставиться у відповідність відображення з області D^n в область D (зауважимо, що $D^n = \{(x_1, \ldots, x_n) | \, x_1 \in D, \ldots x_n \in D)\}$.

3 Кожному n-місному предикатному символу ставиться у відповідність відображення D^n в ІСТИНО або ХИБНО (див. рис. 3.1).

Для кожної інтерпретації з області D формула може набувати істинносного значення ІСТИНО або ХИБНО згідно з наступними правилами:

1) якщо дано значення формул G та H, то істинносні значення формул $\neg G$, $(G \vee H)$, $(G \wedge H)$, $(G \to H)$ та $(G \leftrightarrow H)$ отримуються за даними табл. 2.1 та 2.3-2.6.

2) формула $(\forall x)G$ набуває значення ІСТИНО, якщо G набуває значення ІСТИНО для кожного x з області D.

3) Формула $(\exists x)G$ набуває значення ІСТИНО, якщо G набуває значення ІСТИНО хоча б для одного x з області D, в противному випадку вона набуває значення ХИБНО.

Формула, що містить вільні змінні, не може набути істинносного значення.

Формула G є *несуперечливою* (*здійсненною*) тоді і тільки тоді, коли існує така інтерпретація I_1, за якої вона матиме значення ІСТИНО. Якщо формула G є істинною в інтерпретації I_1, то стверджують, що I_1 є *моделлю* формули G або I1 задовольняє G.

Формула G є *суперечливою* (*нездійсненною*) тоді і тільки тоді, коли не існує інтерпретації, яка її задовольняє.

Формула G є *загальнозначимою* тоді і тільки тоді, коли не існує жодної інтерпретації, яка б не задовольняла формулі G.

Формула G є *логічним наслідком* формул F_1, F_2, …, F_n тоді і тільки тоді, коли для кожної інтерпретації I_1 формула G є істинною в інтерпретації I_1. Тільки у випадку, якщо кон'юнкція формул $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ є істиною в інтерпретації I_1.

Якщо формула в логіці першого порядку не містить змінних і кванторів, її можна розглядати як формулу в численні висловлювань.

3.3 Предварені нормальні форми

У логіці предикатів першого порядку існує поняття «предвареної нормальної форми», як аналог кон'юнктивної та диз'юнктивної нормальних форм у численні висловлювань.

Говорять, що формула F у логіці першого порядку знаходиться в *предвареній нормальній формі* тоді і тільки тоді, коли формула має вигляд $(Q_1x_1) \dots (Q_nx_n)(M)$, де кожне (Q_ix_i), i=1,…,n є або $\forall x_i$ або $\exists x_i$, а M є формулою, яка не містить кванторів. Частина формули $(Q_1x_1) \dots (Q_nx_n)$ називається *префіксом*, а формула M називається *матрицею* формули F.

Приклад 3.8 Наведемо формули в предвареній нормальній формі: $\forall x\exists y(P(y, x) \wedge Q(y))$; $\forall x\forall y(\neg P(y, x) \wedge Q(y))$; $\forall x\forall y\exists z(Q(y, x) \to R(z))$.

Для переведення формули у предварену нормальну форму використовують закони числення висловлювань (див. основні еквівалентності 1-22 в п. 2.3.1 Нормальні форми у численні висловлювань) та закони логіки першого порядку.

Наведемо закони логіки першого порядку (для зручності використання у практичних роботах позначимо їх літерами):

а) $Qx\ F[x] \vee G \equiv Qx\ (F[x] \vee G)$;

б) $Qx\ F[x] \wedge G \equiv Qx\ (F[x] \wedge G)$;

в) $\neg(\forall x\ F[x]) \equiv \exists x\neg(F[x])$;

г) $\neg(\exists x\ F[x]) \equiv \forall x\neg(F[x])$;

д) $\forall x\ F[x] \wedge \forall x\ H[x] \equiv \forall x\ (F[x] \wedge \forall x\ H[x])$;

е) $\exists x\ F[x] \vee \exists x\ H[x] \equiv \exists x\ (F[x] \vee \forall x\ H[x])$;

ж) $Q_1x\ F[x] \vee Q_2x\ H[x] \equiv Q_1x\ Q_2z\ (F[x] \vee H[z])$;

з) $Q_3x\ F[x] \wedge Q_4x\ H[x] \equiv Q_3x\ Q_4z\ (F[x] \wedge H[z])$,

де Q ,Q_1, Q_2, Q_3, Q_4 є або квантором існування або квантором загальності.

Зауваження. Кожна зв'язана зміна (тобто зміна, що стоїть під знаком квантора) може бути інтерпретована як «місце» для підстановки будь-якої предметної змінної або предметної константи, тому зв'язану змінну x (зв'язане входження змінної x) можна замінити іншою змінною, яка не використовувалася у формулі раніше.

Розглянемо формулу $\forall x\, F[x] \vee \forall x\, H[x]$. Можна записати еквівалентність: $\forall x\, H[x] \equiv \forall z\, H[z]$, де z – змінна, яка не використовується у функції $F[x]$, тоді можна записати наступні тотожності для перетвореннч цієї формули:

$$\forall x\, F[x] \vee \forall x\, H[x] \equiv \forall x\, F[x] \vee \forall z\, H[z] \equiv \forall x \forall z\, (F[x] \vee H[z]),$$

аналогічно маємо:

$$\exists x\, F[x] \wedge \exists x\, H[x] \equiv \exists x\, F[x] \wedge \exists z\, H[z] \equiv \exists x \exists z\, (F[x] \wedge H[z]).$$

В загальному випадку, якщо позначити квантори існування та загальності через Q_1, Q_2, Q_3, Q_4, маємо закони ж) та з).

Використовуючи закони числення висловлювань та закони логіки предикатів першого порядку, можна надати формулі предвареної нормальної форми.

Практична робота 3. Переведення формули у предварену нормальну форму

Наведемо алгоритм процедури надання формулам предвареної нормальної форми:

Крок1 Використання законів числення висловлювань (основні еквівалентності 15, 16) $H_1 \to H_2 \equiv \neg H_1 \vee H_2$, $H_1 \leftrightarrow H_2 \equiv H_1 H_2 \vee \neg H_1 \neg H_2$.

Крок2 Використання основних еквівалентностей 1, 12:

- $\neg (\neg H_1) \equiv H_1$ (правило зняття подвійного заперечення),
- $\neg (H_1 \wedge H_2) \equiv \neg H_1 \wedge \neg H_2$ (правило де Моргана),
- $\neg (H_1 \vee H_2) \equiv \neg H_1 \vee \neg H_2$ (правило де Моргана),

а також законів логіки предикатів першого порядку в) та г):

- $\neg(\forall x\, F[x]) \equiv \exists x \neg(F[x])$,
- $\neg(\exists x\, F[x]) \equiv \forall x \neg(F[x])$.

Крок3 Перейменування зв’язаних змінних, якщо це потрібно.

Крок4 Використання законів логіки предикатів першого порядку а), б), д)-з):

- $Qx\, F[x] \vee G \equiv Qx\, (F[x] \vee G)$;
- $Qx\, F[x] \wedge G \equiv Qx\, (F[x] \wedge G)$;
- $\forall x\, F[x] \wedge \forall x\, H[x] \equiv \forall x\, (F[x] \wedge \forall x\, H[x])$;
- $\exists x\, F[x] \vee \exists x\, H[x] \equiv \exists x\, (F[x] \vee \forall x\, H[x])$;

– $Q_1x\ F[x] \vee Q_2x\ H[x] \equiv Q_1x\ Q_2z\ (F[x] \vee H[z])$;

– $Q_3x\ F[x] \wedge Q_4x\ H[x] \equiv Q_3x\ Q_4z\ (F[x] \wedge H[z])$.

Зауваження. Під час застосування заперечення до кванторів слід враховувати наступні еквівалентності: $\neg\ (\forall x) \equiv \exists x$; $\neg\ (\exists x) \equiv \forall x$.

Задача 3.2 Перевести у предварену нормальну форму формулу $\forall x\ F(x) \to \exists x\ P(x)$.

Розв'язання.

$\forall x\ F(x) \to \exists x\ P(x) \equiv^{(11)} \neg(\forall x\ F(x)) \vee \exists x\ P(x) \equiv^{(в)} \exists x\ \neg(F(x)) \vee \exists x\ P(x)) \equiv$

$\equiv^{(3)} \exists x\ (\neg(F(x)) \vee P(x))$.

В отриманій формулі $\exists x$ – префікс, $(\neg(F(x)) \vee P(x))$ – матриця.

Зауваження. В подальшому помітка біля знаку тотожності у еквівалентних перетвореннях формул означає номер закону числення висловлювань (основної еквівалентності) або літеру закону логіки предикатів першого порядку, які були використані під час відповідного перетворення.

Задача 3.3 Перевести у предварену нормальну форму формулу $\forall x \forall y\ ((\exists z\ P(x, z) \wedge P(y, z)) \to \exists u\ Q(x, y, z))$.

Розв'язання.

$\forall x \forall y\ ((\exists z\ P(x, z) \wedge P(y, z)) \to \exists u\ Q(x, y, u)) \equiv$

$\equiv^{(11)} \forall x \forall y\ (\neg\ (\exists z\ P(x, z) \wedge P(y, z)) \vee \exists u\ Q(x, y, u)) \equiv$

$\equiv^{(8)} \forall x \forall y\ ((\forall z\ (\neg\ P(x, z) \vee \neg\ P(y, z))) \vee \exists u\ Q(x, y, u)) \equiv$

$\equiv^{(3)} \forall x \forall y \forall z \exists u\ (\neg\ P(x, z) \vee \neg\ P(y, z) \vee Q(x, y, u))$. В отриманій формулі $\forall x \forall y \forall z \exists u$ – префікс, $(\neg\ P(x, z) \vee \neg\ P(y, z) \vee Q(x, y, u))$ – матриця.

Зауваження. Останнє перетворення є еквівалентним, оскільки формула $(\neg\ P(x, z) \vee \neg\ P(y, z))$ не містить змінної u (тобто дана формула може вважатися сталою відносно цієї змінної), а формула $Q(x, y, u)$ не містить змінної z.

3.4 Логічне виведення. Принцип резолюції

Дві формули під час доведення можуть бути *резольвіровані* (зчеплені) одна з одною, якщо одна з них містить позитивний літерал, а інша – такий

самий, але негативний літерал з одним і тим самим предикатом та з однаковими (уніфікованими) аргументами.

Приклад 3.9 Розглянемо теорію з двох формул:

$$P(x) \vee \neg Q(y, z) \quad (3.1)$$

$$Q(y, z) \vee \neg R(y, z) \quad (3.2)$$

В (3.1) міститься негативний літерал $\neg Q(y, z)$, а в (3.2) – відповідний позитивний літерал $Q(y, z)$ й аргументи обох літералів уніфіковані (тобто y уніфікується з y, а z уніфікується з z), тоді (3.1) може бути резольвірована з (3.2), в результаті отримуємо нову формулу теорії так звану резольвенту:

$$P(x) \vee \neg R(y, z) \quad (3.3)$$

Надалі можна скористатися будь-якою з формул: (3.1), (3.2) або (3.3).

Наведені нижче формули (3.4) і (3.5) не резольвуються одна з одною, так як аргументи літералів Q не уніфікуються:

$$P(x) \vee \neg Q(y, z) \quad (3.4)$$

$$Q(y, y) \vee \neg P(y, z) \quad (3.5)$$

У формульній формі неявно застосовується квантор існування для квантифікації всіх змінних. Наприклад, у формулі (3.2) мається на увазі наявність кванторів $\forall x \forall y\ (Q(y, z) \vee \neg R(y, z))$. Такі формули ще мають назву «фраза».

Зауваження. Якщо в якості аргументу виступає змінна, то вона уніфікується з будь-якою константою.

Зауваження. Якщо одна й та сама змінна в фразі використовується більше ніж один раз і дана змінна в процесі резолюції була уніфікована з константою, то після уніфікації в процесі резолюції дана змінна скрізь залишається уніфікованою з цією самою константою.

На принципі резолюції ґрунтується алгоритм доведення того, що певна фраза є наслідком існуючої множини фраз (теорії).

Для знаходження такого рішення використовується стратегія «від зворотного», що полягає в наступному. Нехай слід довести, що фраза $P(a)$ є наслідком існуючої множини фраз:

1 до теорії додається заперечення цієї фрази $\neg P(a)$;

2 відшукується фраза для виконання резолюції на основі фрази $\neg P(a)$;

3 у кожній наступній резолюції бере участь резольвента попередньої резолюції з підстановками замість змінних уніфікованих констант...

Висновки:

а) якщо в ході знаходження отримана порожня фраза, це означає, що виявлено протиріччя, то вихідна фраза $P(a)$ вважається наслідком наявної теорії;

б) якщо під час знаходження порожня фраза не виявляється, то вихідна фраза $P(a)$ не вважається наслідком наявної теорії.

Описана стратегія розв'язання задач має такі властивості:

– дана стратегія є низхідною, оскільки вона починає процес розв'язання з заперечення висновку, а в подальшому процесі розв'язання використовують інші фрази теорії;

– дана стратегія називається «пошуком у глибину», так як результат останньої резолюції завжди використовується в наступній за нею резолюції.

Практична робота 4. Логічне виведення. Метод резолюцій

Метод резолюцій передбачає прості синтаксичні перетворення формул для перевірки нездійсненності формул.

Резольвентою двох диз'юнктів: $D_1 = L \vee Q_1$ та $D_2 = \neg L \vee Q_2$, називається диз'юнкт $Q_1 \vee Q_2$.

Розглянемо теореми, які дозволяють використовувати поняття резольвенти для виведення нових фраз та доведення істинності теорій.

Теорема 1. Резольвента є логічним наслідком диз'юнктів, що її породжують.

Доведення. Проведемо доказ від зворотного, для цього перевіримо неможливість виконання наступної фрази:

$$F \equiv (L \vee Q_1) \wedge (L \vee Q_2) \wedge \neg (Q_1 \vee Q_2).$$

У наведеній фразі останній диз'юнкт свідчить про заперечення того, що резольвента є наслідком диз'юнктів D_1 та D_2.

Приведемо фразу F до диз'юнктивної нормальної форми:

$$F \equiv \boldsymbol{L\, \neg L}\, \neg Q_1\, \neg Q_2 \vee L\, \boldsymbol{Q_2}\, \neg Q_1\, \boldsymbol{\neg Q_2} \vee \boldsymbol{Q_1}\, \neg L\, \boldsymbol{\neg Q_1}\, \neg Q_2 \vee \boldsymbol{Q_1}\, Q_2\, \boldsymbol{\neg Q_1}\, \neg Q_2 \equiv \varnothing.$$

Зауваження. В даному записі знак кон'юнкції є $\wedge$ відсутнім для спрощення запису так само, як може бути відсутнім знак множення в математичних формулах.

Кожен кон'юнкт цієї фрази (адже диз'юнктивна нормальна форма це диз'юнкція елементарних кон'юнктів) містить літерал та його заперечення, а значить є нездійсненним, тому відповідно і вся фраза є нездійсненою $F \equiv \varnothing$.

Ми прийшли до протиріччя, це означає, що резольвента є наслідком диз'юнктів, що її породжують.

Теорему доведено.

Таким чином доведено, що $Q_1 \vee Q_2$ є логічним наслідком з диз'юнктів $D_1 = L \vee Q_1$ та $D_2 = \neg L \vee Q_2$.

Теорема 2. Якщо до логічної функції кон'юнктивно приєднати її логічний наслідок, то значення функції не зміниться.

Доведення. Доведемо твердження: якщо $A \to B$, то $A \equiv A \wedge B$ (тобто з твердження, що з фрази A випливає фраза B, логічно слідує, що фраза A та $A \wedge B$ є рівносильними).

За визначенням логічного наслідку: на всіх інтерпретаціях, де $A = 1$ (ІСТИНА), також має бути $B = 1$ (ІСТИНА). Побудуємо таблицю істинності для перевірки твердження теореми (табл. 3.2).

Таблиця 3.2 – Таблиця істинності фрази $A \wedge B$

$x\ y\ z$	$A = \ldots$	$B = \ldots$	$A \wedge B$
000	0	1	0
001	0	1	0

010	1	1	1
011	0	1	0
100	0	0	0
101	1	1	1
110	0	0	0
111	1	1	1

З таблиці видно, що на тих інтерпретаціях, де $A = 0$ (ХИБНО) також $A \wedge B = 0$ (ХИБНО), і навпаки, там де $A = 1$ (ІСТИНА) також $A \wedge B = 1$ (ІСТИНА). Значить фрази A та $A \wedge B$ є рівносильними.

Теорему доведено.

Наведемо *особливості методу резолюцій*:

– метод резолюцій використовується для перевірки нездійсненності формул;

– під час перевірки початкова формула переводиться у кон'юнктивну нормальну форму, тобто представляється у вигляді кон'юнкції елементарних диз'юнктів $D_1 \wedge D_2 \wedge \ldots \wedge D_n$;

– для двох будь-яких диз'юнктів $D_j = L \vee Q_j$ та $D_k = L \vee Q_k$, що містять протилежні літерали можна побудувати резольвенту: $Q_j \vee Q_k$;

– оскільки резольвента є логічним наслідком диз'юнктів, що її породжують, її можна кон'юнктивно приєднати до інших диз'юнктів, при цьому значення початкової формули не зміниться;

– резолютивне виведення – це послідовність диз'юнктів $D_1, D_2, \ldots, D_n$, таких, що D_n – порожній диз'юнкт, а D_i ($i = 1, \ldots, n\text{-}1$) або початковий диз'юнкт або резольвента двох буд-яких попередніх диз'юнктів;

– будь-яка формула є *нездійсненною* тоді і тільки тоді, коли має порожню резольвенту.

Задача 3.4 Доведемо нездійсненність формули, наданої в кон'юнктивній нормальній формі:

$$F \equiv (q_1 \vee \neg q_2) \wedge (\neg q_1 \vee q_2) \wedge (q_2 \vee \neg q_3) \wedge (q_3 \vee \neg q_2) \wedge (q_1 \vee q_3) \wedge (\neg q_1 \vee \neg q_3).$$

Розв'язання.

Запишимо F, як набір диз'юнктів та побудуємо резолютивне виведення:

1 $q_1 \vee \neg q_2$ – початковий диз'юнкт;

2 $\neg q_1 \vee q_2$ – початковий диз'юнкт;

3 $q_2 \vee \neg q_3$ – початковий диз'юнкт;

4 $q_3 \vee \neg q_2$ – початковий диз'юнкт;

5 $q_1 \vee q_3$ – початковий диз'юнкт;

6 $\neg q_1 \vee \neg q_3$ – початковий диз'юнкт;

7 $q_1 \vee \neg q_3$ – резольвента диз'юнктів 1 та 3;

8 q_1 – резольвента диз'юнктів 5 та 7;

9 q_2 – резольвента диз'юнкта 2 та фрази 8;

10 q_3 – резольвента фрази 9 та диз'юнкта 4;

11 $\neg q_1$ – резольвента фрази 10 та диз'юнкта 6;

12 [ ] – резольвента фраз 11 та 8 (*порожня* резольвента).

Наведемо опис логіки виведення. Серед початкових диз'юнктів 1-6 шукаємо такі, в які певна змінна входить із запереченням, а в інший без заперечення. Отриману таким чином резольвенту приєднуємо до вже існуючих фраз, тобто кон'юнктивно приєднуємо резольвенти 7-12 до початкової формули. В результаті маємо формулу рівносильну початковій. Отримана формула містить порожню резольвенту, тобто вона є нездійсненною, це означає, що і початкова формула також є нездійсненною.

Задача 3.5 Припустимо є наступні твердження: «Яблуко червоне та запашне», «Якщо яблуко червоне, то яблуко смачне». Довести методом резолюцій твердження «Яблуко смачне».

Розв'язання.

Сформуємо множину формул, що описують прості висловлювання, що відповідають наведеним твердженням:

X_1 – «Яблуко червоне»;

X_2 – «Яблуко запашне»;

X_3 – «Яблуко смачне»;

Тоді твердження, які наведені в умовах задачі, можна записати у вигляді наступних формул:

$X_1 \wedge X_2$ – «Яблуко червоне та запашне»;

$X_1 \to X_3$ – «Якщо яблуко червоне, то яблуко смачне».

Твердження, яке слід довести виражено формулою X_3.

Доведемо, що X_3 є логічним наслідком формул $(X_1 \wedge X_2)$ та $(X_1 \to X_3)$. Складемо множину формул із запереченням висловлювання, яке слід довести: $F = (X_1 \wedge X_2 \wedge X_1 \to X_3 \wedge \neg X_3)$.

Отриману приводимо до кон'юнктивної нормальної форми, використовуючи основні еквівалентності числення висловлювань:

$$F = (X_1 \wedge X_2 \wedge \neg X_1 \vee X_3 \wedge \neg X_3).$$

Запишимо F, як набір диз'юнктів та побудуємо резолютивне виведення:

1 X_1 – початковий диз'юнкт;

2 X_2 – початковий диз'юнкт;

3 $\neg X_1 \vee X_3$ – початковий диз'юнкт;

4 $\neg X_3$ – початковий диз'юнкт;

5 X_3 – резольвента диз'юнктів 1 та 3;

6 [ ] – резольвента диз'юнктів 4 та 5 (*порожня* резольвента).

Таким чином, виведено порожній диз'юнкт, тобто формула із запереченням висловлювання X_3 є нездійсненною, значить висловлювання X_3 доведено.

3.4.1 Особливості логічних методів подання знань.

Логічні методи подання знань мають такі особливості:

1 За логічних методів подання знань здійснюється перехід від процедурного способу завдання інформації до специфікації відношень між об'єктами реального світу.

2 За логічних методів подання знань програма є логічною специфікацією завдання, конкретне обчислення має вигляд запиту до цієї програми, а база знань предметної області є набором логічних формул.

3 Наявність регулярних методів виведення, в термінах яких можна визначати процедуру доведення є основною перевагою логічних методів подання знань.

4 Інша перевага логічних методів полягає в можливості використання семантики, яка припускає різне трактування залежно від мети логічних уявлень. Процедурна семантика забезпечує використання обчислювальних

машин для доказу доведеності тверджень з логічно заданих закономірностей про предметну область.

5 Третя перевага логічного подання полягає в простоті, лаконічності та одноманітності нотації, що використовується, для подання знань про предметну область, що забезпечує можливість створення описів баз знань, які однозначно інтерпретуються.

6 Логічні методи подання знань передбачають відсутність принципів структуризації логічних формул, які складають основу бази знань.

3.5 Продукційна модель представлення знань

У загальному випадку продукційна система завдається за допомогою сукупності правил виду:

Якщо A_1, то B_1, інакше…

…

Якщо A_n, то B_m,

де A_i – опис певної ситуації;

B_i – сукупність дій, які слід виконати за ситуації A_i.

Правила складаються з двох частини: антецеденти та консеквенти.

У продукційній системі розв'язання задачі відбувається шляхом зіставлення зі зразком, а необхідні знання представлені у вигляді набору правил.

Продукція – це пара «умова-дія», яка визначає одну порцію знань, необхідних для розв'язання задачі.

Існує чітка границя між антецедентом (гіпотезою), та консеквентом (висновком). Антецедент носить передбачуваний характер, оскільки він отриманий на підставі фактів, які можуть бути як істинними так і хибними. Консеквент є фактом, що отриманий шляхом доведення на основі фактів, які вважаються істинними.

Під продукцією розуміється вираз вигляду:

$$(i);\ Q;\ P;\ A \rightarrow B;\ N$$

де $A \rightarrow B$ – ядро, яке є основним елементом продукції, зазвичай воно інтерпретується фразою «Якщо А, то В»;

(i) – ім’я продукції, за допомогою якого дана продукція відокремлюється з множини продукцій;

Q – сфера використання продукції, яка описує предметну область або ситуацію;

P – умова використання ядра продукції (предикат), якщо P є істиним, ядро продукції активізується;

N – постумова продукції, що є процедурою, яку слід виконати після успішної реалізації ядра (необов’язково відразу).

Зауваження. Усі частини продукції, окрім ядра, є факультативними.

Основною проблемою виведення знань в системі продукцій є обрання наступної продукції для її аналізу. Обрання наступної продукції можна виконати з використанням таких принципів:

1 Принцип пріоритетного обрання: пріоритет може встановлюватися статично (відповідно до специфіки предметної області) або динамічно (в процесі функціонування системи продукцій, наприклад, залежно від часу знаходження продукції в обробці), статичні пріоритети використовуються в системі PROLOG.

2 Принцип метапродукцій: на підставі аналізу ознак в умовах продукцій, які знаходяться в обробці, метапродукції установлюють порядок їх виконання.

3 Принцип стопки книг: ґрунтується на тому, що продукція, яка найчастіше використовується, є найкориснішою, продукції утворюють «стопку», в якій порядок визначається частотою використання продукцій у минулому.

4 Принцип найдовшої умови: обирається та продукція, у якій стала істиною найдовша умова виконання ядра, принцип базується на тому, що правила, які стосуються вузького класу ситуацій, є важливішими за загальні, які стосуються широкого класу ситуацій.

Одним з найважливіших питань, що виникають під час проектування механізму логічного виведення систем, заснованих на знаннях, є обрання методу відшукування розв’язку, тобто вибір стратегії виведення. Під час розробки стратегії управління виведенням необхідно відповісти на два питання:

1 Яку точку в просторі станів необхідно прийняти за початкову? Від обрання точки залежить і метод здійснення знаходження розв'язку (в прямому або в зворотному напрямку).

2 Як підвищити ефективність знаходження розв'язку?

