А. С. Савченко, О. О. Синельніков

Методи та системи штучного інтелекту

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний авіаційний університет

А. С. Савченко, О. О. Синельніков

МЕТОДИ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Навчальний посібник

Київ 2017

УДК 004.8(075.8)
ББК 3 813я7
С 137

Рецензенти:
Л. Н. Беркман — д-р техн. наук, проф.,
(Навчально-науковий інститут телекомунікацій
та інформатизації Державного університету
інформаційно-комунікаційних технологій);

В. Г. Абакумов — д-р техн. наук, проф.
(Національний технічний університет України
«Київський політехнічний інститут»)

Затверджено методично-редакційною радою Національного авіаційного університету (протокол № 1/13 від 14.02.2013 р.).

Савченко А. С.

С 137 Методи та системи штучного інтелекту: навч. посібник / А. С. Савченко, О. О. Синельніков. — К. : НАУ, 2017. — 176 с.

ISBN 978-966-932-044-5

Містить опис основних методів подання знань у системах штучного інтелекту, основні відомості щодо проектування систем, що ґрунтуються на знаннях, та методи пошуку розв'язків у таких системах.

Для студентів напряму підготовки 6.050101 «Комп'ютерні науки».

УДК 004.8(075.8)
ББК 3 813я7

ISBN 978-966-932-044-5

ПЕРЕДМОВА

Дисципліна «Методи та системи штучного інтелекту» передбачає аудиторні заняття і самостійну роботу із засвоєння теоретичних положень дисципліни, набуття практичних навичок на лабораторних заняттях, виконання домашнього завдання та іспит.

Головною *метою* викладання дисципліни є надання студентам базових теоретичних знань щодо методів проектування систем штучного інтелекту та набуття початкових практичних навичок проектування інтелектуальних інформаційних управляючих систем та технічних автоматизованих систем.

У загальному випадку предмет дослідження штучного інтелекту — будь-яка інтелектуальна діяльність людини, що підпорядковується заздалегідь невідомим законам.

Штучний інтелект — це галузь досліджень, що перебуває на стику наук. Фахівці, що працюють у цій галузі, намагаються зрозуміти, яка поведінка вважається розумною (аналіз), і створити працюючі моделі цієї поведінки (синтез). Дослідники ставлять питання про те, як за допомогою нових теорій і моделей навчитися розуміти принципи й механізми інтелектуальної діяльності. Практичною метою є створення методів і техніки, необхідних для програмування «розумності» і її передачі комп'ютерам, а через них — різним системам і засобам. Інженерні методи і навички у галузі штучного інтелекту стали називати технологією знань — *knowledge engineering*.

Отже, *завданнями* вивчення дисципліни «Методи та системи штучного інтелекту» є:

– дослідження та осмислення фундаментальних понять штучного інтелекту;

– дослідження методів та моделей подання знань у системах штучного інтелекту (СШІ);

– дослідження принципів побудови СШІ, зокрема, експертних систем (ЕС);

– формування навичок із самостійного оволодіння сучасними технологіями побудови інтелектуальних систем, подання їх у загальній структурі інформаційних управляючих технологій.

У результаті вивчення даної навчальної дисципліни студент повинен *знати*:

– основні методи подання та використання знань;

– основи теорії логічного виведення;

– методи виведення на фреймових та сіткових структурах;

– сучасні програмні та інструментальні засоби для проектування СШІ;

– методи та етапи розробки ЕС.

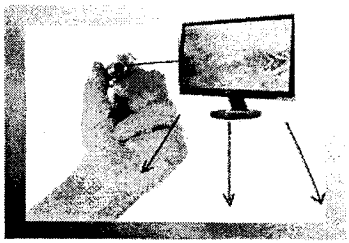
Успішне засвоєння дисципліни «Методи та системи штучного інтелекту» ґрунтується на знаннях, здобутих із курсів «Вища математика», «Дискретна математика», «Інтелектуальний аналіз даних», «Проектування інформаційних систем», «Організація баз даних та знань».

Навчальний посібник розроблений відповідно до робочої навчальної програми дисципліни «Методи та системи штучного інтелекту», яка призначена для студентів напряму підготовки 6.050101 «Комп'ютерні науки».

Мета створення навчального посібника — надання студентам теоретичних знань щодо методів проектування систем штучного інтелекту, моделей подання знань у таких системах та методів пошуку розв'язків.

Навчальний посібник розроблений відповідно до вимог кредитно-модульної системи оцінювання знань. У процесі вивчення першого модуля студенти ознайомлюються з основними термінами, поняттями та напрямами досліджень у галузі штучного інтелекту, методами пошуку розв'язків у просторі станів, моделями подання знань, розглядають продукційну модель подання знань та методи управління пошуком у продукційних системах.

При вивченні другого модуля увага студентів зосереджена на вивченні семантичних сіток та фреймів як методів подання знань, пошуку розв'язків у системах штучного інтелекту. Розглядаються основні складові, функції, етапи розробки ЕС; методи створення та галузі застосування штучних нейронних мереж; онтології як спосіб подання знань.



ВСТУП

У сучасному світі прогрес продуктивності інженера в галузі інформаційних управляючих технологій практично досягається лише в тих випадках, коли частину інтелектуального навантаження беруть на себе комп'ютери. Одним із способів досягти максимального прогресу в цій сфері є «штучний інтелект», коли комп'ютер бере на себе не лише однотипні, багато разів повторювані операції, але й сам може навчатися. Центральні задачі штучного інтелекту полягають у тому, щоб зробити обчислювальні машини корисніші і зрозуміти принципи, що лежать в основі інтелекту. Для цього необхідно знати, як штучний інтелект може допомогти у вирішенні складних проблем.

Дослідження у галузі штучного інтелекту поділяють на два основні напрями: прагматичний і біонічний.

Прагматичний напрямк ґрунтується на припущенні про те, що розумова діяльність людини — «чорний ящик». Але якщо результат функціонування штучної системи в деякому сенсі збігається з результатом діяльності експерта, то таку систему можна визнати інтелектуальною незалежно від способів отримання цього результату. За такого підходу не ставиться питання про адекватність структур і методів, що використовуються у комп'ютері, тим структурам і методам, якими користується в аналогічних ситуаціях людина, а розглядається лише кінцевий результат розв'язування конкретних задач.

З погляду кінцевого результату в прагматичному напрямі можна виділити три цільові галузі:

– *створення інструментарію*, тобто:

- мови для систем штучного інтелекту;
- дедуктивних та індуктивних методів автоматичного синтезу програм;
- лінгвістичних процесорів;

- систем аналізу й синтезу мови;
- бази знань;
- оболонок, прототипів систем;
- систем когнітивної графіки;

– *розробка методів подання й обробки знань* є однією з основ сучасного періоду розвитку штучного інтелекту;

– *інтелектуальне програмування*, яке розбивається на декілька груп. До них відносять:

- ігрові програми;
- природно-мовні програми (системи машинного перекладу, автоматичного реферування, генерації текстів);
- розпізнавальні програми;
- програми створення творів живопису й графіки.

Загальним для перелічених програм є широке використання пошукових процедур і методів розв'язування задач на перебирання, пов'язаних із пошуком і переглядом великої кількості варіантів. Ці методи застосовуються при машинному розв'язанні ігрових задач, у задачах вибору розв'язків, при плануванні доцільної діяльності в інтелектуальних системах.

Біонічний напрям досліджень у галузі штучного інтелекту ґрунтується на припущенні про те, що якщо в штучній системі відтворити структури й процеси людського мозку, то й результати розв'язування задач такою системою будуть подібні до результатів, які отримує людина. У цьому напрямі досліджень виокремлюють:

– *нейробіонічний підхід*, в основі якого лежать системи елементів, які здатні так само, як і нейрони головного мозку, відтворювати деякі інтелектуальні функції. Прикладні системи, розроблені на основі цього підходу, називаються нейронними мережами;

– *структурно-евристичний підхід*, в основі якого лежать знання про поведінку об'єкта або групи об'єктів, за якими спостерігають, і міркування про ті структури, які могли б забезпечити реалізацію форм поведінки, яку спостерігають. Прикладом таких систем є мультиагентні системи;

– *гомеостатичний підхід* — це коли задача, яку розв'язують, формулюється в термінах еволюціонуючої популяції організмів — сукупності підсистем, що ворогують і взаємодіють, у результаті функціонування яких забезпечується потрібна рівновага (стійкість)

усієї системи в умовах впливів середовища, яке постійно змінюється. Такий підхід реалізовано в прикладних системах на основі генетичних алгоритмів.

Навчальна дисципліна «Методи та системи штучного інтелекту» є теоретичною основою сукупності знань та вмінь, що формують профіль фахівця в галузі комп'ютерних наук.

Основна частина даного навчального посібника «Методи та системи штучного інтелекту» складається з семи розділів.

У розділі 1 розкрито сутність основних понять та напрямів дослідження в галузі штучного інтелекту, наведено коротку історію розвитку досліджень у цій сфері та галузі застосування.

Розділ 2 присвячено методам пошуку роз'вязків у системах штучного інтелекту. Проаналізовано способи подання інтелектуальних задач, методи інформованого та неінформованого пошуку роз'вязків у просторі станів, які застосовуються в системах штучного інтелекту. Також розглянуто методи пошуку роз'вязків інтелектуальних задач у разі зведення задачі до сукупності підзадач.

У розділі 3 описано основні моделі подання знань у системах штучного інтелекту. Розглянуто теоретичні основи логічних моделей подання знань — числення висловлювань та предикатів першого порядку. Викладено основні відомості щодо продукційних, фреймових моделей та семантичних сіток. Розглянуто методи управління пошуком роз'вязків у продукційних системах.

У розділі 4 наведено основні відомості про ЕС. Розглянуто відмінність ЕС від інших програм та програм у галузі штучного інтелекту, призначення та принципи побудови статичних та динамічних ЕС, основні етапи розробки та базові функції. Визначено класи задач, для яких використання технології ЕС є ефективним.

Розділ 5 присвячено нейронним мережам. Розглянуто історію досліджень у цьому напрямі, галузі застосування, класифікацію за методами навчання та напрямом поширення помилки, особливості та алгоритми функціонування нейронних мереж різних типів.

У розділі 6 розкрито сутність онтологічного підходу до подання та інтеграції знань у розподілених інформаційних середовищах, зокрема Інтернеті. Розглянуто класифікацію онтологій за ступенем формальності, метою створення та за наповненням. Наведено лексичні онтології для обробки текстів природною мовою.

У розділі 7 розглянуто основні засоби створення систем штучного інтелекту. Наведено коротку характеристику мов функціо-

нального та логічного програмування. Описано особливості створення систем штучного інтелекту за допомогою декларативної мови *Prolog*. Наведено приклади створення програм у середовищі *Visual Prolog*.

Після кожного розділу подано висновки щодо викладеного матеріалу, запитання та завдання для самоперевірки, а також вправи для закріплення знань.

Знання, здобуті в процесі вивчення дисципліни «Методи та системи штучного інтелекту», є базовими для опанування таких дисциплін, як «Проектування баз даних та експертних систем», «Корпоративні інформаційні системи» тощо.

РОЗДІЛ 1

ОСНОВНІ ПОНЯТТЯ В ГАЛУЗІ ШТУЧНОГО ІНТЕЛЕКТУ

1.1. Поняття штучного інтелекту

Термін інтелект *intelligence* походить від латинського *intellectus*, що означає розум, розумові здібності людини.

У межах дисципліни «Методи та системи штучного інтелекту» ***інтелектом*** називатимемо здатність мозку розв'язувати (інтелектуальні) задачі шляхом набуття, запам'ятовування і цілеспрямованого перетворення знань у процесі навчання на досвіді й адаптації до різноманітних обставин.

Відповідно, ***штучний інтелект*** (ШІ) (*artificial intelligence — AI*) розумітимемо як властивість автоматичних систем брати на себе окремі функції інтелекту людини, наприклад, вибирати й ухвалювати оптимальні рішення на підставі раніше одержаного досвіду й раціонального аналізу зовнішніх дій.

У наведеному вище визначенні поняття інтелекту під терміном *знання* мається на увазі не лише та інформація, яка надходить до мозку через органи чуття. Такі знання надзвичайно важливі, але недостатні для інтелектуальної діяльності. Об'єктам нашого навколишнього середовища притаманна властивість не лише впливати на органи чуття, але й перебувати один із одним у певних відносинах. Щоб здійснювати в навколишньому середовищі інтелектуальну діяльність (або хоча б просто існувати), необхідно мати в системі знань модель цього світу.

У цій інформаційній моделі навколишнього середовища реальні об'єкти, їх властивості й відносини між ними не лише відображаються і запам'ятовуються, але і, як на цьому наголошено в даному визначенні інтелекту, можуть «*цілеспрямовано перетворюватися*». При цьому важливим є те, що модель зовнішнього середовища формується «в процесі навчання на досвіді й адаптації до різноманітних обставин».

1.2. Історія розвитку досліджень у галузі штучного інтелекту

Етап I — кінець 50-х рр. XX ст. — початок досліджень у галузі штучного інтелекту.

1956 р. — першою програмою штучного інтелекту стала програма «Логік-Теоретик», призначена для доведення теорем в численні висловлювань.

1957 р. — створено програму для гри в шахи *NSS* (Ньюел, Шо, Саймон), яка надалі привела до концепції *Універсального вирішувача задач*. Ця програма, аналізуючи відмінності між ситуаціями й конструюючи цілі, добре вирішувала головоломки типу «Ханойська башта» або обчислювала невизначені інтеграли.

Евристичному методу розв'язування задачі, властивому людському мисленню «взагалі», для якого характерне виникнення припущень про шлях розв'язання задачі з подальшою їх перевіркою, протиставлявся *алгоритмічний метод*, який інтерпретувався як механічне здійснення заданої послідовності кроків, що детерміновано приводить до правильної відповіді.

1965 р. — поява *методу резолюцій* Дж. Робінсона, що ґрунтується на доведенні теорем у логіці предикатів шляхом приведення до суперечності. На основі даного методу побудовано мову *Prolog*.

Дослідницьким полігоном для розвитку методів ШІ на першому етапі стали ігри, головоломки, математичні задачі. Деякі з цих задач стали класичними в літературі зі ШІ (задачі про мавпу і банани, місіонерів і людоїдів, Ханойську башту, гра в 15 тощо). Вибір таких задач зумовлювався простотою і зрозумілістю предметного середовища (середовища, у якому розв'язується задача), її відносно малою громіздкістю, можливістю достатньо легкого підбору і навіть штучного конструювання «під метод». Розквіт таких досліджень припадає на кінець 60-х років XX ст., після чого було зроблено перші спроби використання розроблених методів для задач, розв'язуваних не в штучних, а в реальних предметних середовищах.

Етап II. На початку 70-х рр. XX ст. відбувся якісний стрибок у дослідженнях.

По-перше, дослідники поступово усвідомили, що всім раніше створеним програмам бракує найважливішого — глибоких знань у відповідній галузі. Тому для істотного поліпшення результатів

роботи якої-небудь програми ШІ вимагається не просто удосконалити евристику або якісь числові коефіцієнти, з якими працює програма, а навпаки, необхідно *використовувати в ній методи логічних міркувань і накопичені в досвіді знання, подані в символьній формі.*

По-друге, виникає конкретна проблема: як передати ці знання програмі, якщо її безпосередній творець ними не володіє. Тобто сама програма повинна їх виділяти з даних, одержуваних від експерта. Дослідники стикнулися з необхідністю забезпечити системи штучного інтелекту можливостями, яких немає в звичних мовах програмування. У даному разі є розмежування між висновком про якийсь факт і використовуванням цього факту. На противагу цьому звична мова програмування дозволяє виражати лише здійснимі завдання або вказівки.

До 1970 р. було створено безліч програм, що ґрунтувалися на цих ідеях. Перша з них — програма *DENDRAL*. Вона призначена для породження структурних формул хімічних з'єднань на підставі інформації, що надходить від мас-спектрометра.

Дослідження систем ШІ при їх функціонуванні в реальному світі привело до постановки задачі створення *інтегральних роботів.*

Описані вище результати починають використовуватися в робототехніці при керуванні роботою нерухомих або мобільних роботів, які діють у реальному тривимірному просторі.

При цьому постає проблема створення штучних органів сприйняття. У системах технічного зору сприймальним пристроєм є телекамера. При розпізнаванні зорових образів значну роль відіграють методи аналізу зорових сцен, пов'язані з визначенням контурів предметів. Якість розв'язання таких задач відтоді значно підвищилась.

Проведення експериментів довело необхідність вирішення таких проблем:

– представлення знань про середовище функціонування;

– зорове сприйняття;

– побудова складних планів поведінки в динамічних середовищах;

– спілкування з роботами природною мовою.

Ці проблеми були ясніше сформульовані й поставлені перед дослідниками на наступному етапі.

Етап III почався з середини 70-х рр. XX ст. Його характерною ознакою стало зміщення центру уваги дослідників зі створення автономних систем, які самостійно вирішували поставлені перед ними завдання в реальному середовищі, до *створення людино-машинних систем*, які інтегрували в одне ціле інтелект людини і здатності обчислювальної машини для досягнення загальної мети — вирішення завдання, поставленого перед системою.

Таке зрушення обумовлене двома причинами:

– реальні задачі не можуть бути розв'язані методами, розробленими для експериментальних задач;

– підвищення ефективності внаслідок взаємодоповнення можливостей людини й обчислювальних машин. На перший план виходить розробка засобів взаємодії.

Розвиток досліджень із ШІ в даному напрямі зумовлювався також різким зростанням виробництва засобів обчислювальної техніки та їх здешевленням. Цей напрям досі лишається найбільш перспективним.

Підсумовуючи викладене, можна виділити три основні підходи в моделюванні ШІ.

У рамках ***першого підходу*** об'єктом досліджень є *структура і механізми роботи мозку людини*, а кінцева мета полягає в розкритті таємниць мислення.

Необхідними етапами досліджень у цьому напрямі є побудова моделей на основі психофізіологічних даних, проведення експериментів із ними, висунення нових гіпотез щодо механізмів інтелектуальної діяльності, вдосконалення моделей.

Другий підхід як об'єкт дослідження розглядає ШІ. Тут ідеться про *моделювання інтелектуальної діяльності за допомогою обчислювальних машин*.

Метою робіт у цьому напрямі є створення алгоритмічного та програмного забезпечення обчислювальних машин, що дозволяє розв'язувати інтелектуальні задачі не гірше за людину.

Третій підхід орієнтований на *створення змішаних людино-машинних*, або, як ще говорять, *інтерактивних інтелектуальних систем*, на симбіоз можливостей природного і штучного інтелекту.

Найважливішими проблемами в цих дослідженнях є оптимальний розподіл функцій між природним та штучним інтелектом і організація діалогу між людиною і машиною.

1.3. Поняття інтелектуальної системи та інтелектуальної задачі

Для з'ясування, чим інтелектуальні задачі відрізняються від просто задач, необхідно ввести термін «алгоритм» — один із наріжних термінів кібернетики.

Під ***алгоритмом*** розуміють точний припис про виконання в певному порядку системи операцій для розв'язання будь-якої задачі з деякого даного класу (множини) задач.

Знаходження алгоритмів є природною метою людини при розв'язуванні нею різноманітних класів задач і пов'язане з тонкими і складними міркуваннями, що потребують великої винахідливості і високої кваліфікації. Прийнято вважати, що така діяльність вимагає участі інтелекту людини.

Задачі, пов'язані з відшуканням алгоритму розв'язування класу задач певного типу, називатимемо ***інтелектуальними задачами.***

Щодо задач, алгоритми розв'язання яких вже встановлено, то, як відзначає відомий фахівець у галузі ШІ М. Мінський, «надмірно приписувати їм таку властивість, як "інтелектуальність"». Якщо алгоритм вже знайдено, процес розв'язання відповідних задач стає таким, що його можуть виконати людина, обчислювальна машина (належним чином запрограмована) або робот, що не має ані найменшого уявлення про сутність самої задачі.

Діяльність мозку (з інтелектом), направлену на розв'язання інтелектуальних задач, називатимемо ***мисленням***, або ***інтелектуальною діяльністю***. Характерними рисами інтелекту, що виявляються в процесі розв'язання задач, є здібність до навчання, узагальнення, накопичення досвіду (знань і навичок) і адаптації до умов, що змінюються, в процесі розв'язування задач.

Існують також суто поведінкові (функціональні) визначення інтелекту. Наприклад, за О. М. Колмогоровим, будь-яка матеріальна система, з якою можна достатньо довго обговорювати проблеми науки, літератури й мистецтва, є інтелектуальною.

Прикладом поведінкового трактування інтелекту може стати відомий тест британського математика Алана Т'юринга, описаний у 1950 р. Його значення полягає в такому. У різних кімнатах перебувають люди й машина. Вони не можуть бачити один одного, але можуть обмінюватися інформацією (наприклад, за допомогою електронної пошти).

Якщо в процесі діалогу між учасниками гри людям не вдається встановити, що один з учасників — машина, то таку машину можна вважати з інтелектом. Слід зауважити, що до сьогодні цей тест не вдалося пройти жодній програмі з галузі ШІ.