Існують такі стратегії управління виведенням:

1 *Пошук у глибину*: в просторі станів перевага завжди, коли це можливо, надається тій, яка відповідає наступному, більш детальному рівню опису завдання. Машина завжди першою досліджує невідому ліву гілку дерева. Коли процес заходить в тупик, то він повертається нагору в останній пункт обрання, де ще є невивчені альтернативні варіанти, а потім виконується наступний варіант обрання.

2 *Пошук у ширину*: спочатку аналізуються всі дані, що знаходяться на одному рівні простору стану, і лише потім здійснюється перехід до наступного рівня.

3 *Розбиття на підзадачі*: у початковій задачі виділяються підзадачі, розв'язання яких розглядається як досягнення проміжних цілей на шляху до кінцевої цілі.

4 *Альфа – бета алгоритм*: задача зводиться до зменшення простору станів шляхом видалення в ньому гілок неперспективних для відшукування успішного розв'язання, тому проглядаються тільки ті вершини, в які можна потрапити в результаті наступного кроку, після чого неперспективні напрями виключаються з подальшого розгляду.

Продукційні системи бувають двох діаметрально протилежних типів – з прямими і зворотними виведеннями. Розглянемо детальніше кожну концепцію, що визначає стратегію обрання в конкретних умовах.

За продукційного представлення область знань є множиною продукційних правил «ЯКЩО …, ТО…», а дані надані у вигляді множини фактів про поточну ситуацію. Сутність механізму виведення полягає у співставленні кожного правила, яке зберігається у базі знань з фактами, які містяться в базі даних. Коли частина правила ЯКЩО (умова) співпадає з фактом, це правило спрацьовує і його частина ТО (дія) виконується. Правило, яке спрацювало може змінити множину фактів шляхом додання нового факту. Такий механізму виведення ще має назву процедура

«зіставлення-спрацьовування». Цикл виведення через процедуру «зіставлення-спрацьовування» показано на рисунку 3.2.

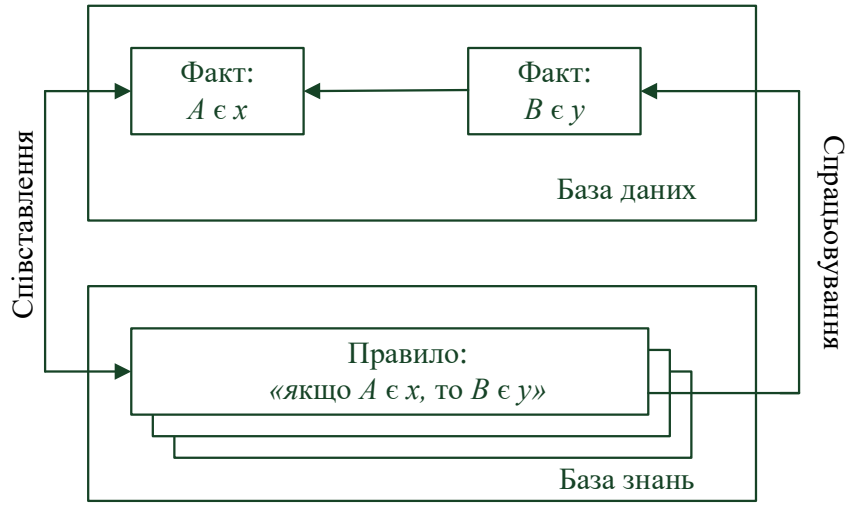


Рисунок 3.2 – Цикл виведення через процедуру «зіставлення-спрацьовування»

Зіставлення частин ЯКЩО правил з фактами утворює *ланцюг виведення*, який показує яким чином інтелектуальна система застосовує правила для отримання висновку.

Для ілюстрації методу виведення через процедуру «зіставлення-спрацьовування» розглянемо приклад побудови ланцюга виведення.

Приклад 3.10 Припустимо, що спочатку база даних містить правила *A*, *B*, *C*, *D* та *E*, а база знань містить лише три правила:

Правило 1 $Y \wedge D \rightarrow Z$

Правило 2 $X \wedge B \wedge C \rightarrow Y$

Правило 3 $A \rightarrow X$

Ланцюг виведення, що демонструє застосування інтелектуальними системами правила для отримання висновку, наведено на рисунку 3.3.

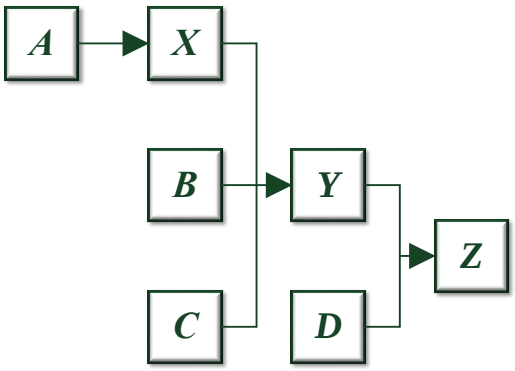


Рисунок 3.3 – Приклад ланцюга виведення

Опишімо процедуру отримання висновку покроково:

1) спрацьовує Правило 3 для виведення факту *X* із вже існуючого в базі даних факта *A*;

2) спрацьовує Правило 2 для виведення факту *Y* із вже існуючих в базі даних фактів *B* та *C*, а також факта *X*, який став відомим на попередньому кроці;

3) спрацьовує Правило 1 для виведення висновку *Z* із вже існуючого в базі даних факта *D* та факта *Y*, який став відомим на попередньому кроці.

Інтелектуальні системи можуть відображати весь ланцюг виведення для пояснення того, яким саме чином було отримано той або інший розв'язок (висновок). Ця можливість є основною частиною пояснювальних здібностей інтелектуальних систем.

Як вже зазначалося, у механізмі виведення слід передбачити спосіб вибору правил, які повинні спрацювати. Існує два принципово відмінних способи, за якими обираються правила:

– *прямий ланцюг виведення* або ланцюг умовно-виведення (Forward chaining);

– *зворотній ланцюг виведення* або ланцюг цілі-виведення (Backward chaining).

Продукційні системи, в яких спочатку аналізується антецендетна частина (тобто умова) є системами з антецендент-вивідною архітектурою. Якщо під час виведення в системі спочатку аналізується консеквентна частина (тобто дія) така система є системою з консеквент-вивідною

архітектурою. Розглянемо приклад продукційної системи з консеквент-вивідною архітектурою.

Приклад 3.11 Припустимо, що спочатку база даних містить правила *A* та *F*, а база знань містить правила:

Правило 1 $A \wedge B \wedge C \rightarrow D$

Правило 2 $D \wedge F \rightarrow G$

Правило 3 $A \wedge J \rightarrow G$

Правило 4 $B \rightarrow C$

Правило 5 $F \rightarrow B$

Правило 6 $L \rightarrow J$

Правило 7 $G \rightarrow H$

Припустимо, ціль виведення – доведення істинності *H*. Опишімо процедуру отримання висновку покроково:

1 перевірка находження факта *H* в базі даних: *нема*;

2 спроба виведення *H* через правила, що містять *H* в правій частині: *Правило 7*;

3 спроба виведення факта *G*:

а) перевірка наявності факта *G* в базі даних: *нема*;

б) спроба виведення *G* через правила, що містять *G* в правій частині: *Правило 2* та Правило 3;

Зауваження. Будемо вважати, що правила впорядковані за пріоритетом, чим менший номер правила, ти вищий його приоритет.

4 спроба виведення фактів *D* та *F*:

а) факт *D* є істинним фактом через *Правило 1*:

– факт *A* є істинним, оскільки він *міститься в базі даних*;

– факт *B* є істинним через *Правило 5* (факт *F* міститься в базі даних);

– факт *C* є істинним через *Правило 4* (факт *B* було доведено на попередньому кроці);

б) факт *F* є істинним, оскільки він *міститься в базі даних*;

5 спроба виведення факта *G*: факт *G* є істинним через *Правило 2*;

6 спроба виведення факта *H*: факт *H* є істинним через *Правило 7*.

Практична робота 5. Прямий та зворотній ланцюг міркувань

За використання прямого ланцюга міркувань розв'язується наступна задача: за відомих умов знайти наслідки. За використання зворотного ланцюга міркувань розв'язується задача: за відомих наслідків знайти умови, що їх спричинили.

Розглянемо міркування за зворотного ланцюга міркувань на прикладі розв'язання задачі призначення на посаду за використання дерева рішень (рис. 3.4).

В даному випадку дерево рішень є спеціальним графом представлення рішень, що має вершини двох типів:

1 *вершина рішення,* яка містить питання, та з якої виходять гілки, що відповідають можливим варіантам відповіді (на рисунку позначено овалами);

2 *вершина цілі* (висновку), що містить логічні висновки, які відповідають певним варіантам відповідей.

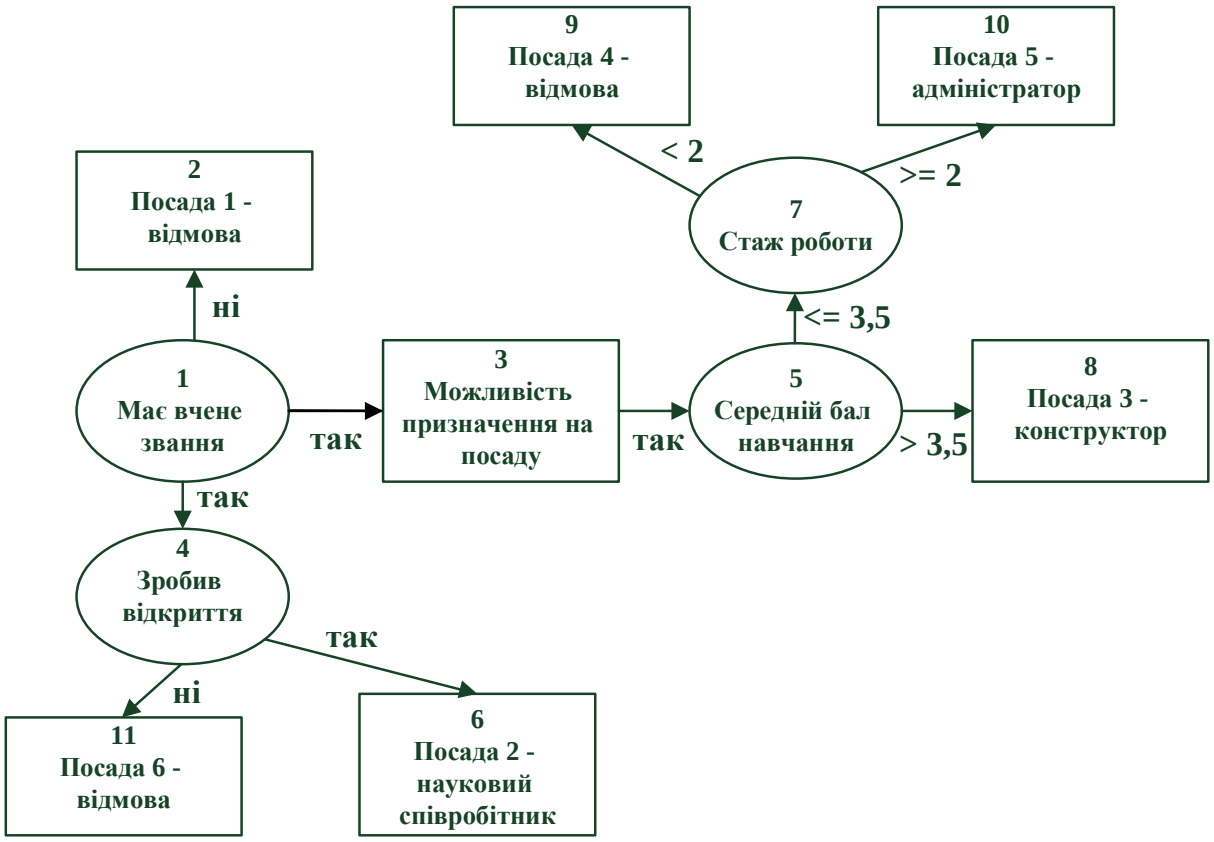


Рисунок 3.4 – Дерево рішень призначення на посаду

Наведене дерево рішень може бути збережене у вигляді правил, наданих у табличному вигляді (табл. 3.3).

Таблиця 3.3 – Правила призначення на посаду

Номер вершини	Змінна логічного висновку	Значення змінної	Номер попередньої вершини	Умова переходу (гілка)	Тип вершини
1	звання	-	-	-	рішення
2	посада 1	відмова	1	«ні»	цілі
3	можливість	так	1	«так»	цілі
4	відкриття	-	1	«так»	рішення
5	середній бал	-	3	«так»	рішення
6	посада 2	науковий співробітник	4	«так»	цілі
7	стаж	-	5	«< 3,5»	рішення
8	посада 3	конструктор	5	«> 3,5»	цілі
9	посада 4	відмова	7	«< 2»	цілі
10	посада 5	адміністратор	7	«> 2»	цілі
11	посада 6	відмова	4	«ні»	цілі

Для збереження результатів виведення створюємо таблицю висновків (табл. 3.4). В початковий момент таблиця порожня.

Припустимо, що висновком виведення є відмова у прийнятті на посаду. Тоді задача за зворотного ланцюга виведення є визначення умов, які призвели до цього результату. Спочатку слід знайти входження: «змінна_посада-значення_відмова». Таких входжень три, які відповідають змінним з номерами 2, 9 та 11.

Таблиця 3.4 – Зворотній ланцюг виведення

Номер етапу	Номер поточної змінної (вершини)	Номер та значення попередньої змінної (вершини)
Входження через змінну 2		
1	2: «посада 1» = «відмова»	1: «звання» = «ні»
2	1: «звання»	-
Входження через змінну 9		
1	9: «посада 4» = «відмова»	7: «стаж» = «< 2»
2	7: «стаж»	5: «середній бал» = «< 3,5»
3	5: «середній бал»	3: «можливість» = «так»
4	3: «можливість»	1: «звання» = «так»
5	1: «звання»	-
Входження через змінну 11		
1	11: «посада 6» = «відмова»	4: «відкриття» = «ні»
2	4: «відкриття»	1: «звання» = «так»
3	1: «звання»	-

Наведена таблиця виведення побудована за наступним алгоритмом:

Крок 1 Визначити змінну логічного висновку та її значення.

Крок 2 Знайти перше входження цієї змінної із завданим значенням та типом вершини:

– якщо входження знайдено, привласнити параметру «Входження через змінну» номер змінної входження;

– якщо входження не знайдено, ціль вважається недосягнутою.

Крок 3 Визначити попередню вершину:

– якщо така вершина визначена, параметру «Номер поточної змінної (вершини)» привласнюємо значення параметру «Номер та значення попередньої змінної (вершини)», а параметру «Номер та значення попередньої змінної (вершини)» привласнюємо номер та значення гілки, яка призвела до поточної вершини;

– якщо така вершина не визначена, переходимо до Кроку 5.

Крок 4 Зробити попередню вершину поточною та перейти до Кроку 3.

Крок 5 Знайти наступне входження цієї змінної із завданим значенням та типом вершини:

– якщо входження знайдено, привласнити параметру «Входження через змінну» номер змінної входження та перейти до Кроку 3;

– якщо входження не знайдено, ціль вважається досягнутою.

Розглянемо міркування за прямого ланцюга міркувань на прикладі розв'язання задачі призначення на посаду за використання дерева рішень (рис. 3.4).

В даному випадку починаємо з вершини 1 та просуваємося по відповідних гілках. Наприклад, 1: «звання» = «так» → «відкриття» = «ні» → → «посада 11» = «відмова».

Задачі для самостійного розв'язання

Задача 1 Завдано двомісну форму висловлювання «x є дільником y»; припустимо x приймає значення з множини $M_x = \{1;2;3\}$, а y – з множини $M_y = \{2;4;6;8\}$. Знайти множину істинності предикатів $P_1(x; y)$ та $P_2(y; x)$.

Задача 2 Подати у вигляді формул предикати, завдані наступними формами висловлювань: «існує хтось, хто прочитав всі книги», «нема нікого, хто б прочитав всі книги», «існує книга, яку прочитали всі», «не існує книги, яку б всі прочитали».

Задача 3 Подати у вигляді формули предикат, завданий формамою висловлювань: «якщо z є батьком x та y, то x є братом або сестрою y».

Задача 4 Навести приклади одно-, дво-, три- та чотирьохмісних предикатів.

Задача 5 Перевести у предварену нормальну форму формулу $\forall x\, \exists y\, P(x, y) \wedge \neg(\exists x\, \forall y\, Q(x, y))$.

Задача 6 Перевести у предварену нормальну форму формулу $\forall x\, (\neg\, P(x) \to \exists y\, (\neg Q(x, y))) \to (Q(z) \to P(z))$.

Задача 7 Довести тотожність $\neg(p \to q) \equiv p \wedge \neg q$.

Задача 8 Надано твердження «Сократ – людина», «Людина – жива істота», «Всі живі істоти смертні». Методом резолюцій довести твердження «Сократ смертний».

Задача 9 За даними практичної роботи 5 «Прямий та зворотній ланцюг міркувань» (рис. 3.4) побудувати прямий та зворотній ланцюги виведення для входження «змінна_посада-значення_адміністратор».

Задача 10 За даними практичної роботи 5 «Прямий та зворотній ланцюг міркувань» (рис. 3.4) побудувати прямий та зворотній ланцюги виведення для входження «змінна_посада-значення_науковий співробітник».

Питання для самоконтролю

1 В чому полягає необхідність використання мови логіки предикатів, як способу подання знань?

2 Дати визначення поняття «предикатний символ». Навести приклади предикатів.

3 Дати визначення поняття «функціональний символ». Навести приклади структур.

4 Дати визначення понять «сигнатура» та «терм».

5 Дати визначення поняття «предикат». Поясніть сутність цього поняття на прикладі.

6 Дати визначення понять «впорядкована n-ка», «компонента» та « декартовий добуток». Яким чином вони використовуються у концепції предикатів?

7 Які існують способи завдання предикатів?

8 Дати графічне пояснення сутності та способам завдання предикатів.

9 В чому полягає відмінність між предикатом-властивістю та предикатом-відношенням? Навести приклади обох видів предикатів.

10 В чому полягає призначення кванторів? Навести приклади використання кванторів у висловлюваннях.

11 Дати визначення поняття «формула логіки першого порядку».

12 З яких процедур складається процес інтерпретації формул логіки першого порядку?

13 За яких умов формула логіки першого порядку є несуперечливою, суперечливою, загальнозначимою, логічним наслідком множини формул?

14 Навести закони логіки першого порядку.

15 З яких кроків складається процедура надання формулам предвареної нормальної форми?

16 В чому полягає принцип резолюції? Яким чином він використовується?

17 Дати визначення поняття «резольвента двох диз'юнктів».

18 Навести особливості методу резолюцій.

19 Які є особливості логічних методів подання знань?

20 Яким чином завдається продукційна система в загальному випадку?

21 Дати вербальне та формульне визначення поняття «продукція».

22 Навести принципи обрання наступної продукції під час виведення знань.

23 Які існують стратегії управління виведенням?

24 Навести схему циклу виведення через процедуру «зіставлення-спрацьовування».

25 В чому полягає різниця між прямим та зворотнім ланцюгами міркувань?

26 З вершин яких типів складається дерево рішень?

27 З яких кроків складається процедура міркувань за звороьнім ланцюгом?

РОЗДІЛ 4
ПОДАННЯ ЗНАНЬ В ВИГЛЯДІ СЕМАНТИЧНИХ МЕРЕЖ ТА ФРЕЙМІВ

4.1. Семантичні мережі

За словником іншомовних слів слово «семантика» (від грецького σημαντικός – означальний) має три тлумачення: перше – це розділ мовознавства, що вивчає значення слів та виразів і зміну цих значень; друге – це значення (слова або виразу); третє – це розділ логіки, що вивчає відношення виразів логічної мови до позначуваних ними об'єктів і змісту, який вони виражають (використовується в металогіці). Наука семантика – це наука, що встановлює відношення між символами та об’єктами, які вони позначають, тобто це наука, яка визначає сенс знаків.

Зазвичай під семантичною мережею розуміють систему знань у вигляді орієнтованого графу, вершини якого відповідають об'єктам (поняттям), а дуги – відносинам між об'єктами (приклад семантичної мережі наведено на рисунку 4.1).

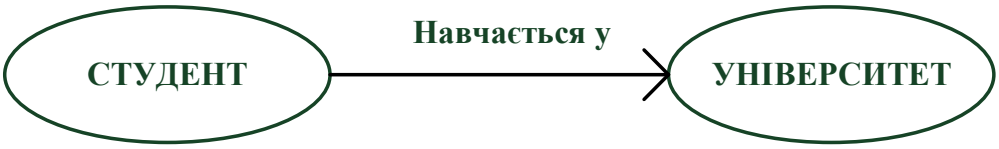


Рисунок 4.1 – Приклад семантичної мережі

В якості об’єктів зазвичай приймаються абстрактні поняття або конкретні об’єкти реального світу (події, дії, узагальнені поняття або властивості об'єктів). Відносини – це зв’язки, які мають різний сенс, наприклад: «це», «має частиною», «належить», «полюбляє», «кохає», «є частиною», «АКО» – належить до класу (від англ. «a kind of»), тощо. Характерною відмінністю семантичних мереж є обов’язкова наявність зв’язків трьох типів:

1 КЛАС (елемент класу або АКО), наприклад, метелик належить до класу безхребетних членистоногих тварин, за цим типом зв’язків об’єкт наслідує властивості класу;

2 ВЛАСТИВІСТЬ (атрибутивні зв'язки), наприклад, у метелика є властивість – колір, ця властивість може мати значення «жовтий»;

3 ПРИКЛАД (значення властивості) прикладом метелика може бути Ліса Патрокл (Lyssa Patroclus).

Відповідно до типів відношень між поняттями семантичні мережі мають наступну класифікацію:

1) за кількістю типів відносин:

– однорідні (з одним типом відносин);

– неоднорідні (з різними типами відносин);

2) за типами відносин:

– бінарні (відносини пов'язують два об'єкти);

– *N*-арні (відносини пов'язують більше двох понять).

Найчастіше в семантичних мережах використовуються наступні види відношень:

1) «частина-ціле» (ієрархічні відносини, що визначають підклас класу або елемент множини);

2) «виробляє», «впливає» (функціональні зв’язки, які визначаються подібними дієсловами);

3) «більше», «менше», «дорівнює», «небільше», «неменше» (кількісні відносини, що позначають порівняння за кількісними показниками);

4) «далеко від», «близько до», «за», «під», «над» (просторові відносини);

5) «раніше», «пізніше», «протягом» (часові відносини);

6) «мати властивість», «мати значення» (атрибутивні відносини);

7) «та», «або», «ні», «тоді і тільки тоді» (логічні відносини).

Наведемо семантичну мережу для прикладу про метелика, наведеного вище, в описі обов’язкових типів зв’язків (рисунок 4.2).

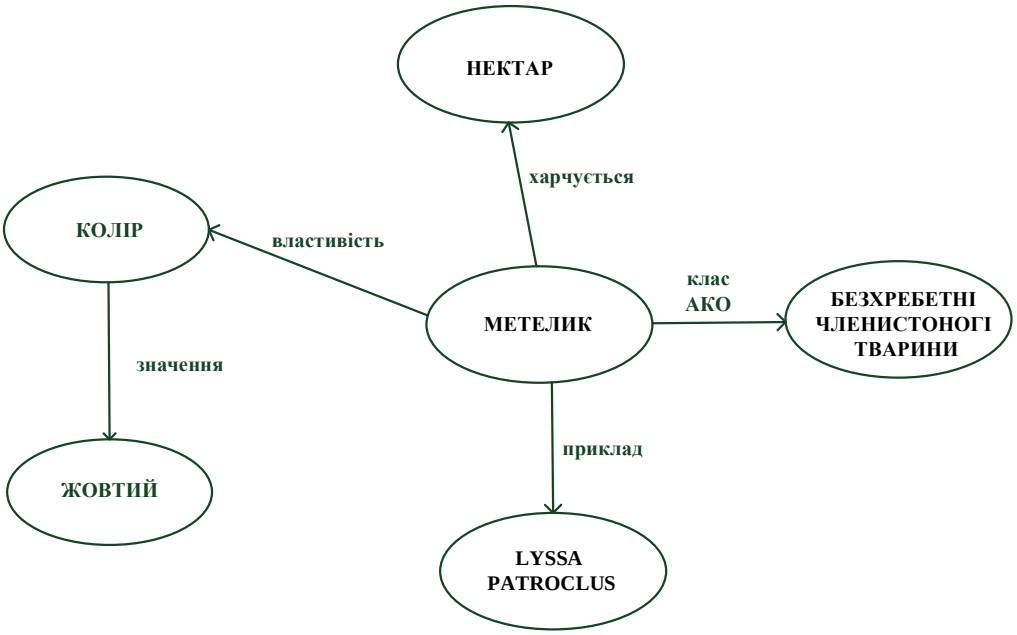


Рисунок 4.2 – Семантична мережа «Метелик»

Іншим прикладом семантичної мережі може бути семантична мережа «Модель» (рисунок 4.3), яка описує ситуацію розробки студенткою Коваленко Г.В. імітаційної моделі матеріальних потоків виробничого підприємства. Модель побудована за допомогою методу виробничої динаміки, яка була розроблена Дж. Форестерм. Ця методологія передбачає представлення опису моделі в математичному та в графічному вигляді.

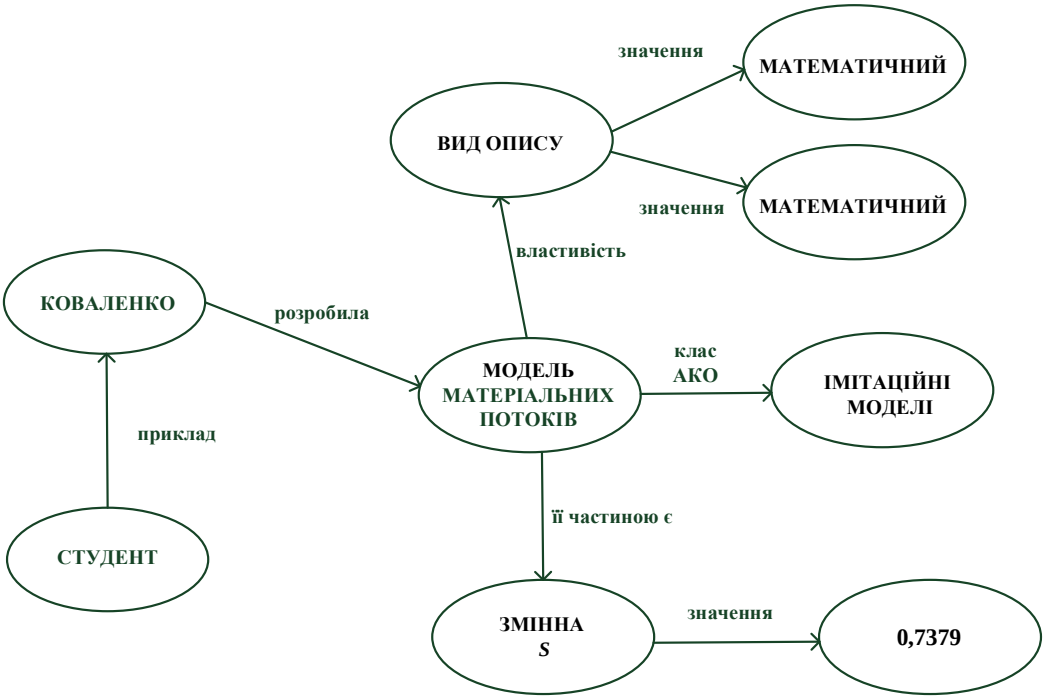


Рисунок 4.3 – Семантична мережа «Модель»

Приклади семантичних мереж наведені з метою підкреслити різноманітність предметних областей, в яких може застосовуватися даний метод представлення даних.

Представлення даних за допомогою семантичних мереж відповідають сучасним уявленням про організацію довгострокової пам’яті людини. Недоліком такого представлення даних є складна процедура пошуку висновка. В базах даних семантичних мереж пошук відбувається шляхом порівняння фрагмента мережі, який завдає користувач, з тими фрагментами, які вже існують в базі даних.

Для реалізації семантичних мереж розроблені спеціальні програмні засоби: NET – комп'ютерні мови програмування, які використовуються для створення бібліотек та програм, що задовольняють вимогам Common Language Infrastructure, SIMER+MIR – засіб для формування моделі предметної області, орієнтований на предметні області з нечітким описом об’єктів).

Прикладами використання семантичних мереж є експертні системи, які застосовують їх як мову представлення знань: PROSPECTOR, CASNET, TORUS.

4.2 Фрейми

Теорія фреймів була запропонована та докладно описана в 1975 році американським вченим в галузі штучного інтелекту Марвіном Лі Мінським (Minsky M.: A Framwork for Representing Knowledge in The Phychology of Computer Vision, P.H. Winston (ed.), McGraw-Hill, 1975). Теорія фреймів належить до психологічних понять, які торкаються сприйняття та усвідомлення того, що ми чуємо та бачимо.

Теорія фреймів ґрунтується на сприйнятті фактів шляхом співставлення інформації отриманої зовні з конкретними елементами та значеннями, а також стереотипами, тобто рамками, визначеними для кожного концептуального об’єкта в нашій пам’яті. Структурами, якими представлені ці рамки є фрейми.

Фрейм – абстрактний образ для представлення деякого стереотипу сприйняття. Коли людина чує якесь слово її уява малює певний стереотипний образ.

Уявіть людину, яка не знає що таке кішка (наприклад мала дитина), чуючи слово «кішка» в мозку такої людини не виникає жодного образу. Дати людині поняття про об’єкт реального світу – кішку (навчити людину) можна двома способами. Перший спосіб полягає в представлені прикладів намальованих або живих кішок або фотографій кішок з відповідними коментарями. Другий – дати опис цього об’єкта: що це тварина середнього розміру, яка має хутро, два вуха, чотири лапи та довгий хвіст. У випадку навчання поняттю «кішка» перший спосіб є ефективнішим (але за умови надання достатньо великої кількості прикладів), другий формує більш абстрактний образ, який може через недоліки сприйняття або опису сформувати образ дещо не зовсім адекватний реальному об’єкту.

Будь-яким чином в процесі навчання абстрактний образ (стереотип або рамка) потрібного об’єкта буде сформований. Цей абстрактний образ буде мати певні атрибути та їх значення. Наприклад, атрибут «розмір» буде мати значення «середній», атрибут «хутро» мати значення «є», «кількість лап» дорівнювати 4, «хвіст» мати значення «є», «кількість вух» дорівнювати 2, тощо. Навчаючись далі людина дізнається більше про цей об’єкт і значення атрибутів можуть корегуватися (наприклад, якщо людина дізнається про породу Сфінкс, у якої хутро відсутнє, або породу Мейн-кун, розмір якої більший за середній) або кількість атрибутів може збільшуватися (наприклад якщо в школі дитині розповіли що кішки належать до класу ссавців). В цьому випадку прийнято навіть так і казати, що навчання «розширює рамки сприйняття» або «ламає стереотипи».

У своїй роботі Мінський писав, що фрейм (рамка) – це одиниця представлення знань, яку було запам’ятовано в минулому, деталі якої можуть змінюватися відповідно до поточної ситуації. Кожен фрейм може бути доповнений різною інформацією, яка може стосуватися:

- способів використання даного фрейму;
- наслідків такого використання (отриманих висновків);
- дій, які необхідно виконати за справдження прогнозів, тощо.

Кожен фрейм можна розглядати як мережу, яка складається з вершин та відношень. На найвищому рівні фрейму представлена інформація: факт, що стосується стану об’єкта, який зазвичай вважається істинним. На наступних рівнях розташована множина термінальних слотів, які обов’язково повинні бути заповнені конкретними значеннями та даними. Для наведеного вище прикладу опису кішки можна навести дану мережу у вигляді показаному на рисунку 4.4.

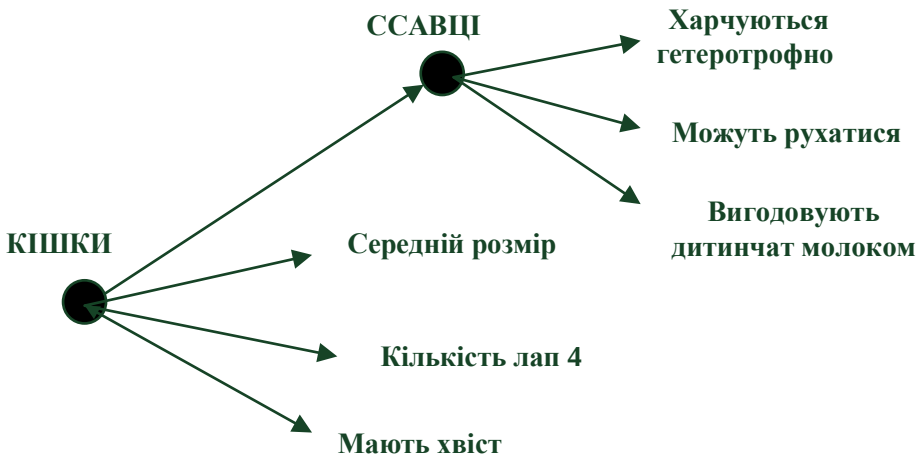


Рисунок 4.4 – Фрейм «кішка» у вигляді частини мережі

В одній системі різні фрейми можуть мати однакові слоти, завдяки чому можливе зв’язування інформації, отриманої з різних точок зору. Приклад фреймової системи, що залежить від точки зору наведено на рисунку 4.5. Цей приклад наведено у роботі М. Мінського для представлення даних про арку, що складається з трьох елементів: дві колони *B* та *C*, а також елементу звіда *A*.

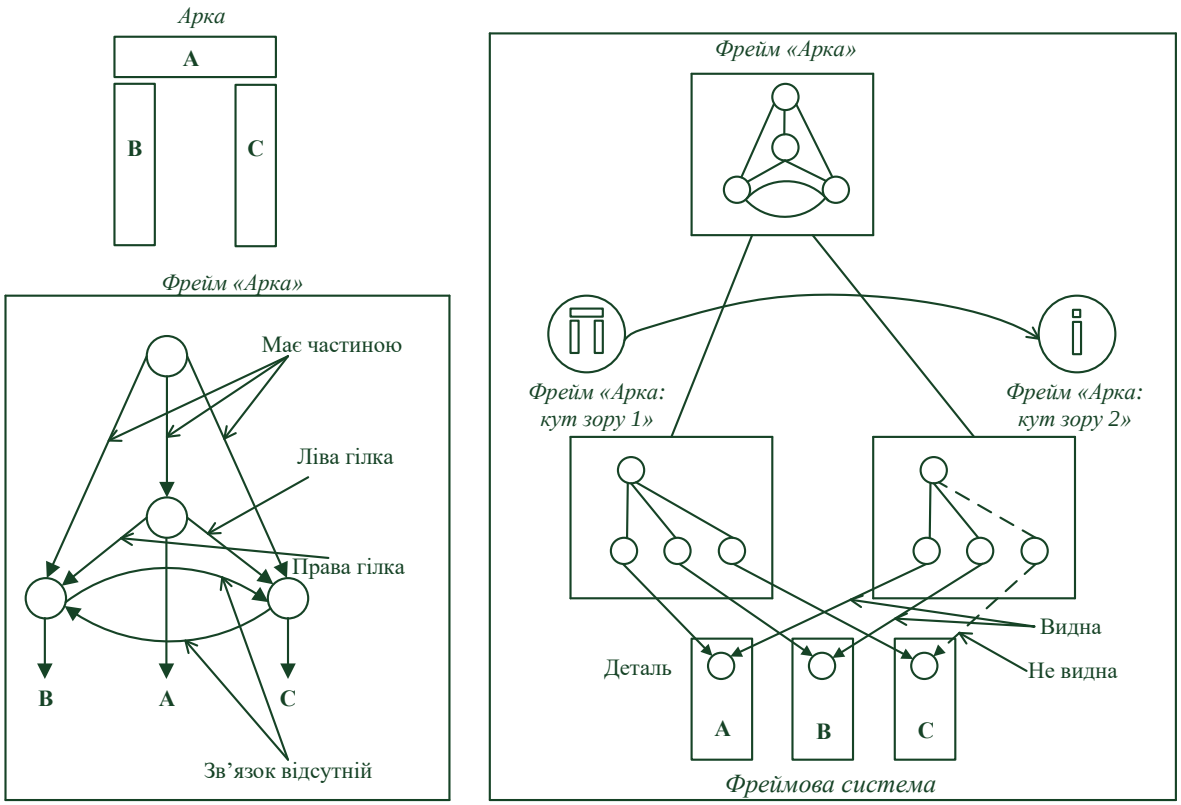


Рисунок 1.5 – Приклад фреймової системи, що залежить від точки зору

Кілька слотів одного фрейма як правило визначаються заздалегідь значеннями за замовченням (так фреми підкласу, ті що пов'язані відношенням АКО, наслідують певні значення класу). Такий метод широко використовується для представлення інформації загального типу. Значення за замовченням не пов'язано жорстко з певним слотом, його можна змінювати на нову інформацію. Використовуючи приклад опису кішки для представлення даних про кішок породи Мейн-кун слот «розмір» за замовченням буде мати значення «середній», але може бути змінений на «великий».

Значення за замовченням може бути використано в якості змінної або логічного обмеження. Так фрейм «учень» є підкласом класу фрейма «дитина», для якого слот «вік» має значення «від 0 до 18 років», тоді значення цього слоту для фрейма «учень» має потрапляти у відповідний інтервал і може мати значення «від 6 до 18 років».

В якості значення слота може бути використані: ім'я іншого фрейма, числа, формули, текст природньою мовою або програмний код.

Можливі наступні способи отримання значень слотом фрейма-екземпляра:

1) за замовченням від фрейма-зразка;
2) через успадкування властивостей від фрейма, який указано в слоті АКО (наслідування властивостей класу);
3) за формулою, яка указана в слоті;
4) шляхом виконання приєднаної до слота процедури;
5) через діалог з користувачем;
6) з бази даних.

Для фреймового (об'єктно-орієнтованого) подання знань характерним є використання механізму успадкування атрибутів, коли значення атрибутів передаються по ієрархії від класів, що знаходяться на вищих рівнях до класів, що знаходяться на більш низьких рівнях. Мережа фреймів, що описує будь-яку предметну область, є ієрархічною структурою, в якій фрейми з'єднуються за допомогою родовидових зв'язків. На верхньому рівні ієрархії знаходиться фрейм, що містить найбільш загальну інформацію, істинну для всіх інших фреймів. Розглянемо мережу фреймів для ілюстрації отримання слотами значень шляхом наслідування властивостей класу (рисунок 4.6).

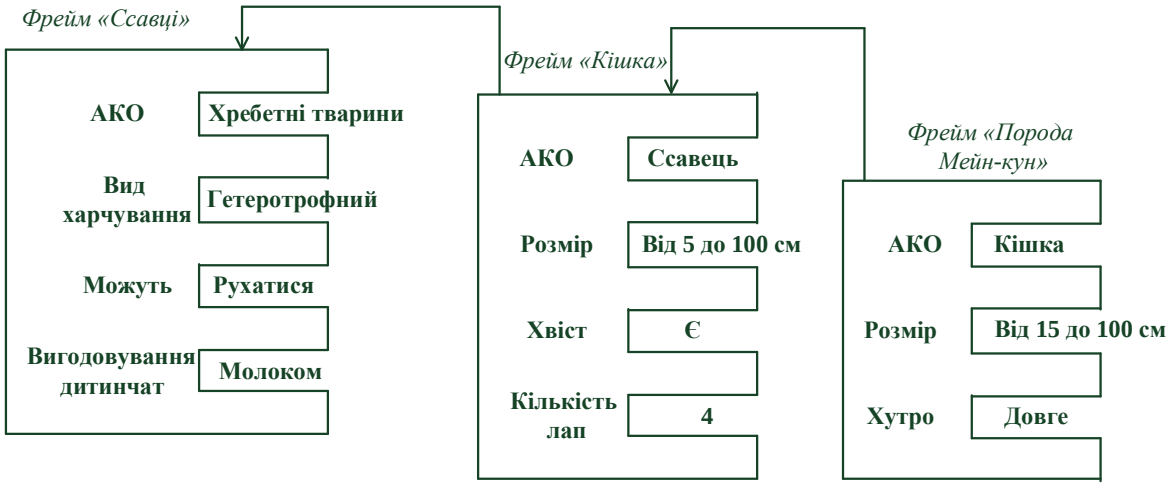


Рисунок 4.6 – Мережа фреймів з наслідуванням властивостей класу

Фрейм «Порода Мейн-кун» успадковує властивості фреймів «Ссавці» та «Кішка», які знаходяться на більш високому рівні ієрархії. Це відбивається на процедурі виводу висновків. Так, на питання: «Чи

вигодовують кішки породи Мейн-кун дитинчат молоком?» слід відповісти «Так», оскільки це властивість всіх ссавців, яка успадкована також кішками і зокрема кішками породи Мейн-кун.

Наслідування властивостей може бути частковим. Наприклад розмір кішок породи Мейн-кун більший і тому вказується в окремому слоті відповідного фрейма. Слоти, значення яких залишаються за замовченням, у фреймів-екземплярів можуть бути не відображенні. У фреймі «Порода Мейн-кун» значення слота «розмір» слід уточнити, тому він повторений.

Фрейм як модель представлення даних є досить універсальною, оскільки дозволяє відобразити всю різноманітність та багатогранність знань. Таке відображення відбувається через:

1 Фрейми-структури, які використовуються для означення об'єктів та понять (наприклад, інформаційна система, академічна група, модель);

2 Фрейми-раси, які використовуються для означення суб'єктів та їх груп (наприклад, програміст, менеджер, студент);

3 Фрейми-сценарії, які використовуються для означення процесів та явищ (наприклад, розробка моделі, банкрутство, складання сесії);

4 Фрейми-ситуації, які використовуються для означення станів певних об'єктів або суб'єктів (наприклад, робочий стан, відмова в роботі, радість).

Структура фрейму може бути подана одним з наступних способів.

Спосіб 1. У вигляді переліку властивостей.

назва фрейму:
ім'я 1 слота: значення 1 слота
ім'я 2 слота: значення 2 слота
…….
ім'я n слота: значення n слота

Спосіб 2. У табличному вигляді (таблиця 4.1).

Таблиця 4.1 – Подання структури фрейма

Назва фрейму			
Ім’я слота	Значення слота	Спосіб отримання значення слота	Процедура, що приєднана до слота
…	…	…	…

До переваг моделі подання знань у вигляді фреймів належать наступні її властивості:

– модель відбиває концепцію організації пам’яті людини;

– модель є гнучкою та надійною;

– модель використовує родовидові зв’язки, які змінюються не часто і тому опис предметної області має небагато винятків;

– модель може враховувати невизначеність опису знань шляхом неповного заповнення значень слотів, а також фреймова модель здатна здійснювати припущення про значення даних завдяки механізму наслідування властивостей в ієрархії узагальнення.

Недоліком моделі подання знань у вигляді фреймів є її відносна складність.

Існують спеціальні мови представлення знань в мережах фреймів: FRL (Frame Representation Language), KRL (Knowledge Representation Language), Kappa, а також фрейм-орієнтовані експертні системи: ANALYST, MODUS, TRISTAN, ALTERID.

Задачі для самостійного розв’язання

Задача 1 Виконати порівняльний аналіз подання знань у вигляді семантичних мереж та фреймів.

Задача 2 Побудувати семантичну мережу «Заклад вищої освіти».

Задача 3 Побудувати фреймову ієрархічну структуру для поняття «студент» з вказанням наслідування класу.

Задача 4 Побудувати фреймову ієрархічну структуру для поняття «стілець» з урахуванням різних точок зору на це предмет.

Задача 3 Побудувати фреймову ієрархічну структуру будь-якого абстрактного поняття «студент» з урахуванням різних точок зору на це поняття.

Питання для самоконтролю

1 Які визначення поняття «семантика» Вам відомі? Навести приклади семантичної мережі.

2 Наявність зв’язків яких трьох типів є характерною відмінністю семантичних мереж? Навести приклади кожного з типів зв’язків.

3 Навести класифікацію семантичних мереж за типами відношень між поняттями.

4 Які види відношень використовуються в семантичних мережах найчастіше? Навести приклади.

5 Навести приклад графу семантичної мережі з неменше як 10 вершинами.

6 В чому полягає недолік представлення даних з використанням семантичних мереж?

7 Які розроблені спеціальні програмні засоби для реалізації семантичних мереж?

8 На якому принципі ґрунтується теорія фреймів?

9 Дати визначення поняття «фрейм».

10 Якого типу інформацією може бути доповнений фрейм?

11 З яких елементів складається мережа, яка описує фрейм? Навести приклад.

12 Які існують способи отримання значень слотом фрейма-екземпляра?

13 Навести приклад фрейми з'єднуються за допомогою родовидових зв'язків.

14 Які види фреймів використовують для відображення всієї різноманітності знань?

15 Які Вам відомі способи подання структури фреймів?

16 Якими властивостями забезпечуються переваги моделі подання знань у вигляді фреймів?

РОЗДІЛ 5
ЕКСПЕРТНІ СИСТЕМИ

5.1 Структура та класифікація експертних систем

Експертні системи (ЕС) – це складні програмні комплекси, які акумулюють знання та емпіричний досвід фахівців в конкретних предметних областях, та які тиражують ці знання та досвід під час консультування менш кваліфікованих користувачів.

Якщо розглядати ЕС з точки зору інтелектуальних систем та інженерії знань можна надати наступне визначення.

ЕС – це втілення в ЕОМ компоненти досвіду експерта, яка заснована на знаннях, в такій формі, що машина може дати інтелектуальну пораду або прийняти інтелектуальне рішення відносно функції, що обробляється.

Головною властивістю ЕС є здібність системи за вимогою пояснити хід своїх міркувань зрозумілим для користувача чином. Ця властивість грає подвійну роль: з одного боку підвищує довіру користувача до наданих порад, з іншого дозволяє експертові визначити правильність використання наданих ним знань. Забезпечується ця властивість в результаті програмування, яке засноване на формальних правилах.

Структура ЕС складається з таких базових елементів:

1 користувацький інтерфейс – комплекс програм, який реалізує діалог користувача з ЕС, та є необхідним для правильної передачі відповідей користувачеві в зручній для нього формі та здійснення експертом маніпуляцій зі знаннями;

2 база знань – ядро ЕС, яке містить процедурні знання про предметну область, які записані на машинному носії у формі, яка є зрозумілою користувачу та експерту;

3 база даних – блок, який містить фактичні знання про поточну ситуацію;

4 механізм логічних виведень (інші назви: машина виведення, інтерпретатор, блок логічного виведення) – блок ЕС, який містить правила отримання висновків;

5 модуль порад і пояснень – програма, яка надає інтелектуальну пораду або розв’язок за допомогою механізму логічного виведення, а також коментарі, що пояснюють хід наведених міркувань, надають користувачеві відповіді на питання: «Як була отримана та або інша порада?», «Чому система прийняла те або інше рішення?»;

6 модуль набуття знань (або інтелектуальний редактор бази знань) – програма, яка дозволяє інженеру зі знань створювати ЕС в діалоговому режимі, містить правила отримання від експерта нових знань.

Структуру ЕС та взаємодію операторів з нею можна подати графічно у вигляді схеми (рис. 5.1).

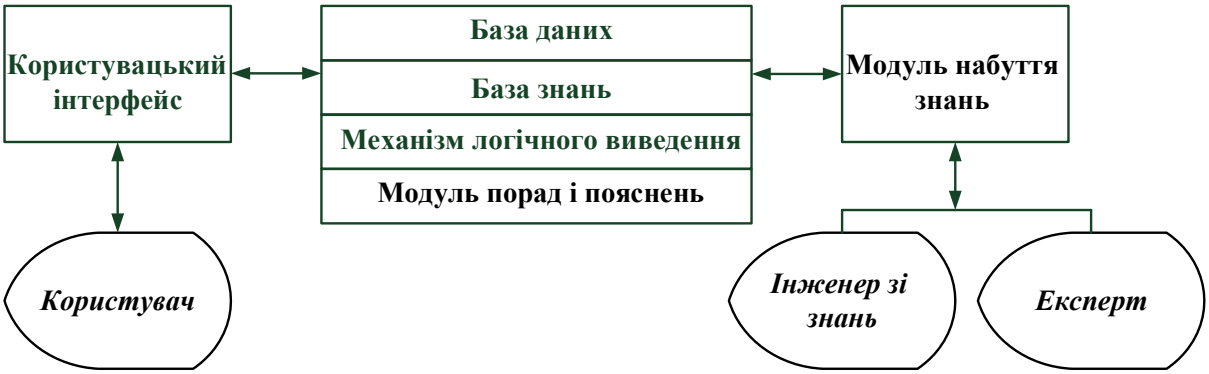


Рисунок 5.1 – Структура ЕС

Користувач – фахівець у відповідній предметній області, для роботи якого призначена ЕС.

Інженер зі знань – фахівець в області штучного інтелекту, що виконує роль посередника між експертом та базою знань ЕС.

Експерт – фахівець у відповідній предметній області, що має унікальні знання та досвід, які мають бути перенесені у базу знань ЕС.

Одним з класифікаторів ЕС є *сфери їх застосування* (таблиця 5.1).

Таблиця 5.1 – Сфери застосування ЕС

№ з/п	Сфера застосування	Сутність завдань, що вирішуються
1	Інтерпретація	Побудова описів ситуацій за спостережуваними даними
2	Прогноз	Надання висновків з імовірних наслідків за заданих ситуацій
3	Діагностика	Надання висновків про порушення в системі, виходячи зі спостережень
4	Проектування	Побудова конфігурації об’єктів за певних обмежень
5	Планування	Побудова плану дій для досягнення певної цілі
6	Моніторинг	Порівняння спостережень з критичними точками плану
7	Налагодження	Вироблення рекомендацій щодо усунення несправностей
8	Ремонт	Виконання плану застосування виробленої рекомендації
9	Навчання	Діагностика, налагодження та виправлення поведінки учня
10	Управління	Інтерпретація, прогноз, ремонт та моніторинг поведінки системи

ЕС, що застосовуються для вирішення перелічених типів завдань є інтерпретуючими, прогнозуючими, діагностуючими, тощо.

За *типом моделі предметної області* ЕС поділяють на:

– статичні (знання, що інтерпретуються ЕС не змінюються у часі);

– квазідинамічні (знання, що інтерпретуються ЕС змінюються у часі з певною періодичністю);

– динамічні (знання інтерпретуються ЕС неперервно, а сама ЕС працює в режимі реального часу).

У статичній предметній області її модель (тобто набір сутностей, їх атрибути, зв’язки між ними, тощо) залишається незмінною протягом всього часу виконання завдання. Якщо дана умова не виконується, то предметна область є динамічною.

За *видами даних і знань*, що використовуються ЕС поділяються на:

1 системи з детермінованими (чітко визначеними) знаннями;

2 системи з невизначеними знаннями.

Виокремлюють наступні типи невизначеності знань (даних):

– неповнота (відсутність),

– недостовірність (неточність вимірювання),

– двозначність (багатозначність понять),

– нечіткість (якісна оцінка замість кількісної).

За *способом формування рішення* ЕС поділяються на:

1 аналітичні (здійснюють обрання рішень з багатьох відомих альтернатив, тобто визначають характеристики об’єктів);

2 синтетичні (здійснюють генерацію невідомих рішень, тобто формують об’єкти).