Запрограмований таким чином комп'ютер повинен володіти перерахованими нижче засобами:

1) *обробки текстів природними мовами*, що дозволяють успішно спілкуватися з комп'ютером, наприклад, англійською мовою;

2) *подання знань*, за допомогою яких комп'ютер може записати в пам'ять те, що він дізнається або прочитає;

3) *автоматичного формування логічних висновків*, що забезпечують можливість використовувати інформацію, яка зберігається для пошуку відповідей на запитання і виведення нових висновків;

4) *машинного навчання*, які дозволяють пристосовуватися до нових обставин, а також знаходити й екстраполювати ознаки стандартних ситуацій.

У повному тесті Т'юринга передбачено також використання відеосигналу для того, щоб експериментатор міг перевірити здібності до сприйняття випробовуваного об'єкта. Для цього комп'ютеру необхідний:

5) *машинний зір* для сприйняття об'єктів;

6) *засоби робототехніки* для маніпулювання об'єктами і переміщення в просторі.

Перелічені шість напрямів досліджень становлять основну частину ШІ.

Враховуючи викладене вище, наведемо декілька визначень інтелектуальної системи.

Визначення 1. ***Інтелектуальною*** називається система, здатна цілеспрямовано, залежно від стану інформаційних входів, змінювати не лише параметри функціонування, але й сам спосіб своєї поведінки, причому спосіб поведінки залежить не лише від поточного стану інформаційних входів, але також і від попередніх станів системи.

Наприклад, будь-який живий організм — інтелектуальна система, якій притаманна довготривала пам'ять і здібність до самонавчання. Щеня, вперше погнавшись за кішкою, одержить по морді. Зустрівшись із нею наступного разу, воно навряд чи повторить свої дії, найімовірніше воно втече або покаже зуби чи проявить ще одну з тисяч можливих реакцій.

Технічні ж системи найчастіше не є інтелектуальними, тобто реакція на одну й ту ж подію кардинально не відрізнятиметься. Річ у тім, що живий організм, крім того, що запам'ятовує параметри і ситуації, може виробити нові правила поведінки.

Визначення 2. ***Інтелектуальною*** називається система, що моделює на комп'ютері поведінку людини.

Друге визначення з'явилося в 60-х рр. XX ст., коли вважалося, що мозок людини можна змоделювати на комп'ютері. Клітини мозку — нейрони — програмно описувалися спеціальними математичними методами. На вхід програми подавалися деякі дані (на вхід клітини мозку в живому організмі подається електричний сигнал), на виході знімалися результати, які звірялися з еталоном. Залежно від того, наскільки одержані результати відхилялися від еталону, до розрахункових коефіцієнтів вносилися зміни. Залежно від кількості циклів такого «навчання» результати роботи програми поступово дедалі більш наближалися до результатів роботи дуже маленького елемента мозку людини.

Ідея про можливість повторити мозок на комп'ютері дотепер зазнавала невдач. Проте теорія нейронних мереж, нейромережевий підхід довели свою корисність на низці практичних додатків. Позитивні результати одержано передусім на задачах прогнозу значень параметрів і розпізнавання образів.

Визначення 3. ***Інтелектуальною*** називається система, що дозволяє підсилити інтелектуальну діяльність людини внаслідок ведення з нею осмисленого діалогу.

До кінця 80-х рр. XX ст. стало абсолютно очевидно, що створити універсальний штучний розум неможливо. Більше того, з'ясувалося, що це абсолютно не потрібно. Слід створювати вузькоспеціалізовані інтелектуальні системи, які не замінюють людину, але доповнюють її. Необхідно, щоб комп'ютер швидко аналізував ситуацію, генерував варіанти дії на основі величезної пам'яті і пропонував їх людині, а людина розглядала запропоновані варіанти і пояснювала, чому той або інший варіант поганий. Комп'ютер, враховуючи одержані роз'яснення, знову аналізував би всі варіанти дії і видавав нові, а людина вибирала відповідний варіант і відповідала б за його реалізацію.

Наприклад, система автоматичного наведення ракет знайшла ціль. Ціль було виявлено практично миттєво, людина навіть не

встигла її помітити. Ракета була автоматично наведена на ціль. Цілі був посланий запит «свій-чужий». Ціль з'явилася на пульті керування перед оператором, людина ухвалила рішення про ураження, вибравши тип зброї і натиснувши на кнопку «знищити». У разі повністю автоматичного ведення цілі існувала б реальна небезпека знищити свій літак. І навпаки, якби людина наводила ракету на ціль, надсилала запит, було б утрачено дорогоцінний час.

Таким чином, сьогодні ШІ — інструмент, що сам навчається і призначений для підсилення діяльності людини з генерації та ухвалення рішень.

1.4. Галузі застосування систем штучного інтелекту

Застосування ШІ **ефективне** для:

– *слабкоструктурованих наочних галузей*, тобто галузей, алгоритм дії у яких наперед невідомий. Для таких галузей характерна неясність і нечіткість у вхідних даних, наявність різних шумів. Разом із тим рішення вимагається ухвалювати однозначні, чіткі й зрозумілі, бажано також передбачити ефективність цих рішень. До таких галузей відносяться медицина, економічний менеджмент, управління складними технічними об'єктами, психологія, лінгвістика тощо;

– *галузі освіти*. У низці галузей навчання припускає дуже велику кількість циклів повторень навчальної інформації і дій в спеціально сконструйованій навчальній ситуації з різними варіаціями: керування рухом літака, заучування іноземних слів і фраз щодо певної ситуації тощо. Навчання окремим навичкам і умінням за допомогою обчислювальних машин набагато дешевше, а головне ефективніше;

– *систем пошуку інформації*, оскільки вони є основою для побудови глобальних інформаційних сховищ. Одним із найяскравіших прикладів може бути «м'який пошук».

Галузями **неефективного** застосування методів ШІ є добре структуровані предметні галузі. До них належать передусім точні та інженерні науки (математика, фізика, опір матеріалів тощо). Це зумовлено тим, що для розв'язування задач у цих науках вже існують свої власні надійні алгоритми й методи, випробувані протягом десятків або навіть сотень років.

Далі наведено деякі галузі застосування ШІ з прикладами реалізації.

1. *Системи, що імітують творчі процеси:*

– доведення теорем на основі дедукції;

– автоматичний синтез програм, доведення їх правильності;

– ігрові програми. Розвиток теорії пошуку в просторі вирішення;

– програми генерації прозаїчних і поетичних текстів, музичних творів;

– інтелектуальні навчальні системи (т'ютори).

2. *Сприйняття і розпізнавання образів*:

– розпізнавання мови;

– розпізнавання тексту машинописного і рукописного (*ABBYY FineReader*);

– розпізнавання образів, сцен.

3. *Інформаційні системи, що ґрунтуються на знаннях*:

– експертні системи;

– системи машинного перекладу;

– системи автоматичного реферування — перетворення первинної інформації для її стиснення зі збереженням основної ідеї документа;

– індуктивні системи — виведення закономірностей на підставі даних спостереження або експериментальних даних.

4. *Інтелектуальні інформаційні системи* — великі й дуже великі програми, призначені для розв'язування задач у предметній галузі на основі математичних і алгоритмічних моделей, які можуть вести осмислений діалог із користувачем задля спрощення управління, скорочення обсягу роботи людини, підвищення якості тощо:

– системи підтримки ухвалення рішень для людей, що працюють в екстремальних умовах (дефіцит інформації і часу, висока ціна помилки) — оператори керування рухом повітряних суден, льотчики, військові при протизенітному маневрі;

– системи інтелектуального моніторингу: стеження за станом пацієнтів у реанімації, контроль технологічних процесів на екологічно небезпечних підприємствах (хімічні заводи, атомні електростанції). Монотонність процесу нагляду «присипляє» оператора. При аварії він від «раптовості» переходить у стресовий стан і неадекватно реагує на ситуацію.

5. *Робототехніка.* З погляду «інтелектуальності» розрізняють декілька поколінь роботів:

– перше покоління — роботи-маніпулятори, які діють згідно із затвердженою і незмінною програмою (наприклад, подаючи заготовки до верстата). Якщо в процесі роботи з будь-яких причин зміниться відстань до заготовки, робот її втратить;

– друге покоління — адаптивні роботи, оснащені великою кількістю датчиків. Застосовуються, наприклад, для зварювання кузовів автомобілів;

– третє покоління — робото-технічні системи (дослідницькі роботи на планетах, роботи, що працюють за несприятливих умов середовища — радіація, хімічне ураження).

6. *Системи спілкування з обчислювальними машинами природною мовою* — системи типу запитання-відповідь, синтезатори мови.

Висновки

1. Штучним інтелектом вважається властивість автоматичних систем брати на себе окремі функції інтелекту людини.

2. Для інтелектуальної діяльності системі необхідні знання про навколишнє середовище, тобто модель предметної галузі.

3. Історію розвитку досліджень у галузі ШІ можна розділити на три основні етапи:

– І етап (кінець 50-х рр. XX ст.) — початок досліджень. Головна увага приділяється евристичному методу розв'язування задачі як властивому людському мисленню «взагалі». Розробка універсального вирішувача задач, поява методу резолюцій. Дослідження проводились здебільшого на головоломках та іграх;

– ІІ етап (початок 70-х рр. XX ст.) — якісний стрибок у дослідженнях. Зміщення центру уваги дослідження в реальні предметні галузі. Розробка інтегральних роботів;

– ІІІ етап (з середини 70-х рр. XX ст.) — головна увага приділяється не системам, які функціонують автономно і самостійно вирішують поставлені перед ними завдання в реальному середовищі, а створенню людино-машинних систем, що інтегрують в єдине ціле інтелект людини і здатності обчислювальної машини для досягнення загальної мети — вирішення завдання, поставленого перед системою.

4. Виокремлюють три основні підходи в моделюванні штучного інтелекту: моделювання структури й механізми роботи мозку людини, моделювання інтелектуальної діяльності, створення змішаних людино-машинних систем.

5. Інтелектуальними задачами вважаються задачі, пов'язані зі знаходженням алгоритму розв'язання класу задач певного типу.

6. Інтелектуальна система — це система, яка здатна:

– цілеспрямовано, залежно від стану інформаційних входів, змінювати не лише параметри функціонування, але й сам спосіб своєї поведінки;

– моделювати поведінку людини на комп'ютері;

– підсилити інтелектуальну діяльність людини внаслідок ведення з нею осмисленого діалогу.

7. Застосування ШІ ефективне для слабо структурованих наочних галузей, сфери освіти та систем пошуку інформації.

8. Недоцільно використовувати методи ШІ в структурованих предметних галузях, передусім у точних та інженерних науках.

Вправи

1. Запропонуйте власне визначення штучного інтелекту.
2. Визначте, у якій із предметних галузей застосування методів та СШІ найбільш доцільно.
3. Наведіть критичні зауваження щодо т'юрингівського критерію «розумності» комп'ютерної програми.
4. Назвіть та обґрунтуйте потенційно негативні наслідки розвитку штучного інтелекту для суспільства.
5. Який із трьох головних підходів у моделюванні ШІ є найбільш перспективним? Обґрунтуйте відповідь.

Запитання та завдання для самоперевірки

1. У чому полягає сутність тесту Т'юринга? Що він доводить?
2. Яка система може називатися інтелектуальною?
3. Схарактеризуйте три основні етапи розвитку досліджень у галузі ШІ.
4. Які основні підходи виділяють у моделюванні ШІ?
5. Назвіть приклади ефективного застосування систем ШІ.
6. Чим зумовлена висока ефективність застосування методів ШІ для слабкоструктурованих галузей?
7. Чому застосування методів ШІ неефективне для точних та інженерних наук?

РОЗДІЛ 2

МЕТОДИ ПОШУКУ РОЗВ'ЯЗКІВ У СИСТЕМАХ ШТУЧНОГО ІНТЕЛЕКТУ

2.1. Способи подання інтелектуальних задач, їх переваги та недоліки

Виокремлюють локальний та системний підходи до подання інтелектуальних задач.

Локальний, або «задачний», підхід ґрунтується на тому, що для кожної задачі, властивої творчій діяльності людини, можна знайти спосіб її розв'язання на електронній обчислювальній машині (ЕОМ), який, будучи реалізований у вигляді програми, дає результат або подібний результату, отриманому людиною, або навіть кращий.

Системний, або заснований на знаннях, підхід пов'язаний із уявленням про те, що вирішення окремих творчих завдань не вичерпує всієї проблематики ШІ.

Природний інтелект людини здатний не лише розв'язувати творчі завдання, а за потреби навчається того чи іншого виду творчої діяльності. Тому і програми ШІ повинні бути орієнтовані не лише або не стільки на розв'язання конкретних інтелектуальних задач, скільки на створення засобів, що дозволяють автоматично будувати програми розв'язання інтелектуальних задач, коли в таких програмах виникне потреба.

У такому підході проблема створення інтелектуальних систем розглядається як частина загальної теорії програмування. При цьому підході для складання інтелектуальних програм використовують звичайні програмні засоби, що дозволяють писати потрібні програми за описами задач професійною природною мовою.

Усі метазасоби, що виникають при цьому на базі часткового аналізу природного інтелекту, розглядають тут лише з точки зору створення інтелектуального програмного забезпечення, тобто комплексу засобів, що автоматизують діяльність самого програміста.

2.2. Пошук розв'язків інтелектуальних задач у просторі станів

У більшості СШІ інформації, доступної стратегії керування недостатньо для того, щоб вибрати відповідне правило на кожному кроці процесу. Тому самі процедури виведення є, як правило, пошуковими процедурами.

Оскільки поняттям знання про завдання можуть відповідати стани завдання, а правилам виведення — оператори переходу з одного стану в інший від завдання до підзадач, то процедура пошуку розв'язків називається ***пошуком у просторі станів***.

Пошук розв'язків у просторі станів зводиться до визначення послідовності операторів, які відображають початкові стани в цільові. Причому якщо така послідовність не одна й задано критерій оптимальності, то пошук зводиться до визначення оптимальної послідовності операторів, які забезпечують оптимум заданого критерію оптимальності.

Методи пошуку розв'язків у просторі станів зручно розглянути, використовуючи *дерево* (*граф*) *станів*. На дереві станів (рис. 2.1) пошук розв'язків зводиться до визначення шляху (оптимального, якщо задано критерій оптимальності) від кореня дерева до ***цільової вершини*** *A*, тобто до вершини, яка відповідає цільовому стану. Вершини *B* і *C* є *вершинами, що розкриваються* (обчислюваними, проміжними). Вершина *D термінальна*, тобто завершальна. Ребра, що з'єднують вершини, означають приєднані процедури, які необхідно виконати, щоб перейти до наступного стану (вершини).

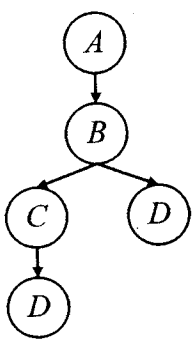


Рис. 2.1. Приклад дерева станів

Процес застосування приєднаних процедур називають породженням вершин або *перебиранням варіантів*. При породженні нової вершини обов'язково запам'ятовується покажчик на стару. У кінці перебирання сукупність цих покажчиків утворює шлях розв'язування задачі, який записується разом з іменами виконаних приєднаних процедур. Для знаходження розв'язку необхідно знову і знову вибирати, перевіряти й розкривати вузли доти, доки не буде знайдено розв'язок або не залишиться більше станів, які можна було б розгорнути. Порядок, у якому відбувається розгортання станів, визначається *стратегією пошуку*.

Результатом застосування будь-якого алгоритму розв'язування задачі є або невдале завершення, або отримання розв'язку. Деякі алгоритми можуть входити до нескінченого циклу й не повертати ніякого результату. Продуктивність алгоритму оцінюється за допомогою чотирьох показників:

– *повнота* дає відповідь на запитання, чи гарантує алгоритм виявлення розв'язку, якщо він є;

– *оптимальність* відповідає за те, чи забезпечує дана стратегія знаходження оптимального розв'язку (тобто такого, що має найменшу вартість шляху серед усіх інших розв'язків);

– *затрачений час*, за який алгоритм знаходить розв'язок;

– *необхідні ресурси*, тобто який обсяг пам'яті необхідний для здійснення пошуку.

Методи пошуку в одному просторі призначені для використання за таких умов: області невеликої розмірності, повнота моделі, точні та повні дані.

Стратегії пошуку в одному просторі можна класифікувати так:

1. Неінформований пошук («сліпі» методи):

– у ширину;

– у глибину (з обмеженням глибини, з ітеративним поглибленням).

2. Інформований пошук (евристичний пошук).

Пошук може здійснюватись у різних напрямках.

Прямий пошук йде від вихідного стану і, як правило, використовується тоді, коли цільовий стан задано неявно.

Зворотний пошук йде від цільового стану і використовується тоді, коли початковий стан задано неявно, а цільовий — явно.

Двонапрямлений метод пошуку дозволяє одночасно проводити два пошуки в прямому й зворотному напрямку, зупиняючись після того, як два процеси пошуку зустрінуться на середині.

У такому разі передбачається перевірка в одному або в обох процесах пошуку кожного вузла перед його розгортанням для визначення того, чи не знаходиться він на периферії іншого дерева пошуку. У разі позитивного результату перевірки розв'язок знайдено і пошук припиняється.

Переваги цього методу:

– незначний час пошуку, оскільки перевірка належності вузла до іншого дерева пошуку може бути виконана за сталий час за допомогою хеш-таблиці;

– повнота методу;
– оптимальність.

Серед недоліків слід відзначити значну витрату пам'яті, оскільки необхідно зберігати принаймні одне з дерев пошуку для того, щоб можна було виконати перевірку належності до іншого дерева.

На рис. 2.2 показано схематичне зображення двонапрямленого пошуку в тому стані, коли він має успішно закінчитися після того, як одна з гілок, яка виходить із початкового вузла, зустрінеться з гілкою, що виходить із цільового вузла.

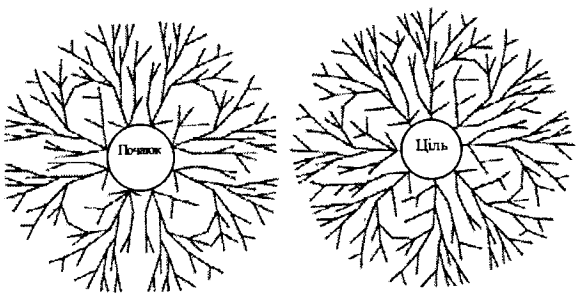


Рис. 2.2. Приклад двонапрямленого пошуку

2.3. Методи «сліпого» пошуку

Методи ***неінформованого***, або «сліпого», пошуку (повного перебирання) означають, що в даних стратегіях не використовується додаткова інформація про стани, крім тієї, яка подана у завданні. Такі стратегії можуть лише виробляти наступників і відрізняти цільовий стан від нецільового. Потребують великої витрати часу.

Пошук ***у ширину*** є досить простою стратегією, у якій спочатку розгортається кореневий вузол, потім — усі його наступники (рис. 2.3). Після цього розгортаються наступники цих наступників і т. д. Тобто перш ніж відбувається розгортання будь-яких вузлів на наступному рівні, розгортаються всі вузли на даній конкретній глибині в дереві пошуку. Перевага — повнота.

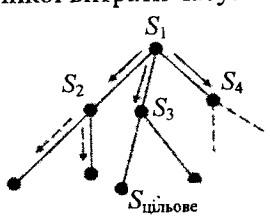


Рис. 2.3. Дерево пошуку в ширину

Недоліки: неоптимальний, значні витрати часу та пам'яті, оскільки потрібно зберігати всі проміжні значення.

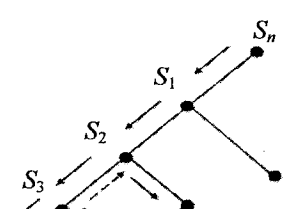


Рис. 2.4. Дерево пошуку в глибину

Пошук ***у глибину*** завжди розгортає найглибший вузол у дереві пошуку (рис. 2.4). Пошук безпосередньо переходить на найглибший рівень дерева пошуку, на якому вузли не мають наступників.

У міру того, як ці вузли розгортаються, вони видаляються з периферії, тому надалі пошук «відновлюється» з наступного поверхневого вузла, який все ще має недосліджених наступників. Перевага — незначні потреби в пам'яті (зберігання лише єдиного шляху від кореня до листового вузла). Недолік — неоптимальний, може бути зроблений неправильний вибір і перехід у тупикову ситуацію, пов'язану з проходженням вниз по дуже довгому (чи навіть нескінченному) шляху, при тому, що інший варіант міг би привести до розв'язку, що знаходиться недалеко від кореня дерева пошуку.

Пошук ***із обмеженням глибини*** передбачає застосування під час пошуку заздалегідь певної межі глибини l. Це дозволяє вирішити проблему необмежених дерев (тобто нескінченного шляху). Вузли на глибині l розглядаються як такі, що не мають наступників. Однак за неправильного вибору $l < d$, коли поверхнева ціль виходить за межі глибини, розв'язок не буде знайдено. Така ситуація цілком імовірна, якщо значення d невідомо. Крім того, пошук із обмеженням глибини буде неоптимальним при виборі значення $l > d$.

Пошук у глибину ***з ітеративним поглибленням*** передбачає поступове збільшення глибини пошуку (яка спочатку дорівнює 1, потім 2, 3 і т. д.) доти, доки не буде знайдено ціль. Така подія відбувається після того, як межа глибини досягає значення d, глибини поверхневого цільового вузла.