За *ступенем інтерпретації знань* ЕС поділяються на:

1 автономні – ЕС, які призначені для розв’язання специфічних задач, в яких не застосовується традиційні методи обробки даних, таку як розрахунки, моделювання, тощо;

2 гібридні – ЕС, які застосовують стандартні пакети прикладних програм та засоби маніпулювання знаннями.

5.2 Методологія розробки експертних систем

Процес розробки ЕС умовно можна поділити на п’ять основних етапів: ідентифікація, концептуалізація, формалізація, реалізація, випробування. Розглянемо детальніше кожен з цих етапів.

Етап 1 *Ідентифікація*. На цьому етапі визначаються:

– цілі й завдання побудови ЕС;

– предметна область ЕС;

– ресурси необхідні для створення ЕС (час, обчислювальні засоби);

– учасники процесу створення ЕС (наприклад, додаткові експерти).

Етап 2 *Концептуалізація*. На цьому етапі визначаються: основні поняття, відносини й характер інформаційних потоків, необхідних для опису процесу розв'язання задач у даній предметній області.

Результатом етапу концептуалізації проблемної області є її представлення у вигляді наочних графічних схем на об’єктному, функціональному й поведінковому рівні за допомогою відповідних моделей (табл. 5.2).

Таблиця 5.2 – Рівні моделювання етапу концептуалізації

Назва моделі	Предмет відображення моделі	Опис моделі
Об’єктна	Структура предметної області	Відображає фактуальні знання про склад об'єктів, їх властивості та зв'язки. *Елементарною одиницею* є факт, що описує одну властивість або один зв’язок об'єкта у вигляді предикату
Функціональна	Дії та перетворення об’єктів	Відображає перетворення фактів, залежності між ними, що показують, як одні факти утворюються з інших. *Елементарною одиницею* є функціональна залежність між фактами у вигляді імплікації: $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow B$
Поведінкова	Взаємодії об’єктів у часовому аспекті	Відображає зміну станів об’єктів у результаті виникнення деяких подій, що ведуть до виконання певних процедур. Задача полягає у визначенні зв’язків подій з поведінкою об’єктів та зміною їх станів. *Елементарною одиницею* є подія у формі повідомлення, що посилається об'єкту.

Функціональна модель будується шляхом послідовної декомпозиції цілей. Кожній цілі (підцілі) відповідає певна задача (підзадача), яка не може бути вирішена, поки не будуть досягнені її підцілі (вирішені підзадачі), що знаходяться на нижчому рівні. Зазвичай функціональні залежності фактів наводяться графічно у вигляді дерев цілей або графів "І" / "Або", в яких кожен залежний факт являє собою цільову зміну – кореневу вершину, а факти-аргументи, що її визначають, пов'язують з коренем підлеглі вершини.

Етап 3 *Формалізація*. На етапі формалізації бази знань здійснюється:

– вибір методу подання знань;

– проектування логічної структури бази знань;

– вибір інструментальних засіб розробки ЕС;

– введення основних понять та відносин в рамках обраного формалізму.

Для розробки ЕС можуть бути використані такі моделі формалізації та подання знань:

– логічна модель: реалізує об’єкти й правила за допомогою предикатів першого порядку, є строго формалізованою моделлю з універсальним дедуктивним і монотонним методами логічного виводу;

– продукційна модель реалізує евристичні методи виведення на правилах і дозволяє:

а) обробляти невизначеності у вигляді умовних ймовірностей або коефіцієнтів упевненості (наприклад, умовні ймовірності Байеса, або методи нечіткої логіки Заде);

б) виконувати монотонний або немонотонний вивід;

– семантична мережа відображає різноманітні відносини об’єктів у вигляді предикатів, ім’я яких є позначеною дугою між двома вузлами графа, які відповідають двом пов’язаним об’єктам;

– фреймова модель використовує для реалізації операційного знання приєднані процедури та всі атрибути об’єктів, які об’єднуються в одну структуру даних – фрейм, значеннями атрибутів якого є дані або дії, спрямовані на отримання цих значень (невизначеність опису знань реалізується в результаті неповного заповнення всіх слотів).

Етап 4 *Реалізація*. На цьому етапі здійснюється:

– наповнення бази знань;

– комбінація та реорганізація формалізованих знань.

Результатом цього етапу є програма-прототип, яку можна виконувати та піддавати контрольним випробуванням.

Етап 5 *Випробування*. На цьому етапі здійснюється оцінка роботи програми-прототипу. Як правило, експерт дає оцінку роботи програми й допомагає інженеру з питань щодо знань в наступних її модифікаціях.

До цього етапу можна віднести також процес підтримки та ведення ЕС, головною задачею якого є витяг знань. Класифікацію методів витягу знань можна подати схематично (рис. 5.2).

Рисунок 5.2 – Класифікація методів витягу знань

До комунікативних пасивних методів належать: спостереження, протокол «думки вголос» (експерт розмірковує вголос під час виконання певних дій), лекції.

До комунікативних активних групових методів належать: мозковий штурм, круглий стіл, рольові ігри.

До комунікативних активних індивідуальних методів належать: анкетування, інтерв'ю, діалог, експертні ігри.

До текстологічних методів належать: аналіз підручників, аналіз літератури, аналіз документів, аналіз інтернет джерел.

Окремим питанням витягу та інтерпретації знань є робота з *нечіткими знаннями*, які складають цілий клас описів якісних характеристик об'єктів. Наприклад, «багато», «мало», «сильний», «дуже сильний», «дорослий».

Більш того, іноді доводиться використовувати знання, які не можуть бути інтерпретовані як істинні або хибні, наприклад, «достовірність певного факту приблизно дорівнює 0,7».

Для роботи з нечіткими даними в 70-х роках XX сторіччя була запропонована нечітка арифметика та нечітка логіка. Головне поняття нечіткої логіки – це «лінгвістична змінна».

Лінгвістична змінна – це змінна, значення якої визначається набором вербальних характеристик певної властивості. Наприклад лінгвістична змінна «зріст» може визначатися словами: «низький», «високий», «середній», «карликовий», «дуже високий», «гігантський».

Кожне значення лінгвістичної змінної визначається нечіткою множиною. Нечіткі множини визначаються на базовому наборі значень або на базовій шкалі.

Наприклад лінгвістична змінна «вік» може мати наступний базовий набір значень: «дитячий», «юнацький», «молодий», «дорослий», «літній». Наведемо приклад базової шкали для значень «дитячий» та «юнацький» (рис. 5.3).

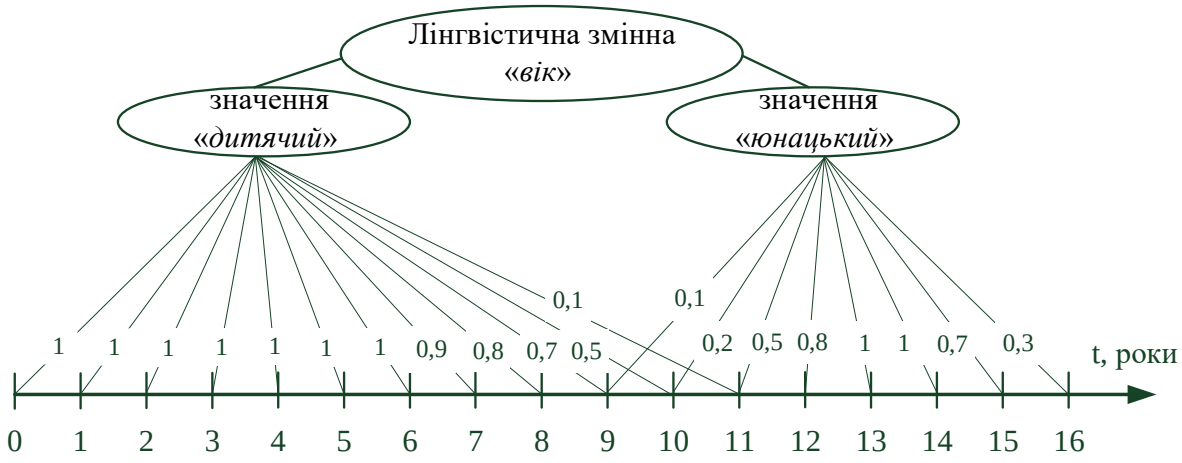


Рисунок 5.3 – Базова шкала значень лінгвістичної змінної

Числа на лініях з’єднання базових значень лінгвістичної змінної зі значеннями років позначають значення функції приналежності нечітких чисел.

Функція приналежності нечітких чисел визначає суб’єктивну ступінь впевненості експерта в тому, що дане конкретне значення базової шкали відповідає предметній нечіткій множині. Позначається функція приналежності $\mu(X)$, $1 \geq \mu(X) \geq 0$.

Нечітка множина – сукупність пар виду $\{\mu(X)/X\}$, де X – значення базової шкали, а $\mu(X)$ – відповідне цьому значенню шкали значення функції приналежності. Наприклад, «дитячий вік» визначається нечіткою множиною {1/0; 1/1; 1/2; 1/3; 1/4; 1/5; 1/6; 0,9/7; 0,8/8; 0,7/9; 0,5/10; 0,1/11}.

Нечітку множину можна також задати графіком її функції приналежності (рис. 5.4).

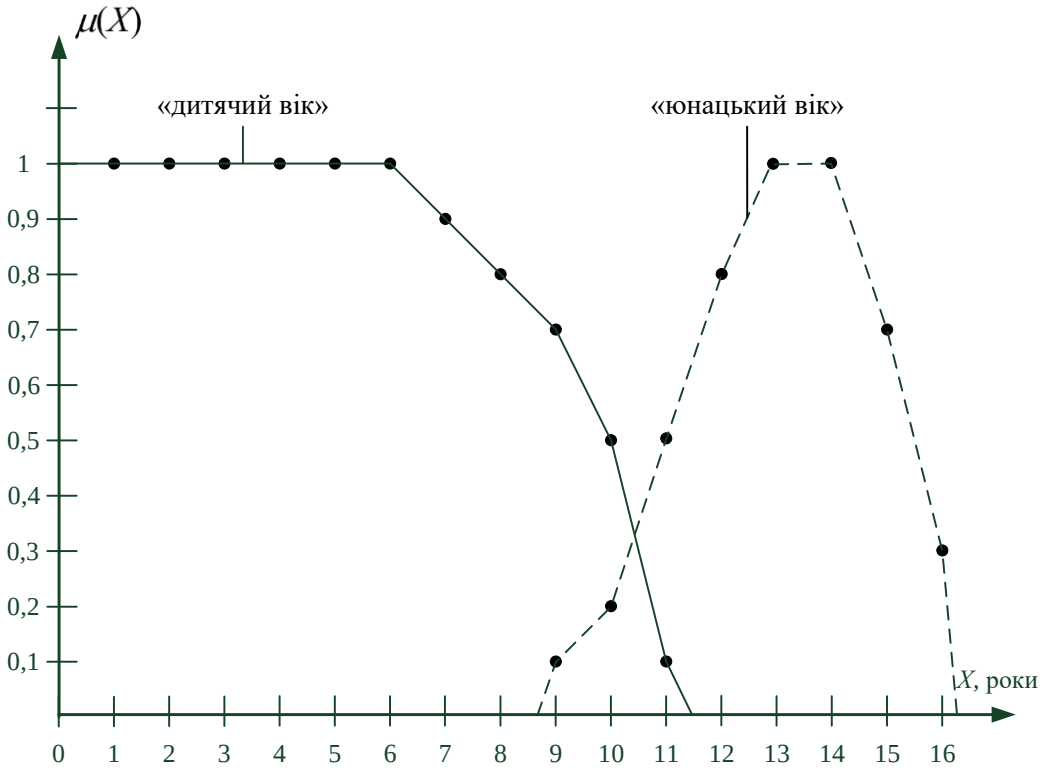


Рисунок 5.4 – Графіки функцій приналежності

На кожному з етапів можливе повернення на кілька етапів назад, завдяки чому ЕС еволюціонує та поступово ускладнюється її організація та вдосконалюється подання знань.

5.3 Інструментальні засоби розробки ЕС

ЕС належать до інтелектуальних систем, інструментальні засоби розробки яких докладно розглянуті в п. 1.6 Інструментальні засоби розробки штучного інтелекту.

Під час розробки ЕС слід враховувати такі її особливості:

1 адаптивність – властивість ЕС, що передбачає можливість покращення програми за рахунок легкої зміни правил користувачем, або шляхом отримання ЕС нових знань в результаті аналізу початкових знань;

2 універсальність відносно проблемної області – властивість, яка дає змогу ЕС виконувати маніпуляції зі знаннями таким чином, щоб стати експертом в іншій проблемні області (наприклад, ЕС з діагностування хвороб за змін медичних знань на знання інженерних споруд може стати ЕС

з діагностування стану інженерних споруд), така властивість досягається універсальністю механізмів аналізу знань та виведення висновків.

На сьогодні відомі три основні види розробки ЕС:

1 ЕС, які виконані у вигляді окремих програм певною алгоритмічною мовою. База знань таких ЕС є частиною програми. Такі ЕС призначені для вирішення завдань в одній фіксованій предметній області. В цьому випадку для створення ЕС використовують або звичайні процедурні мови програмування (PASCAL, C) або спеціалізовані мови штучного інтелекту (LISP, PROLOG).

2 Оболонки ЕС – програмний продукт, засоби якого дозволяють шляхом формалізації та введеня знань представляти та аналізувати знання з різних предметних областей. Прикладами оболонок ЕС є PC+, VP-Expert.

3 Генератори ЕС – потужні програмні продукти, що дозволяють розробляти оболонки для подальшого представлення знань залежно від предметної області. Прикладами генераторів ЕС є KEE, ART.

Ситуація, в якій доцільно та можливо створювати та застосовувати ЕС виникає за наявності наступних умов:

– необхідність символьних міркувань, тобто, наприклад, недоцільно створювати ЕС для ведення розрахунків (знаходження коренів системи алгебраїчних рівнянь, розв’язку оптимізаційних задач, тощо);

– наявність експертів, що є компетентними у відповідній предметній галузі та згодні співпрацювати у створені ЕС;

– складність завдань, що потребують вирішення: завдання повинні бути або достатньо складними (такими, що не можуть бути вирішені звичайними фахівцями), або такими, що є трудомісткими та вирішуються доволі часто;

– обмеженість кола питань, що складають проблемну область, яка запобігає «комбінаторному вибуху» обсягу інформації, необхідної для компетентного вирішення поставлених завдань;

– наявність погодженості експертів щодо фактів та правил виведення висновків, які мають бути застосовані під час створення ЕС;

– достатність початкових даних для перевірки працездатності ЕС в певній предметній області.

Цінність застосування ЕС визначаються наступними трьома аспектами:

1 акумуляція, оперативне уточнення та розповсюдження експертних знань;

2 ефективне розв'язання проблем, складність яких перевищує можливості людини та потребує експертних знань в кількох предметних областях;

3 реалізації колективної пам'яті, що є запорукою якісно кращого вирішення складних та слабоформалізованих проблем.

5.4 Логіка побудови ЕС

Логіку побудови ЕС розглянемо на прикладах, які були описані в книзі Кріса Нейлора «Як побудувати свою експертну систему»[1], перше видання якої датується 1987 роком. В прикладі 5.1 описано статистичний підхід до створення ЕС, в прикладі 5.2 описано процедуру побудови правил виведення висновків.

Приклад 5.1 Припустимо необхідно створити ЕС для відповіді на питання «Чи буде завтра дощ?».

На це питання може бути два варіанти відповіді «ТАК» або «НІ».

Побудуємо експертну матрицю (табл. 5.3), рядки якої відповідають можливим питанням, відповіді на які є фактами для виведення висновку щодо дощової погоди, а стовпці – всі можливі відповіді.

Таблиця 5.3 – Експертна матриця прикладу 5.1

№ з/п	Спостереження	Завтра дощ	Завтра нема дощу
1	Дощ: Сьогодні дощ?	*a*	*b*
2	Холодно: Сьогодні холодно?	*c*	*d*
3	Вітряно: Сьогодні вітер?	…	…
4	Хмарно: Сьогодні хмарно?	…	…
5	Туманно: Сьогодні туманно?	…	…

[1] Нейлор К. Как построить свою экспертную систему: Пер. с англ. – М.: Энергоатомиздат, 1991. – 286 с.

6	…	…	…
7	…	…	…
8	…	…	…
9	…	…	…
10	…	*s*	*t*

Літери *a*, *b*, …, *s*, *t* позначають можливі варіанти правильних відповідей на питання в другому стовпчику, які поки що невідомі.

Задача ЕС – обрати один з двох стовпчиків: «Завтра дощ» або «Завтра нема дощу» для відповіді на зазначені у другому стовпчику питання (1-10).

Припустимо, виходячи із статистичної інформації про спостереження в минулому, експерти визначають, що якщо сьогодні був холодний день, то наступного дня дощитиме у 60% випадків. Тоді $a = 0{,}6$, аналогічно можуть бути визначені відповіді, що позначені іншими літерами.

Таким чином створені база знань та область запитів ЕС. *Область запитів* визначає круг питань, які можна поставити ЕС. В даному випадку область запитів – це погода або, якщо бути точніше, дощ. База знань містить всі знання ЕС про предмет області запитів (поки що база знань є порожньою). Співвідношення бази знань та області запитів можна подати схематично (рис. 5.5).

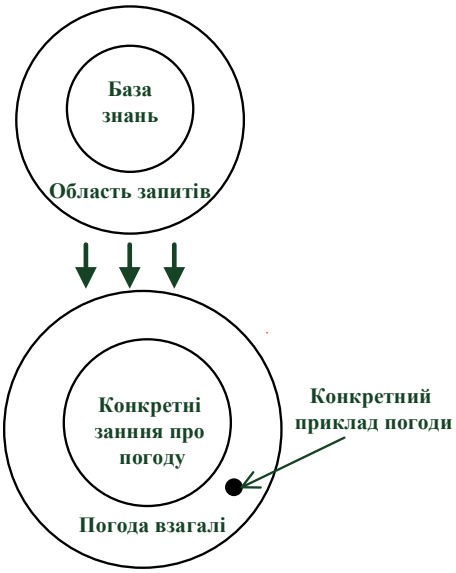


Рисунок 5.5 – Співвідношення понять «база знань» та «область запитів»

Область запитів містить все, що треба було б знати, для того, щоб бути експертом в даній предметній області. В ідеальному випадку база знань повинна співпадати за ємністю з областю запитів, але на практиці в базі знань ЕС міститься лише деяка частина знань з області запитів. Тому, коли виникає конкретне питання, воно має потрапити в область запитів, але воно може не потрапити до бази знань.

Можна розрізнити два типи інформації:

– постійна інформація – це інформація містить незмінні дані, для наведеного прикладу це інформація про погоду взагалі (повинна потрапляти в базу знань);

– змінна інформація – це інформація, що пов'язана зі специфікою задачі, що була поставлена в даний момент, для наведеного прикладу це інформація, яка дозволяє відповісти на питання «Чи буде завтра дощ?» і ця інформація надається для відповіді на це питання саме сьогодні.

Постійна інформація є аналогом програми, тоді змінна інформація є аналогом початкових даних, що вводяться в програму для розв'язання конкретної задачі. Однак межа між постійною та змінною інформацією має бути нечіткою, тобто має бути можливість змінювати постійну інформацію, що забезпечує процес вдосконалення та навчання ЕС в міру накопичення досвіду.

Припустимо, на підставі статистичних даних, було встановлено:

– якщо сьогодні іде дощ, то завтра в 60% випадків буде дощ, а в 40% випадків буде сухо;

– якщо сьогодні сухо, то завтра в 45% випадків буде дощ, а в 55% випадків буде сухо.

Наведена інформація є постійною і має потрапити до бази знань ЕС.

Тоді перш ніж вирішити задачу ЕС має запитати «Чи є сьогодні дощ?», відповідь на це питання є змінною інформацією. Якщо відповідь стверджувальна, то ЕС може надати відповідь «Завтра буде дощ», тому що вона є найвірогіднішою.

Наведене відсоткове значення випадків дощу або сухої погоди завтра є умовними ймовірностями. Тобто, наприклад 0,6 – це ймовірність настання дощу завтра, *за умови*, що дощ іде сьогодні. Тоді, якщо позначити через *A* подію, яка полягає в тому, що завтра буде дощ, а через *B* – сьогодні іде дощ,

то P(*A*/*B*) – ймовірність настання події *A*, обрахована за умови, що подія *B* вже відбулася.

Нагадаємо, якщо P(*A*) – ймовірність настання події *A*, а P(*B*) – ймовірність настання події *B*, то P(*AB*) – *сумісна* ймовірність (настання одночасно події *A* і події *B*); P(*A*/*B*) – *умовна* ймовірність події *A* за визначеної *B*.

Аналізуючи таблицю 5.3 можна з'ясувати, що дані позначені літерами *a, b, …, s, t* є відповідними умовними ймовірностями, тобто *a* = P(*A*/*B*). Слід визначити ймовірність настання дощу завтра P(*A*) враховуючи декілька показників погоди сьогодні: Дощ, Холодно, Вітряно, Хмарно, Туманно... При цьому слід враховувати кореляцію цих показників між собою. Ця задача може бути розв'язана наступним чином:

1 Розглянути всі наявні змінні (показники погоди), припустимо їх *n*.

2 Розглянути всі можливі комбінації вказаних змінних. Загальна кількість цих комбінацій обраховується як сума всіх комбінацій «з *n* по *x*», де *x* змінюється від 0 до *n*, тобто:

$$n!/((n-0)!0!) + n!/((n-1)!1!) + n!/((n-2)!2!) +\ldots,$$

у випадку *n* = 4 така сума дорівнює 16 (якщо змінні приймають тільки значення «Так» або «Ні»).

3 Створити нову експертну матрицю, яка містить не тільки змінні, але і їх комбінації, а також ймовірності для змінних та їх комбінацій, за умов здійснення тих або інших комбінацій змінних.

4 Тоді робота ЕС зводиться до пошуку у розширеній експертній матриці комбінації, що відповідає стану погоди на сьогодні та визначити відповідні їй умовні ймовірності для кожного з можливих наслідків.

В даному контексті нагадаємо теорему Байєса, яку для даного прикладу можна записати в наступному вигляді:

$$P(A/X) = P(X/A)P(A) / (P(X/A)P(A) + P(X/\text{ні } A)\, P(\text{ні } A)).$$

Якщо *A* – подія, яка полягає в тому, що завтра буде дощ, а *X* – певна комбінація подій, що описують погоду сьогодні, тоді ймовірність настання дощу завтра, за умови, що погода сьогодні описується комбінацією *X*, дорівнює ймовірності настання події *X* сьогодні, за умови, що завтра буде дощ, помноженої на ймовірність дощу сьогодні, поділеної на повну ймовірність настання події *X*.

Наведений приклад логіки побудови ЕС свідчить про необхідність залучання саме експертних знань, що обумовлено нелінійним зростанням кількості статистичних розрахунків за зростання кількості змінних, від яких залежить висновок, який має надати ЕС.

Розглянемо приклад, який демонструє логіку формування правил виведення в ЕС з елементами навчання.

Приклад 5.2 Слід побудувати ЕС, яка на підставі відповідей користувача визначає який предмет було «загадано». За предмети для прикладу візьмемо «Літак» та «Птах». Параметри, якими можна описати ці предмети та їх значення для кожного предмету, тобто експертна матриця, наведені в табл. 5.4.

Таблиця 5.4 – Експертна матриця прикладу 5.2

№ з/п	Параметр	Наявність параметру в предмета «Літак»	Наявність параметру в предмета «Птах»
1	Крила	1	1
2	Хвіст	1	1
3	Дзьоб	0	1
4	Двигун	1	0
5	Пір’я	0	1
6	Шасі	1	0

Принцип роботи ЕС полягає в отримані можливих варіантів предметів, а також назв параметрів, що описують ці предмети та їх значень, в ході діалогу. Після чого застосовуються правила виведення, сформовані на даному етапі навчання ЕС, робиться припущення щодо можливого висновку і користувачу надається варіант відповіді. Якщо користувач погоджується з наданою відповіддю відбувається перехід до опитування щодо наступного предмету, якщо ні – відбувається корекція правил виведення висновку, після чого відбувається перехід до наступного предмету.

Роботу описаної ЕС можна подати у вигляді спрощеної блок-схеми алгоритму (рис. 5.6).

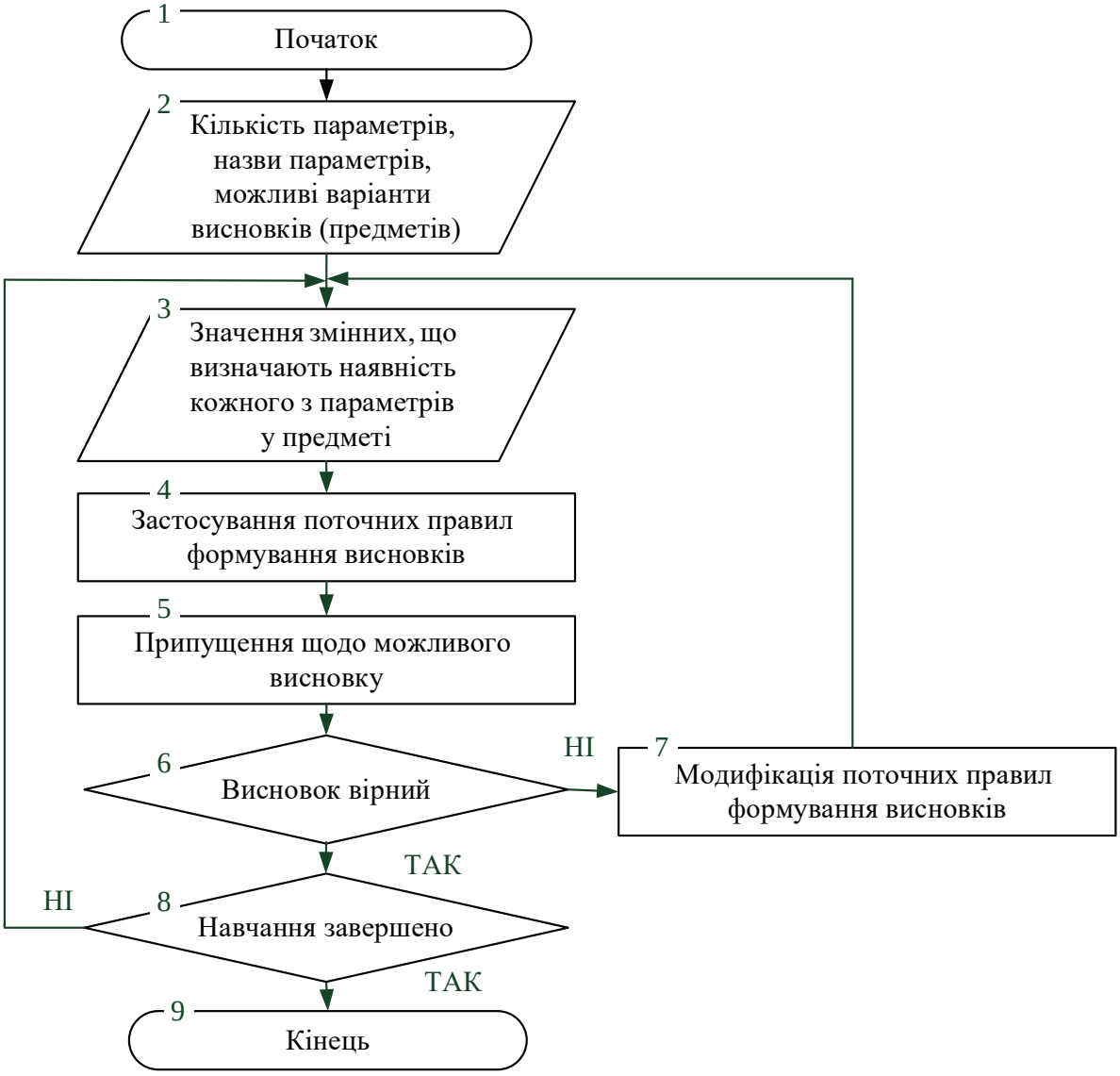


Рисунок 5.6 – Блок-схема алгоритму формування правил

Далі наведемо текст програми, яка реалізує даний алгоритм формування правил виведення.

```
10    CLS
20    INPUT «ВВЕДІТЬ КІЛЬКІСТЬ ПАРАМЕТРІВ»; VAR
30    DIM VALUE(VAR), RULES(VAR), VARS(VAR)
40    FOR I=1 TO VAR
50    VALUE(I)=0
60    RULES(I)=0
70    NEXT: PRINT
```

```
80    PRINT “ВВЕДІТЬ НАЗВИ ПАРАМЕТРІВ”
90    FOR I=1 TO VAR
100   INPUT “НАЗВА ПАРАМЕТРА:”; VARS(I)
110   NEXT: PRINT
120   PRINT “ВВЕДІТЬ МОЖЛИВІ ВАРІАНТИ ВИСНОВКІВ:”
130   INPUT “ПЕРШИЙ ВИСНОВОК:”; OUTCOME1S
140   INPUT “ДРУГИЙ ВИСНОВОК:”; OUTCOME2S
150   PRINT
160   FOR I=1 TO VAR
170   VALUE(I)=0
180   PRINT “ПАПАМЕТР:”; VARS(I)
190   INPUT “ЦЕЙ ПАРАМЕТР Є НАЯВНИМ <Y/N>”; AS
200   IF AS= “Y” OR AS= “y” THEN VALUE(I)=1
210   NEXT
220   DECISION=0
230   FOR I=1 TO VAR
240   DECISION= DECISION+VALUE(I)*RULES(I)
250   NEXT
260   PRINT “МОЖЛИВИЙ ВИСНОВОК:”; : IF DECISION >0 THEN
PRINT OUTCOME1S ELSE PRINT OUTCOME2S
270   INPUT “ВИСНОВОК Є ВІРНИМ <Y/N>”; AS
280   IF AS= “Y” OR AS= “y” THEN 150
290   IF DECISION>0 THEN
FOR I=1 TO VAR: RULES(I)= RULES(I)-VALUE(I): NEXT
ELSE FOR I=1 TO VAR: RULES(I)= RULES(I)+VALUE(I): NEXT
300   GOTO 150
```

Зауважимо, що наведена програма не містить команд виходу, тому для відповідності алгоритму (рис 5.6) слід додати команди припинення роботи програми після 270-го рядка програми.

В програмі використані три масиви:

RULES – масив, що містить правила виведення висновків;

VARS – масив, що містить назви параметрів, які описують предмет;

VALUE – масив, що містить значення параметрів, які описують предмет.

Якщо проаналізувати роботу програми для даних, наведених в експертній матриці (табл. 5.4), то можна надати наступний опис значень її змінних (табл. 5.5). Припустимо, що виконання блоку 2 (відповідно рядки коду 20-140) призводить до наступного визначення значень змінних:

– VAR = 6;

– VALUE(I) = 0, для I=1…VAR;

– RULES(I) = 0, для I=1…VAR;

– VARS(1) = “КРИЛА”, VARS(2) = “ХВІСТ”, VARS(3) = “ДЗЬОБ”, VARS(4) = “ДВИГУН”, VARS(5) = “ПІР’Я”, VARS(6) = “ШАСІ”

– OUTCOME1S = “ЛІТАК”, OUTCOME2S = “ПТАХ”.

Тоді роботу програми можна подати наступним описом значень її змінних.

Таблиця 5.5 – Значення змінних програми

№ рядка	Змінні				
	VALUE(I)	RULES(I)	DECISION	OUTCOME_S	AS
150-280	{0;0;0;0;0;0} {1;1;0;1;0;1} Загадали літак		0 0	DECISION=0 OUTCOME2S= =“ПТАХ”	N
290		**{1;1;0;1;0;1}**			
300	повернення до рядка 150				
150-280	{0;0;0;0;0;0} {1;1;0;1;0;1} Загадали літак		0 4	DECISION>0 OUTCOME2S= =“ЛІТАК”	Y
300	повернення до рядка 150				
150-280	{0;0;0;0;0;0} {1;1;1;0;1;0} Загадали птаха		0 2	DECISION>0 OUTCOME2S= =“ЛІТАК”	N
290		**{0;0;-1;1;-1;1}**			
300	повернення до рядка 150				

150- 280	{0;0;0;0;0;0} {1;1;1;0;1;0} Загадали птаха		0 -2	DECISION<0 OUTCOME2S= =“ПТАХ”	Y
300	повернення до рядка 150				
150- 280	{0;0;0;0;0;0} {1;1;0;1;0;1} Загадали літак		0 2	DECISION>0 OUTCOME2S= =“ЛІТАК”	Y
Навчання системи завершено					

Масив RULES місить правила виведення висновків, тому коли його елементи приймають значення {0;0;-1;1;-1;1} ЕС припиняє робити помилки та її навчання вважається завершеним.

Розглянемо змінну DECISION:

1 якщо масив VALUE заповнено значеннями, що відповідають предмету «Птах» {1;1;1;0;1;0}, то:

$$DECISION = 1*0+1*0+1*(-1)+0*1+1*(-1)+0*1= -2,$$

за умовами рядка 260 буде виведено висновок OUTCOME2S = “ПТАХ”;

2 якщо масив VALUE заповнено значеннями, що відповідають предмету «Літак» {1;1;0;1;0;1}, то:

$$DECISION = 1*0+1*0+0*(-1)+1*1+0*(-1)+1*1=2,$$

за умовами рядка 260 буде виведено висновок OUTCOME1S = “ЛІТАК”.

В результаті отримали ЕС, яка є коректно навченою на розглянутому наборі параметрів, а будь-які помилки є виключеними.

Зазначимо, що розглянута ЕС є універсальною відносно предметної області та кількості параметрів, які описують предмет.

Задачі для самостійного розв’язання

Задача 5.1 Описати логіку побудови ЕС для відповіді на питання «Чи складе студент сесію вчасно?», розглянувши не менше п’яти змінних, що обумовлюють відповідь на це питання.

Задача 5.2 Заповнити таблицю значень змінних для навчання ЕС з прикладу 5.2 після виключення параметрів, що не є принциповими для визначення різниці між предметами «Літак» та «Птах».

Задача 5.3 Змінити предметну область ЕС з прикладу 5.2, з кількістю параметрів опису предметів не менше 7.

Задача 5.4 Сформулювати базовий набір значень лінгвістичної змінної «зріст». Побудувати графік функцій приналежності, задавши значення шкали та відповідні значення функцій приналежності самостійно.

Задача 5.5 Сформулювати базовий набір значень лінгвістичної змінної «температура тіла». Побудувати графік функцій приналежності, задавши значення шкали та відповідні значення функцій приналежності самостійно.

Питання для самоконтролю

3 Дати визначення поняття «експертна система».

4 Дати визначення поняття «експертна система» з точки зору інженерії знань.

5 В чому полягає головна властивість ЕС?

6 З яких базових елементів складається структура ЕС? Дати визначення кожного з елементів.

7 Навести схему структури ЕС.

8 Які ролі у функціонуванні ЕС відіграють користувач, інженер зі знань, експерт?

9 Навести класифікацію ЕС за сферами застосування.

10 Навести класифікацію ЕС за типом моделі предметної області.

11 Навести класифікацію ЕС за видами даних і знань. Навести типи невизначеності знань.

12 Навести класифікацію ЕС за способом формування рішення.

13 Навести класифікацію ЕС за ступенем інтерпретації знань.

14 На які етапи можна умовно поділити процес розробки ЕС? Охарактеризуйте кожен з етапів.

15 Які виокремлюють рівні моделювання етапу концептуалізації?

16 Які моделі формалізації та подання знань використовують для розробки ЕС?

17 Навести класифікацію методів витягу знань. Дати характеристику кожного з методів.

18 Дати визначення поняття «лінгвістична змінна». Навести приклади лінгвістичних змінних та їх базових шкал.

19 Дати визначення поняття «функція приналежності».

20 Дати визначення поняття «нечітка множина». Навести приклади нечітких множин та їх графіків для визначення лінгвістичних змінних.

21 Які особливості ЕС слід враховувати під час їх розробки?

22 Які насьогодні існують види розробки ЕС?

23 За яких умов виникає ситуація, в якій доцільно та можливо створювати та застосовувати ЕС?

24 Якими аспектами визначається цінність застосування ЕС?

25 Яке співвідношення понять «база знань» та «область запитів» відбувається у процесі розробки ЕС?

26 Описати логіку побудови ЕС.

27 З якими типами інформації працює ЕС? Наведіть приклади інформації кожного типу. Інформація якого типу повинна потрапляти до бази знань?

28 Описати логіку формування правил виведення в ЕС з елементами навчання.

29 Навести блок-схему алгоритму формування правил.

РОЗДІЛ 6
МОВА ЛОГІЧНОГО ПРОГРАМУВАННЯ PROLOG

6.1 Основна ідея логічного програмування

Основна ідея логічного програмування полягає у використанні комп'ютера для виведення висновків з декларативного опису предметної області. В процесі логічного програмування опис предметної області та виконання логічного виведення базується на логіці предикатів першого порядку (див. Розділ 3 Логіка предикатів першого порядку).

Логічне програмування пов'язують з системами програмування, які в своїй роботі використовують спеціальні класи логічних формул, так званих клаузів Хорна або хорновських диз'юнктів.

Хорновський диз'юнкт – диз'юнктивний одночлен з не більш ніж одним додатнім літералом (атомом). Диз'юнктивний одночлен (ще має назви елементарна диз'юнкція, диз'юнкт, макстерм, клауза від англ. clause) – це диз'юнкція літералів, тобто змінних та заперечень змінних, який може приймати значення ХИБНО тільки за єдиного з усіх можливих наборів значень змінних, які до нього входять. Якщо диз'юнктивний одночлен містить одночасно змінну та її заперечення, то його істинностне значення завжди ІСТИНА. Приклад диз'юнктивного одночлена: $\neg X \vee Y \vee Z$.

Наведемо ще один приклад визначеного хорновського диз'юнкта: $\neg X \vee \neg Y \vee \ldots \vee \neg Z \vee U$. Ця формула може бути перетворена в еквівалентну формулу з імплікацією: $(X \to U) \vee (Y \to U) \vee \ldots \vee (Z \to U)$ або $(X \wedge Y \wedge \ldots \wedge Z) \to U$. В логічному програмуванні визначений хорновський диз'юнкт використовується як процедура редукції цілі (або процедурв виведення висновку). Редукція цілі – це представлення цілі (в дереві цілей) у вигляді вичерпного комплексу підцілей наступного рівня. Така процедура працює наступним чином: «для того щоб довести U, слід довести X, Y,…, Z», тобто $U \leftarrow (X \wedge Y \wedge \ldots \wedge Z)$ на PROLOG цей запис набуває вигляду: U : – X, Y, …, Z.

Логічна модель обчислень та спосіб виконання логічної програми базуються на спеціальних методах логічного виведення, що є варіантами методу резолюцій (див. п. 3.4 Логічне виведення. Принцип резолюції).

Мова логічного програмування PROLOG (від англ. Programming in Logic) є декларативною мовою, це означає, що замість чіткого алгоритму дій, який вказує комп'ютеру що за чим слід робити для розв'язання задачі, програма на PROLOG складається з опису задачі.

Програма на PROLOG задається хорновськими диз'юнктами в дещо зміненому записі, як це було показано вище. Виконання програми є визначеним певним чином упорядкованим процесом побудови дерева виведення з використанням методу резолюції.

Нагадаємо, що метод резолюцій є узагальненням методу "доказу від протилежного". Замість виведення деякої формули-гіпотези з наявної несуперечливої множини аксіом до множини аксіом додається заперечення формули і виконується спроба вивести з неї протиріччя. У випадку успішної спроби приходять до висновку, що вихідна формула є такою, що виводиться з даної множини аксіом.

6.2 Компоненти PROLOG програми

Прогами на PROLOG містять такі основні компоненти: константи, відношення (предикати), факти, правила, запити, змінні, коментарі.

6.2.1 Відношення (предикати).

PROLOG є декларативною мовою, тобто розробник програми намагається описати предметну область за допомогою цієї мови. Для опису предметної області слід визначити об'єкти які є її елементами та відношення (зв'язки) між ними. Після цього можна встановити правила, за яких ці відношення є істинними.

В PROLOG такі відношення між об'єктами описуються предикатами, тобто функціями, що можуть набувати лише двох значень: ІСТИНО або ХИБНО, і які призначені для вираження властивостей об'єктів або зв'язків між ними.

Приклад 6.1 Наведемо приклад: в предметній області існує студент Юра, який боїться висоти. В даному випадку маємо об'єкти «Юра» та «висота» і відношення між ними «боїться». Тобто є речення «Юра боїться висоти», яке може бути подано предикатом:

боїться(юра, висота).

Тут «боїться» – символьне ім’я предиката, «юра» та «висота» – аргументи предиката. Нагадаємо, що предикати бувають нульмісними, одномісними або *n*-місними. В даному випадку маємо двомісний предикат.

Можна також записати предикат:

боїться(Студент, висота).

Зауважимо, що в першому предикаті власне ім’я «Юра» написано з маленької літери, тому що це предметна константа, в другому слово «студент» з великої, тому що це змінна, тобто не зрозуміло про якого саме студента йдеться в реченні. Якщо замість змінної «Студент» підставити ім’я студента, який справді боїться висоти, то речення буде справедливим, тобто істинностне значення предиката буде ІСТИНА. Якщо до предиката підставити ім’я студента, який не боїться висоти, то істинностне значення предиката буде ХИБНО.

Нагадаємо, що в термінах математичної логіки предикат (*n*-парний або *n*-місний) це функція з областю значень {0,1}, яка є визначеною на декартовій множени *n*-й ступені.

6.2.2 Факти.

Якщо у відношенні (предикаті) всі змінні замінено на константи і воно є істинним, то таке відношення – факт. Нагадаємо, що в логіці предикатів такий вираз є висловленням.

В PROLOG те, що вже достеменно відомо про об’єкти та відносини предметної області є фактом. Факти також відбивають властивості об’єктів.

Наведемо кілька прикладів фактів:

боїться(юра, висота).
студент(юра).
блакитний(небо).

В термінах логіки предикатів факт може бути визначений як формула $P(t_1,t_2,...t_n)$, де P – символ предикату, t_n – терми, що складаються зі змінних, констант та функціональних символів.

6.2.3 Правила.

Правила дозволяють виводити з одних фактів інші факти.

Приклад 6.2 Наведемо кілька правил, що стосуються відношень «боїться», «студент» та «подобається»:

Каті подобаються всі студенти.

Толя боїться всього того, чого боїться Юра.

В PROLOG ці правила виглядатимуть приблизно наступним чином:

подобатися(катя, Хто) якщо студент(Хто).

боїться(толя, Що) якщо боїться(юра, Що).

З наведених у прикладі 6.1 фактів за описаними правилами можна зробити наступні висновки, які також є фактами: «Каті подобається Юра» та «Толя боїться висоти» або

боїться(толя, висота).

подобається(катя, юра).

Наведені у прикладі правила можна подати у вигляді наступних процедур:

– для того, щоб довести, що Толя чогось боїться, слід довести, що Юра боїться того самого;

– для того, щоб довести, що Каті хтось подобається, слід довести, що цей хтось є студентом.

У PROLOG всі правила мають дві частини: голову (Head) та тіло (Body), які розділені спеціальною лексемою «:-», аналогічною слову «якщо», і яка призначена для того, щоб відділити дві частини правила: голову та тіло. *Голова* (Head) – це факт, який буде істинним, якщо певний

набір умов є істинним. *Тіло* (Body) – це набір умов, який повинен бути істинним, щоб механізм логічного виведення PROLOG міг довести, що голова правила є істинною.

Наведені в прикладі 6.2 правила в PROLOG матимуть наступний вигляд:

подобатися(катя, Хто) :– студент(Хто).
боїться(толя, Що) :– боїться(юра, Що).

В першому правилі об'єкт «Хто» позначає множину людей, які є студентами, в другому об'єкт «Що» позначає множину речей, яких боїться Юра. Ці множини можна позначити *X* та *Y*, але це значно знижує читабельність програми.

В термінах логіки предикатів правило – це формула $A_1 \wedge A_2 \wedge \ldots \wedge A_{n0} \rightarrow A_0$, де A_i –атомарні формули.

Мовою PROLOG формули A_0:–$A_1, A_2, \ldots, A_{n0}$ – це безквантворні імплікації вигляду: кон'юнкція атомарних формул → атомарна формула. Якщо у цьому правилі зустрічаються змінні $X_1, X_2, \ldots, X_m$, то говорять, що «для всіх об'єктів $X_1, X_2, \ldots, X_m$, якщо є істинними твердження $A_1, A_2, \ldots, A_{n0}$ то є істинним також твердження A_0».

6.2.4 Запити

Після того, як буде введено набір фактів та правил можна ставити питання стосовно цих фактів. Такий процес відомий як «надання запитів в PROLOG-системі». Відповідаючи на запити PROLOG-програма спирається на факти та правила виведення нових фактів.

Для наведених вище прикладів можна сформулювати питання «Чи подобається Каті Юра?», яке у PROLOG набуло б вигляду:

подобається(катя, юра).

Спираючись на введені правила відповідь була б «Так».

Можна сформулювати складніше питання «Хто подобається Каті»:

подобається(катя, Хто).

Відповідь PROLOG в цьому випадку:

Хто = Юра
1 рішення

Якщо до наведених у попередніх прикладах фактів додати факт

боїться(юра, землетрус).

Та сформулювати питання

боїться(толя, Що).

Відповідь PROLOG матиме вигляд:

Що = висота
Що = землетрус
2 рішення

В термінах логіки предикатів запит до логічної програми – це кон'юнкція атомарних формул, тобто формула $A_1 \wedge A_2 \wedge \ldots \wedge A_{n0}$. Якщо у запиті немає змінних, то його читають як «Чи вірно, що є істинними A_1 та A_2 та … та A_{n0} ?» Якщо запит містить у собі змінні X_1, X_2,…, X_m, то його інтерпретують як «Для яких об'єктів X_1, X_2,…, X_m, є істинними A_1 та A_2 та … та A_{n0} ?».

Виконати PROLOG програму значить знайти всі значення вільних змінних, за яких запит логічно слідує з наведених в програмі фактів та правил.

6.2.5 Змінні

З опису наведених вище прикладів вже відомо, що аргументами предиката можуть бути константи та змінні. Ім'я змінної повинно

починатися з великої літери (або знаку підкреслення «_»), після якої може йти будь-яка кількість малих або великих літер, чисел або знаків підкреслення. Слід зауважити, що сумлінний підбір імен змінних підвищує читабельність програми.

PROLOG не має оператора присвоювання, замість цього змінні отримують значення шляхом порівняння (або ще кажуть «зіставлення», або англ. matching) з константами у фактах та правилах. Перш ніж змінна отримує значення, вона є *вільною*, після отримання значення, вона стає *зв'язаною*. Змінна залишається зв'язаною, доки не знайдено чергову відповідь на запит, тобто не досягнена ціль пошуку. Після досягнення цілі пошуку PROLOG звільняє змінну, виконує відкат та шукає альтернативні розв'язки.

Приклад 6.3 Наведемо приклад, який продемонструє, як і коли змінні отримують значення.

```
PREDICATES
nondeterm study (symbol,symbol)
CLAUSES
study (hellen, algebra).
study (robert, geometry).
study (jonn, history).
study (robert, geography).
study (hellen, geometry).
study (eric, algebra).
GOAL
study (Person, geometry), study (Person, algebra).
```

Ціль наведеної програми є питанням «Яка особа вивчає і геометрію і алгебру?». Для пошуку відповіді на це питання PROLOG порівнює другий аргумент фактів з ціллю та зв'язує змінну *Person* з відповідною константою у випадку співпадіння. При цьому відшукуються диз'юнкти, що відповідають обом частинам запиту.

Спочатку змінна *Person* є вільною (її значення є невідомим). Зустрівши факт *study (robert, geometry)* програма зв'язує змінну *Person* з константою *robert* (тобто привласнює змінній *Person* значення *robert*).

PROLOG розміщує покажчик в списку фактів, який показує точку, до якої дійшов процес пошуку та починає пошук співпадіння з другою частиною цілі, тобто шукає факт *study (robert, algebra)* з самого початку списку фактів. Але такого у списку фактів нема, тобто друга частина цілі є хибною за змінної *Person* уніфікованої з константою *robert.* Тоді PROLOG робить змінну *Person* знову вільною та намагається знайти інші співпадіння з першою частиною запиту, починаючи з рядка, в якому встановлено покажчик. Таке повернення до останнього позначеного місця має назву «перебір з поверненнями» (backtracking).

Коли PROLOG знаходить факт *study (hellen, geometry)*, він зв'язує змінну *Person* з константою *hellen* та шукає факт *study (hellen, algebra)*. Після знаходження відповідного факту ціль буде виконана повністю, змінна *Person* буде уніфікована з константою *hellen*, а програма перейде до пошуку інших розв'язків цілі.

В даному випадку відповідь матиме вигляд:

Person=hellen
1 Solution

Якщо до списку фактів додати *study (robert, algebra)* відповідь матиме вигляд:

Person=robert
Person=hellen
2 Solutions

В PROLOG існує таке поняття як «анонімні змінні», які дозволяють упорядковувати програми та прибирати зайву інформацію. Вони позначаються знаком підкреслення «_».

Анонімні змінні використовують у випадку, якщо у запиті необхідна лише певна інформація, а значення, в яких немає необхідності слід

ігнорувати. У PROLOG анонімна змінна зіставляється з усіма фактами і правилами та може бути використана замість будь-якої іншої змінної. Анонімна змінна ніколи не отримує значення.

Наведемо приклади використання анонімних змінних у фактах:

```
study(_).
haveto(_, eat).
```

Перший з наведених фактів читається як «кожен навчається», другий – «кожен має їсти».

Якщо у прикладі 6.3 за ціль ввести *study(Student, _)*, то відповідь матиме вигляд:

```
Student=hellen
Student=robert
Student=jonn
Student=robert
Student=hellen
Student=eric
6 Solutions
```

Наведемо приклад використання анонімних змінних у правилах виведення PROLOG програм.

Приклад 6.4. Програма PROLOG містить факти про кількість очок п’яти найкращих тенісисток світу за версією Жіночої тенісної асоціації WTA (Women`s Tennis Association) станом на 2020 рік.

Ціль програми є відповіді на питання «Хто грає за нашу країну?» та «За яку країну грає найкраща тенісистка?», для спрощення будемо вважати, що найкращою є тенісистка, що має понад 8000 очок.

Нижче наведено лістинг програми.

```
DOMAINS
country,tennis_player=symbol
points=unsigned
```

```
PREDICATES
player(country,points)
nondeterm playing_for(tennis_player,country)
nondeterm good_player(country)
CLAUSES
playing_for("Ashley Barty", "Australia").
playing_for("Simona Halep", "Romania").
playing_for("Caroline Plishkova", "Czech Republic").
playing_for("Sofia Kenyn", "USA").
playing_for("Elina Svitolina", "Ukraine").
player ("Australia", 8717).
player("Romania",6076).
player("Czech Republic",5205).
player("USA",4590).
player("Ukraine",4580).
good_player(Country):- playing_for(_,Country), player(Country, Score), Score>8000.
GOAL
playing_for(Player_of_our_country,"Ukraine"),
good_player(Country_winner).
```

В даному випадку відповідь матиме вигляд:

```
Player_of_our_country=Elina Svitolina, Country_winner=Australia
1 Solution
```

У предикаті, який визначає кого вважати найкращою тенісисткою (*good_player(Country):- playing_for(_,Country), player(Country, Score), Score>8000*) використано анонімну зміну, оскільки у запиті не питається про ім’я чи прізвище тенісистки, лише про країну за яку вона грає, тому ці дані можна ігнорувати та замінити їх анонімною змінною.