У пошуку з ітеративним поглибленням поєднуються переваги пошуку в глибину та пошуку в ширину. Переваги методу:

– незначні вимоги до пам'яті, як у пошуку в глибину;

– повнота, як і в пошуку в ширину, якщо коефіцієнт розгалуження кінцевий;

– оптимальність, якщо вартість шляху становить неспадну функцію глибини вузла.

Ця стратегія може здатися занадто витратною, оскільки одні й ті стани формуються декілька разів. Але, як виявилося, такі повторні операції не є надто дорогими.

Причина полягає в тому, що в дереві пошуку з одним і тим самим (або майже одним і тим самим) коефіцієнтом розгалуження на кожному рівні більшість вузлів перебуває на нижньому рівні, тому не має великого значення те, що вузли на верхніх рівнях формуються багаторазово. У пошуку з ітеративним поглибленням вузли на нижньому рівні (з глибиною d) формуються один раз. Вузли, які перебувають на рівні, що передує нижньому, формуються двічі і т. д., до дочірніх вузлів кореневого вузла, які формуються d разів.

2.4. Методи евристичного пошуку

Методи повного перебирання гарантують розв'язання задачі, якщо воно існує, а за наявності декількох розв'язків гарантує оптимальний. Однак на практиці ці методи використовуються для розв'язування лише невеликих за розмірами графів станів.

Для реальних випадків найчастіше використовується додаткова інформація, що ґрунтується на попередньому досвіді або отримана на підставі теоретичних висновків. Така інформація називається ***евристичною***, а організована в правила — *евристичними методами пошуку* або *евристиками*.

Евристична інформація має суто спеціальний характер і може застосовуватися лише в рамках цього завдання, в кращому випадку в рамках завдань цього класу, вона робить перебирання впорядкованим.

У алгоритмах евристичного пошуку список відкритих вершин впорядковується за зростанням деякої *оціночної функції*, що формується на основі евристичних правил. Оціночна функція може включати дві складові, одна з яких називається евристичною і характеризує близькість поточної та цільової вершин. Що менше значення евристичної складової оціночної функції, то ближче розглянута вершина до цільової вершини.

Евристичні методи найчастіше застосовують не для пошуку єдиного правильного (оптимального розв'язку), а для пошуку першого розв'язку, що задовольняє деякий критерій за певних обмежень. Як приклад розглянемо відому задачу про комівояжера. Торговець повинен побувати в кожному з N міст по одному разу і повернутися в початкове місто. Бажано, щоб маршрут був мінімальним за протяжністю (рис. 2.5).

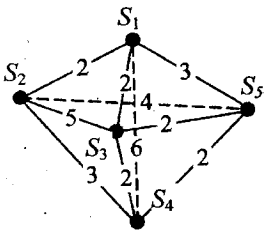


Рис. 2.5. Граф станів для задачі про комівояжера

Фрагмент простору станів зображено на рис. 2.6. Починаючи перебирання в ширину, на першому ж рівні отримуємо можливі шляхи різної довжини: 2, 2, 6, 3. Якщо виходити з евристики «на кожному кроці вибирати шлях мінімальної довжини», слід зробити кроки S_1 — S_2 та S_2 — S_3, потім S_3 — S_4 або S_3 — S_5, потім S_4 — S_5 і т. д. Добудувавши дерево до кінця, можна переконатися, що шлях, знайдений таким методом, не завжди буде найкоротшим. Більш правильною була б евристика «вибирати так, щоб мінімальним був сумарний шлях» (принцип Р. Белмана).

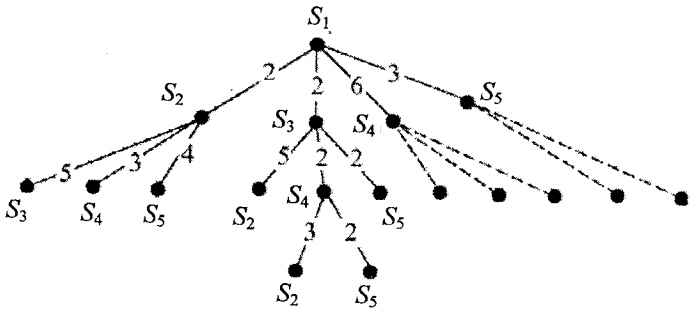


Рис. 2.6. Фрагмент дерева пошуку для задачі про комівояжера

Граф повного перебирання для попереднього прикладу, що включає всі можливі шляхи комівояжера, міститиме $(N — 1)!$ варіантів. Якщо відкинути зворотні шляхи, то найбільше $(N — 1)!/2$.

На реальних завданнях подібного типу, які часто вирішуються при створенні електронних карт (розмісити передавачі пейджингового зв'язку так, щоб відстань від будь-якої точки заданої зони не перевищувала X, або розмістити банкомати так, щоб мінімізувати шлях інкасатора (не обов'язково за протяжністю) тощо), N зазвичай дорівнює декільком десяткам об'єктів, що для сучасних ЕОМ може створити проблему. Евристичні алгоритми найчастіше застосовуються не для пошуку єдино правильного (оптимального рішення), а для пошуку першого рішення, що задовольняє деякому критерію за заданих обмежень.

Так, наприклад, людина, відправляючись до магазину, не ставить собі завдання купити найдешевше молоко, вона готова купити молоко не нижче заданої якості і не дорожче деякої (можливо, нечіткої) оцінки, і упевнена в тому, що не стане витрачати на покупку більше 5 хв (обходити декілька магазинів).

На сьогодні розроблено десятки корисних евристик, які не завжди є очевидними і доказовими, але дозволяють значно скоротити перебирання.

2.5. Методи пошуку розв'язків інтелектуальних задач у разі зведення задачі до сукупності підзадач

Пошук розв'язків при зведенні задач до підзадач полягає в послідовному розбитті вихідної задачі на простіші доти, доки не будуть отримані лише елементарні задачі. Частково впорядкована сукупність таких задач становитиме розв'язок вихідної задачі.

Метод пошуку в такому разі спирається на *граф редукції* завдання, який є ***графом «І-АБО»***. Кожній вершині цього графа ставиться у відповідність опис деякої задачі (підзадачі).

Побудова графа редукції задачі аналогічна побудові графа пошуку розв'язку у просторі станів. Мета пошуку — показати, що початкова вершина розв'язана. Процедуру пошуку можна інтерпретувати як побудову дерева розв'язків. ***Дерево розв'язків*** — це піддерево (підграф) графа редукції задачі з коренем у початковій вершині, що складається з розв'язних вершин.

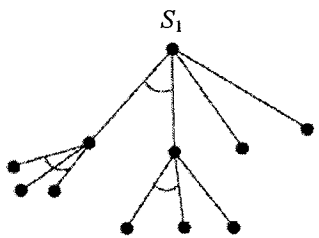


Рис. 2.7. Дерево пошуку в глибину при розбитті задачі на підзадачі

Між отриманими при розбитті підзадачами можуть бути відносини *узгодженості* (одночасності) їх розв'язків — відношення «І», позначені одинарною або подвійною дугою, що зв'язує ребра графа, або відношення *альтернативності* — відношення «АБО» (рис. 2.7). Таким чином, вихідна задача представлена підзадачами, що мають альтернативний «АБО»-характер, а самі підзадачі, своєю чергою, — підзадачами з відношеннями типу «і». *Пошук на графі редукції* відрізняється від пошуку на графі станів тим, що він включає процедури перевірок розв'язності й нерозв'язності вершин S_i замість процедури перевірок відповідності стану цільовому.

Вершина є *розв'язною*, якщо виконується одна з таких умов:

1) вершина S_i є заключною (термінальною);

2) наступні за S_i вершини є вершинами типу «АБО» і при цьому хоча б одна з них розв'язна;

3) наступні за S_i вершини є вершинами типу «І» і при цьому кожна з них розв'язна.

Розв'язним графом називається підграф, що складається з розв'язних вершин із коренем у початковій вершині. ***Редукція*** (розкриття вершин) закінчується, якщо встановлюється розв'язність або нерозв'язність початкової вершини.

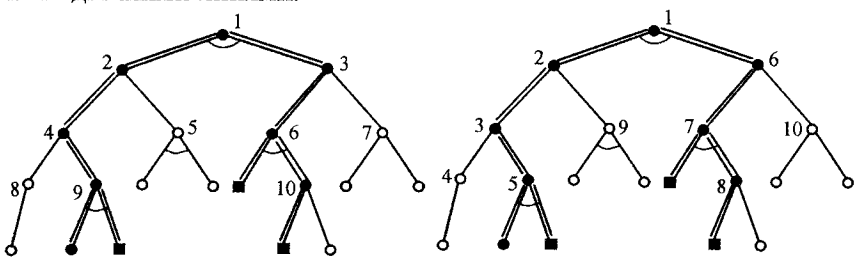
Методи пошуку для графа «І-АБО», так само як для пошуку у просторі станів, можна розділити на такі групи:

– пошук у глибину і в ширину;

– пошук у прямому і в зворотному напрямку;

– методи «сліпого» і впорядкованого (евристичного) пошуку.

Методи «сліпого» пошуку. На рис. 2.8 наведено приклад пошуку в ширину і глибину. Вершини пронумеровано в тому порядку, у якому вони розкривалися. Кінцеві вершини позначено квадратами, розв'язні вершини затемнено, дуги розв'язного графа виділено подвійними лініями.



а *б*

Рис. 2.8. Приклади пошуку при розбитті задачі на підзадачі:
а — пошук у ширину; *б* — пошук у глибину

Евристичні методи пошуку. Для упорядкування вершин, що розкриваються на дереві типу «І-АБО», використовується так зване *оптимальне потенційне дерево* розв'язків, для виділення якого застосовується евристична функція. При пошуку розв'язку у просторі станів евристична функція визначається як оцінка вартості оптимального шляху від заданої вершини до цільової. У деревах типу «І-АБО» для визначення евристичної функції використовується *оцінка вартості дерева розв'язку*. Розрізняють *сумарну* вартість дерева розв'язку, що становить собою суму вартості всіх дуг у дереві розв'язку, і *максимальну* вартість, яка дорівнює вартості шляху між двома вершинами дерева розв'язку, що має максимальну вартість.

Вартість шляху визначається як сума вартостей дуг, що входять до цього шляху. Якщо вартість дуг дорівнює одиниці, то сумарна вартість — кількості дуг у дереві розв'язку, а максимальна вартість — кількості дуг на шляху між двома найвіддаленішими вершинами дерева розв'язку. Дерево розв'язку, яке має мінімальну вартість (сумарну або максимальну, залежно від того, яка з них узята за критерій оптимальності), називається ***оптимальним***.

Висновки

1. Розрізняють локальний та системний підходи до подання інтелектуальних задач.

2. Пошук у просторі станів — це пошук у просторі, де поняттям знання про завдання відповідають стани завдання, а правилам виведення — оператори переходу з одного стану в інший від завдання до підзадач.

3. Пошук розв'язків інтелектуальних задач у просторі станів проводять за допомогою дерева (графа) станів.

4. Пошук зводиться до визначення послідовності операторів, які відображають початкові стани в цільові.

5. Порядок, у якому розгортаються стани, визначається стратегією пошуку. Розрізняють стратегії неінформованого та інформованого (евристичного) пошуку.

6. Методи неінформованого пошуку означають, що в даних стратегіях не використовується додаткова інформація про стани, окрім наведеної у завданні. До таких методів відносять стратегію пошуку в ширину і в глибину.

7. Методи евристичного пошуку використовують додаткову інформацію, що ґрунтується на попередньому досвіді або отримана на підставі теоретичних висновків.

8. Існує прямий, зворотний та двонапрямлений пошук.

9. Пошук розв'язку при зведенні задач до підзадач полягає в послідовному розбитті вихідної задачі на простіші доти, поки не будуть отримані лише елементарні задачі. Частково впорядкована сукупність таких задач становитиме розв'язок вихідної задачі.

10. Метод пошуку в такому разі спирається на граф редукції задачі, який є графом «І-АБО». Кожній вершині цього графа відповідає опис деякої задачі (підзадачі).

11. Побудова графа редукції задачі аналогічна побудові графа пошуку розв'язку у просторі станів. Мета пошуку — показати, що початкова вершина розв'язна.

Вправи

1. Складіть дерево пошуку шляху від вершини А до вершини З (рис. 2.9) за стратегією в глибину.

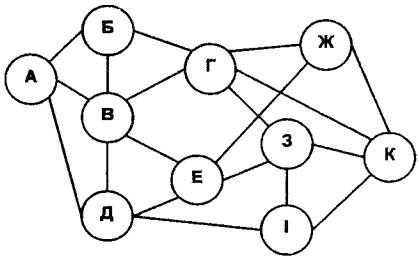


Рис. 2.9. Нерозмічений граф

2. Побудуйте дерево пошуку шляху від вершини А до вершини Е (рис. 2.9) за стратегією в ширину.

3. Побудуйте дерево пошуку шляху від вершини А до вершини К (рис. 2.9) за стратегією двонапрямленого пошуку.

4. Намалюйте дерево пошуку за стратегією в глибину для задачі про комівояжера (див. рис. 2.5).

5. Знайдіть мінімальний шлях від вершини А до вершини К (рис. 2.9).

Запитання та завдання для самоперевірки

1. Чим відрізняються системний та локальний підходи до подання інтелектуальних задач?
2. Як реалізується пошук у просторі станів?
3. Опишіть стратегію пошуку в глибину та в ширину.
4. Які існують модифікації методу пошуку в глибину?
5. У чому полягає відмінність евристичного пошуку від неінформованого?
6. Проаналізуйте прямий, зворотний та двонапрямлений напрями пошуку.
7. Як проводиться пошук розв'язку задачі на графі редукції?
8. Опишіть стратегії «сліпого» та евристичного пошуку для дерева типу «І-АБО».

РОЗДІЛ 3

ПОДАННЯ ЗНАНЬ У СИСТЕМАХ ШТУЧНОГО ІНТЕЛЕКТУ

3.1. Знання та моделі подання знань у системах штучного інтелекту

Усім інтелектуальним системам необхідні знання про світ (зокрема, про предметну галузь) для того, щоб вони могли виробляти адекватні рішення. Слід зазначити, що до появи СШІ використовувалися здебільшого поняття «дані» і «база даних». Розглянемо їх відмінність від понять «знання» та «база знань».

Дані (*Data*) — це те, що може реєструватися в тій або іншій формі органами чуття людини або приладами. Тиск, вологість, яскравість, радіаційний фон тощо. ***База даних*** — це певним чином організована сукупність даних, що дозволяє забезпечити ефективний пошук, розміщення і модифікацію даних.

Знання (*Knowledge*) — це зафіксована закономірність щодо фактів, процесів, явищ і причинно-наслідкових відносин між ними. ***База знань*** (*Knowledge Base*) — організована певним чином сукупність знань, що дозволяє забезпечити ефективне логічне виведення розв'язку поставленої задачі, розміщення, модифікації і поповнення знань.

У системах штучного інтелекту знання є основним об'єктом формування, обробки і дослідження. Будь-яка СШІ містить початкову інформацію, тобто знання, які зберігаються в базі мовою подання, і певну систему логічного виведення. Ефективна робота забезпечується внаслідок збереження одержаної інформації в базі, використання механізму логічного виведення для отримання нових знань і використання їх для прийняття рішення.

Механізм виведення нових знань побудовано за типом людського мислення, що дотримується законів логіки. Основна відмінність у логічних теоріях полягає у виборі аксіом і визначенні правил виведення.

Виявлення джерел знань і робота з ними — основне завдання інженера знань. *Інженер знань* повинен добре орієнтуватися у предметній галузі, спілкуватися з експертом у цій галузі та вміти структурувати знання для зберігання і роботи з ними. Розрізняють такі види знань:

– *глибинні знання* — це категорії, абстракції та аналогії, за допомогою яких експерт приходить до розуміння структури та призначення поточних уявлень. Ці знання використовуються переважно за нестандартних ситуацій;

– *поверхневі знання* (експертні знання) — це навички, відповідні знання на рівні рефлекторних реакцій, відпрацьованих дій. Сюди ж можна віднести правила та асоціації для стандартних міркувань і ситуацій.

Розрізняють *формальні* (логічні) методи, в основі яких лежить чітка математична теорія, та *неформальні* — моделі такої теорії не дотримуються. На рис. 3.1 зображено найвідоміші моделі подання знань (МПЗ).

Рис. 3.1. Класифікація моделей знань

В основі ***логічного методу*** подання знань у СШІ лежить формальна система — логіка предикатів І порядку, яка мовою теорії множин описується такою четвіркою:

$$M = <T, P, A, B>,$$

де T — множина базових елементів; P — множина синтаксичних правил, за допомогою яких із елементів множини T утворюють синтаксично правильні сукупності; A — елементи цієї множини

утворюють аксіоми, визначені на множині P; $P(A)$ — процедура, що визначає належність до A; B — множина правил виведення, які застосовують до елементів множини A.

Вибір МПЗ є важливим питанням при створенні СШІ. При використанні *логіки* предикатів І порядку база знань (БЗ) може розглядатися як сукупність логічних формул, які забезпечують частковий опис предметного середовища. Перевага — можливість безпосередньо запрограмувати механізм виведення. Недоліки: відсутність наочності та зручності читання, громіздкість запису, що призводить до складності знаходження помилок.

Великі труднощі виникають при створенні моделей неповних, нечітких знань. Моделі на основі *нечіткої логіки* Л. Заде дозволяють оперувати розмитими поняттями, проте такі результати інтерпретувати важче і навіть не завжди можливо.

Семантичні сітки дозволяють описувати властивості й відношення об'єктів, подій, понять, ситуацій або дій за допомогою направленого графа, що складається з вершин і помічених ребер. Переваги: наочність, зрозумілість, зручність для програмування. Недолік — втрата наочності при розширенні моделі.

Фрейми, як і семантичні сітки, є декларативно-процедурними структурами. Можливе наслідування атрибутів більш абстрактних об'єктів. Перевага — економне розміщення БЗ у пам'яті комп'ютера. Недолік — через складність зменшується швидкість роботи механізму виведення.

Продукційні моделі використовують найчастіше. У БЗ містяться правила продукцій, а в базі даних (БД) — інформація, яка відображає поточний стан розв'язуваної задачі.

Ініціалізацію необхідного правила здійснює інтерпретатор або блок керування. Перевага — природність виведення знань. Недолік — процес виведення менш ефективний, ніж в інших системах, та важко піддається керуванню.

3.2. Логіка числення висловлювань

У цьому підрозділі розглядається аксіоматика і правила виведення різних логічних теорій. Семантика *логіки числення висловлювань* дуже близька до природної мови, тому результати формального виведення легко інтерпретувати.

Висновки, одержані на основі числення предикатів I порядку, вже не повною мірою збігаються з семантикою природної мови, їх складніше інтерпретувати.

Проте моделі на основі числення предикатів виходять набагато компактнішими і, як наслідок, осяжними.

3.2.1. Основні поняття числення висловлювань

Числення висловлювань (його також називають пропозиціональним численням — від лат. *propositio* — пропозиція, думка, вислів) — найпростіший розділ математичної логіки, що лежить в основі решти її розділів.

Основними об'єктами розгляду є ***висловлювання*** — це речення (вислів), про яке можна сказати одне з двох: істинне воно або хибне. Позначаються: I (істина — *true*) і X (хибне — *false*); або числа 1 і 0.

Приклади висловлювань: «Вода мокра» — істинне, «У Африці мороз» — хибне, «Кисень є газом» — істинне, «Двічі по два — одинадцять» — хибне.

Висловлювання типу «кисень є газом» або «двічі по два — одинадцять» становлять собою приклади так званих елементарних (неподільних) постійних висловлювань — ***атомів*** (грец. *atomus* — неподільний).

Зауважимо, що істинність або хибність висловлювань залежить від предметної галузі. Наприклад, кисень за певних умов може бути не лише газом, але й рідиною і навіть твердим тілом.

Для позначення елементарних *постійних* висловлювань, значення яких «Істина» або «Хибність» не змінюється, використовують великі літери латинського алфавіту. Для позначення так званих *змінних* висловлювань, замість яких може бути підставлено будь-який конкретний постійний вислів, використовують малі літери.

3.2.2. Логічні операції

Із елементарних висловлювань будуються складніші висловлювання за допомогою логічних зв'язок і правил. Таким чином отримують *правильно побудовані формули*. У табл. 3.1 наведено п'ять основних операцій для зв'язки логічних висловлювань.

Заперечення або доповнення. Висловлювання $\neg A$ істинне, якщо висловлювання A хибне; $\neg A$ хибне, якщо A — істинне.

Кон'юнкція (лат. *conjunctio* — сполучник, зв'язок) або логічне множення. Відповідає сполучнику «І». Позначення — A і B, $A \wedge B$, AB. Це висловлювання істинне тоді і лише тоді, коли істинні як A, так і B, A і B називаються *кон'юнктивними членами*.