Зауваження. В наведеному прикладі текстові константи взяті у лапки, тому що для наочності вони написані з великої літери та містять пробіли.

Якщо зняти лапки, то для правильної роботи програми вони повинні мати такий вигляд: *ashley_barty, czech_republic*, тощо.

6.2.6 Коментарі

Сумлінний стиль програмування передбачає включення до тексту програми коментарів, які пояснюють неочевидні моменти. Завдяки коментарям підвищується читабельність та зрозумілість програми. Не слід жалкувати зусиль та часу на написання коментарів, оскільки те, що зрозуміло розробнику програми в момент її створення, може забутися через деякий час.

Якщо коментар містить лише один рядок, то на його початку встановлюється символ %, якщо коментар багатосрроковий, то він починається сполученням символів /*, а закінчується сполученням символів */. Наведемо приклади коментарів:

```
/* Ця програма працює добре*/
% Ця програма також працює добре
/*============================
    Ці три рядки є коментарем
============================*/
```

Завдяки тому, що PROLOG є декларативною мовою коментарів потрібно небагато, оскільки програма сама себе «пояснює». Але таке «пояснення» відбувається лише за зрозумілого надання імен змінних, предикатів, доменів. Такі імена повинні описувати логіку та зміст тих елементів, яким вони надані.

6.3 Основні розділи PROLOG програми

Зазвичай програма на PROLOG містить чотири основні розділи:

1 CLAUSES – розділ диз'юнктів. Цей розділ є ядром програми на PROLOG, який містить факти та правила, якими оперує PROLOG для досягнення цілі програми.

2 PREDICATES – розділ предикатів. Містить об’явлення предикатів, що використовуються в програмі (вбудовані предикати об’являти не потрібно).

3 DOMAINS – розділ доменів. Цей розділ містить опис будь-яких доменів (типів аргументів, що використані в програмі). Стандартні домени PROLOG описувати не обов’язково.

4 GOAL – розділ цілей. Містить вбудовану в програму ціль, що забезпечує виконання програми незалежно від середовища PROLOG.

Кім наведених в програмах на PROLOG існують також додаткові розділи: CONSTANTS, DATABASE, GLOBAL.

Розділи наведені за порядком їх важливості для створення програми, з порядком розташування розділів у програмі можна ознайомитися з прикладів текстів програм. Розглянемо детальніше наведені розділи.

6.3.1 Розділ CLAUSES

В розділі CLAUSES програміст описує всі факти та правила, що утворюють програму.

Диз’юнкти, що належать до певного предиката, повинні бути розміщені у цьому розділі разом. Послідовність диз’юнктів, які описують певний предикат має назву *процедура*. Процедура визначає яким чином буде викликано предикат в процесі виконання програми.

Коли PROLOG робить спробу задовольнити ціль, він починає перегляд з початкових диз’юнктів розділу, перевіряючи кожен факт і правило в пошуках збігу. Як тільки PROLOG доходить до кінця розділу диз’юнктів, він розміщує внутрішні покажчики на наступні диз’юнкти за тими, які були зіставлені з поточною підціллю. Якщо диз’юнкт не є частиною логічного виведення висновку, що приводить до розв’язку, то PROLOG повертається на встановлений покажчик та відшукує інший збіг. Такий процес пошуку розв’язків має назву перебір або пошук з поверненнями (backtracking), який був розглянутий детальніше під час опису прикладу 6.3 цього розділу.

6.3.2 Розділ PREDICATES

Якщо програміст застосовує в розділі CLAUSES свій власний предикат, то він повинен описати його в розділі PREDICATES. Під час опису предикату слід визначити до яких доменів належать його аргументи.

Опис предиката починається з його імені, за яким у дужках наведені домени аргументів предиката (дужки можуть бути порожніми). Опис предиката, на відміну від речень у розділі CLAUSES, не завершується крапкою.

Домени аргументів предиката можуть бути або стандартними доменами, або доменами, які оголошені в розділі DOMAINS. Ім’я предиката повинно починатися з літери та містити послідовність літер, цифр і символів підкреслення (максимум 250 символів). В імені предиката не можна використовувати пробіл, мінус, * та інші алфавітноцифрові символи.

Приклад 6.5 Наприклад, опис відношення «зріст людини» може мати наступний вигляд:

```
PREDICATES
man_height(symbol, integer)
```

При цьому немає потреби оголошувати домени аргументів предиката у відповідному розділі, оскільки вони стандартні. Проте, такий опис не є вдалим з точки зору читабельності програми, тому кращим буде опис наступного виду:

```
DOMAINS
name = symbol
height = integer
PREDICATES
man_height(name, height)
```

З позиції детермінізму (однозначності або неоднозначності результату виведення висновку) в PROLOG існує кілька видів предикатів, різницю між якими зручно подати таблицею (табл. 6.1).

Таблиця 6.1 – Види предикатів PROLOG

Признак відмінності	Вид предикату			
	procedure	multi	determ	nondeterm
Чи може предикат створювати розгалуження всередині себе (мати кілька варіантів успішного проходження)	–	+	–	+
Чи може проходження предикату завершуватися неуспіхом	–	–	+	+

Наведемо декілька вбудованих предикатів PROLOG (табл. 6.2).

Таблиця 6.2 – Вбудовані предикати PROLOG

№ з/п	Вбудований предикат	Дія предикату
1	2	3
1	nl	перехід на новий рядок
2	write	вивід повідомлення
3	readchar	читання символу
4	asserta	додавання нового факту в базу даних перед існуючим фактом для даного предикату
5	assertz	додавання нового факту в базу даних після існуючих фактів для даного предикату
6	consult	завантаження з файла фактів бази даних
7	retract	знищення факту з бази даних
8	retractall	знищення всіх фактів з бази даних
9	save	збереження динамічної бази на диск
10	free(Var)	перевірка чи є змінна Var вільною
11	bound(Var)	перевірка чи є змінна Var зв’язаною
12	findall	формування списку із всіх можливих рішень, які забезпечують істинність деякого неоднозначного предикату (іншими словами це пошук всіх рішень для цілі одночасно)
13	length	обчислення довжини списку
14	member	приналежність елемента списку
15	append	приєднання одного списку до іншого
16	fail	визначення того, що поточний пошук рішення для деякої підцілі немає успіху, та необхідно почати новий пошук

Продовження таблиці 6.2

1	2	3
17	cut (або !)	примусове завершення всіх пошуків
18	not	визначення того, що поточний пошук рішення для деякої підцілі дає успіх, коли пов'язана з ним підціль не може бути доведена

6.3.3 Розділ DOMAINS

Домени в PROLOG еквівалентні типам даних в імперативних мовах програмування. В програмах на PROLOG об'єкти у відношеннях, тобто аргументи предикатів, належать доменам, які можуть бути стандартними або спеціальними, тобто такими, що визначаються програмістом під час створення програми. У розділі DOMAINS описуються області даних для предикатних змінних, задається область інтерпретації предметних змінних.

Розділ DOMAINS в програмах на PROLOG відіграє дві важливі ролі:

1 дає можливість визначити для доменів змістовні імена, які відтворюють логіку програми (це підвищує читабельність програми), навіть в тому випадку, коли домени співпадають з вже існуючими стандартними доменами (див. приклад 6.5);

2 дає можливість опису структур даних, які не можуть бути визначені за допомогою стандартних доменів.

Приклад 6.6 Наприклад, для визначення того, які саме аргументи мають бути присутні в описі предиката, що відображає факт «Юра – студент, який має зріст 182 сантиметри», слід сформулювати розділи DOMAINS та PREDICATES наступним чином:

```
DOMAINS
name, occupation, sex = symbol
height = integer
PREDICATES
person(name, occupation, sex, height)
```

Наведемо декілька стандартних доменів PROLOG (табл. 6.3).

Таблиця 6.3 – Стандартні домени PROLOG

№ з/п	Найменування домену	Характеристика домену
1	2	3
1	short	коротке знакове число 32768..32767
2	ushort	коротке беззнакове число 0..65535
3	long	довге знакове число -2147483647..2147483648
4	ulong	довге беззнакове число 0..4294967295
5	integer	ціле зі знаком -32768..32767
6	unsigned	ціле без знака 0..4294967295
7	byte	байт, беззнакове число 0.. 255
8	char	символ, подається в якості unsigned byte. синтаксично він записується як символ, взятий в одинарні лапки: ‘a’
9	real	число з плаваючою точкою -10^{307} …10^{308}
10	symbol	символи, довільні текстові рядки
11	string	символи, довільні текстові рядки, в яких текст взятий у подвійні лапки

6.3.4 Розділ GOAL

Ціль програми на PROLOG може інтерпретуватися як запит на виконання певного правила. Між ціллю та правилом є дві відмінності:

1 ціль не містить лексеми :- (якщо);

2 після запуску програми PROLOG відробляє ціль автоматично.

Цілі в програмі на PROLOG можуть бути зовнішніми або внутрішніми.

Зовнішні цілі завдаються вже під час роботи програми. Такі цілі вводяться у діалоговому режимі у вікні Dialog у відповідь на підказку Goal. Зовнішні цілі зручні в тому випадку, коли програма завжди запускається у середовищі розробки PROLOG.

Внутрішні цілі завдаються у тексті програми в розділі GOAL. Такі цілі є частиною початкового тексту програми та компілюються разом з іншими її частинами.

Вміст розділу GOAL є переліком підцілей. Робота програми на PROLOG передбачає виконання підцілей з цього переліку, тобто задоволення тіла цільового правила. Якщо під час роботи програми

досягаються всі підцілі розділу GOAL, то робота програми завершується успіхом і виводяться всі можливі розв'язки (Solutions). Якщо під час роботи програми хоча б одна з підцілей не досягається, то робота програми завершується неуспіхом, про що програма надає відповідне повідомлення (No Solution).

Наприклад, якщо у прикладі 6.3 факт *study (hellen, geometry)* замінити на факт *study (hellen, geography)*, то робота програми з ціллю *study (Person, geometry), study (Person, algebra)* закінчиться неуспіхом. У розділі GOAL програми цього ж прикладу можна замінити ціль на запит *study (hellen, algebra)*, тобто запитати «Чи вивчає Хелен алгебру?», в цьому випадку відповідь програми буде :Yes.

Ціль є *складною*, якщо вона складається з двох або більше частин, що мають назву підцілі (subgoal). Так у прикладі 6.3 ціль програми складається з двох частин: *study (Person, algebra)* та *study (Person, geometry)*. Програма успішно завершує роботу тільки у випадку досягнення обох підцілей шляхом пошуку співпадаючих диз'юнктів, тобто, коли знаходить особу, яка вивчає і алгебру і геометрію.

Якщо під час пошуку розв'язку складної цілі, її підцілі поєднані за допомогою кон'юнкції, то в тексті програми необхідно відокремлювати такі підцілі комою. Якщо ж підцілі поєднані за допомогою диз'юнкції, то в тексті програми необхідно відокремлювати підцілі крапкою з комою.

Повернимося до прикладу 6.3, розділ GOAL якого має вигляд:

```
GOAL
study (Person, algebra), study (Person, geometry).
```

В даному випадку підцілі поєднані за допомогою кон'юнкції, тобто програма має знайти особу, яка б вивчала алгебру ТА геометрію, тому відповідь буде мати вигляд:

```
Person=hellen
1 Solution
```

Якщо замінити кому на крапку з комою, то підцілі будуть поєднані за допомогою диз'юнкції. В цьому випадку програма шукає особу, яка вивчає алгебру АБО геометрію, і відповідь буде мати наступний вигляд:

Person=hellen
Person=eric
Person=robert
Person=hellen
4 Solutions

6.3.5 Розділ CONSTANTS

Розділ CONSTANTS призначений для оголошення констант. Таке оголошення має наступний вигляд:

<ім'я константи> = <значення>

Оголошення констант має задовольняти таким вимогам:

– ім'я константи є ідентифікатором, тобто воно може складатися з англійських літер, цифр та знаку підкреслення, і не може починатися з цифри;

– кожне оголошення константи закінчується символом кінця абзацу, тобто повинно розміщуватись в окремому рядку.

Наведемо кілька прикладів оголошення констант.

CONSTANTS
number = 3
hundred = (10*(10-1)+10)
name = Юра

Перш ніж компілювати програму PROLOG замінить кожний ідентифікатор константи на відповідний рядок, який йому відповідає.

На використання констант в PROLOG накладаються наступні обмеження:

1 оголошення констант не може посилатися на себе;

2 програма може містити кілька розділів CONSTANTS, але константи мають бути оголошені преш ніж їх використовує програма;

3 ідентифікатори констант є глобальними, тому можуть використовуватися у оголошені лише один раз (не дотримання цього обмеження призводить до видачі повідомлення «Constant identifier can only be declared once»);

4 коли ідентифікатор константи використовується розділом диз'юнктів програми, перша літера повинна бути рядковою, щоб відрізнити константу від змінної.

Нагадаємо, що використання констант під час розробки програм, необхідно для завдання параметрів моделі, тобто тих даних, що не обов'язково змінюються від одного програмного експерименту з моделлю до іншого (як, наприклад, вхідні дані). Проте параметри змінюються під час зміни умов проведення програмного експерименту з моделлю, тому, щоб не редагувати значення параметрів вручну по всьому тексту програми, їх слід описувати за допомогою констант.

6.3.6 Розділ DATABASE

Програма на PROLOG є сукупністю фактів та правил. Якщо в процесі виконання програми може виникнути потреба у заміні, видалені або додаванні фактів, то в такому випадку факти утворюють *динамічну* або *внутрішню* базу даних.

Розділ DATABASE призначено для оголошення фактів, які складають динамічну базу даних. Оголошення фактів через ключове слово DATABASE дозволяє:

1 додати факти до бази даних з клавіатури під час виконання програми за допомогою предикатів *asserta* і *assertz*;

2 видалити існуючі факти за допомогою предикатів *retract* і *retractall*;

3 модифікувати вміст бази даних спочатку видаленням факту, а потім додаванням нової версії цього факту (предикат *consult* читає факти з файлу і вставляє їх у внутрішню базу даних, а предикат *save* зберігає зміст розділу динамічної бази даних у файлі).

Існують обмеження на використання предикатів бази даних:

– можна додати до бази даних лише факти, але не правила;

– факти бази даних не можуть мати вільну змінну.

Програма може містити декілька розділів бази даних, кожному з яких необхідно явно задати ім’я. Наприклад:

DATABASE – mydatabase
myFirstRelation (integer)
mySecondRelation (real, string)
myThirdRelation (string)

Дане оголошення створює базу даних з ім’ям *mydatabase*. Якщо ім’я для динамічної бази даних не задано, то за замовчуванням буде привласнене стандартне ім’я – *dbasedom*. Імена предикатів бази даних повинні бути унікальними в модулі (початковому файлі). Не можна використовувати однакове ім’я предиката бази даних у двох різних розділах бази даних.

Зауважимо, що деякі версії PROLOG дозволяють оголошувати певні домени, предикати, факти та правила як *глобальні*. Для цього слід на початку тексту програми сформулювати відповідні розділи: GLOBAL DOMAINS, GLOBAL PREDICATES, GLOBAL DATABASE.

6.4 Складові об’єкти

Об’єктами відносин, тобто аргументами предикатів є дані. Прості дані представляють самі себе і є простими об’єктами. Структура, що складається з простих об’єктів є простою структурою. В якості простих об'єктів даних в PROLOG використовуються константи і змінні.

Наприклад,

боїться(юра, висота)

Якщо ж об’єкт представляє інший об’єкт або сукупність об’єктів, то такий об’єкт є складеним, а структура, що складається з таких об’єктів є складеною структурою.

Наприклад,

вивчає(юра, математика, мови програмування, ризикологія)

всі дисципліни можна об’єднати в одну структуру:

дисципліна(математика, мови програмування, ризикологія)

і тоді отримаємо один складовий об’єкт, що пояснює відповідне відношення:

вивчає(юра, дисципліна(математика, мови програмування, ризикологія))

Таким чином, складові об'єкти даних дозволяють інтерпретувати деякі частини інформації як єдине ціле, а потім легко розділяти їх знову. До складових об'єктів даних в PROLOG відносяться структури та списки.

6.4.1 Структура

Структура – це складовий об’єкт в PROLOG. Ім’я структури має назву «функтор». Функтор визначає вигляд складеного об’єкту даних та поєднує разом його аргументи. Після функтора у круглих дужках міститься список аргументів, розділених комою.

Аргументами структури можуть бути числа, атоми, змінні, структури та інші складові об'єкти. Кількість аргументів структури фіксується під час оголошення і не може змінюватися в процесі виконання програми.

Оголошення структури подібно до оголошення предикату, тому, для того щоб PROLOG міг їх розрізняти, всі структури повинні бути описані в розділі DOMAINS разом з користувацькими типами даних, а всі предикати, які використані в програмі, (крім вбудованих) – в розділі PREDICATES.

Наприклад, дата грудень 5, 2012, можна описати наступним чином:

DOMAINS
date_omp=date(string,unsigned, unsigned)
D=date(“December”, 5, 2012)

Складовий об’єкт може бути уніфікований з простою змінною або з іншим складовим об’єктом. Це означає, що складовий об’єкт можна використовувати для того, щоб передавати цілий набір значень як єдиний об’єкт, а потім використати уніфікацію для їх розділення.

6.4.2 Список

Список – це об’єднання довільної кількості об’єктів, розділених комами і обмежених квадратними дужками.

Наведемо кілька прикладів списків в PROLOG:

- список, елементами якого є символи [one, two, three];
- список, елементами якого є рядки [“One”, “Two”, “Three”];
- список, який не має елементів – порожній список [].

Списки оголошуються у розділі DOMAINS наступним чином:

DOMAINS
<ім’я спискового домену> = <ім’я домену елементів списку> *

Зірочка після імені домену вказує на те, що описується список, який складається з об’єктів відповідного типу. Наприклад:

mylist = elementDom*

У цьому прикладі *mylist* – домен, що складається зі списку елементів з домену *elementDom*. Домен *elementDom* може бути або визначеним користувачем доменом, або одним із стандартних доменів. Наприклад:

numberlist = integer *

це оголошення домену для списку цілих чисел [1, -5, 2, -6].

Під час роботи зі списками до одного списку не можна розміщати елементи, що належать різним доменам, в такому разі слід скористатися складеним доменом. Наведемо приклад:

```
DOMAINS
%элементы списка - целые числа
intlist=integer*
%элементы списка - символы
symlist=symbol*
%элементы списка - строки
strlist=string*
%элементы списка - или целые числа или символы или строки
mixlist=mixdomain*
mixdomain =int(integer); sym(symbol); str(string)
```

Зауваження. Під час оголошення складеного домену *mixdomain* були використані функтори, тому що запис *mixdomain = integer; symbol; string* призвів би до синтаксичної помилки.

Список є об'єктом рекурсивного типу, який має наступне визначення. Список – це структура даних , що визначається таким чином:

1 порожній список [] є списком;

2 структура виду [H | T] є списком, якщо H (голова списку) – перший елемент списку (або кілька перших елементів списку, перелічених через кому), а T (хвіст списку) – список, що складається з елементів, які залишилися від початкового списку.

Наведемо приклади до даного визначення:

– список [1, 2, 3] можна розподілити на: голову списку [1] та хвіст списку [2, 3], тобто [1 | [2, 3]];

– список [1] можна розподілити на: голову списку [1] та хвіст списку [], тобто [1 | []];

– список [1, 2, 3] можна розподілити або на [1 | [2, 3]], або на [1 | [2 | [3]]], або на [1 | [2 | [3 | []]]].

Порожній список розподілити на голову та хвіст неможливо.

Наведемо кілька прикладів уніфікації у списках (табл. 6.4).

Таблиця 6.4 – Приклади уніфікації у списках

Уніфікація	Результат уніфікації
[1, 2, 3]=[Elem1, Elem2, Elem3]	Elem1=1, Elem2=2, Elem3=3
[1, 2, 3]=[Elem1, Elem2, Elem3 | T]	Elem1=1, Elem2=2, Elem3=3, T=[ ]
[1, 2, 3]=[Head | Tail]	Head=1, Tail=[2, 3]
[1, 2, 3]=[Elem1, Elem2 | T]	Elem1=1, Elem2=2, T=[3]
[1, 2, 3]=[4 | T]	Помилка
[ ]=[H | T]	Помилка

Оскільки список є рекурсивною структурою даних, то для обробки списків використовують рекурсію, яка полягає в наступному:

1 відділити від списка голову,

2 виконати необхідні дії з головою списку,

3 перейти до хвоста списка, який в свою чергу також є списком і знову виконувати обробку, починаючи з першого пункту.

Обробка завершується, коли у хвості залишається порожній список.

Приклад 6.7. Для ілюстрації рекурсивної обробки списків наведемо програму, що поелементно виводить на екран список, елементами якого є цілі числа.

```
DOMAINS
intlist=integer*

PREDICATES
printlist (intlist)

CLAUSES
%якщо список порожній нічого не робимо
printlist ([ ]):- !.
/* якщо список не порожній, слід відділити голову та вивести її на екран, потім використати той самий предикат для виводу хвоста списку, тобто рекурсивно викликати предикат printlist, надав в якості аргумента хвіст списка */
printlist ([H | T]):- write (H), nl, printlist (T).
```

```
GOAL
printlist ([1, 2, 3]).
```

Результати виконання програми мають наступний вигляд:

```
1
2
3
yes
```

Зауваження. Оскільки показником закінчення рекурсивної обробки списків є наявність у хвості порожнього списку, слід передбачити предикати роботи як з не порожнім списком, так і предикати для роботи з порожнім списком.

Для роботи зі списками використовують вбудований предикат *findall*, який збирає всі розв'язки для певного цільового твердження в один список. Предикат *findall* має три аргументи:

VarNam (ім'я змінної) – визначає параметр, який необхідно зібрати в список;

MyPredicate (мій предикат) – визначає предикат, з якого треба зібрати значення;

ListParam (список параметрів) – містить список значень, зібраних методом пошуку з поверненням.

Приклад 6.8. Роботу предиката *findall* проілюструємо програмою, яка визначає середній бал успішності академічної групи.

```
DOMAINS
lastname,firstname = symbol
rating = integer
list = rating*

PREDICATES
nondeterm person(lastname,firstname,rating)
```

```
sumlist(list,rating,integer)

CLAUSES
sumlist([],0,0).
sumlist([H|T],Sum,N):- sumlist(T,S1,N1), Sum=H+S1, N=1+N1.
person("Kovalenko", "Igor", 4).
person("Petrenko", "Katya", 3).
person("Soloshenko", "Olena", 5).
person("Slushko", "Ivan", 4).
person("Ivanchenko", "Ganna", 5).

GOAL
findall(Rating,person(_, _, Rating),L),
sumlist(L,Sum,N),
Rating = Sum/N,
write("Average rating =", Rating),nl.
```

Результати виконання програми мають наступний вигляд:

```
Average rating =4.2
L=[4,3,5,4,5], Sum=21, N=5, Rating=4.2
1 Solution
```

Зауваження. Щоб відповідь в прикладі 6.8 була цілим числом слід символ ділення у формулі обчислення змінної *Rating* замінити оператором *div*.

6.5 Управління виведенням висновків

6.5.1 Використання предикату *fail*

Внутрішня ціль програми передбачає закінчення процесу пошуку розв’язків після першої успішної уніфікації цілі. Таким чином PROLOG знаходить перший розв’язок і на цьому завершує виконання програми. Існує

два способи «змусити» PROLOG шукати всі розв’язки, що уніфікуються з даною ціллю:

1 метод повернення після невдачі (реалізується за допомогою предиката *fail*);

2 метод переривання та повернення (реалізується за допомогою предиката *cut*).

Спеціальний предикат *fail* штучно викликає ситуацію неуспішного виконання програми і таким чином ініціює пошук нового розв’язку. Дія предикату *fail* є аналогічною підцілі, яка ніколи не може бути виконана (наприклад порівняння 1=4).

Приклад 6.9. Роботу предиката *fail* проілюструємо на прикладі програми, яка виводить на екран список студентів та їх оцінок.

```
DOMAINS
lastname,firstname = symbol
rating = integer
list = rating*

PREDICATES
nondeterm person(lastname,firstname,rating)
nondeterm print_person

CLAUSES
person("Kovalenko ", "Igor ", 4).
person("Petrenko ", "Katya ", 3).
person("Soloshenko ", "Olena ", 5).
person("Slushko ", "Ivan ", 4).
person("Ivanchenko ", "Ganna " , 5).

print_person:-person(Lastname, Firstname, Rating),
write(Lastname, Firstname, Rating), nl, fail.
print_person.

GOAL
```

```
write("Here are the student sand their rating:"),nl,
nl, print_person.
```

Результати виконання програми мають наступний вигляд:

```
Here are the student sand their rating:

Kovalenko Igor 4
Petrenko Katya 3
Soloshenko Olena 5
Slushko Ivan 4
Ivanchenko Ganna 5
yes
```

Ця програма виводить на друк усі можливі розв’язки запиту. На першому етапі змінна *Lastname* уніфікується з рядком "Kovalenko ", змінна *Firstname*, відповідно з рядком "Igor ", а змінна Rating уніфікується зі значенням 4. Після чого змінні виводяться на екран та відбувається повернення до наступного факту, який би міг задовольнити ціль. Таким чином було знайдено всі можливі розв’язки, тобто було виведено на екран дані про всіх студентів.