Таблиця 3.1

Основні операції логіки числення висловлювань

Назва операції	Зв'язка	Позначення	Відповідна математична операція
Заперечення	НІ	$\neg$	
Кон'юнкція	І	$\&$ $\wedge$ $\cdot$	множення
Диз'юнкція	АБО	$\vee$ $+$	складання
Імплікація	ЯКЩО — ТО	$\rightarrow$ $\supset$	слідування
Тотожність (еквівалентність)	ТОДІ-І-ТІЛЬКИ-ТОДІ	$\leftrightarrow$ $\equiv$	

Диз'юнкція (лат. *disjunctio* — роз'єднання, розділення, розрізнення) або логічне додавання. Відповідає сполучнику «АБО». Позначення — $A \vee B$. Це висловлювання істинне, якщо істинне хоча б одне з висловлювань A і B. A і B називаються диз'юнктивними членами.

Імплікація (лат. *implicito* — тісно пов'язую) або слідування. Відповідає підрядному сполучнику «якщо ... то ...». Позначення — $A \rightarrow B$, $A \supset B$. Це висловлювання хибне тоді і лише тоді, якщо A — істинне і B — хибне.

Тотожність (еквівалентність). Позначення $A \equiv B$, $A \leftrightarrow B$. Читається: «A істинне тоді і лише тоді, коли істинне B».

Це висловлювання істинне тоді і лише тоді, коли A і B мають одне й те саме значення.

3.2.3. Таблиця істинності

Із кожною логічною зв'язкою можна співставити функцію двох змінних $f(x, y)$ — *функцію істинності*. Область визначення аргументів функції і область значення функції складається з двох значень: «істина» і «хибність». Функцію істинності легко зобразити в табличному вигляді.

Таку таблицю називають *таблицею істинності*.

Будь-яке складне висловлювання, створене з деяких початкових висловлювань за допомогою логічних операцій, називається *пропозиційною формулою* (формою). Кожна формула задає функцію істинності, яку можна зобразити таблицею істинності. У табл. 3.2 показано таблиці істинності для п'яти основних зв'язок у численні висловлювання.

Таблиця 3.2

Таблиця істинності

A	B	$\neg A$	$A \,\&\, B$	$A \vee B$	$A \supset B$	$A \equiv B$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Кількість рядків у такій таблиці визначається кількістю різних висловлювань, що входять у формулу. Якщо кількість аргументів дорівнює N, то кількість рядків у таблиці дорівнює кількості різних комбінацій аргументів 2^N, тобто при великих значеннях N це число може бути достатньо великим.

Наприклад, із простих висловлювань A, B, C можна побудувати складні висловлювання:

1) $\neg (A \,\&\, B) \vee C$;

2) $A \rightarrow [B \leftrightarrow (A \vee C)]$.

Логічне значення складеного висловлювання залежить від структури висловлювання, вираженої формулою, і логічних значень утворюючих його елементарних висловлювань. Якщо припустити, що $A = 0$, $B = 1$ і $C = 0$, тоді перше й друге висловлювання будуть істинними.

1) $\neg (A \,\&\, B) \vee C = \neg (0\& \; 1) \vee 0 = \neg 0 \vee 0 = 1$;

2) $A \rightarrow [B \leftrightarrow (A \vee C)] = 0 \rightarrow [1 \leftrightarrow (0 \vee 0)] =$

$= 0 \rightarrow [1 \leftrightarrow 0] = 0 \rightarrow 0 = 1$.

3.2.4. Основні закони числення висловлювань

Щоб визначити, чи є дві формули еквівалентними, можна використовувати два підходи:

– порівняти *таблиці істинності.* Якщо вони рівні, то формули еквівалентні. Цей громіздкий шлях можна використовувати при

кількості аргументів не більше п'яти. Для функції з N аргументів наборів буде 2^N;

– використати *алгебричний підхід*, тобто перетворити одні формули в інші за допомогою законів. Таку алгебру називають *бульовою* на честь розробника — ірландського математика і логіка Джорджа Буля (1815–1864 рр.).

Далі наведено основні закони числення висловлювань.

Комутативний закон (закон переміщення):

$$a \,\&\, b = b \,\&\, a; \qquad a \vee b = b \vee a.$$

Асоціативний закон (сполучний):

$$a \,\&\, (b \,\&\, c) = (a \,\&\, b) \,\&\, c; \quad a \vee (b \vee c) = (a \vee b) \vee c.$$

Дистрибутивний закон (розділення):

$$a \,\&\, (b \vee c) = a \,\&\, b \vee a \,\&\, c; \quad a \vee (b \,\&\, c) = (a \vee b) \,\&\, (a \vee c).$$

Закони де Моргана

$$\neg(a \vee b) = \neg a \,\&\, \neg b; \qquad \neg(a \,\&\, b) = \neg\, a \vee \neg b; \qquad \neg(\neg a) = a.$$

Принцип подвійності: будь-яка теорема булевої алгебри залишається істинною, якщо поміняти місцями операції кон'юнкції і диз'юнкції, константи *true* і *false* у всій теоремі. Наприклад:

$$a \vee \neg\, a = true \qquad a \,\&\, \neg\, a = false$$

$$a \vee true = true \qquad a \,\&\, \text{false} = false$$

$$\neg\,(a \vee b) = \neg a \,\&\, \neg b \qquad \neg\,(a \,\&\, b) = \neg a \vee \neg b$$

Числення висловлювань містить 10 схем **аксіом**.

У схему аксіоми замість змінних A, B, С можна підставляти будь-які конкретні формули.

1. $A \supset (B \supset A)$.
2. $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$.
3. $(A \,\&\, B) \supset A$.
4. $(A \,\&\, B) \supset B$.
5. $A \supset (B \supset (A \,\&\, B))$.
6. $A \supset (B \vee A)$.
7. $A \supset (A \vee B)$.
8. $(A \supset B) \supset ((C \supset B) \supset ((A \vee C) \supset B))$.
9. $(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$.
10. $A \supset A$.

Правило виведення одне — виключення імплікації (*modus ponens — mp*)

$$\frac{A, A \supset B}{B}.$$

Додаткові правила виведення:

виключення винятків (modus tollens)

$$\frac{A \supset B, \neg B}{\neg A}$$

транзитивність імплікації

$$\frac{A \supset B, B \supset C}{A \supset C}$$

Доведенням формули Φ_n називається така послідовність формул $\Phi_0, \Phi_1, \ldots, \Phi_n$, що кожна Φ_i, $i \le n$ або є аксіомою, або отримана з деяких формул $\Phi_j, \Phi_k, j, k < i$ за правилом виведення.

3.2.5. Зведення виразів до нормальних форм

Для розв'язування логічних задач важливо вміти знаходити для кожної формули числення висловлювань еквівалентну найпростішу формулу. Існує дві канонічні форми:

– кон'юнктивна нормальна форма (КНФ) $D_1 \& D_2 \& \ldots \& D_m$;

– диз'юнктивна нормальна форма (ДНФ) $K_1 \vee K_2 \vee \ldots \vee K_m$.

K_i і D_i — кон'юнктивний та диз'юнктивний терми, що становлять кон'юнкцію (диз'юнкцію) змінних і їх заперечень. Змінні — елементарні висловлювання, які називаються ***атомами***. Вони містять лише три операції — $\vee$, &, $\neg$, причому заперечення стоїть безпосередньо перед атомом.

Для зведення формули до КНФ і ДНФ використовують такі тотожності:

1. $A \supset B = \neg A \vee B$.
2. $\neg\neg A = A$.
3. $\neg(A\&B) = \neg A \vee \neg B$.
4. $\neg(A \vee B) = \neg A \& \neg B$.
5. $A \vee A = A$.
6. $A \& A = A$.
7. $A \equiv B = (A \supset B) \& (B \supset A)$.

Враховуючи комутативність, ідемпотентність і правила, що містять константи, можна домогтися, щоб в одному і тому самому кон'юнктивному або диз'юнктивному термі не було однако-

вих змінних. Диз'юнктивні (кон'юнктивні) форми з цими додатковими умовами називаються ***нормальними формами***.

Щоб встановити еквівалентність двох формул, необхідно привести обидві формули до однієї і тієї ж нормальної форми. Якщо нормальні форми еквівалентні, то еквівалентні й вихідні формули.

Зауважимо, що, як випливає з формул, які містять константи, наявність хоча б одного диз'юнкта зі значенням *false* означає хибність всієї формули.

Відповідно, кон'юнкт зі значенням *false* забезпечує істинність нормальної кон'юнктивної форми.

Приклад. Розглянемо розв'язання логічної задачі за допомогою числення висловлювань із застосуванням прямого методу виведення.

Умова. Маємо такі істинні висловлювання.

1. Якщо студент X цікавиться логікою, то він або запишеться на курси з логіки, або він ледачий.

2. Якщо студент X самостійно вивчає літературу з логіки, то він цікавиться логікою.

3. Студент X не ледачий.

4. Студент X самостійно вивчає літературу з логіки.

Запитання. Чи запишеться студент X на курси з логіки?

Формалізація задачі. Виділимо висловлювання і позначимо їх:

P — студент X цікавиться логікою;

Q — студент X запишеться на курси;

S — студент X ледачий;

D — студент X самостійно вивчає літературу з логіки.

Запишемо логічні формули, що відповідають вказаним висловлюванням:

1. $P \supset Q \vee S$.

2. $D \supset P$.

3. $\neg S$.

4. D.

Необхідно довести істинність Q.

Отже, маємо умову задачі, записану в символьній формі:

$D,\ \neg S, D \supset P, P \supset Q \vee S \blacktriangleright Q$.

Доведення. Записуємо послідовно всі дані за умовою задач посилки і використовуємо одне правило виведення — виключення імплікації (*modus ponens — mp*).

1. $P \supset Q \vee S$.
2. $D \supset P$.
3. $\neg S$.
4. D.
5. Застосовуючи правило *mp* до п. 4 і п. 2, маємо: $\dfrac{D, D \supset P}{P}$.
6. Застосовуючи правило *mp* до п. 5 і п. 1, отримуємо: $\dfrac{P, P \supseteq Q \vee S}{Q \vee S}$.
7. Проведемо тотожну заміну: $Q \vee S = S \vee Q = \neg S \supset Q$.
8. Застосовуючи правило *mp* до п. 3 і п. 7, дістаємо: $\dfrac{\neg S, \neg S \supset Q}{Q}$.

Таким чином, доведено істинність Q.

Відповідь. Студент X запишеться на курси з логіки.

3.2.6. Непрямі методи виведення

Методи виведення поділяються на прямі й непрямі.

Прямий метод виведення дозволяє за допомогою правил виведення безпосередньо перейти від початкових посилок і аксіом до цілі.

Непрямі методи виведення дозволяють замінити один ланцюжок виведення іншим, у якому початкові посилки і виведення відрізняються від початкових. При цьому з істинності другого виведення випливає істинність першого. Але одержати друге виведення значно легше. Розрізняють чотири основні стратегії:

- введення припущення;
- метод міркування шляхом розбору випадків;
- доведення від протилежного;
- метод резолюції.

Метод виведення «введенням припущення» ґрунтується на теоремі про дедукцію: якщо є виведення формули B із множини посилок A і Q, то можна побудувати виведення формули $A \supset B$ із множини посилок Q. Записується так:

$$\frac{A, Q \mapsto B}{Q \mapsto A \supset B} \quad \text{або} \quad \frac{Q \mapsto A \supset B}{A, Q \mapsto B}.$$

Для доведення імплікації $A \supset B$ допускається, що A істинна, тобто A приймається як додаткова посилка, і робляться спроби довести істинність B.

Приклад. Покажемо, що формулу $\Phi \supset (\Psi \supset (\Phi \,\&\, \Psi))$ можна довести. За визначенням доведення формули зводиться до доведення можливості її виведення з порожньої множини припущень, тобто

$$\blacktriangleright\ \Phi \supset (\Psi \supset (\Phi \,\&\, \Psi)).$$

За теоремою про дедукцію достатньо показати, що виводиться

$$\Phi, \Psi \blacktriangleright\ \Phi \,\&\, \Psi,$$

оскільки

1. $\Phi, \Psi \blacktriangleright\ \Phi \,\&\, \Psi$;
2. $\Phi \blacktriangleright\ \Psi \supset (\Phi \,\&\, \Psi)$;
3. $\blacktriangleright\ \Phi \supset (\Psi \supset (\Phi \,\&\, \Psi))$.

Замість того, щоб доводити формулу 3, доведемо формулу 1.

1. Посилка Φ
2. Посилка Ψ
3. Аксіома 5 $A \supset (B \supset (A \,\&\, B))$ або $\Phi \supset (\Psi \supset (\Phi \,\&\, \Psi))$
4. Правило *mp* до 1 і 3: $(\Psi \supset (\Phi \,\&\, \Psi)$
5. Правило *mp* до 2 і 4: $\Phi \,\&\, \Psi$

Отже, виведення має такий вигляд:

$$\Phi, \Psi, \Phi \supset (\Psi \supset (\Phi \,\&\, \Psi)), (\Psi \supset (\Phi \,\&\, \Psi)), \Phi \,\&\Psi.$$

Метод виведення шляхом «розбору випадків» грунтується на правилі виведення: щоб побудувати виведення з посилок $A \vee B$, Q, достатньо одержати два виведення: C — з A, Q, і C — з B, Q.

$$\frac{A,Q \mapsto C;\quad B,Q \mapsto C}{A \vee B, Q \mapsto C}.$$

Доведення від протилежного. З визначення виведення випливає, якщо $\{Q_1, Q_2, \ldots, Q_n\} \mid\!- A$, то справедливо, що $(Q_1 \& Q_2 \&, \ldots, \& Q_n) \mid\!- A$, або $\mid\!- (Q_1 \& Q_2 \&, \ldots, \& Q_n) \supset A$.

За принципом дедукції, якщо формула A є логічним наслідком кінцевої множини Q, тоді і лише тоді $Q \cup \{\neg A\}$ нездійсненно:

$$\{Q_1, Q_2, \ldots, Q_n\} \mid\!- \mathrm{A} \quad \epsilon \quad \{Q_1, Q_2, \ldots, Q_n, \neg \mathrm{A}\} \mid\!- 0.$$

Тобто, якщо потрібно одержати виведення A з Q, то можна заперечення формули A додати до Q і намагатися одержати з Q, $\neg A$ суперечність. Якщо вийшло, то можна побудувати виведення A з Q.

Приклад. Розглянемо доведення $A \vee B, A \supset \neg C, B \supset D \blacktriangleright \neg C \vee D$.

Записуємо складові

1. $A \vee B = \neg A \supset B$.
2. $A \supset \neg C = C \supset \neg A$.
3. $B \supset D$.
4. Додаємо заперечення до правої частини: $\neg (\neg C \vee D)$.
5. Використовуємо правило транзитивності імплікації до 1 і 3:

$$\frac{\neg A \supset B, B \supset D}{\neg A \supset D}.$$

6. Використовуємо правило транзитивності імплікації до 2 і 5:

$$\frac{C \supset \neg A, \neg A \supset D}{C \supset D}.$$

7. Оскільки $C \supset D = \neg C \vee$ D, тоді 4 і 6 $\neg C \vee D$, $\neg(\neg C \vee D) \blacktriangleright 0$,

що й потрібно було довести.

Метод резолюцій ґрунтується на доведенні від протилежного і містить лише одне правило виведення — правило резолюції.

Метод резолюції працює на формулах, поданих у кон'юнктивній нормальній формі D_1 & D_2, ..., & D_n.

D_i — ***диз'юнкт*** формули, який містить лише елементарні висловлювання — *літерали* — у стверджувальній L *і* заперечній $\neg L$ формі, поєднані знаком диз'юнкції: $L_1 \vee L_2 \vee, \ldots, \vee L_k$.

Припустимо, що є два диз'юнкти D_1, D_2, такі, що в один із них входить деякий літерал L, а в іншій входить його заперечення.

Тоді з цих двох висловлювань можна вивести третє висловлювання C_3. Висловлювання C_3 називається ***резольвентою*** висловлювань D_1 і D_2, а D_1, D_2 — ***батьківськими вислювлюваннями*** C_3. Правило резолюції

$$\frac{D_1, D_2}{C_3} \text{ або } \frac{F \vee A, G \vee \neg A}{F \vee G}.$$

Одержана резольвента включається до загального списку диз'юнктів і сама може стати батьківським вислювлюванням.

Правило резолюції застосовується доти, поки не зустрінуться два одиночні диз'юнкти з протилежними літералами A та $\neg A$. Їх резолюція дає порожній диз'юнкт □, що означає суперечність.

$$\frac{A, \neg A}{\square}.$$

Алгоритм застосування правила резолюції.

1. Зводимо всі посилки і заперечення висновку, прийнятого як додаткова посилка, до нормальної кон'юнктивної форми:

– усуваємо символи еквівалентності й імплікації за допомогою формул еквівалентності:

$$A \equiv B = (A \supset B) \,\&\, (B \supset A);$$

$$A \supset B = \neg A \vee B;$$

– просуваємо всередину за допомогою законів де Моргана:

$$\neg (A \vee B) = \neg A \,\&\, \neg B;$$

$$\neg (A \& B) = \neg A \vee \neg B;$$

– застосовуємо дистрибутивний закон для виразів типу

$$A \vee (B \,\&\, C) = (A \vee B) \,\&\, (A \vee C).$$

2. Записуємо всі диз'юнкти з нового рядка, оскільки кон'юнкція істинна лише тоді, коли істинні всі диз'юнкти.

3. Усі диз'юнкти містять або стверджувальні літерали, або їх заперечення. Шукаємо пару диз'юнктів, що містять один і той самий літерал, але з протилежним знаком. Застосовуємо для них правило резолюції. Одержаний новий диз'юнкт додаємо до наявної сукупності диз'юнктів.

4. Продовжуємо цей процес доти, доки не отримаємо порожній диз'юнкт □, що означає суперечність.

Приклад.

Умова. Дано істинні висловлювання.

1. Або Петро й Іван брати, або вони однокурсники.
2. Якщо Петро й Іван брати, то Сергій і Іван — не брати.
3. Якщо Петро й Іван однокурсники, то Іван і Михайло теж однокурсники.

З цього випливає, що або Сергій і Іван — не брати, або Іван і Михайло — однокурсники.

Формалізація задачі. Виділимо висловлювання і позначимо їх.

1. A — Петро й Іван брати.
2. B — Петро й Іван однокурсники.

3. C — Сергій і Іван брати.

4. D — Іван і Михайло однокурсники.

Запишемо логічні формули, що відповідають складним висловлюванням:

1. $A \vee B$.

2. $A \supset \neg C$.

3. $B \supset D$.

Довести: $\neg C \vee D$.

Записуємо диз'юнкти, приводимо їх до нормальної кон'юнктивної форми і додаємо заперечення висновку:

1. $A \vee B$.
2. $A \supset \neg C = \neg A \vee \neg C$.
3. $B \supset D = \neg B \vee D$.
4. $\neg(\neg C \vee D)$.

Виводимо резольвенти:

5. П. 1 і п. 2 $\dfrac{A \vee B, \neg A \vee \neg C}{B \vee \neg C}$;
6. П. 5 і п. 3 $\dfrac{B \vee \neg C, \neg B \vee D}{\neg C \vee D}$;
7. П. 4 і п. 6 $\dfrac{\neg(\neg C \vee D), \neg C \vee D}{\square}$.

Що й треба було довести за умовою задачі.

3.3. Логіка числення предикатів

3.3.1. Поняття предиката

Предикат (лат. *praedicatum* — сказане) — те, що висловлюється (стверджується або заперечується) в судженні про об'єкт. Предикат відображає наявність або відсутність тієї чи іншої ознаки у предмета.

Функція P, яка набуває одного зі значень, 0 або 1, аргументи якої набувають значення із довільної множини M, називається *предикатом P в проблемній області M.* Кількість аргументів предиката $P(x_1, x_2, ..., x_k)$ називається його *порядком*. Множина M, на якій визначено предикат, називається *проблемною областю*. Підмножина $Q \supset M$, для якої $P(x)$ істинно, називається *екстенсіоналом*.

Розрізняють предикати: унарні (наприклад, чоловік (X)), бінарні (наприклад, батько (X, Y)), тернарні (наприклад, громадянин (f, dr, mr)), n-арні предикати. Предикат n-го порядку $P(x_1, x_2, ..., x_k)$ визначає n-арне відношення R у M: якщо $P(c_1, c_2, ..., c_k) = 1$, то $(c_1, c_2, ..., c_k)$ перебуває у відношенні R, що визначається цим предикатом. Якщо значення предиката в цій точці дорівнює 0, то ці елементи не перебувають у відношенні R.

3.3.2. Квантори спільності й існування

Квантори (лат. *quantum* — скільки) — логічні оператори, які несуть інформацію про кількісну характеристику логічного висловлювання, перед якими вони поставлені.

Квантор спільності. Знак $\forall$ називається ***квантором спільності*** та ставиться при спільних судженнях. Символом квантора спільності узято перевернуту букву A (перша літера німецького слова *alle* — всі). Вислів «для будь-якого елемента x належного M властивість R здійснена» домовились позначати як

$$(\forall x \in M)\,(R(x) = 1).$$

Знак квантора ставиться перед висловлюванням. Праворуч від знака ставиться літера, яка називається *кванторною змінною* та є неодмінною складовою написання квантора. Квантор спільності можна розглядати як узагальнення кон'юнкції. Якщо проблемна область скінченна і складається з елементів $c_1, c_2, ..., c_k$, то формула $(\forall x)(F(x))$ рівносильна кон'юнкції

$$F(c_1)\&F(c_2)\&, ... ,\&F(c_k).$$

Для нескінченних предметних областей квантор спільності відіграє роль нескінченної диз'юнкції.

Наприклад, розглянемо такі висловлювання природною мовою та їх запис за допомогою кванторів.

«Кожна людина смертна» записується як

$$(\forall x)(\textit{людина}(x) \supset \textit{смертна}(x)).$$

«Будь-яке число x , яке більше за 1, більше за квадратний корінь із нього» записується як

$$(\forall x)(\textit{більше}(x, 1) \supset \textit{більше}(x, \textit{sqrt}(x)).$$

Квантор існування. Знак $\exists$ називається ***квантором існування*** і застосовується при поодиноких судженнях. Символом кванто-

ра існування узято перевернуту літеру E (перша літера німецького слова *Existieren* — існувати). Висловлювання «існує принаймні один елемент $x \in M$, якому притаманна властивість R» позначають як

$$(\exists x \in M)(R(x)=1).$$

Квантор існування ставиться при поодиноких судженнях, в яких стверджується чи заперечується про частини предметів якого-небудь класу предметів, наприклад, «деякі метали плавають на воді».

Якщо необхідно підкреслити, що існує єдиний x, такий що $R(x)$, тоді запис набирає такого вигляду: $\exists !xR(x)$.

Квантор існування $(\exists x)$ можна розглядати як узагальнення диз'юнкції, при цьому записи

$$(\exists x)\,(F(x)) \quad \text{і} \quad F(c_1)\vee F(c_2)\vee \ldots \vee F(c_k)$$

рівносильні. Для нескінченних предметних галузей квантор існування відіграє роль нескінченної диз'юнкції.

Наприклад, висловлювання «У кожної людини є батько» інтерпретується як «існує такий x , що…». Відповідний запис

$$\forall x\ \exists y\ (\textit{людина}(x) \supset \textit{батько}(y, x)).$$

3.3.3. Операції з кванторами

Закон заперечення кванторів. Закон *заперечення квантора спільності*: «Неправильно, що кожен предмет має якість x тоді і тільки тоді, коли існують предмети, що не мають цієї якості». Символічно цей закон записують так:

$$\neg\ \forall x\, A(x) = \exists x\, \neg A(x).$$

Закон *заперечення квантора існування*. Перед квантором ставиться знак логічного заперечення і записується це так: $\neg\exists\ x$, що означає «Не існує такого x, що ….».

$$\neg\ \exists x\, A(x) = \forall x\, \neg A(x).$$

Приклад.

До висловлювання «Всі червоні яблука солодкі», яке записується мовою числення предикатів як $(\forall x)(\textit{червоне\_яблуко}(x) \supset \textit{солодке\_яблуко}(x))$, застосувати закон заперечення.

Розв'язання:

1. Застосовуємо закон заперечення кванторів:

$$\neg\,(\forall x)(\textit{червоне\_яблуко}(x) \supset \textit{солодке\_яблуко}(x)) =$$
$$= \exists x(\neg((\textit{червоне\_яблуко}(x) \supset \textit{солодке\_яблуко}(x)).$$

2. Замінюємо імплікацію на диз'юнкцію:

$\exists x(\neg((червоне\_яблуко(x) \supset солодке\_яблуко(x))=$

$= \exists x(\neg(\neg(червоне\_яблуко(x) \vee солодке\_яблуко(x)).$

3. Внесемо заперечення всередину дужок:

$\exists x(\neg(\neg(червоне\_яблуко(x) \vee солодке\_яблуко(x))=$

$= \exists x((червоне\_яблуко(x) \& \neg солодке\_яблуко(x)).$

Відповідь. Хибно, що «Всі червоні яблука солодкі», якщо існує «Червоне яблуко, і воно не солодке».

Комутативний закон — це закон, за яким можна квантори, що стоять перед висловлюваннями, міняти місцями:

$\forall x \forall y P(x, y) = \forall y \forall x P(x, y)$; або $\exists x \exists y P(x, y) = \exists y \exists x P(x, y)$.

Але квантори різних типів некомутативні.

Наприклад, висловлювання «у кожної людини є батько»

$$\forall x \exists y (людина(x) \supset батько(y, x))$$

при перестановці кванторів місцями

$$\exists y \forall x (людина(x) \supset батько(y, x))$$

означатиме «у всіх людей спільний батько», тобто не буде тотожне початковому виразу.

Дистрибутивний закон — це закон, за яким квантори, що стоять перед складними висловлюваннями, можна відносити до компонентів цих складних висловлювань (за членами складного висловлювання).

$$\forall x (A(x) \supset B(x)) = (\forall x A(x) \supset (\forall x B(x));$$

$$\exists x (A(x) \supset B(x)) = (\exists x A(x) \supset (\exists x B(x)),$$

або

$$\forall x (A(x) \& B(x)) = (\forall x A(x) \& (\forall x B(x));$$

$$\exists x (A(x) \vee B(x)) = (\exists x A(x) \vee (\exists x B(x)).$$

Щоб задати формальну логічну теорію, необхідно визначити $M = (T, P, A, F)$, де T — множина термінальних символів, P — множина правильно побудованих формул, A — множина аксіом, F — множина правил виведення.

Термом можуть бути:

1) *предметні константи* $c_1, c_2, \ldots, c_n$ — конкретні значення імен елементів предметної галузі. Будь-яка предметна константа є термом;

2) *предметні змінні* $x_1, x_2, \ldots, x_n$ — змінні, що набувають значення з предметної галузі. Будь-яка предметна змінна є термом;

3) *функції* f_1, f_2 у предметній галузі — це відображення одних об'єктів предметної галузі через інші. Наприклад, функція *plus* (2, 3)→5. Результатом функції є об'єкт (ім'я об'єкта). Тому можна вважати, що *plus* (2, 3) це ім'я «5», висловлене через операцію додавання. Якщо f — функціональна літера і $t_1, t_2, ..., t_k$ — терми, то $f(t_1, t_2, ..., t_k)$ — терм.

Таким чином, ***терм*** — це ім'я елемента предметної галузі, вказане безпосередньо константою або змінною, а також побудоване за допомогою функції.

У численні предикатів ***формулою*** вважається:

– будь-яка елементарна формула;

– кожне з висловлювань $\neg A$, $A \vee B$, $A\&B$, $A \supset B$, $A \equiv B$, $\forall x\,(A(x))$, $\exists x\,(A(x))$, якщо A і B — формули, а x — предметна змінна.

Виведенням формули Ф із множини формул Г називається така послідовність $\Phi_0, ..., \Phi_n$ формул, що $\Phi_n = \Phi_i$ для кожної $i \leq n$:

– Φ_i аксіома,

– $\Phi_i \in \Gamma$,

– отримується з $\Phi_j, j < i$ за одним із правил, причому x не повинна входити до формули вільно.

Доведення формули — це виведення з порожньої множини $\Gamma = \varnothing$.

3.3.4. Зв'язані та вільні предметні змінні

Значення предиката (істина або хибність) визначається після підставлення замість змінних конкретних значень. Наприклад,

$$+(складова\_1, складова\_2, сума),$$

$+(2,3,5)$ — істинне, $+(2,2,5)$ —хибне.

Вільно обираючи значення змінних, можна отримувати різні значення предиката. Тому такі змінні називають *вільними*.

Суттєвою властивістю кванторів є те, що вони виключають залежність значення предиката від змінних, перетворюють вільні змінні на зв'язані змінні у тих висловлюваннях, перед якими стоять квантори. Предметна змінна, що входить до формули, називається *вільною*, якщо вона не йде безпосередньо за квантором і не входить до області дії квантора за цією змінною. Усі інші формули, що входять до формули, називаються *зв'язаними*.

Областю дії квантора називається та частина формули, на яку поширюється дія будь-якого квантора.

Так, у формулі:

$$\forall x\,(\forall x(P(x)\vee Q(x))\vee P(x)\to Q(x))$$

областю дії квантора $\forall x$ є вся частина формули, розташована праворуч від цього квантора областю дії другого квантора $\forall x$ є лише те, що міститься у перших дужках праворуч від цього квантора.

Будь-яка формула з вільними змінними задає деяке відношення в предметній галузі, яку іноді називають *областю інтерпретації.*

Підформула, що не містить зв'язану змінну, називається *константою* відносно квантора $\forall\ x\ (\exists\ x)$.

Винесення константи:

$$\forall x(A(y)\ \&\ B(x) = A(y)\ \&\ \forall x\ B(x);$$

$$\forall x(A(y)\vee B(x) = A(y)\vee \forall x\ B(x);$$

$$\exists x(A(y)\ \&\ B(x) = A(y)\ \&\ \exists x\ B(x);$$

$$\exists x(A(y)\vee B(x) = A(y)\vee \exists x\ B(x).$$

Правило перейменування (еквівалентності змінних):

$$\forall x\ \Phi = \forall y\ [\Phi]_y^x\ ; \qquad \exists x\ \Phi = \exists y\ [\Phi]_y^x.$$

Запис $[\Phi]_y^x$ позначає результат підстановки терма y замість усіх вільних входжень у Ф змінної x.

При підстановці необхідно дотримуватися правила — всі вільні змінні після підстановки мають лишитися вільними.

3.3.5. Метод резолюцій у численні предикатів

Виведення в численні предикатів переважно збігається з виведенням у численні висловлювань. Відмінності пов'язані з наявністю кванторів. Алгоритм застосування методу резолюції в численні предикатів.

1. Перенести цільове твердження в посилки із запереченням

$$\Phi_1, \Phi_2, \ldots, \Phi_n\ , \neg Q \vdash .$$

2. У всіх формулах виключити еквівалентності, використовуючи рівність

$$A \equiv B = (A \supset B)\ \&\ (B \supset A).$$

3. У всіх формулах виключити імплікації, використовуючи рівність

$$A \supset B = \neg A \vee B.$$

4. Перемістити всередину формул знак заперечення, використовуючи формули де Моргана. У складних виразах краще починати із зовнішніх знаків заперечення, оскільки внутрішні можуть скоротитися:

$$\neg (A \vee B) = \neg A \,\&\, \neg B; \qquad \neg (A \,\&\, B) = \neg A \vee \neg B;$$
$$\neg \forall x\, A(x) = \exists x \neg A(x); \quad \neg \exists x\, A(x) = \forall x \neg A(x).$$

5. Виключити квантори існування $\exists x\, P(x, y)$.

6. Розкрити кон'юнкції, що потрапили всередину диз'юнкцій, використовуючи дистрибутивний закон

$$P \vee (Q \,\&\, R \,\&\, \ldots) = (P \vee Q) \,\&\, (P \vee R) \ldots .$$

7. Винести всі квантори спільності з перейменуванням змінних (якщо в цьому є потреба). Тут доречно провести аналогію з локальними змінними. Локальну змінну можна замінити на будь-яку іншу, аби лише не виникало колізій (суміщення) з іншими змінними в тій самій області дії, наприклад,

$$\forall x\, (rod(x, y), m(x) \supset father(x, y) \Rightarrow$$
$$\Rightarrow \forall x_1\, (rod(x_1, y_1), m(x_1) \supset father(x_1, y_1);$$
$$\forall x\, (rod(x, y), f(x) \supset mother(x, y) \Rightarrow$$
$$\Rightarrow \forall x_2\, (rod(x_2, y_2), f(x_2) \supset mother(x_2, y_2).$$

8. Записати всі диз'юнкти.

9. Застосувати правило резолюції до отримання порожнього диз'юнкта. Для збігу літералів застосувати підстановку й уніфікацію.

Розглянемо більш детально процедури виключення кванторів існування (п. 5) та підстановки й уніфікації (п. 9), оскільки вони безпосередньо позв'язані з наявністю кванторів і змінних.

Виключення кванторів існування $\exists x$. Метод виключення запропонував логік Сколем. На його честь процес виключення квантора назвали «сколемізацією». Правила «сколемізації»:

- якщо змінна x позв'язана квантором існування $\exists x$, який сам не перебуває в області дії ніякого квантора спільності, то $\exists x$ стирається, а всі входження x, які були позв'язані цим квантором, замінюються на нову «сколемівську константу». Її ім'я має бути відмінне від всіх інших.

Наприклад,

$$\exists x\, Q(x) \Rightarrow Q(a) \qquad \exists x\, P(x, y) \Rightarrow P(a, y).$$

Константа a становить собою деяке (невідоме) значення, для якого твердження істинне. Важливо те, що підставляємо одне й те ж невідоме значення на всі місця, де використовується x;

• якщо $\exists x$ перебуває в області дії кванторів спільності $\forall y \exists x P(x, y)$, то a функціонально залежить від y – $a = f(y)$. У цьому разі $\exists x$ стирається, а всі входження x, які були позв'язані цим квантором, замінюються на «сколемівську функцію» $f(y)$, також невідому, але всюди одну й ту саму. Наприклад,

$$\forall y \exists x P(x, y) \Rightarrow P(f(y), y).$$

Для скінченної множини області визначення y можна записати

$$\forall y \exists x P(x, y) = \exists x P(x, y_1) \,\&\, \exists x\, P(x, y) \ldots \,\&\, \exists x\, P(x, y_n) =$$
$$= P(a_1, y_1) \,\&\, P(a_2, y_2) \ldots \,\&\, P(a_n, y_n).$$

Константа a функціонально залежить від $y - a = f(y)$.

Приклад. Проведемо виключення кванторів існування для такого виразу

$$\exists z\, P(z) \,\&\, (\forall y)\,((\exists x)\, Eq(y, plus(x, 1)) \,\&\, \exists x\, less(x, plus(x, y))).$$

Розв'язання.

1. $\exists z$ прибираємо і замінюємо z на a.
2. $\exists x$ замінюємо x на функцію $f(y)$.
3. $\exists x$ замінюємо x на функцію $g(y)$.

Таким чином, отримуємо новий вираз

$$P(a) \,\&\, (Eq(y, plus(f(y), 1)) \,\&\, less(g(y), plus(g(y), y))).$$

Підстановка та уніфікація. Для застосування методу резолюції необхідно визначити два диз'юнкти, що містять літерали з протилежними знаками: $F(t_1, t_2, \ldots, t_n)$ і $\neg F(t_1, t_2, \ldots, t_n)$. Потрібно, щоб збігалися не лише імена предикатів, але і їх аргументи. Нагадаємо, що аргументи предиката — це терми: константи, змінні та функції (складні імена). Константи змінити неможливо, а замість змінних можна підставити будь-які терми, використовуючи правило:

$$\forall\ x\, P(x) \supset P(a).$$

Тобто якщо предикат істинний для всіх x, то він істинний і для конкретного значення змінної.

При цьому треба робити таку підстановку, щоб літерали повністю збігалися. Процес визначення найбільш спільної підстановки, що робить два літерали однаковими, називають *уніфікацією*.

Наприклад, якщо дано вирази

$$F(x, y) \vee M(x) \quad \text{та} \quad \neg M(jon),$$

то підстановка константи *jon* замість змінної x

$$F(jon, y) \quad \text{та} \quad M(jon)$$

робить літерали однаковими і до них можна застосувати правило резолюції:

$$\{F(jon, y) \vee M(jon)\ ;\ \neg M(jon)\}$$

$F(jon, y)$ — резольвента.

3.3.6. Хорнівські диз'юнкти

Основна проблема в методі резолюцій — визначити послідовність диз'юнктів, що беруть участь у резолюції. Існує клас висловлювань, для яких ця задача спрощується. Їх детально дослідив логік Альфред Хорн. Хорнівський диз'юнкт містить не більше одного додатного літерала.

Формула $P_1 \,\&\, P_2 \,\&\, \ldots \,\&\, P_n \supset B$ перетворюється на диз'юнкт вигляду

$$\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_n \vee B,$$

у якому лише один додатний літерал.

Вираз $P_1 \,\&\, P_2 \,\&\, , \ \ldots \ , \& P_n \supset B$ називають *продукцією*. Багато знань існують у вигляді продукцій. Наприклад, «Якщо птахи літають низько і вітер ущух, то скоро піде дощ». Існують ще два види формул, які можна перетворити на хорнівські.

1. $P_1 \supset B_1 \& B_2$ розбивається на два диз'юнкти: $P_1 \supset B_1$ і $P_1 \supset B_2$.

Доведення.

$$P_1 \supset B_1 \,\&\, B_2 = \neg P_1 \vee (B_1 \,\&\, B_2) = (\neg P_1 \vee B_1) \,\&\, (\neg P_1 \vee B_2) =$$

$$= (P_1 \supset B_1) \,\&\, (P_1 \supset B_2).$$

Узагальнення $P_1 \supset B_1 \,\&\, B_2 \,\&\ldots\, \&\, B_n$.

2. $(Q \vee S) \supset B$ розбивається на два диз'юнкти: $Q \supset B$ і $S \supset B$.

Доведення.

$$(Q \vee S) \supset B = \neg (Q \vee S) \vee B = (\neg Q \,\&\, \neg S) \vee B =$$

$$= (\neg Q \vee B) \,\&\, (\neg S \vee B) = (Q \supset B) \,\&\, (S \supset B).$$

Узагальнення $(Q \vee S \vee, \ldots, \vee R) \supset B$.

Крім того, цільове твердження формулюється зазвичай за допомогою квантора існування $\exists x(R(x)\&Q(x)...)$. Заперечення квантора існування також перетворюється на хорнівський диз'юнкт:

$$\neg(\exists\, x\,(R(x)\ \&\ Q(x)...)) = \forall\, x\,(\neg R(x) \vee \neg Q(x)).$$

Заборонений варіант формули: $P_1 \supset Q \vee S$.

Приклад. Розглянемо виведення розв'язку в логічній моделі на основі методу резолюцій.

Маємо такі істинні твердження:

«Сократ — людина»;

«Людина — це жива істота»;

«Всі живі істоти смертні».

Необхідно довести методом резолюцій твердження «Сократ смертний».

Розв'язання.

Крок 1. Запишемо умову задачі в предикатній формі.

$\forall(X)(Людина\ (X) \rightarrow Жива\_істота\ (X))$.

$Людина\ (Сократ)$.

$\forall(Y)(Жива\_істота\ (Y) \rightarrow Смертна\ (Y))$.

Крок 2. Перетворимо висловлювання на диз'юнктивну форму.

$\neg(Людина\ (X) \vee Жива\_істота\ (X))$.

$Людина$ (Сократ).

$\neg(Жива\_істота\ (Y) \vee Смертна\ (Y))$.

Крок 3. Запишемо заперечення цільового твердження (висновку, що необхідно довести), тобто «Сократ безсмертний».

$Смертна\ (Сократ)$.

Крок 4. Складемо кон'юнкцію всіх диз'юнктів (тобто побудуємо КНФ), включивши в неї заперечення цільового твердження.

$\neg(Людина\ (X) \vee Жива\_істота\ (X)) \wedge$

$\neg(\ Жива\_істота\ (Y) \vee Смертна\ (Y)) \wedge$

$Людина\ (Сократ) \wedge$

$\neg Смертна\ (Сократ)$

Крок 5. У циклі проведемо операцію пошуку резольвент над кожною парою диз'юнктів:

1-й диз'юнкт	2-й диз'юнкт
$\neg(Людина(X) \vee Жива\_істота\ (X))$	$\neg(Жива\_істота\ (Y) \vee Смертна\ (Y))$

Резольвента: $\neg$*Людина* (*Y*) $\vee$ *Смертна* (*Y*)	
$\neg$*Людина* (*Y*)$\vee$ *Смертна* (*Y*)	*Людина* (*Сократ*)
Резольвента: *Смертна* (*Сократ*)	
Смертна (*Сократ*)	$\neg$*Смертна* (*Сократ*)
Резольвента: □	

Отримання порожнього диз'юнкта означає, що висловлювання «Сократ безсмертний» хибне, отже, вислів «Сократ смертний» є істинним.

Загалом метод резолюцій цікавий завдяки простоті та системності, але може застосовуватися лише до обмеженої кількості випадків (доведення не повинно мати велику глибину, а кількість потенційних резолюцій не має бути великою). Крім методу резолюцій та правил виведення існують інші методи отримання висновків у логіці предикатів.

3.4. Основні поняття нечіткої логіки

Найбільш вражаючою властивістю людського інтелекту є здатність приймати правильні рішення в умовах неповної і нечіткої інформації. Побудова моделей наближених роздумів людини і використання їх у комп'ютерних системах становить сьогодні одну з найважливіших проблем науки.

Основи нечіткої логіки були закладені наприкінці 60-х рр. XX ст. у працях відомого американського математика Л. Заде. Необхідність такого дослідження була викликана зростаючим невдоволенням ЕС.

«Штучний інтелект», що досі легко міг впоратися із задачами керування складними технічними комплексами, був безпорадним при найпростіших висловлюваннях повсякденного життя на кшалт «Якщо машиною перед тобою керує недосвідчений водій — тримайся від неї подалі».

Для створення дійсно інтелектуальних систем, здатних адекватно взаємодіяти з людиною, необхідний був новий математичний апарат, що перекладає невиразні й неоднозначні життєві твердження мовою чітких і формальних математичних формул.

Першим серйозним кроком у цьому напрямку стала теорія нечітких множин, розроблена Л. Заде.