Якщо прибрати предикат *fail*, то результати виконання програми матимуть наступний вигляд:

```
Here are the student sand their rating:

Kovalenko Igor 4
yes
```

Тобто буде виведено перший розв’язок, який уніфікується з ціллю, після чого програма завершується успіхом.

Програму, наведену у прикладі 6.9 можна використовувати також для виведення даних про студентів, які задовольняють певним вимогам.

Наприклад, для виведення даних про відмінників слід змінити предикат *print_person* наступним чином:

```
print_person:-person(Lastname, Firstname, 5),
write(Lastname, Firstname, 5), nl, fail.
print_person.
```

або

```
print_person:-person(Lastname, Firstname, Rating), Rating = 5,
write(Lastname, Firstname, Rating), nl, fail.
print_person.
```

Результати виконання програми в цьому випадку матимуть наступний вигляд:

```
Here are the student sand their rating:

Soloshenko Olena 5
Ivanchenko Ganna 5
yes
```

Після знаходження першого розв’язку, який задовольняє запиту предикат *fail* забезпечує повернення до моменту виведення розв’язку на екран та пошуку іншого розв’язку, який би задовольняв би запит.

6.5.2 Переривання пошуку з поверненням

Метод переривання та повернення є другим способом «змусити» PROLOG шукати всі розв’язки, що уніфікуються з даною ціллю. Він реалізується за допомогою вбудованого предиката *cut* (вкраїнською «відсікання»), та позначається в програмі PROLOG знаком оклику !.

Цей предикат завжди завершується успішно. Його дія полягає у встановленні бар’єру для запобігання перегляду фактів, які розташовані раніше ніж той факт, який вже відповідає певним підцілям програми. Тобто

таке відсікання запобігає повторному перегляду фактів, які вже були переглянуті для досягнення певних підцілей. На перегляд підцілей, які розташовані правіше за тестом програми, відсікання не впливає.

Якщо програма один раз проходить крізь відсікання, то вже неможливо повернутися до підцілей, які розташовані перед відсіканням . Існує два випадки використання відсікання:

1 *зелене відсікання*: коли заздалегідь відомо, що в даному випадку не може бути альтернативних розв'язків (за відкидання таких відсікань програма продовжує видавати ті самі рішення, що й за їх наявності);

2 *червоне відсікання*: коли логіка програми вимагає відсікання, щоб запобігти розгляду альтернативних підцілей.

Відсікання відповідає оператору *goto* в процедурних мовах програмування, який багато чого дозволяє, але робить програму більш важкою для розуміння.

Приклад 6.10. Роботу предиката *cut* проілюструємо на прикладі програми, яка передбачає пошук машини з умовами на її марку, колір та ціну.

```
PREDICATES
buy_car(symbol, symbol)
nondeterm car (symbol, symbol, integer)
colors(symbol, symbol)

CLAUSES
buy_car (Model,Color):-car (Model, Color, Price), colors (Color, sexy),!,
Price > 25000.
car(maserati, green, 25000).
car(corvette, black, 24000).
car(corvette,red,26000).
car(porsche, red, 24000).
colors(red, sexy).
colors (black,mean) .
colors (green, preppy).
```

```
GOAL
  buy_car (corvette, Y).
```

Результати виконання програми мають наступний вигляд:

```
  Y=red
  1 Solution
```

За умовами програми машина повинна бути марки "corvette", кольору з параметром "sexy" та ціною вищою за 25000. Перша підціль досягається за потрапляння до перегляду другого факту: car(corvette, black, 24000). Проте колір даної машини не відповідає параметру "sexy" через наявність факту colors (black,mean). Тоді PROLOG шукає інші факти для уніфікації змінної *Model* та переходить до факту car(corvette,red,26000). В даному випадку задовольняються підцілі щодо марки та кольору машини. В програмі встановлюється бар'єр, який під час перегляду фактів для досягнення підцілі щодо ціни машини, завадить PROLOG переглядати факти, які розташовані в програмі вище факта car(corvette,red,26000). Цей факт є розв'язанням, оскільки він задовольняє умові на ціну машини. Внутрішня ціль містить зміну *Y*, яка згідно з фактом унифікується зі значенням "red".

6.5.3 Використання предикату NOT

Вбудований предикат *not* змінює логіку обирання фактів PROLOG під час їх перегляду. Особливістю предиката *not* є те, що він є успішним, якщо не може бути доведена істинність підцілі, з якою він асоціюється. Пояснемо ці твердження на наступному прикладі: слід записати твердження про те, що Юра не хоче бути дантистом, коли вирасте (тобто він згоден бути ким завгодно крім дантиста). Записати твердження про те, що Юра згоден бути ким завгодно легко:

```
future_profession (jura, X):-job(X),
```

а для виключення професії дантиста потрібен предикат, який виключає її з розгляду:

```
future_profession (jura, X):-job(X),
not (dentist(X)).
```

В даному випадку унарний предикат *not* буде успішно виконуватися, якщо підціль, що є його аргументом є хибною.

Наведемо ще один приклад використання предикат *not*.

Приклад 6.11. Програма виводить перелік студентів, середній бал яких є більшим за 3,5. З переліку виключаються студенти які наразі проходять практику.

```
DOMAINS
name= symbol
gpa= real

PREDICATES
nondeterm honor_student(name)
nondeterm student(name, gpa)
probation (name)

CLAUSES
honor_student (Name) :-student (Name, GPA),
GPA>=3.5, not (probation(Name)).
student("Betty Blue", 3.6).
student("David Smith", 4.0).
student("Martin Smith", 2.0).
student("John Johnson", 3.7).
probation("Betty Blue").
probation("David Smith").

GOAL
honor_student(X).
```

Результати виконання програми мають наступний вигляд:

Who=John Johnson, GPA=3.7
1 Solution

Якщо з предикату *honor_student* прибрати предикат *not*, то програма буде виводити прізвища всіх студентів середній бал яких є вищим за 3,5. Результати виконання програми матимуть наступний вигляд:

X=Betty Blue
X=David Smith
X=John Johnson
3 Solutions

6.5.4 Використання правил для умовного розгалуження

Зазвичай у процедурних мовах умовне розгалуження реалізується операторами *case* та *goto*. Проте PROLOG є дескриптивною мовою, тому кожне правило є певною умовою за виконання якої програмою робиться той або інший розв'язок. Реалізацію в PROLOG класичного варіанту використання оператора *case* можна наступним прикладом.

Приклад 6.12. Потрібно вивести на екран фразу, яка відповідає вибору користувача.

```
PREDICATES
Nondeterm action (integer)

CLAUSES
action(1):-nl,write("You typed 1."), nl.
action(2):-nl,write("You typed 2."), nl.
action(3):-nl,write("Three was what you typed."), nl.
action(N):-nl,N<>1, N<>2, N<>3,
write("I don't know that number! ").
```

```
GOAL
write("Type a number from 1 to 3: "),
readint(Num),
action(Num).
```

Результатом роботи програми буде одна з чотирьох фраз, яка відповідає значенню введеному користувачем. Якщо користувач вводить будь яке число крім 1, 2 або 3 результат роботи виглядатиме наступним чином:

```
Type a number from 1 to 3:12
I don't know that number!
Num=12
1 Solution
```

Предикат перевірки введеного числа action(N):-nl,N<>1, N<>2, N<>3, виглядає доволі громіздким. Для вдосконалення програми можна скористатися анонімною змінною та замінити цей предикат наступним:

```
action(_):-write("I don't know that number!").
```

В цьому випадку PROLOG не зважаючи на те яке число введено буде видавати фразу "I don't know that number!":

```
Type a number from 1 to 3:2
You typed 2
Num=2
I don't know that number!
Num=2
2 Solutions
```

Щоб відкорегувати роботу програми слід скористатися предикатом *cut*, який може вважатися аналогом оператора *goto*. Отримаємо наступний вигляд програми:

```
PREDICATES
nondeterm action( integer)

CLAUSES
action(1):-!, nl, write("You typed 1").
action(2):-!,nl, write ("You typed 2").
action(3):-!,nl,write("Three was what you typed.").
action(_):-write("I don't know that number!").

GOAL
write("Type a number from 1 to 3:"),
readint (Num),
action(Num),nl.
```

В цьому випадку предикат cut є червоним відсіканням, яке не дозволяє виводити фразу "I don't know that number!", якщо користувач ввів число від 1 до 3.

6.5.5 Використання рекурсії

Рекурсія може бути визначена як процес визначення чогось з використанням самого себе, або коли об’єкт є частиною самого себе. Поняття «рекурсія» використовується в багатьох сферах: інформатиці, лінгвістиці, математиці, логіці і навіть в живописі. В цій роботі рекурсія була використана для визначення деяких понять.

В інформатиці здебільшого використовують для визначення функцій. Функція є рекурсивною, якщо оператори в тілі функції викликають цю ж саму функцію. Наприклад для обрахування факторіалу цілого невід’ємного числа можна скористатися рекурсивною функцією, логіка роботи якої може бути описана наступною функцією:

$$n! = \begin{cases} n(n-1)!, & n > 1; \\ 1, & n = 1. \end{cases}$$

Кожне наступне звернення до функції відбувається з аргументом на одиницю меншим за попереднє значення. Оскільки *n* є цілим невід'ємним числом, через *n* рекурсивних звернень обрахунки дійдуть до випадку 0!=1, яким роботу рекурсивної функції буде завершено.

Іншим прикладом рекурсивних обрахунків може слугувати піднесення числа в ступінь. В цьому випадку рекурсивно викликається функція множення попереднього результата на це число. Так, наприклад, число 2 в п'ятому ступені це число 2 в четвертому ступені помножене на 2. В PROLOG така функція може бути записана наступним предикатом:

```
st(X, Y, Z) :- YY=Y-1, st(X, YY, ZZ), Z=ZZ*X
```

За визначення внутрішньої цілі st(2, 5, Z) відбудиться рекурсивний виклик цілі з аргументами st(2, 4, Z), потім з аргументами st(2, 3, Z) і т.д.

Для зупинки рекурсивного виконання функції необхідно передбачити випадок піднесення числа в першу ступінь з уніфікацією цілі без додаткових викликів:

```
st(X, 1, X)
```

Таким чином можна зробити висновок, що рекурсія в PROLOG складається з двох частин:

1 *крок рекурсії* – правило, яке забезпечує послідовний рекурсивний виклик функції, в тілі якого обов'язково міститься в якості підцілі виклик предиката, що визначається;

2 *базис рекурсії* – речення, яке забезпечує коректне завершення рекурсивної функції, визначає початкову ситуацію або ситуацію в момент припинення.

Процес уніфікації змінних в PROLOG відбувається послідовно зверху в низ з ліва на право, тому базис рекурсії слід розташовувати раніше за крок рекурсії.

Класичним прикладом для демонстрації використання рекурсії в програмах є задача про Ханойські вежі.

Приклад 6.13. Умови задачі про Ханойські вежі полягають в наступному: надано три стрижні, на лівому стрижні пірамідою розташовано кілька дисків різного розміру. Потрібно перемістити всі диски на правий стрижень в тому ж самому порядку. Можна переміщувати лише по одному диску та класти лише диск меншого розміру на диск більшого розміру. Задача може бути розв'язана інтуїтивно за відносно невеликої кількості дисків. Кількість операцій в розв'язанні задачі визначається формулою 2^n, де n – кількість дисків.

Наведемо текст програми PROLOG для розв'язання даної задачі.

```
DOMAINS
loc = right; middle; left

PREDICATES
nondeterm hanoi(integer)
move(integer, loc, loc, loc)
inform(loc, loc)

CLAUSES
hanoi (N):-move (N, left,middle,right).
move(1,A,_,C):-inform (A,C),!.
move(N,A,B,C):-N1=N-1, move (N1,A,C,B),
inform(A,C), move(N1,B,A,C).
inform(Loc1, Loc2):-nl,
write("Move a disk from ", Loc1, "   to    ", Loc2).

GOAL
hanoi(3).
```

Програма містить три предикати:

1 предикат *hanoi*, через який визначається кількість дисків в задачі;

2 предикат *move*, який визначає стратегію переміщення диску з між початковим, цільовим та проміжним стрижнями;

3 предикат *inform*, який визначає та виводить на екран кожну окрему операцію переміщення диску.

Рядок «move(1,A,_,C):-inform (A,C),!.» визначає ситуацію, якщо диск лише один.

Задачі для самостійного розв’язання

Задача 6.1 Написати програму за індивідуальним варіантом предметної області (табл. 6.5), яка б містила не менше 10 фактів та 3 правил виведення. Номер варіанта визначити за формулою:

$$N_{вар} = 10 * \{N_{студ} / 10\}+1,$$

де $N_{студ}$ – номер студента за списком у журналі академічної групи;

{} – дробова частина числа (0<{a}<1).

Таблиця 6.5 – Варіанти предметних областей

Номер варіанта	Назва предметної області	Запит до програми
1	Дружні зв’язки між особами однієї групи	Хто дружить з певною особою? З ким дружить певна особа, якщо вона дружить лише з друзями іншої певної особи?
2	Рейтинг кращих футболістів світу	За яку команду грає найкращій футболіст? Які гравці грають за певну команду?
3	Інформація про улюблені фільми	Хто є актором головної ролі та режисером певного фільму? В яких фільмах грав певний актор?
4	Родинні зв’язки	Хто є бабусею певної особи? Хто є дітьми певної особи?
5	Турнірна таблиця шахістів	Хто з ким грає у певний час? З ким грає певний шахіст?
6	Книга адрес	Хто мешкає за певною адресою? За якою адресою мешкає певна особа?
7	Каталог продукції	Яку продукцію виробляє певна фірма? Яка вартість на певну продукцію?
8	Організаційна структура підприємства	Чи є менеджером певна особа? Хто є менеджером самого верхнього рівня?
9	Бібліотечний каталог	Які є книги певного автора? До якої секції належить певна книга?
10	Розклад занять	Заняття з яких дисциплін відвідують студенти певної групи по понеділках? По яких днях тижня викладається певна дисципліна?

Додати власні запити до цілей створених програм.

Задача 6.2 Написати програму поелементного виведення списку днів тижня.

Задача 6.3 Написати програму для визначення середнього віку групи людей.

Задача 6.4 Написати програму для виведення на екран даних про дисципліни, які викладаються певною кафедрою (дані мають містити не менше 4 параметрів).

Задача 6.5 Написати програму для виведення на екран даних про дисципліни, які передбачають складання екзамену.

Задача 6.6 Написати програму для виведення на екран даних про дисципліни, які передбачають викладання 120 годин.

Задача 6.7 Написати програму обрахунку факторіала цілого невід'ємного числа.

Задача 6.8 Написати програму піднесення цілого невід'ємного числа в певну ступінь.

Задача 6.9 В прикладі 6.13 визначити складові рекурсії: крок та базис. Змінити код програми таким чином, щоб цільовим був середній стрижень.

Питання для самоконтролю

1 В чому полягає основна ідея логічного програмування?

2 Дати визначення поняття «хорновський диз'юнкт». Наведіть приклад хорновського диз'юнкту.

3 Якого вигляду набуває запис хорновського диз'юнкту у PROLOG?

4 В чому полягає особливість програмування декларативною мовою PROLOG?

5 З яких основних компонентів складаються програми, написані мовою PROLOG?

6 Які функції виконують відношення (предикати) у програмах написаних мовою PROLOG? Навести приклади використання предикатів.

7 Які функції виконують факти у програмах написаних мовою PROLOG? Навести приклади використання фактів.

8 Які функції виконують правила у програмах написаних мовою PROLOG? Навести приклади використання правил.

9 Які функції виконують запити у програмах написаних мовою PROLOG? Навести приклади використання запитів.

10 Які функції виконують змінні у програмах написаних мовою PROLOG? Навести приклади використання змінних.

11 Які функції виконують анонімні змінні у програмах написаних мовою PROLOG? Навести приклади використання анонімних змінних.

12 Які функції виконують коментарі у програмах написаних мовою PROLOG? Навести приклади використання коментарів.

13 Які розділи програми мовою PROLOG Вам відомі? Описати призначення кожного з розділів.

14 Які функції виконують процедури у програмах написаних мовою PROLOG?

15 Який синтаксис має опис предиката в розділі PREDICATES? Навести приклад.

16 Які види предикатів PROLOG Вам відомі? В чому полягає відмінність кожного з видів?

17 Навести основні вбудовані предикати PROLOG. Які призначення мають ці предикати в програмах PROLOG?

18 Які ролі в програмах на PROLOG відіграє розділ DOMAINS?

19 Який синтаксис описів в розділі DOMAINS? Навести приклади описів.

20 Які стандартні домени PROLOG Вам відомі?

21 В чому полягають відмінності між ціллю та правилом?

22 В чому відмінність між зовнішньою цілю та внутрішньою цілю?

23 В якому випадку робота програми мовою PROLOG завершується успіхом, а в якому неуспіхом?

24 В чому відмінність синтаксису поєднання підцілей за допомогою кон’юнкції від поєднання підцілей за допомогою диз’юнкції у розділі GOAL? В чому буде полягати відмінність у результатах роботи програми в та іншому випадках?

25 Який синтаксис має оголошення констант у відповідному розділі програм мовою PROLOG? Яким вимогам мають задовольняти оголошення констант?

26 Навести обмеження, які накладаються на використання констант в програмах мовою PROLOG.

27 Які можливості надає оголошення фактів у розділі DATABASE ?

28 Які існують обмеження на використання предикатів бази даних?

29 Навести основні вбудовані предикати роботи з базою даних в PROLOG.

30 Яке призначення мають складові об'єкти в програмах мовою PROLOG? Навести приклад.

31 Який синтаксис оголошення структури в програмах мовою PROLOG? В якому розділі відбувається таке оголошення? Навести приклад.

32 Який синтаксис оголошення списків в програмах мовою PROLOG? В якому розділі відбувається таке оголошення? Навести приклади списків.

33 Дати визначення поняття «список» як об'єкту рекурсивного типу в PROLOG. Навести приклади до даного визначення.

34 Навести приклади уніфікації у списках.

35 В чому полягає рекурсивна процедура обробки списків? Навести приклади такої процедури.

36 Який вбудований предикат використовують для роботи зі списками? Описати призначення аргументів вбудованого предикату роботи зі списками. Навести приклад використання предикату.

37 Які способи Управління виведенням висновків Вам відомі?

38 Яке призначення вбудованого предикату *fail*? Навести приклад використання вбудованого предикату *fail*.

39 Яке призначення вбудованого предикату *cut*? Навести приклад використання вбудованого предикату *cut*.

40 Які існують випадки використання відсікання за допомогою предиката *cut*?

41 Яке призначення вбудованого предикату *not*? Навести приклад використання вбудованого предикату *not*.

42 Навести приклади використання правил для умовного розгалуження в програмах мовою PROLOG.

43 Дати визначення поняття «рекурсія». Навести приклади рекурсивних процедур в різних сферах людської діяльності.

44 З яких частин складається рекурсія в PROLOG? Навести приклади використання рекурсії в програмах мовою PROLOG.

РОЗДІЛ 7
КАРТИ, ЩО САМООРГАНІЗУЮТЬСЯ

7.1 Карти, що самоорганізуються, як різновид штучних нейроних мереж

Карти, що самоорганізуються (SOM – Self Organizing Maps), або карти Кохонена – це різновид нейронної мережі, що використовуються для розв'язання задач кластеризації та сегментації. Алгоритм функціонування карти, що самоорганізується, представляє собою один із варіантів кластеризації багатовимірних просторів. Відмінністю даної технології є те, що під час її навчання використовується метод навчання без учителя, тобто результат навчання залежить тільки від структури вхідних даних.

Штучна нейронна мережа (Artificial Neural Network) –це комп'ютерна система, яка складається з визначеної кількості простих взаємопов'язаних процесорів, які називаються нейронами і які є аналогами біологічних нейронів мозку. Такі мережі реалізують парадигму обробки інформації на основі структури та функцій людського мозку, який складається з великої кількості нейронів, поєднаних один з одним за допомогою синапсів. В даному випадку нейрони – це процесори, які є аналогами біологічних нейронів мозку.

Метод, який реалізується картами Кохонена має наступні переваги:

– стійкість до зашумлених даних;
– швидке навчання, яке не потребує управління;
– можливість візуалізації багатовимірних вхідних даних.

Карти Кохонена можуть бути застосовані в наступних сферах:

1 Розвідувальний аналіз даних. Карта Кохонена здатна розпізнавати кластери в даних, а також встановлювати близькість класів. Таким чином, користувач може краще зрозуміти структуру даних для подальшого уточнення моделей, що розробляються на підставі цих даних. Якщо в даних визначені класи, то їх можна позначити, після чого карта Кохонена зможе вирішувати завдання класифікації самостійно.

2 Прогнозування поведінки клієнта. Якщо побудувати карту Кохонена, що містить кластери для кожної групи клієнтів за ступенем їхньої

лояльності, то з її допомогою можна прогнозувати очікувану поведінку клієнтів для визначення відповідної маркетингової політики щодо цих клієнтів.

3 Виявлення аномалій. Карта Кохонена розпізнає кластери в навчальних даних і відносить всі дані до тих чи інших кластерів. Якщо після цього карта стикається з набором даних, який є несхожим на жодний з відомих їй зразків, то вона не зможе класифікувати такий набір і таким чином виявить його аномальність.

7.2 Навчання та використання карт Кохонена

Розглянемо алгоритм навчання карти Кохонена.

Припустимо завданий вектор чисел $X = (x_1, x_2, \ldots, x_N)$, що є зразком для навчання. Існує мережа нейронів W, яка складається з M нейронів, кожен з яких описується N вагами (розмірність вектора X співпадає з кількістю ваг, що описують положення нейрона) $W = \{w_n^m\}$, $n = 1,\ldots, N$, $m = 1,\ldots, M$.

Задача полягає у відображенні вектора X на графічне зображення мережі нейронів, тобто на карту Коханена.

Для цього:

1 Ваги $\{ w_n^m \}$ ініціюємо випадковим чином.

2 Вводимо параметр часу і покладаємо його рівним 1 ($t = 1$).

3 Обираємо вектор X.

4 Знаходимо такий нейрон W^m*, який знаходиться найближче за значеннями ваг до обраного вектору X. Метрика порівняння може варіюватися залежно від задачі.

5 Корегуємо ваги всіх інших нейронів за правилом

$$w_n^m = w_n^m + \eta(t)h(t, \rho(m, m^*))[x_n - w_n^m],$$

де w_n^m – поточне значення ваг нейрона;

$\eta(t)$ – параметр навчання (його значення зменшується в процесі подальшого навчання);

$h(t, \rho(m, m^*))$ – числова функція, значення якої залежить від часу та відстані між нейроном W^m* та тим нейроном, який корегується;

$[x_n - w_n^m]$ – різниця між числовими значеннями елемента вектора даних та відповідною за номером вагою нейрона.

Наведемо приклади виду функцій, які можуть бути застосовані на кроці 5.

$$\eta(t) = \eta_0 e^{-\alpha t},$$

де $\eta(t)$ – експоненціальна спадна за часом функція;

α – коефіцієнт, що визначає швидкість спадання функції.

$$h(t, \rho) = e^{-\frac{\rho^2}{2\sigma(t)}}, \ \sigma(t) = \sigma_0 e^{-\beta t},$$

де $h(t, \rho)$ – експоненціальна спадна за відстанню функція (зауважимо, що вона має вигляд подібний до функції нормального розподілу);

$\sigma(t)$ – експоненціальна спадна за часом функція;

β – коефіцієнт, що визначає швидкість спадання функції.

6 Збільшуємо значення змінної часу на 1 ($t = t + 1$) та перевіряємо умову: чи скінчився час або за іншими параметрами процес навчання закінчено? За позитивної відповіді карта Кохонена вважається побудованою – кінець алгоритма. За негативної відповіді переходимо до кроку 3, тобто вибору наступних даних (наступного вектора-зразка для навчання).

Після виконання наведеного алгоритму достатньо велику кількість разів отримуємо налаштовану карту Кохонена, яка здатна самостійно розподіляти дані по нейронах. Геометрично близькі нейрони містять дані про не дуже відмінні стани об’єкта, приклади яких наведено в таблиці 7.1.

Таблиця 7.1 – Приклади станів різних видів об’єктів

Об’єкт	Стан	Сфера застосування
Технічна система	Безвідмовна робота	Техніка
Соціально-економічна система	Ефективне функціонування	Економіка
Організм людини	Здоров’я	Медицина
Екосистема (біоценоз)	Чистота довкілля	Екологія

Далі розглянемо використання налаштованої карти Кохонена.

1 Градуємо карту Кохонена. В результаті отримуємо карту, яка показана на рисунку 7.1.

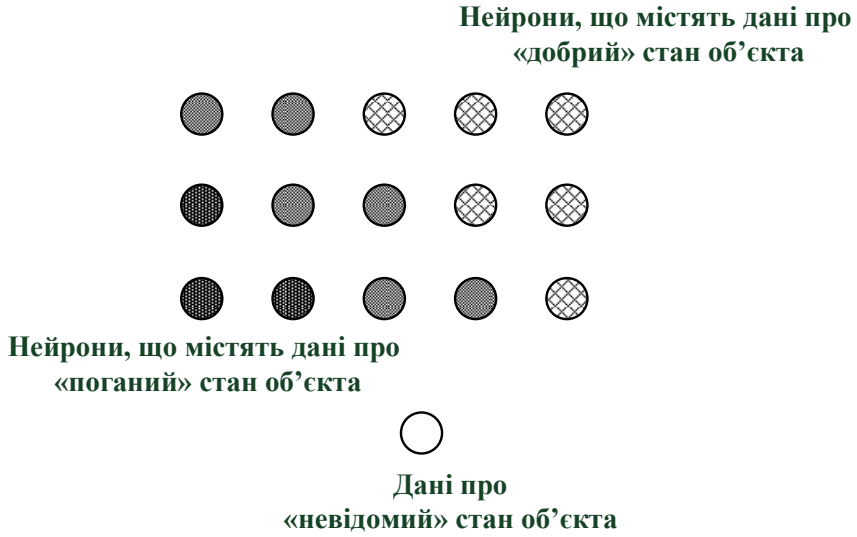


Рисунок 7.1 – Налаштована карта Кохонена

Після налаштування карти Кохрнена (рис. 7.1) дані, що описують «ДОБРИЙ» стан певного об’єкта (позначено світлими кружечками) розташувалися в нейронах, що знаходяться поруч один з одним; дані, що описують «СЕРЕДНІЙ» стан (позначено темнішими кружечками) розташувалися в нейронах, які розташовані трішечки далі, а дані, що описують «ПОГАНИЙ» стан (позначено темними кружечками) – в нейрони, що розташовані якнайдалі від нейронів «доброго стану».