3.4.1. Нечіткі множини

Нечітка множина задається парами вигляду

$$x_1 \left| \mu_X (x_1) \right.,$$

де x_1 — елемент нечіткої множини X, а $\mu_X (x_1)$ — ступінь належності елемента x_1 до нечіткої множини X. Значення $\mu_X (x_1)$ змінюється в інтервалі [0…1].

Наприклад, деяка нечітка множина A задаватиметься виразом вигляду

$$B = \{(x_1 \mid 0{,}1), (x_2 \mid 1), (x_3 \mid 0{,}6)\},$$

де x_1 = 17 (років), x_2 = 24, x_3 = 49 або {(17|0,1), (24|1), (49|0,6)}.

Множина B (рис. 3.2) могла б, наприклад, мати сенс відповіді на запитання: «Скільки років молодій людині?». Люди з різних поколінь дадуть відповідь на нього по-різному, вкладаючи в поняття «молодий» різні обсяг і зміст.

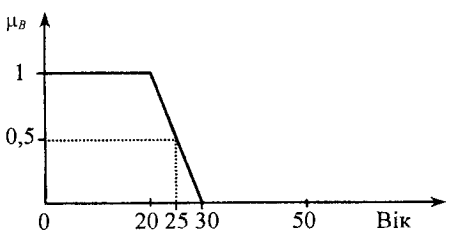


Рис. 3.2. Приклад нечіткої множини для поняття «молода людина»

Для підлітків «молода людина» — це та, кому «мало років», для людей старшого віку — та, кому «небагато років» тощо.

Пенсіонери, на противагу школярам, віднесли б шістнадцятирічного юнака до категорії «підлітки», а безнадійно застарілого, на думку школяра, тридцятирічного викладача — до категорії «досить молоді люди». Думки різних поколінь зійшлися б абсолютно лише щодо «молодої людини у віці 18–25 років».

Така нечіткість завжди є в реальних інженерних задачах, ос кільки граничні поняття «істинне — 1» та «хибне — 0» існують лише в теорії.

Припустимо

$$E = \{x_1, x_2, x_3, x_4, x_5\}, \quad M = [0,1].$$

A — нечітка множина, для якої

$$\mu_A(x_1) = 0{,}3; \quad \mu_A(x_2) = 0;$$

$$\mu_A(x_3) = 1; \quad \mu_A(x_4) = 0{,}5; \quad \mu_A(x_5) = 0{,}9.$$

Тоді A можна подати у вигляді:

$$A = \{0{,}3/x_1;\ 0/x_2;\ 1/x_3;\ 0{,}5/x_4;\ 0{,}9/x_5\}$$

або

$$A = 0{,}3/x_1 + 0/x_2 + 1/x_3 + 0{,}5/x_4 + 0{,}9/x_5,$$

де знак «+» позначає операцію об'єднання, а не додавання.

У табличному вигляді це можна записати як показано у табл. 3.3.

Таблиця 3.3

Значення нечіткої множини

	x_1	x_2	x_3	x_4	x_5
A	0,3	0	1	0,5	0,9

3.4.2. Операції над нечіткими множинами

Чітких математичних методів отримання «міри належності» деякого елемента до множини немає, застосовуються так звані *методи прийняття рішень*.

Іншим важливим запитанням є спосіб оперування з нечіткими (лінгвістичними) оцінками (мірами належності).

Припустимо, є деяка ціль A, досягнення якої залежить від досягнення підцілей A_1 і A_2, причому повне досягнення підцілей можливе лише в окремому випадку. Виникають запитання: як обчислити міру досягнення цілі A_1 і як порівняти дві нечіткі множини?

Множини A і B рівні, якщо

$$\forall x \in M\ \mu_A(x) = \mu_B(x),$$

де M — множина елементів, які є і в A, і в B.

Множини A і B доповнюють одна одну, якщо

$$\forall x \in M\ \mu_A(x) = 1 - \mu_B(x).$$

Перетин A і B

$$\forall x \in M\ \mu_A(x) \cap \mu_B(x) = \min(\mu_A(x), \mu_B(x)).$$

Об'єднання A і B

$$\forall x \in M \ \mu_A(x) \cup \mu_B(x) = \max(\mu_A(x), \mu_B(x)).$$

Уведені операції доповнення, перетину та об'єднання задовольняють закони комутативності, асоціативності, ідемпотентності, дистрибутивності, дії з константами де Моргана, подвійного заперечення (доповнення) і поглинання, тобто виконуються всі основні закони теорії множин.

Геометричну інтерпретацію операцій на нечітких множинах (діаграми Ейлера) показано на рис. 3.3, де зображено множину оцінок віку для поняття «молода людина» і її доповнення.

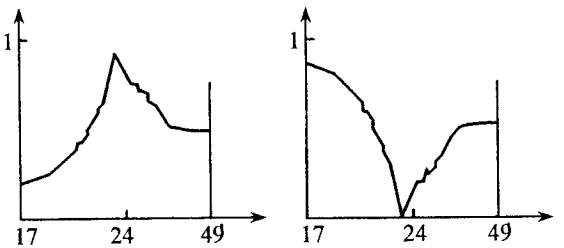


Рис. 3.3. Нечітка множина та її доповнення

На нечітких множинах визначені також операції алгебричного добутку та суми:

$$\forall x \in M \ \mu_{A+B}(x) = \mu_A(x) \cdot \mu_B(x);$$

$$\forall x \in M \ \mu_{A+B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x).$$

3.4.3. Застосування нечітких множин

Сучасні ЕС дедалі частіше використовують замість класичної логічної теорії нечітку логіку Л. Заде.

Найбільш важливим застосуванням теорії нечітких множин є контролери нечіткої логіки. Їх функціонування дещо відрізняється від роботи звичайних контролерів; для опису системи замість диференціальних рівнянь використовуються знання експертів. Ці знання можуть бути виражені за допомогою лінгвістичних змінних, описаних нечіткими множинами.

Перелічимо переваги так званих нечітких систем порівняно з іншими:

- можливість оперувати вхідними даними, заданими нечітко: наприклад, значення, що безупинно змінюються в часі (динамічні

задачі), значення, що неможливо задати однозначно (результати статистичних опитувань, рекламні кампанії тощо);

• можливість нечіткої формалізації критеріїв оцінки і порівняння: оперування критеріями «більшість», «можливе», «переважно» тощо;

• можливість проведення якісних оцінок як вхідних даних, так і виведених результатів: оперуючи не лише власне значеннями даних, але їхнім ступенем вірогідності (не плутати з імовірністю) і її розподілом;

• можливість проведення швидкого моделювання складних динамічних систем і їх порівняльний аналіз із заданим ступенем точності: оперуючи принципами поведінки системи, описаними нечіткими методами, по-перше, не потрібно витрачати багато часу на з'ясування точних значень змінних і складання рівнянь, що їх описують, по-друге, можна оцінити різні варіанти вихідних значень.

3.4.4. Логічне виведення у системах із нечіткою логікою

Припустимо, нечітка система вибирає варіанти розв'язків на основі залежності вихідної величини від декількох вхідних величин. Математичної моделі залежності виходу від входів немає і замість неї використовують базу експертних правил у вигляді нечітких висловлювань «якщо – то» у термінах лінгвістичних змінних та нечітких множин.

Тоді функціональність нечіткої системи прийняття рішень визначається такими кроками:

1) перетворення чітких вхідних змінних на нечіткі, тобто визначення ступеня відповідності входів кожній із нечітких множин;

2) обчислення правил на основі використання нечітких операторів та застосування імплікації для отримання вихідних значень правил;

3) агрегування нечітких виходів правил у загальне вихідне значення;

4) перетворення нечіткого виходу правил на чітке значення.

3.5. Продукційні моделі подання знань

Продукційна модель або модель, що ґрунтується на правилах, дозволяє подати знання у вигляді речень типу:

Якщо (умова), *то* (дія).

Під *умовою* розуміють деяке речення — зразок, за яким здійснюється пошук у базі знань, а під *дією* — дії, що виконуються в разі успішного результату пошуку.

У продукційних моделях процедурна інформація явно виокремлена та описується іншими засобами, ніж декларативна інформація. Замість логічного виведення, що характерне для логічних моделей, в продукційних моделях з'являється *виведення на знаннях.*

У загальному випадку продукційну модель можна подати у вигляді логічного виразу:

$$M = <I, Q, P, A \rightarrow B, N>,$$

де I — ім'я продукції; Q — сфера застосування продукції; P — умова застосовності ядра продукції (логічний вираз типу предиката: $P = 1$ — ядро активується, $P = 0$, то ядро не може бути використане); $A \rightarrow B$ — ядро продукції; N — постумова продукції (активується, коли ядро продукції реалізоване), це дія та умова процедури, які можуть бути виконані після P.

Усі ядра продукції можна розділити на такі групи:

1. *Детерміновані*, де права частина виконується обов'язково ($A=>B$ настане з імовірністю 1), які, свою чергою, поділяються на:

– однозначні (якщо A, то B);

– альтернативні (якщо A, то частіше B_1, рідше B_2).

2. *Недетерміновані* (якщо A виконується, то можливо B).

Можливі також різні оцінки реалізації ядра продукції:

– імовірнісна (якщо A, то з імовірністю P реалізується B);

– лінгвістична (мала, менша). Якщо A, то з більшою часткою впевненості B;

– прогнозувальна (якщо A, то з імовірністю P можна очікувати B).

Переваги продукційних МПЗ: розділення процедурних та декларативних знань; природність виведення знань; гнучкість родовидової ієрархії понять (зміни правил спричиняють зміни в ієрархії).

Недоліки продукційних систем: процес виведення може бути менш ефективний, ніж в інших системах та важко піддається керуванню; лінійне зростання обсягу бази знань у міру включення нових фрагментів знань.

Якщо використовуються дерева розв'язків, то зміни відбуваються за логарифмічним законом. Крім того, при накопиченні

досить великої кількості продукцій вони починають суперечити одна одній.

Продукційна модель найчастіше застосовується в промислових ЕС.

3.5.1. Механізм логічного виведення у продукційних системах

Механізм логічного виведення забезпечує формування висновків, сприймаючи факти, що вводяться, як елементи правил, відшукуючи правила, до складу яких входять введені факти, і актуалізуючи ті частини продукцій, яким відповідають введені факти. Механізм логічного виведення виконує функції пошуку в базі правил, послідовного виконання операцій над знаннями та отримання висновків.

Існує два способи проведення виведення:

– *прямий*, що реалізує стратегію від фактів до висновків;

– *зворотній*, коли висуваються гіпотези ймовірнісних висновків, які можуть бути підтверджені або спростовані на підставі фактів, які надходять до робочої пам'яті.

Функцією, що реалізує роботу механізму логічного виведення, є рекурсивна процедура зіставлення зі зразком.

Рекурсія (лат. *recurso* — біжу назад, повертаюся) — спосіб розв'язання задач, що полягає в розбитті вихідної задачі на підзадачі. Якщо підзадача є зменшеним варіантом вихідної задачі, то спосіб її розбиття і розв'язання ідентичний застосованому до вихідної задачі. Послідовне розбиття приводить до задачі, розв'язуваної безпосередньо. Цей розв'язок є підставою для розв'язування підзадачі верхнього рівня і т. д. доти, поки первісна задача не буде розв'язана.

Процедура логічного виведення включає:

– *робочу пам'ять* (базу даних) — фактичні дані, що описують можливий і поточний стан предметної галузі — зберігається в оперативній пам'яті;

– *базу продукційних правил*, яка містить всі допустимі залежності між фактами предметної галузі та зберігається в довготривалій пам'яті;

– *механізм логічного виведення*.

На рис. 3.4. зображено типову схему продукційної системи, яка включає п'ять основних блоків.

Спираючись на схему продукційної СШІ (рис. 3.4), можна класифікувати ядра продукції.

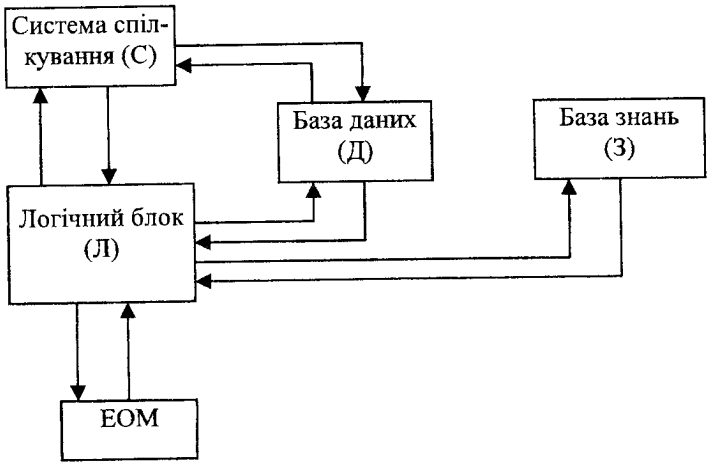


Рис. 3.4. Схема продукційної системи штучного інтелекту

Якщо x і y позначають будь-який із блоків СШІ (С, Д, З, Л), то ядро $A_x \Rightarrow B_y$ означає, що інформація про A береться з блока x, а результат продукції B посилається у блок y. Комбінації x і y, осмислені з точки зору СШІ, відмічені в табл. 3.4 знаком «+».

Таблиця 3.4

Класифікація ядер продукції

A/B	С	Д	З	Л
С		+		+
Д	+	+	+	+
З	+		+	+
Л	+		+	+

Розглянемо, як відбувається виведення у продукційних моделях для двох основних випадків:

– схема $A_З \Rightarrow B_З$, коли з первинних вихідних знань виводяться нові знання;

– схема $A_Д \Rightarrow B_З$, коли з первинних вихідних даних виводяться нові знання.

Розглянемо спочатку перший тип продукції $A_3 \Rightarrow B_3$. У цьому випадку A_3 і B_3 становлять собою деякі фрагменти інформації, що зберігається в базі знань. При мережевому поданні це можуть бути фрагменти семантичної сітки, при логічних моделях — формули того чи іншого числення.

Тоді сенс продукції $A_3 \Rightarrow B_3$ полягає в заміні одного фрагмента бази знань іншим. Для актуалізації цієї продукції необхідно, щоб в базі знань існував фрагмент, що збігається з A. При пошуку в базі знань A відіграє роль зразка, а процедура такого пошуку називається *пошуком за зразком*.

Для ілюстрації пошуку припустимо, що в базі знань для подання знань використовується семантична сітка (рис. 3.5) і продукція (рис. 3.6).

Пошук A у базі знань організується різними способами. Можна, наприклад, спочатку шукати вершину a. Якщо в базі знань такої вершини немає, то пошук закінчується невдачею.

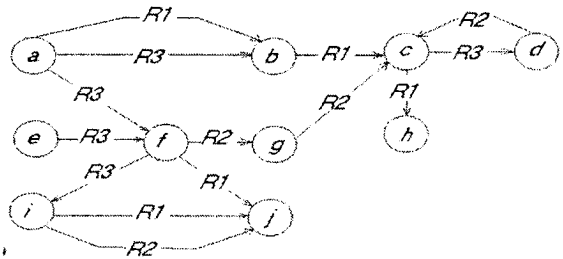


Рис. 3.5. Приклад знань у вигляді семантичної сітки

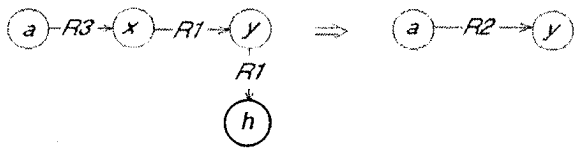


Рис. 3.6. Приклад продукції

Якщо вершину a знайдено, то шукають усі дуги, що виходять з неї, помічені відношенням $R3$, оскільки у зразку праворуч від цієї дуги стоїть вершина x, на місці якої в базі знань може бути будь-яка вершина. Якщо з a не виходить жодної дуги, поміченої відношенням $R3$, то пошук закінчується невдачею. Але якщо такі дуги є, то відбува-

ється перехід у всі вершини, з якими вершину *a* пов'язує відношення *R*3, тобто виникає паралельний процес пошуку.

У прикладі відбудеться перехід від вершини *a* до вершин *b* і *f*, з яких починається пошук дуг, які з них виходять, помічених відношенням *R*1, і які ведуть у будь-яку вершину, оскільки у зразку далі стоїть вершина, якій відповідає вільна змінна *y*. Далі процес триває аналогічно. У прикладі пошук виявляється успішним. Після знаходження *A* в семантичній сітці відбувається заміна, яка визначається правою частиною зразка. У результаті виникає трансформована семантична сітка (рис. 3.7). Другий тип продукції $A_д \Rightarrow B_3$ може відповідати процедурі знаходження закономірностей за емпіричними даними.

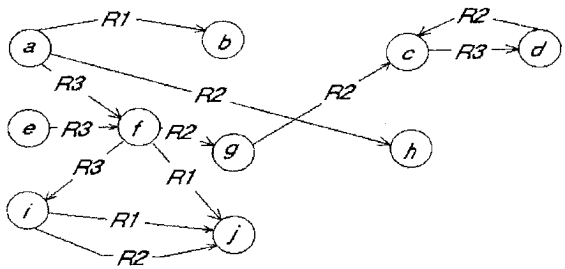


Рис. 3.7. Трансформована семантична сітка

У такому випадку логічний блок на підставі перегляду та аналізу даних висуває гіпотези про наявність закономірностей і, переконавшись у їх прийнятності і достатній обґрунтованості, записує їх до бази знань. Аналогічно можна інтерпретувати інші типи продукцій із табл. 3.4.

3.5.2. Нечітке логічне виведення у продукційних системах

Основою для проведення операції нечіткого логічного виведення є база правил, яка містить нечіткі правила у формі «Якщо — То» і функції належності відповідних лінгвістичних термів. Необхідно дотримуватися таких умов, оскільки в іншому випадку база нечітких правил буде неповна:

1. Існує хоча б одне правило для кожного лінгвістичного терма *вихідної* змінної.

2. Для будь-якого терма *вхідної* змінної є хоча б одне правило, в якому цей терм використовується як передумова (ліва частина правила). Нехай база правил складається з *m* правил вигляду:

R_1: ЯКЩО x_1 – це A_{11} ... І ... x_n це A_{1n}, ТО y це B_1.

...

R_i: ЯКЩО x_1 — це A_{i1} ... І ... x_n це A_{in}, ТО y це B_i.

...

R_m: ЯКЩО x_1 — це A_{i1} ... І ... x_n це A_{mn}, ТО y це B_m,

де x_k, $k = 1 \ldots n$ — вхідні змінні; y — вихідна змінна; A_{ik} — задані нечіткі множини з функціями належності.

Результатом нечіткого виведення є чітке значення змінної y^* на основі заданих чітких значень x_k, $k = 1 \ldots n$. У загальному випадку механізм логічного виведення включає чотири етапи: введення нечіткості (фазифікація), нечітке виведення, композиція і приведення до чіткості або дефазифікація (рис. 3.8).

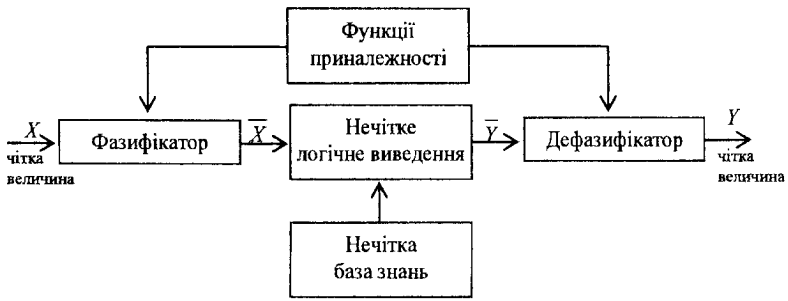


Рис. 3.8. Етапи нечіткого логічного виведення

Алгоритми нечіткого виведення розрізняються видом використовуваних правил, логічних операцій та різновидом методу дефазифікації. Існують моделі нечіткого виведення Мамдані, Сугено, Ларсена, Цукамото.

Докладніше розглянемо нечітке виведення на прикладі механізму Мамдані, оскільки він є найпоширенішим. У цьому методі використовується мінімаксна композиція нечітких множин. Механізм включає в себе таку послідовність дій:

1. Процедура *фазифікації*: визначаються ступені істинності, тобто значення функцій належності для лівих частин кожного правила (передумов). Для бази правил із m правилами позначимо ступені істинності як $A_{ik}(x_k)$, $i = 1 \ldots m$, $k = 1 \ldots n$.

2. *Нечітке виведення*. Спочатку визначаються рівні «відсічення» для лівої частини кожного з правил: $\alpha_i = \min_i(A_{ik}(x_k))$. Далі обчислюються «усічені» функції належності: $B_i^* = \min_i(\alpha_i, B_i(y))$.

3. *Композиція*, або об'єднання отриманих усічених функцій. Для цього використовується максимальна композиція нечітких множин: $MF(y) = \max_i(B_i^*(y))$, де $MF(y)$ — функція належності підсумкової нечіткої множини.

4. *Дефазифікація*, або приведення до чіткості. Існує декілька методів дефазифікації. Найчастіше використовують метод середнього центру, або центроїдний метод: $MF(y) = \max_i(B_i^*(y))$.

Геометричний сенс такого значення — центр тяжіння для кривої $MF(y)$. На рис. 3.9 графічно зображено процес нечіткого виведення за Мамдані для двох вхідних змінних і двох нечітких правил R_1 і R_2.