2 Надходять дані про «невідомий» стан об’єкта (білий кружечок на рис.7.1). Ці дані слід відобразити на карту Кохонена, тобто знайти нейрон

W*, ваги якого найближчі до значень вектора X, який описує певний стан. Залежно від того в які нейрони потрапили ці дані визначається стан об’єкта.

7.3 Класифікація за допомогою карти Коханена

Наведемо концептуальний опис задачі класифікації:

Існує множина об’єктів (точок) *A*, яку слід розбити на класи 1,2,3,4, як це показано на рисунку 7.2.

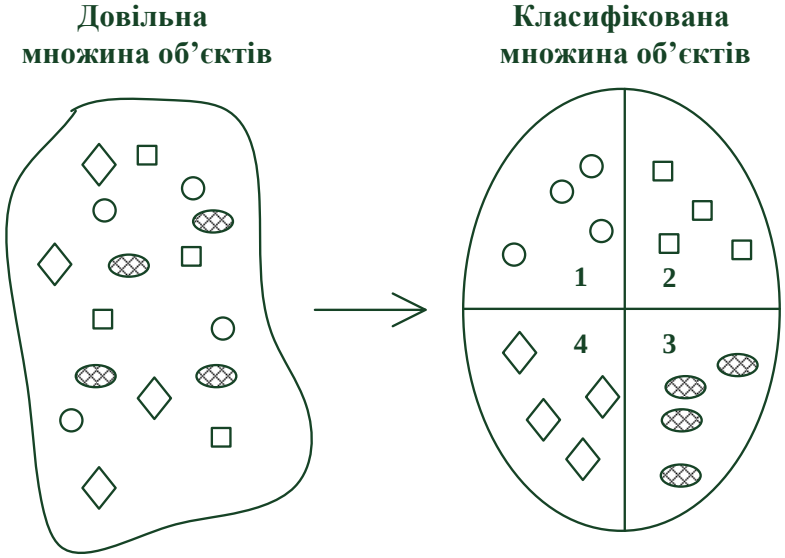


Рисунок 7.2 – Розбиття множини об’єктів на класи

Об’єкти в певний клас розподіляють за значеннями певної властивості (або властивостей, якщо класифікаторів декілька).

Система штучного інтелекту навчається класифікувати об’єкти на прикладі певної вибірки, тобто спочатку слід ввести множину прикладів для навчання (налагодження), а потім використовувати налагоджену систему. Приклад має містити об’єкт та клас, до якого він належить.

На початку досліджень може виникати ситуація, за якої невідомо до якого з класів належить той або інший об’єкт, в цьому випадку потрібно щоб карта Кохонена визначила клас автоматично, тобто щоб вона розбила множину об’єктів *A* на класи самостійно.

Формальна постановка задачі класифікації полягає у наступному: припустимо існує *M* векторів розмірністю *N* тобто матриця *Z* розмірністю

$M \times N$, які потрібно розбити на K класів, де $K > 1$: $X^m = (x_1^m, x_2^m, ...x_N^m),\ m = \overline{1, M}$

Існує також карта Кохонена, яка складається з нейронів з вагами, що подані векторами розмірністю N (це значення співпадає з розмірністю векторів початкових даних, кількість ваг співпадає з кількістю класів):

$W^k = (w_1^k, w_2^k, ...w_N^k),\ k = \overline{1, K}$.

Алгоритм навчання наведено на рисунку 7.3.

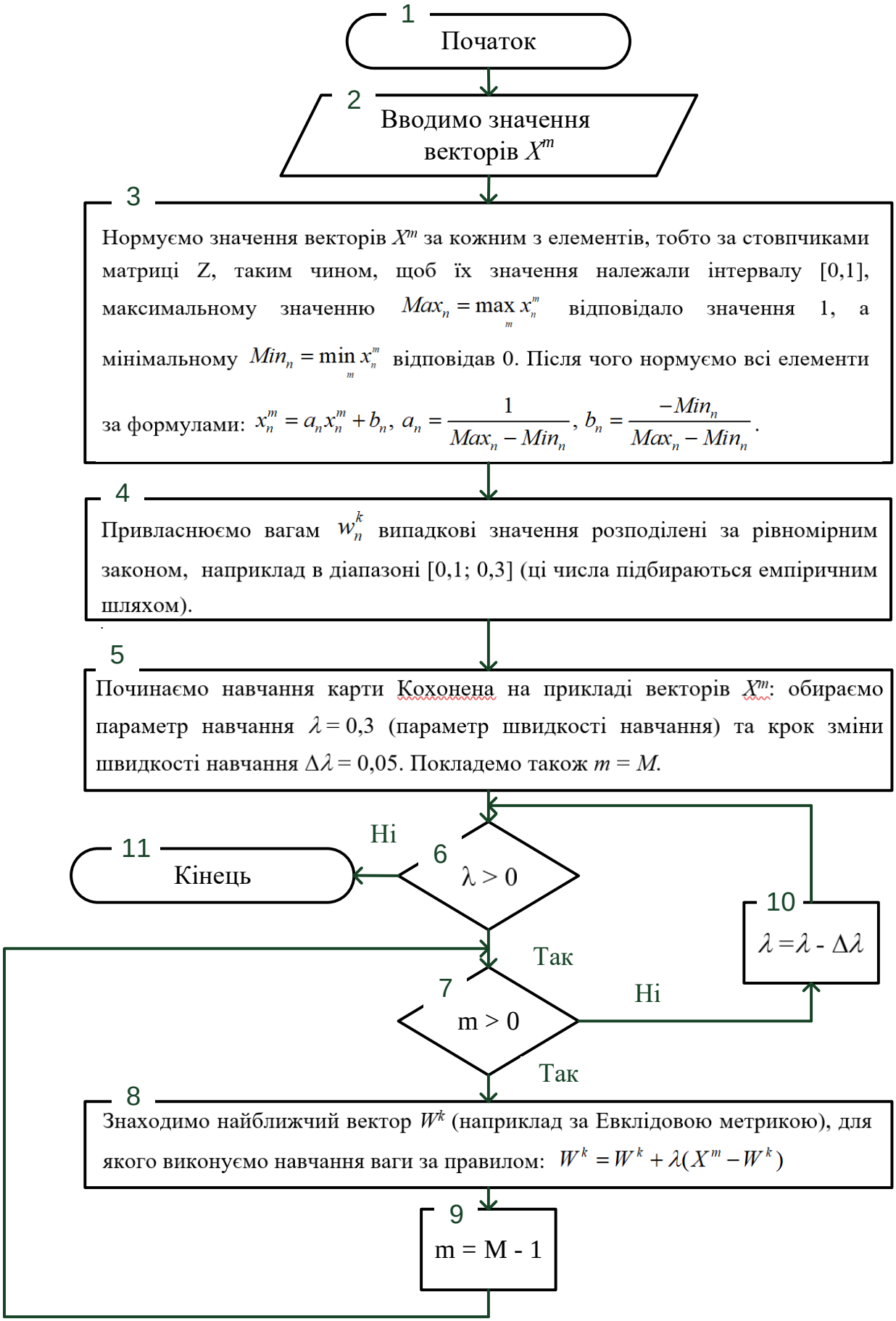


Рисунок 7.3 – Блок-схема алгоритму навчання карти Кохонена

В результаті виконання алгоритму (рис. 3.7) отримуємо $W^k = (w_1^k, w_2^k, ...w_N^k),\ k = \overline{1, K}$ – налаштовані ваги.

Далі розв'язуємо задачу класифікації довільного вектора X^m (у загальному випадку цей вектор не належить но множини векторів навчання) шляхом знаходження нейрона W^k найближчого до вектора. Тоді робимо висновок, що вектор X^m належить класу k. Проте, спочатку для вектора X^m має бути проведена процедура нормування за формулами блока 3 наведеного алгоритму (рис. 7.3).

Оскільки значення ваг на початку обиралися випадковим чином, то номер класа значення не має. Цей номер може змінюватися від одного запуска програми до іншого, але за різних запусків одні й тіж самі вектори X^m будуть належати до тих самих класів.

Крім того, якщо для W^k провести процедуру зворотну до процедури нормування (блок 3), отримаємо значення, що є найбільш типовими для об'єктів k-го класу.

7.4 Мережа Хопфілда, як різновид штучних нейроних мереж

Іншим прикладом нейроних мереж є мережа Хопфілда, яка може застосовуватися для розпізнавання зашумлених, але раніше відомих мережі образів.

Мережа Хопфілда складається з одного шару штучних нейронів, кожен з яких пов'язаний з усіма іншими нейронами крім себе. Мережа Хопфілда дозволяє реалізовувати асоціативну пам'ять людини.

Образ в мережі Хопфілда наданий за допомогою матриці (двомірного вектора) або вектором більшої вимірності. Він повинен якомога повніше заповнювати вектор (не має бути занадто маленьким). Кожен з нейронів мережі може знаходитися в одному з двох станів 1 або -1. Якщо нейрон зберігає частину образу він знаходиться у стані 1, інакше – -1. На рисунку 7.4 наведено приклад образу та його відображення на нейрону мережу Хопфілда.

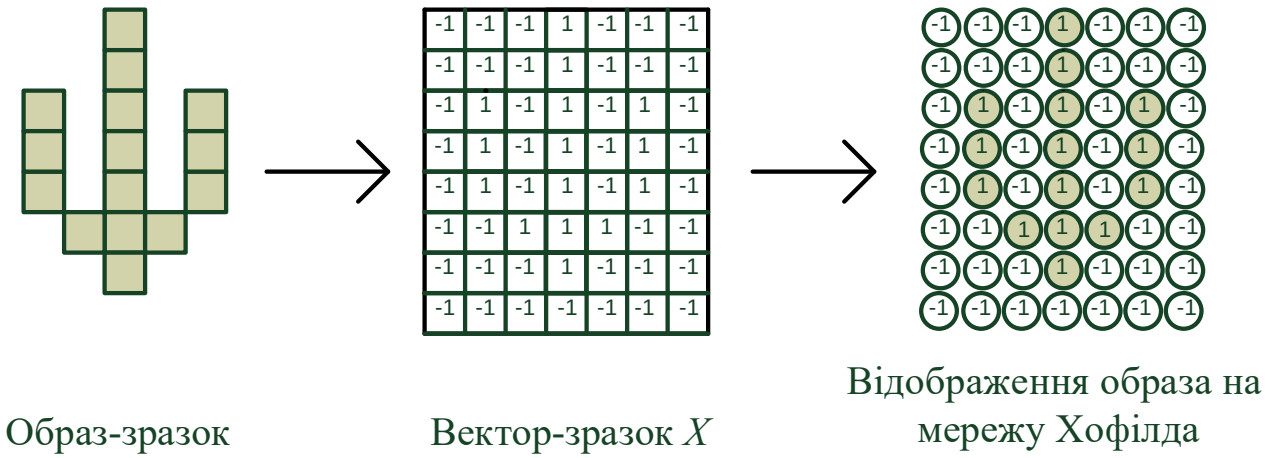


Рисунок 7.4 – Образ та його відображення на нейрону мережу Хофілда

Позначимо вектор яким задано образ через $X=\{x_{[1]}, x_{[2]},\ldots x_{[N]}\}$, де N – кількість нейронів, в яких зберігається образ. Число N визначає об'єм асоціативної пам'яті (для приклада на рисунку 1.4 N = 56, а сам вектор має наступні значення X = {-1, -1,-1,1,-1,-1,-1,-1,1,-1,1,-1,1,-1, -1,1,-1,1,-1,1,-1, -1,1, -1,1,-1,1,-1,-1,-1,1,1,1,1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1, -1,-1,-1,-1,-1,-1,-1,}).

Таких еталонних векторів-образів може бути M штук. Для ефективнішої роботи мережі Хопфілда еталонні образи повинні значною мірою відрізнятися один від одного.

Розглянимо алгоритм побудови мережі Хопфілда (рис. 7.5).

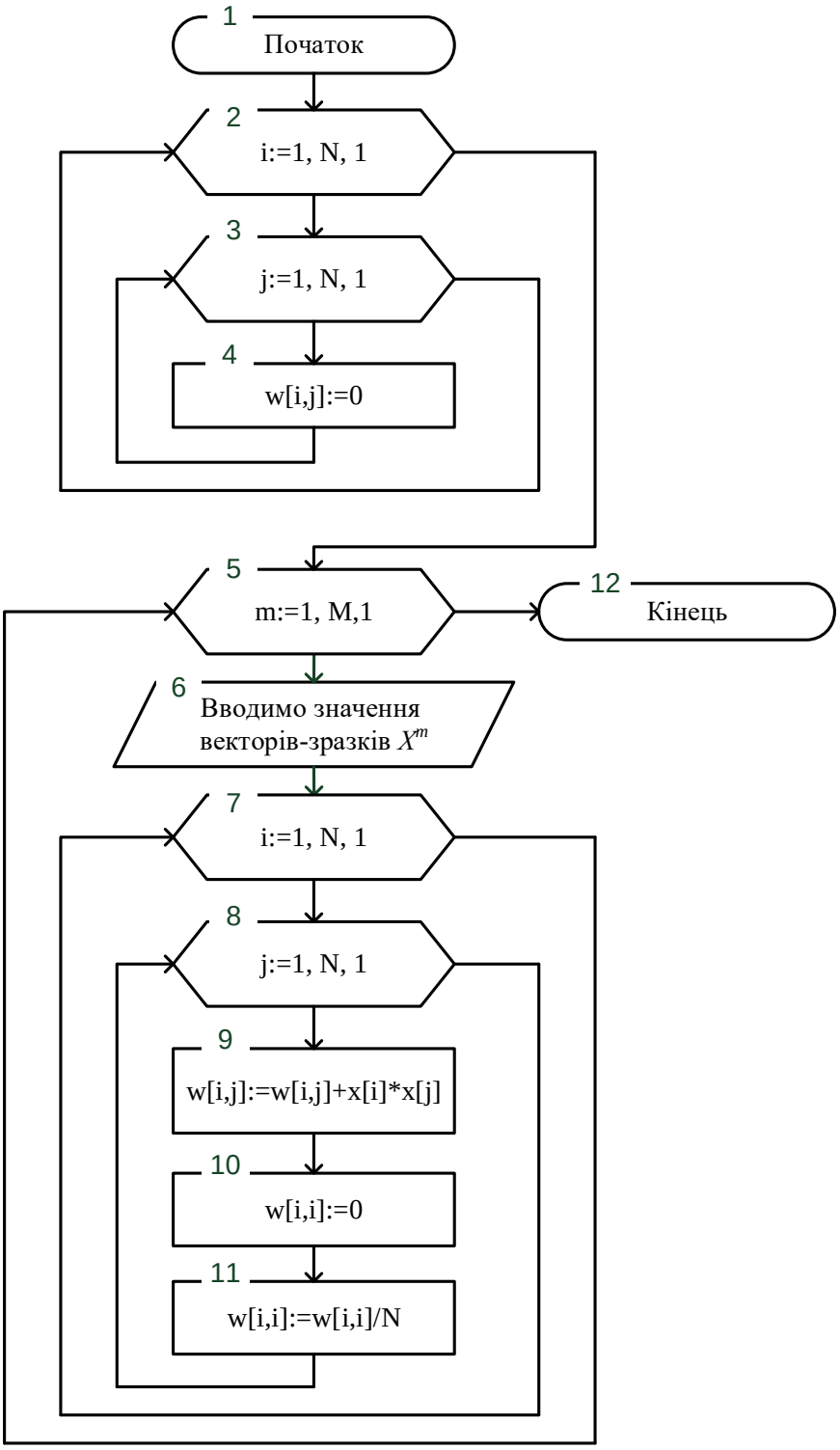


Рисунок 7.5 – Блок-схема алгоритму побудови мережі Хопфілда

Робота з мережею Хопфілда починається з навчання ваг нейронів $w_{[i,j]}$, які представлені матрицею *W*. На початку процедури навчання слід обнулити ваги нейронів (блоки № 2-4). Після чого для кожного з *M* вектоів-образів виконуємо наступну процедуру: вводимо значення вектора;

визначаємо ваги відповідних нейронів за формулою, що наведена в блоці № 9, а вагам $w_{[i,i]}$, що визначають зв'язок нейрона самого з собою привласнюємо нуль (блок №10), оскільки замкнення нейрона на себе неможливе через неінформативність такого зв'язка. Блок № 11 містить нормалізацію значень ваг, ця операція є необов'язковою, але підвищує ефективність алгоритму.

Після навчання мережі Хопфілда можна переходити до її використання для розпізнавання зашумлених або неповних образів. Алгоритм процедури розпізнавання наведено на рисунку 7.6.

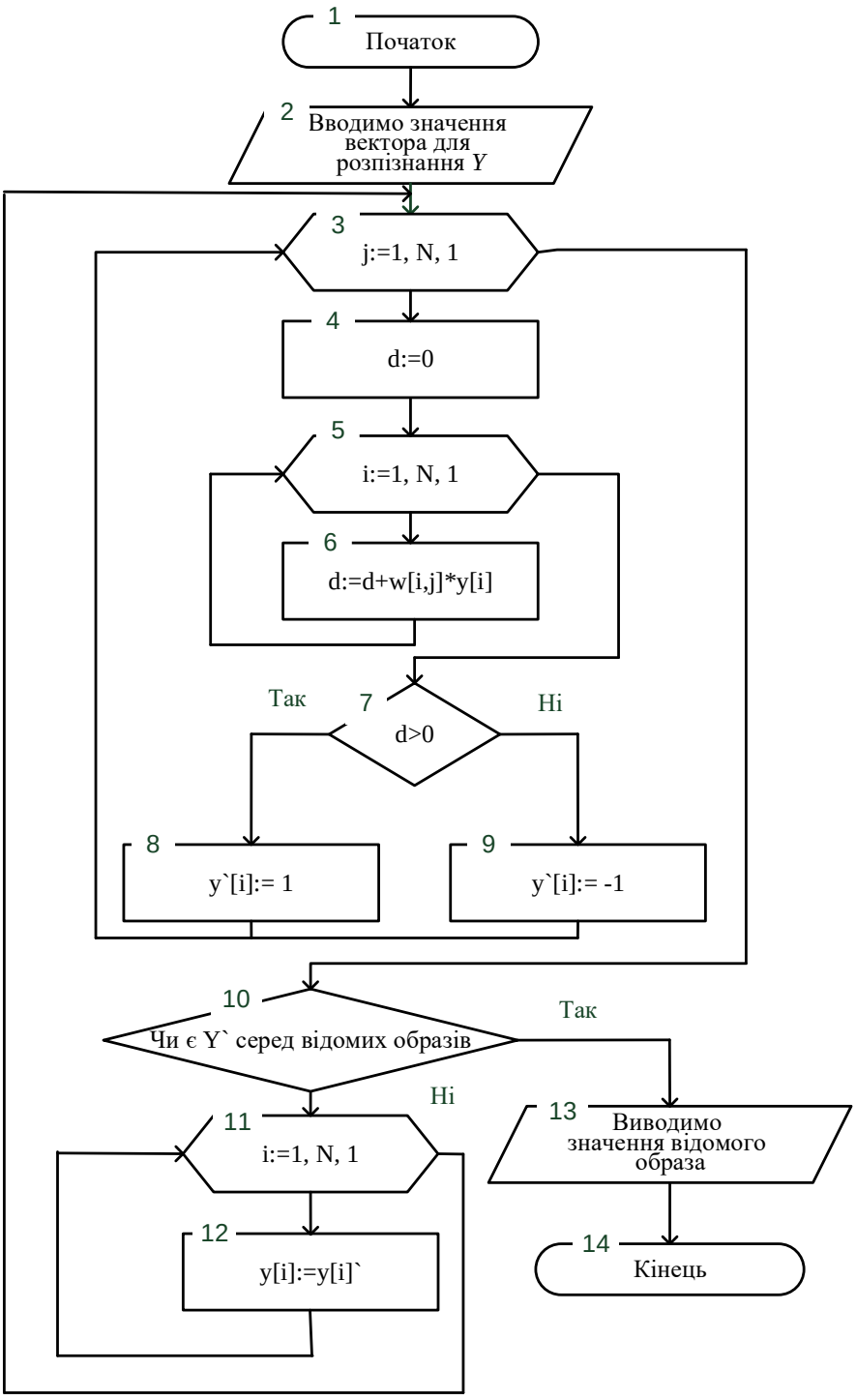


Рисунок 7.6 – Блок-схема алгоритму використання мережі Хопфілда

Припустимо надано зашумлений образ, що описується вектором *Y*. На початку алгоритма обнуляємо додаткову змінну *d*, значення якої визначається в циклі за формулою, яка наведена в блоці № 6. За значеннями змінної *d* формуємо вектор *Y`* (блоки № 7-9). Далі перевіряємо чи містися образ, що описується вектором *Y`* серед *M* векторів які вже відомі мережі

Хофілда (блок № 10). Якщо образ, який сформовано у векторі $Y^`$ відомий мережі виводимо його, інакше переписуємо значення вектора $Y^`$ у вектор Y та повторюємо процедуру розпізнавання починаючи з блока № 3.

Можливі наступні варіанти закінчення процедури використання мережі Хофілда:

1 Значення вектора $Y^`$ зійшлися у вже відомий образ і за наведеним алгоритмом цей образ був виведений.

2 Значення вектора $Y^`$ зійшлися в образ невідомий мережі (відсутній серед M векторів, на прикладі яких було побудовано мережу).

3 Значення вектора $Y^`$ не зійшлися в конкретний образ.

Задачі для самостійного розв'язання

Задача 1. Розробити програмну реалізацію алгоритма, наведеного на рисунку 7.5.

Задача 2. Розробити програмну реалізацію алгоритма, наведеного на рисунку 7.6.

Задача 3. Побудувати блок-схему алгоритму процесу класифікації за допомогою карт Кохонена.

Питання для самоконтролю

4 Дати визначення поняття «штучна нейронна мережа». Навести пояснення до даного визначення.

5 В чому полягають переваги методу, який реалізується картами Кохонена?

6 В яких сферах можуть бути застосовані Карти Кохонена?

7 З яких кроків складається процедура навчання карти Кохонена?

8 З яких кроків складається процедура використання налаштованої карти Кохонена?

9 Навести формальну постановку задачі класифікації. Навести приклади класифікації об'єктів реального світу. Дати визначення поняття «класифікатор». Навести приклади класифікаторів.

10 Навести блок-схему алгоритму навчання карти Кохонена.

11 Яке застосування мереж Хопфілда. Навести приклад відображення об’єкта на нейрону мережу Хофілда в графічному та векторному виглядах.

12 Навести блок-схему алгоритму побудови мережі Хопфілда.

13 Навести блок-схему алгоритму використання мережі Хопфілда.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1 Глибовець М.М., Олецький О.В. Штучний інтелект: Підручн. для студ. вищ. навч. закладів, що навчаються за спец. «Компютерні науки» та «Прикладна математика». – К.: Вид. дім «КМ Академія», 2002. – 366 с.

2 Вовок Е.С. Системи штучного інтелекту. – Львів: Львівська політехніка, 2018. – 334с.

3 Девятков В.В. Системы искусственного интеллекта. – М.:МГУ,2001. – 179с.

4 Люгер Ф. Дж. Искусственный интеллект. Стратегии и методы решения сложных проблем. – М.: «Вильямс», 2003. – 864 с.

5 Рассел С., Норвиг П. Искусственный интеллект: современный подход. – Изд. Диалектика, 2016. – 1406 с.

6 Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. – СПб.:Питер, 2000. – 89 с.

7 Тельнов Ю.Ф. Интеллектуальные информационные системы в экономике:Учебное пособие. – М.:СИНТЕГ, 2002. – 306 с.

8 Гнатієнко Г.М., Снитюк В.Є. Експертні технології прийняття рішень. – К.: Маклаут, 2008. – 444 с.

9 Джексон.Введение в экспертные системы. – М.: Вильямс, 2001. – 701 с.

10 Снитюк В.Є. Прогнозування. Моделі, методи, алгоритми. – К.: Маклаут, 2008. – 364 с.

11 Новожилова М.В., Петрова О.О., Гречко Н.В. Лабораторний практикум «Мова логічного програмування Prolog»: Навчальний посібник. –Х.:ХНУБА, 2012. –132 с.

12 Петрова О.О. Методи та системи штучного інтелекту. Інтелектуальний аналіз даних: Тексти лекцій. – Х.: ХНУБА, 2014р. –111 с.

13 Адаменко А., Кучуков А. Логическое программирование и Visual Prolog. – СПб:БХВ. – Петербург, 2003. – 990 с.

14 Хайкин С. Нейронные сети: полный курс. – М.: Вильямс, 2006. – 1104 с.

15 Потапов А.С. Технологии искусственного интеллекта – СПб: СПбГУ ИТМО, 2010. – 218 с.

16 Норвиг П., Рассел С. Искусственный интеллект: современный подход. – М.: Вильямс, 2007. – 1408с.

ЗМІСТ