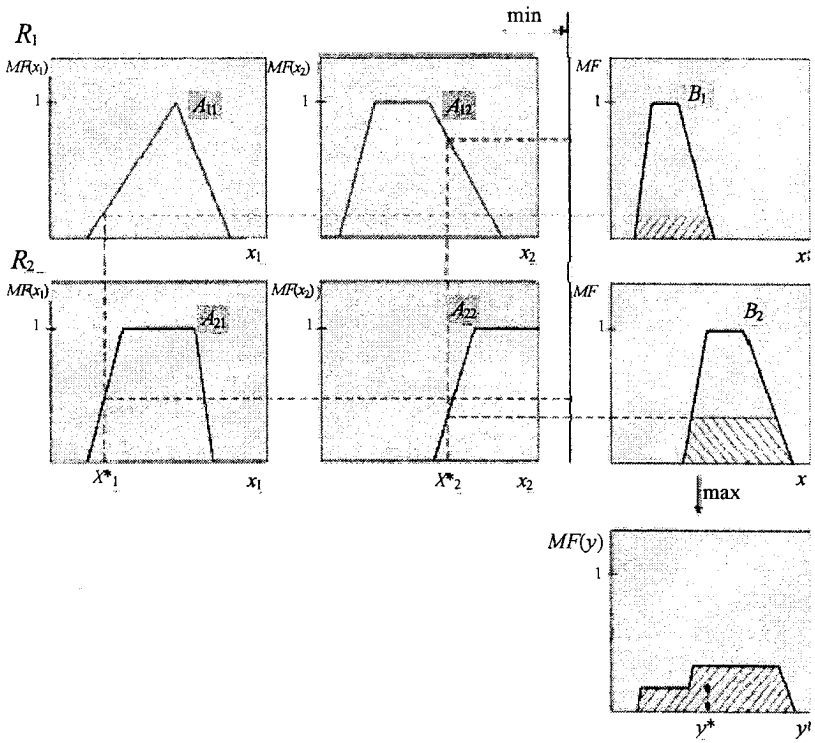


Рис. 3.9. Схема нечіткого виведення за Мамдані

3.6. Керування пошуком розв'язків у продукційних системах

У продукційних системах передбачені додаткові можливості з додавання евристичного керування до алгоритму пошуку. Вони містять:

- вибір стратегії (на основі даних або від цілі);
- вибір структури правил;
- вибір стратегій для вирішення конфліктів.

Керування за допомогою вибору стратегії пошуку. Пошук ***на основі даних*** (*пряме виведення*) починається з опису задачі (наприклад, у вигляді набору логічних аксіом), потім із наявних даних виводяться нові знання. Це здійснюється шляхом застосування правил виведення (наприклад, допустимих ходів у грі або інших операцій), що генерують нові стани в поточному описі предметного середовища, і додавання результатів до опису даної задачі. Цей процес триває доти, доки не буде досягнуто цільового стану.

Таке подання міркувань на основі даних підкреслює їх відповідність моделі виробничої системи. «Поточний стан предметного середовища» (це вихідні дані, які прийняті як істина або виведені як істина за допомогою правил виведення) заноситься до робочої пам'яті. Потім у циклі «розпізнавання-дія» поточний стан порівнюється з упорядкованим набором продукцій. Якщо такі дані відповідають умовами одного з правил виведення (уніфікуються з ними), застосування цього правила додає нову порцію інформації до поточного стану предметного середовища, змінюючи робочу пам'ять.

Усі продукційні правила мають форму УМОВА→ДІЯ. Якщо УМОВА відповідає деяким елементам робочої пам'яті, виконується ДІЯ. Якщо продукційні правила сформульовані як логічні слідування і ДІЯ додає твердження в робочу пам'ять, то активізація правила відповідає застосуванню правила *modus ponens*. При цьому на графі створюється новий стан.

У продукційних системах, які здійснюють ***пошук від цілі*** (*зворотне виведення*), пошук починається з цілі і повертається назад до фактів, які відповідають даній цілі. Щоб реалізувати це в продукційній системі, ціль заноситься до робочої пам'яті, а потім перевіряється її відповідність діям правил виведення. Відповідність дій перевіряється так само, як при пошуку на основі даних — перевіряється виконання УМОВИ (наприклад, за допомогою уніфікації).

Усі продукційні правила, висновки (ДІЇ) яких відповідають цілі, формують конфліктну множину.

Після перевірки відповідності ДІЙ їх УМОВИ додаються до робочої пам'яті, формуючи нові підцілі (або стани) пошуку. Потім перевіряється відповідність нових станів висновкам ДІЙ інших продукційних правил. Цей процес триває доти, доки не буде знайдено факт, що міститься в початковому описі завдання, або, як часто буває в ЕС, факт, безпосередньо отриманий шляхом запиту у користувача необхідної інформації. Пошук припиняється, коли умови всіх продукційних правил, використаних (активізованих) у процесі пошуку, виявляться істинними. Ці умови і ланцюжок правил, що ведуть до вихідної мети, становлять доведення її істинності.

Керування пошуком за допомогою структури правил. Структура правил у продукційній системі, включаючи відмінності між умовою і дією, а також порядок перевірки умов, визначає метод дослідження простору. Числення предикатів як мова подання в продукційних системах дозволяє представити одне істинне твердження декількома альтернативними формами. Еквівалентність цих виразів може бути продемонстрована за допомогою таблиці істинності. Хоча ці формулювання логічно еквівалентні, вони не ведуть до однакових результатів, якщо інтерпретуються як продукції (продукційні правила), оскільки реалізація виробничої системи забезпечує певний порядок перевірки відповідності та активізації правил.

Тому обирають найбільш зручну форму подання правил для конкретної задачі. Таким чином, продукційна система накладає на декларативну мову опису правил процедурну семантику.

Оскільки продукційна система перевіряє правила в певному порядку, програміст може керувати пошуком через структуру і порядок проходження правил у продукційному наборі. Кваліфіковані фахівці кодують найбільш значущі (ключові) евристики, керуючись своїми професійними знаннями. У черговості передумов міститься важлива процедурна інформація, необхідна для успішного вирішення проблеми. Дуже важливо, щоб це формулювання зберігалося при написанні програми.

Керування пошуком через вирішення конфліктів. Продукційні системи (як і всі системи, що ґрунтуються на знаннях) дозволяють представляти евристики безпосередньо в правилах, що описують знання. Крім того, вони пропонують інший метод евристич-

ного керування — через *вирішення конфліктів*. Найпростіша стратегія зводиться до того, щоб вибирати перше відповідне правило з робочої пам'яті. Однак для вирішення конфліктів потенційно може бути застосована будь-яка стратегія.

1. *Рефракція* означає, що після активізації правила воно не може бути використане знову, поки не зміняться елементи робочої пам'яті, що відповідають його умовам. Це запобігає зацикленням.

2. *Новизна*. Стратегія новизни віддає перевагу правилам, умови яких відповідають зразкам, доданим до робочої пам'яті останніми. Це дозволяє зосередити пошук на одній лінії міркування.

3. *Специфічність*. Згідно з цією стратегією, доцільніше використовувати більш конкретне, а не більш загальне правило. Одне правило більш специфічне (конкретне), ніж інше, якщо воно містить більше умов, а отже, відповідає меншій кількості зразків у робочій пам'яті.

3.7. Семантичні сітки як модель подання знань

3.7.1. Основні поняття

Основна ідея підходу до подання знань у вигляді мережевих моделей полягає у тому, щоб розглядати предметне середовище як сукупність об'єктів та зв'язків між ними. Тобто, ***семантична сітка*** — це орієнтований граф, вершини якого — *поняття*, а дуги — *відношення* між ними.

Така конструкція може бути описана мовою теорії множин:

$$H = \{I, C_1, C_2, \ldots, C_n, \Gamma\},$$

де I — множина інформаційних одиниць; $C_1, C_2, \ldots, C_n$ — множина типів зв'язків між інформаційними одиницями; Γ — множина відображення між інформаційними одиницями.

Розглянемо, наприклад, фразу: «Рибак сів на пліт, переплив на інший берег і взяв кошик із рибою». У реченні йдеться про об'єкти, зв'язані відношеннями.

Об'єкти	Відношення
a_1 — рибак,	r_1 — сів на,
a_2 — пліт,	r_2 — переплив,
a_3 — берег,	r_3 — взяв,
a_4 — кошик,	r_4 — знаходитись
a_5 — риба	

Відобразимо об'єкти вершинами графа, а відношення — дугами. Отримаємо семантичну сітку (граф), зображений на рис. 3.10.

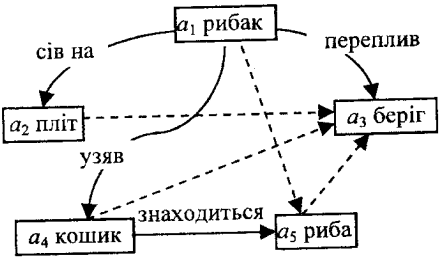


Рис. 3.10. Приклад семантичної сітки

Залежно від типів зв'язків між інформаційними одиницями, які використовуються в моделі, розрізняють такі типи мереж:

1. *Класифікувальні мережі* — застосовують відношення структуризації. Дозволяють у базах знань вводити різні ієрархічні відношення між інформаційними одиницями.

2. *Функціональні мережі* — функціональні відношення між інформаційною одиницею. Прикладом є обчислення, тому можуть виступати ще як обчислювальні. Дозволяють описувати процедури «обчислень» одних інформаційних одиниць через інші.

3. *Сценарії* — часто використовуються казуальні (причинно-наслідкові) відношення між інформаційними одиницями. Крім того, можуть зустрічатися відношення типів: «засіб — результат», «знаряддя — дія».

Якщо в мережевій моделі застосовують відношення всіх типів, то таку мережу називають ***семантичною.***

3.7.2. Типи об'єктів

У сучасних семантичних сітках використовуються три основні типи об'єктів: поняття, події та властивості.

Поняття є константами або параметрами предметної галузі, описуваної семантичною сіткою, і зазвичай вказують предмети й абстракції.

Події становлять собою дії, які можуть відбутися. Методи подання подій:

– *глибинно-характерні семантичні відносини*, які вказують характеристики та діючих осіб цієї події;

– *зміни*, які може спричинити подія. Результатом події є також деяка ситуація, яку можна визначити як зразок у деякій процедурі, що описує таким чином послідовність дій, що приводить до цієї ситуації.

Властивості використовуються для уточнення або модифікації понять, подій та інших властивостей. У разі понять властивості можуть бути особливостями, рисами або характеристиками. У разі подій властивості описують деякі загальні, універсальні, постійні характеристики, наприклад, місце, час, тривалість тощо.

Властивість є бінарним відношенням, яке відображає область власного визначення, тобто вершини, до яких властивість застосовується, в область значень, тобто значення, якого властивість може набувати.

Можливе розширення поняття властивості від бінарного до тернарного і багатомісного відношення. При цьому додаткові характеристики властивості пов'язуються з вершиною властивості дугами, поміченими «щодо» (на рис. 3.10 пунктирна лінія).

Вершини, що входять до семантичної сітки, незалежно від їх типу, поділяють на два класи:

– *загальні* — поняття, події та властивості загального характеру, що описують у сукупності закони, які діють у предметній галузі;

– *фактуальні* — окремі випадки загальних об'єктів, які описують конкретні прояви зазначених вище законів або просто деякі факти.

Іноді для підвищення ефективності виведення вводять *процедурні* вершини.

Важливим є подання в семантичних сітках загального вигляду елементів пізнавальної активності, наприклад, у вигляді логічних міркувань і прикладних програм. Ці елементи можуть бути реалізовані за наявності в семантичній сітці віртуальних відношень, які дозволяють у системі подання знань реалізувати інформаційно-логічний режим функціонування систем подання знань.

3.7.3. Логічне виведення на семантичних сітках

Виведення на семантичних сітках відрізняється повнотою та має концептуальну інтерпретацію. Послідовне застосування правил виведення може привести до утворення так званих «ланцюжків виведення», які в окремих випадках можуть досягати значної довжини.

Особливий тип генерації виведення, що використовується в семантичних сітках, — це так званий *метод «активності, що розповсюджується, і техніки перетинань»*. Цей метод відіграє важливу роль в обробці контекстів.

Процес здійснюється побудовою ланцюжків виведення на основі введених висловлювань в усіх напрямках доти, поки не виявиться перетинання в сітці.

Ці методи подібні до тих, що використовуються в системах подання знань на базі логіки предикатів: розширення семантичних сіток внаслідок введення в них знань про застосування; тематична структуризація; предметно-орієнтована ієрархія, розробка глобальних схем подання, у яких використовувалися б семантичні сітки, що містять локальні знання.

3.7.4. Сценарії

У системі подання знань можуть бути стереотипи знання, що описують відомі стандартні ситуації. Окремим видом семантичних сіток є сценарії. ***Сценарії*** — це формалізований опис стандартної послідовності взаємопов'язаних фактів, що визначають типову ситуацію предметної галузі.

Сценарій включає такі компоненти:

– *початкові умови*, які мають бути істинними при виклику сценарію;

– *результати* або факти, які є істинними, коли сценарій завершується;

– *припущення*, які підтримують контекст сценарію;

– *ролі* є діями, які виконують окремі учасники;

– *сцени*, які є часовими аспектами сценарію.

На рис. 3.11 зображено приклад сценарію у вигляді сітки, де як зв'язки між вершинами використовуються причинно-наслідкові відношення.

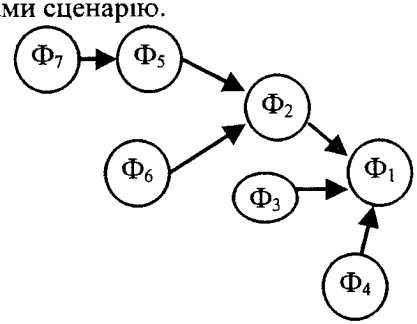


Рис. 3.11. Приклад сценарію

Вершинами є такі факти: $Ф_1$ — студент не виконує завдання на лабораторному занятті; $Ф_2$ — студент відсутній на занятті; $Ф_3$ — не працює комп'ютер; $Ф_4$ — усі

завдання виконані достроково; $Ф_5$ — перерва; $Ф_6$ — студент захворів; $Ф_7$ — студент відпочиває на вулиці або у кав'ярні.

Галузь застосування сценаріїв: процедури навчання, процедури прийняття рішень, процедури розуміння природних текстів тощо.

Застосування семантичних сіток у цілому ефективне в таких галузях:

– системи обробки природної мови: подання семантичних знань, знань про предметне середовище, епізодичних знань (тобто знань про просторово-часові події та стани);

– системи «запитання-відповідь»;

– системи штучного бачення: зберігання знань про структуру, форму і властивості фізичних об'єктів.

Переваги семантичних сіток як моделей подання знань: наочність, зрозумілість, зручність для програмування.

Недоліки: у міру зростання складності моделі втрачається її наочність, галузь застосування мережевих моделей обмежена через недостатню їх структурованість та неможливість подання процедурних знань.

3.8. Фрейми: основні поняття, структура фрейму. Фреймові системи

Фрейм (від англ. *frame* — каркас або рамка) запропонований М. Мінським у 70-ті рр. XX ст. як структура знань для сприйняття просторових сцен. Ця модель, як і семантична сітка, має глибоке психологічне обґрунтування. Фрейм — одиниця подання знань, деталі якої можуть змінюватися відповідно до поточної ситуації.

Фрейми часто використовують як структуру для подання стереотипних ситуацій.

Структура фрейму така, що він складається з характеристик описуваних ситуацій та їх значень, які називаються відповідно *слотом* та *заповнювачем слоту*.

На рис. 3.12 зображено загальну структуру фрейму. Ця структура, доки вона не заповнена якимись значеннями, називається *протофреймом*.

Значення слоту: число, математичний вираз, текст природною мовою, програма для ЕОМ, правила виведення, посилання на інші слоти даного фрейму або інших фреймів.

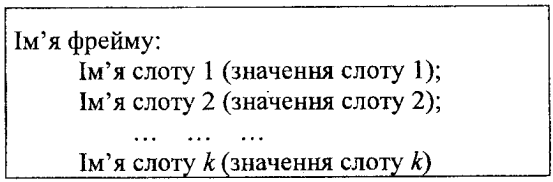


Рис. 3.12. Структура фрейму

При заповненні фрейму йому та слотам присвоюються конкретні імена та відбувається заповнення слотів. З протофрейму отримують ***фрейм-екземпляр*** (экзофрейм).

Фрейми утворюють ієрархію. Ієрархія у фреймових моделях породжує єдину багаторівневу структуру, що описує або об'єкт, якщо слоти описують лише властивості об'єкта, або ситуацію чи процес, якщо окремі слоти є іменами процедур, приєднаних до фрейму і спричинених при його актуалізації.

Формально фрейм — це тип даних вигляду:

$$F=<N, S_1, S_2, S_3>,$$

де N — ім'я об'єкта; S_1 — множина слотів, що містять факти, які визначають декларативну семантику фрейму; S_2 — множина слотів, що забезпечують зв'язки з іншими фреймами (каузальні, семантичні тощо); S_3 — множина слотів, яка забезпечує перетворення, що визначають процедурну семантику фрейму.

Отже, фрейми поділяються на:

– *фрейм-екземпляр* — конкретна реалізація фрейму, яка описує поточний стан у предметній галузі;

– *фрейм-зразок* — шаблон для опису об'єктів або допустимих ситуацій предметної галузі;

– *фрейм-клас* — фрейм верхнього рівня для подання сукупності фреймів-зразків.

Наприклад, у мережі фреймів, зображеній на рис. 3.13, поняття «Учень» наслідує властивості фреймів «Дитина» і «Людина», які перебувають на більш високому рівні ієрархії.

У мережевому і фреймовому поданні знань найчастіше використовуються відносні зв'язки типу IS-A і AKO. Зв'язок типу IS-A означає, що окремий об'єкт «є екземпляром» певного класу. Зв'язок типу AKO (A-KIND-OF) визначає відношення між родовими класами. Загальний клас, на який указує стрілка AKO, назива-

ється *суперкласом*. Якщо суперклас має зв'язок АКО, що вказує на інший вузол, то він є класом суперкласу.

На запитання: «Чи люблять учні солодке?» буде відповідь: «Так», адже ця властивість притаманна всім дітям, що вказано у фреймі «дитина».

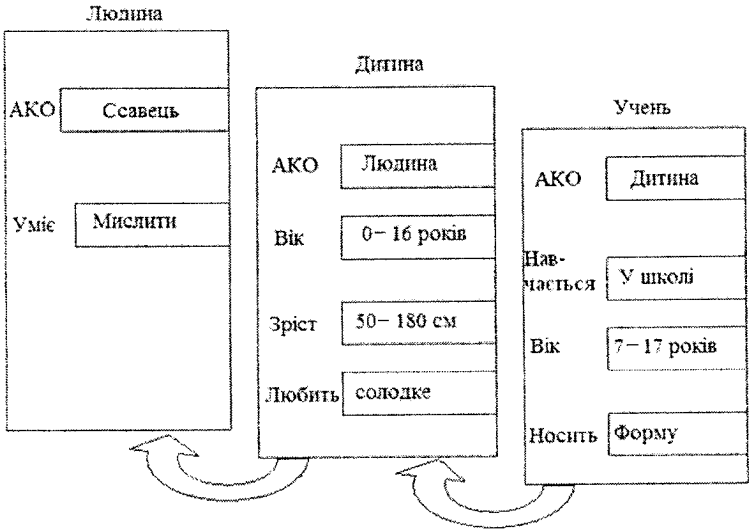


Рис. 3.13. Мережа фреймів

Наслідування властивостей може бути частковим. Так, вік для учнів не наслідується від фрейму «Дитина», оскільки зазначено явно у своєму власному фреймі.

Переваги фреймових систем:

1. Дозволяють маніпулювати як декларативними, так і процедурними знаннями, тобто значення будь-якого слоту може бути обчислене за допомогою відповідних процедур або визначене евристичним методом.

2. Економне розміщення бази знань у пам'яті комп'ютера.

Недоліки фреймових систем:

1. Відносно висока складність цих систем, що призводить до зменшення швидкості роботи механізму виведення.

2. Доцільно застосувати там, де родо-видові зв'язки змінюються нечасто та предметна галузь має небагато винятків.

Висновки

1. Знання є основним об'єктом формування, обробки й дослідження у СШІ. Будь-яка СШІ містить знання, які зберігаються в базі мовою подання, і певну систему логічного виведення.

2. Ефективна робота СШІ забезпечується внаслідок збереження одержаної інформації в базі, використання механізму логічного виведення для отримання нових знань і використання їх для прийняття рішення. Механізм виведення нових знань побудований за типом людського мислення, підпорядкованого законам логіки.

3. Розрізняють формальні (логічні) методи, в основі яких лежить чітка математична теорія, та неформальні — моделі такої теорії не дотримуються.

4. Числення висловлювань є найпростішим розділом математичної логіки, що лежить в основі решти її розділів. Твердження у численні висловлювань доводяться переважно за допомогою десяти основних аксіом та одного з основних правил виведення — *modus ponens*, *modus tollens*, правила транзитивності імплікації.

5. Методи доведення в численні висловлювань поділяються на прямі й непрямі. До непрямих відносяться введення припущення, метод міркування шляхом розбору випадків, доведення від протилежного, метод резолюції.

6. Метод резолюцій числення висловлювань грунтується на доведенні від протилежного і містить одне правило виведення — правило резолюції.

7. Предикат — це те, що висловлюється (стверджується або заперечується) в судженні про об'єкт. Він відображає наявність або відсутність тієї чи іншої ознаки у предмета.

8. У численні предикатів існують квантори — логічні оператори, які несуть інформацію про кількісну характеристику логічного висловлювання, перед якими вони поставлені. Розрізняють квантори спільності, які використовуються при спільних судженнях, та квантори загальності, які застосовуються при поодиноких судженнях.

9. Виведення за методом резолюції в численні предикатів здебільшого збігається з виведенням в численні висловлювань. Відмінності пов'язані з наявністю кванторів.

10. Моделі на основі нечіткої логіки Л. Заде дозволяють оперувати розмитими поняттями, проте такі результати інтерпретувати важче і навіть не завжди можливо.

11. Нечітка множина задається парами з двох складових: елемента нечіткої множини та ступеня належності даного елемента до нечіткої множини, який змінюється в інтервалі від 0 до 1.

12. Сучасні ЕС дедалі частіше замість класичної логіки використовують методи нечіткої логіки. Це зумовлено можливістю оперувати вхідними даними, заданими нечітко, нечіткої формалізації критеріїв оцінки і порівняння та можливістю проведення якісних оцінок як вхідних даних, так і виведених результатів.

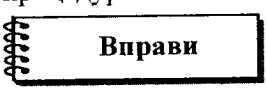
13. У продукційних моделях процедурна інформація явно виокремлена та описується іншими засобами, ніж декларативна інформація. Замість логічного виведення, характерного для логічних моделей, у продукційних моделях використовується виведення на знаннях.

14. У продукційних системах керування алгоритмом пошуку включає вибір стратегії, вибір структури правил, стратегій для вирішення конфліктів.

15. Семантична сітка — це орієнтований граф, вершини якого — поняття, а дуги — відношення між ними. Виведення за допомогою методу «активності, що розповсюджується, і техніки перетинань».

16. Окремим видом семантичних сіток є сценарії — це формалізований опис стандартної послідовності взаємопов'язаних фактів, що визначають типову ситуацію предметної галузі.

17. Фрейм — одиниця подання знань, деталі якої можуть змінюватися відповідно до поточної ситуації. Мають ієрархічну структуру. Дозволяють маніпулювати як декларативними, так і процедурними знаннями.



1. Яке значення матиме вираз $A \to [B \leftrightarrow (A \vee C)]$ — істинне чи хибне, якщо $A = 1$, $B = 1$, $C = 0$?

2. Побудуйте таблицю істинності для виразу $A \to [B \leftrightarrow (A \vee C)]$.

3. Доведіть методом резолюцій: $P \vee Q \vee R$, $P \supset R \mid - Q \& R$.

4. Розв'яжіть методом резолюцій логічну задачу. У злочині підозрюють чотири людини: A, B, C, D. Встановлено істинні висловлювання: якщо A або B винні, то і C винен; якщо A винен, то хоча б один із двох B або C також винен; якщо C винен, то і D винен; якщо A невинен, то D винен. Чи винен D?

5. Подайте у вигляді нечіткої множини поняття «гарячий чай».

6. Запишіть вислів «Усі німці розмовляють однією мовою» за допомогою числення предикатів. Використайте предикат *Speaks* (x, l), який означає, що людина x розмовляє мовою l.

7. Мовою логіки предикатів записно фразу: «Якщо дехто A та B перебувають у шлюбі, то вони люблять одне одного, і навпаки — якщо люблять, то перебувають у шлюбі». $\forall\ \forall\ x, y$ (*шлюб*(x, y) ≡ *любять*(x, y)). За допомогою закону заперечення кванторів виведіть фразу, яка заперечує це висловлювання.

8. Подайте у вигляді семантичної сітки відношення між членами родини.

9. Подайте висловлювання «Програміст сів за комп'ютер та налагодив програму» за допомогою семантичної сітки.

10. Подайте за допомогою фреймової моделі відомості щодо співробітників деякої організації. Використайте, наприклад, слоти «Прізвище», «Стаж роботи» тощо.

11. Побудуйте мережу фреймів для подання предметної галузі «Птахи».

Запитання та завдання для самоперевірки

1. У чому відмінність знань від даних?
2. Назвіть основні моделі подання знань у США.
3. Що таке висловлювання? Наведіть приклади істинних та хибних висловлювань.
4. Що таке таблиця істинності?
5. Назвіть основні закони числення висловлювань.
6. Які особливості непрямих методів виведення?
7. У чому сутність виведення методом резолюції? Де застосовується цей метод?
8. Що таке предикат? Яке його призначення?
9. За допомогою чого в численні предикатів надається інформація про кількісну характеристику логічного висловлювання?
10. У чому особливість застосування методу резолюцій у численні предикатів?
11. Поясніть сутність поняття «нечітка множина». Із яких елементів вона складається?
12. Опишіть основні характеристики продукційної моделі знань.
13. Як у продукційних системах реалізується керування процесом пошуку розв'язку?
14. Які особливості подання знань у вигляді семантичних сіток?
15. Які переваги використання фреймів для подання знань?

РОЗДІЛ 4

ЕКСПЕРТНІ СИСТЕМИ

4.1. Характеристики експертних систем

Експертна система (*expert system*) — комп'ютерна програма, здатна частково замінити фахівця-експерта у вирішенні проблемної ситуації. Вона є інструментом, що підсилює інтелектуальні здібності експерта. Мета розробки ЕС полягає в отриманні програми, яка при розв'язуванні задач, складних для експерта-людини, отримує результати, що не поступаються якістю та ефективністю розв'язкам, отримуваним експертом. Використовують також термін «*інженерія знань*», уведений Е. Фейгенбаумом.

Призначення ЕС полягає у розв'язанні досить складних для експертів задач на основі накопичуваної бази знань, що відображає досвід роботи експертів у даній предметній галузі. Перевага застосування ЕС полягає в можливості прийняття рішень в унікальних ситуаціях, для яких алгоритм заздалегідь не відомий і формується за вихідними даними у вигляді ланцюжка міркувань (правил прийняття рішень) із БЗ. Причому розв'язування завдань передбачається здійснювати в умовах неповноти, недостовірності, багатозначності вихідної інформації і якісних оцінок процесів.

Розглянемо більш докладно, за яких умов комп'ютерну програму можна назвати *експертом*.

1. *Програма повинна володіти знаннями.* Просто здатність виконувати певний алгоритм, наприклад, аналізувати список елементів на наявність певної властивості, не відповідає цій вимозі. Це все одно, що дати першому випадковому перехожому список питань і відповідей і чекати від нього успішного виконання пошуку та усунення несправностей у системах певного типу. Раніше чи пізніше, але він обов'язково стикнеться з ситуацією, не передбаченою в тому списку.

2. *Знання*, якими володіє програма, повинні бути сконцентровані *в певній предметній галузі*. Випадковий набір імен, дат і місць

подій не можна вважати знаннями, які можуть стати основою для програми, що претендує на здатність виконати експертний аналіз. Знання припускають певну організацію та інтеграцію, тобто окремі відомості повинні співвідноситися одна із одною і утворювати щось на зразок ланцюжка, у якому одна ланка тягне за собою наступну ланку.

3. Із наявних знань має *безпосередньо випливати вирішення* проблем. Просто продемонструвати свої знання, що стосуються, наприклад, технічного обслуговування комп'ютерів, — це зовсім не те саме, що привести комп'ютер у робочий стан. Так само отримати доступ до оперативної документації — це зовсім не те саме, що отримати в своє розпорядження фахівця (або програму), здатного впоратися з проблемами, що виникли.

Експертна система може повністю взяти на себе функції, виконання яких зазвичай потребує залучення досвіду людини-спеціаліста, або відігравати роль асистента для людини, що приймає рішення. Експертна система використовується загалом для розв'язання задач прийняття рішень, розпізнавання образів і розуміння людської мови.

Експертна система відрізняється від інших прикладних програм наявністю таких ознак:

1. Експертна система моделює не стільки фізичну (або іншу) природу певної предметної галузі, скільки механізм мислення людини щодо розв'язання задач у цій предметній галузі. Це істотно відрізняє ЕС від систем математичного моделювання або комп'ютерної анімації. Не можна, звичайно, сказати, що програма повністю відтворює психологічну модель фахівця в цій предметної галузі (експерта), але важливо, що основна увага приділяється відтворенню комп'ютерними засобами методики вирішення проблем, яка застосовується експертом, — тобто виконання певної частини завдань так само (або навіть краще), як це робить експерт.

2. Експертна система формує певні міркування та висновки, ґрунтуючись на тих знаннях, які має. Знання в системі подані, як правило, деякою спеціальною мовою і зберігаються окремо від власне програмного коду, який і формує висновки і міркування. Цей компонент програми прийнято називати базою знань.

3. У експертній системі при розв'язанні задач основними є евристичні та наближені методи, які, на відміну від алгоритмічних,

не завжди гарантують успіх. Евристика, по суті, є правилом впливу (*rule of thumb*), яке в машинному вигляді становить деяке знання, набуте людиною в міру накопичення практичного досвіду вирішення аналогічних проблем. Такі методи є приблизними в тому розумінні, що, по-перше, вони не вимагають вичерпної вихідної інформації, і, по-друге, існує певна ступінь впевненості (або невпевненості) у тому, що пропонований розв'язок є правильним.

Експертні системи відрізняються від інших видів програм із галузі ШІ, оскільки:

– оперують предметами *реального світу*, що потребує наявності значного досвіду, накопиченого людиною. Мають яскраво виявлену практичну направленість у науковій або комерційній галузі;

– *високопродуктивні*, тобто система за прийнятний час знаходить розв'язок, не гірший за запропонований спеціалістом у цій предметній галузі;

– можуть *пояснити*, чому запропоновано саме такий розв'язок, і довести його обґрунтованість.

Приклади ЕС:

MYCIN — експертна система для медичної діагностики. Розроблена у Стенфордському університеті для інфекційних захворювань. Ставить відповідний діагноз, виходячи з наданих їй симптомів, і рекомендує курс медикаментозного лікування будь-якої з діагностованих інфекцій. База складається з 450 правил;

PUFF — аналіз порушення дихання. Ця система-аналог *MICIN* для легеневих захворювань;

DENDRAL — розпізнання хімічних структур. Ця система найстаріша з тих, що мають звання експертних. Перші версії системи з'явились ще в 1965 рр. у Стенфордському університеті. Користувач дає системі *DENDRAL* деяку інформацію про речовину, а також дані спектрометрії (інфрачервоного, ядерного магнітного резонансу та мас-спектрометрії). Експертна система видає діагноз у вигляді відповідної хімічної структури;

PROSPECTOR — експертна система, створена для сприяння пошуку комерційно виправданих покладів корисних копалин.

OMEGAMON — сучасна експертна мультиагентна динамічна система, яка працює в реальному часі і відслідковує стан корпоративної інформаційної мережі.

4.2. Призначення та галузі застосування експертних систем

Основні класи задач, для розв'язування яких використовується технологія ЕС, розглянуто нижче.

Діагностика стану систем, у тому числі моніторинг. ЕС виробляють таку діагностику, застосовуючи опис будь-якої ситуації, поведінки або даних про будову різних компонентів, щоб визначити можливі причини несправності діагностованої системи. Прикладами є встановлення обставин захворювання за симптомами, які спостерігаються у хворих (у медицині); визначення несправностей в електронних схемах і визначення несправних компонентів у механізмах різних приладів. Системи діагностики досить часто є помічниками, які не лише ставлять діагноз, але й допомагають в усуненні неполадок. У таких випадках системи цілком можуть взаємодіяти з користувачем, щоб допомогти при пошуку неполадок, а потім навести список дій, необхідних для їх усунення. Багато діагностичних систем розробляються як додатки до інженерної справи і комп'ютерних систем.

Прогнозування розвитку систем на основі моделювання попереднього і теперішнього стану. Експертні системи, що прогнозують що-небудь, визначають імовірнісні умови заданих ситуацій. Прикладами є прогноз збитку, заподіяного урожаю хліба несприятливими погодними умовами, оцінювання попиту на газ на світовому ринку, прогнозування погоди за даними метеорологічних станцій. Системи прогнозування іноді застосовують моделювання, тобто такі програми, які відображають деякі взаємозв'язки в реальному світі, щоб відтворити їх у середовищі програмування, і потім спроектувати ситуації, які можуть виникнути за тих чи інших вихідних даних.

Інтерпретація. Найчастіше застосовують значення різних приладів для опису стану. Інтерпретувальні ЕС здатні обробляти різні види інформації. Прикладом може бути використання даних спектрального аналізу і зміни характеристик речовин для визначення їх складу і властивостей. Також інтерпретацію використовують для показань вимірювальних приладів у котельній для опису стану котлів і води в них. Інтерпретувальні системи найчастіше мають справу безпосередньо з показаннями. У зв'язку з цим виникають труднощі, яких немає в інших видах систем. Оскільки ЕС доводиться інтерпретувати неповну, ненадійну або неправильну інформацію, можуть виникнути або помилки, або значне збільшення обробки даних.

Планування та розробка заходів в організаційному й технологічному керуванні. Експертні системи, призначені для планування, проектують різні операції. Системи зумовлюють практично повну послідовність дій, перш ніж почнеться їх реалізація. Прикладом такого планування подій може бути створення планів військових дій як оборонного, так і наступального характеру на певний термін для отримання переваги перед ворогом.

Проектування або вироблення чітких приписів щодо побудови об'єктів, які задовольняють поставленим вимогам. Експертні системи, що виконують проектування, розробляють різні форми об'єктів з огляду на сформовані обставини і всі супутні фактори. Як приклад можна розглянути генну інженерію.

Автоматичне керування (регулювання). Експертні системи, що здійснюють керування, вельми результативно керують поведінкою системи в цілому. Прикладом є керування різними виробництвами, а також розподіленими комп'ютерними системами. Керуючі ЕС повинні включати в себе компоненти моніторингу для того, щоб контролювати поведінку об'єкта протягом тривалого часу, але вони можуть мати потребу і в інших компонентах із вже проаналізованих типів завдань.

Навчання користувачів за допомогою ЕС ефективне, оскільки для навчання використовується методика багаторазового повторення матеріалу.

Найбільша кількість ЕС використовується у військовій справі, геології, інженерній справі, інформатиці, космічній техніці, математиці, медицині, метеорології, промисловості, сільському господарстві, керуванні процесами, фізиці, філології, хімії, електроніці, юриспруденції.

4.3. Узагальнена архітектура експертної системи

Типова ***статична*** ЕС складається з таких основних компонентів (рис. 4.1): вирішувача (інтерпретатора); робочої пам'яті (РП), яка називається також базою даних (БД); бази знань (БЗ); компонента набуття знань; пояснювального компонента; діалогового компонента.

База даних (*робоча пам'ять*) призначена для зберігання вихідних і проміжних даних розв'язуваного в поточний момент завдання.

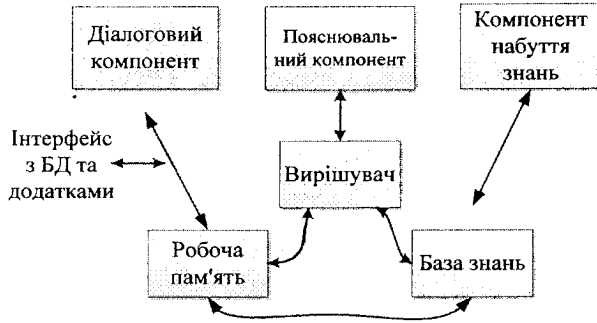


Рис. 4.1. Структура статичної ЕС

База знань в ЕС призначена для зберігання довгострокових (а не поточних) даних, що описують розглянуту галузь, і правил, що описують доцільні перетворення даних цієї галузі.

Вирішувач, використовуючи вихідні дані з робочої пам'яті і знання з БЗ, формує таку послідовність правил, які застосовувані до вихідних даних, приводять до вирішення завдання.

Компонент набуття знань автоматизує процес наповнення ЕС знаннями, здійснюваний користувачем-експертом.

Пояснювальний компонент пояснює, як система отримала розв'язок задачі (або чому вона не отримала розв'язку) і які знання вона при цьому використовувала, що полегшує експерту тестування системи і підвищує довіру користувача до отриманого результату.

Діалоговий компонент орієнтований на організацію дружнього спілкування з користувачем як у ході вирішення завдань, так і в процесі набуття знань і пояснення результатів роботи.

Статичні ЕС використовуються в тих додатках, де можна не брати до уваги зміни навколишнього світу, що відбуваються за час виконання завдання. Перші ЕС, які отримали практичне застосування, були статичними.

На рис. 4.2 показано, що в архітектуру ***динамічної*** ЕС вводяться два додаткові компоненти:

- підсистема моделювання зовнішнього світу;
- підсистема зв'язку із зовнішнім оточенням.

Остання здійснює зв'язки із зовнішнім світом через систему датчиків і контролерів. Крім того, традиційні компоненти статичної ЕС (БЗ і вирішувач) зазнають істотних змін, щоб відобразити часову логіку, яка відбувається в реальному світі подій.

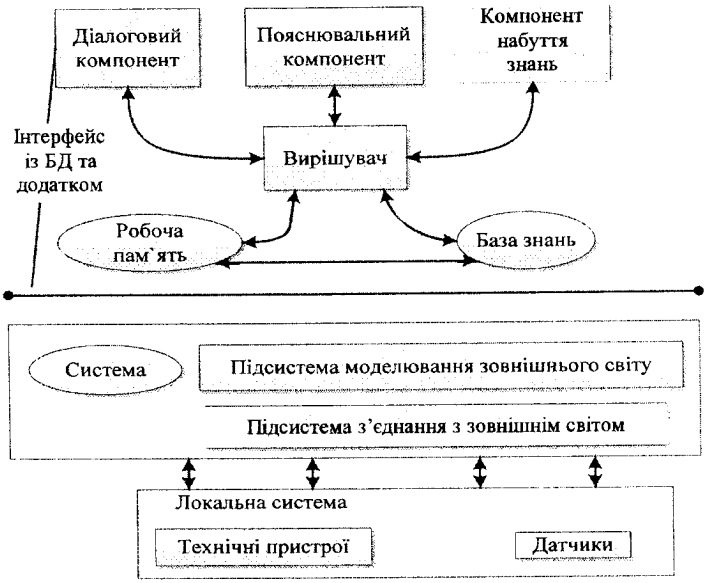


Рис. 4.2. Структура динамічної ЕС

4.4. Класи задач, які розв'язуються за допомогою експертних систем

За ступенем складності розв'язуваних задач ЕС можна класифікувати так:

1. За способом *формування розв'язків* ЕС поділяються на:

– аналітичні — вибір розв'язків з множини існуючих альтернатив (визначення характеристик об'єкта);

– синтетичні — генерація невідомих розв'язків (формування відповіді).

2. За способом *обліку часової ознаки* ЕС можуть бути:

– статичні — вирішують завдання за незмінних у процесі вирішення даних і знань. Статичні системи монотонно, безперервно розв'язують задачі від введення вихідних даних до кінцевого результату;

– динамічні — допускають такі зміни. Динамічні системи допускають можливість перегляду отриманих даних і знань.

3. За *видами використовуваних знань* ЕС можуть бути:

– з детермінованими знаннями;

– з невизначеними даними (під невизначеністю знань або даних розуміється неповнота (відсутність), недостовірність (неточність вимірювань), двозначність (багатозначність), нечіткість (якісна оцінка)).

4. За *кількістю використовуваних джерел даних* ЕС можуть бути побудовані з використанням:

– одного джерела даних;

– декількох джерел даних.

Класифікувальні ЕС. До аналітичних задач передусім відносяться задачі розпізнавання різних ситуацій, коли за набором різних ознак (факторів) виявляється сутність певної ситуації, залежно від якої вибирається певна послідовність дій.

Таким чином, відповідно до вихідних умов, серед альтернативних рішень обирають одне, яке найкраще задовольняє поставлену мету і обмеження. Експертні системи такого виду називають *класифікувальними*, оскільки вони визначають належність аналізованої ситуації до деякого класу.

Як основний метод формування розв'язків використовується метод логічного дедуктивного висновку від загального до часткового, коли шляхом підстановки вихідних даних у деяку сукупність взаємопов'язаних загальних тверджень виходить частковий розв'язок.

Визначальні ЕС. Більш складний тип аналітичних задач становлять задачі, які розв'язуються на основі невизначених вихідних даних і застосовуваних знань. У цьому випадку ЕС повинна довизначити відсутні знання.

У просторі розв'язків може виходити декілька можливих розв'язків із різною ймовірністю або упевненістю в необхідності їх виконання.

Як метод роботи з невизначеністю може використовуватися баєсівський імовірнісний підхід, коефіцієнт впевненості, нечітка логіка. Визначальні ЕС можуть застосовувати для формування розв'язків декілька джерел даних. У цьому випадку можуть використовуватися евристичні прийоми вибору одиниць знань із їх конфліктного набору (наприклад, на основі використання пріоритетів або одержуваного ступеня визначеності результату чи значень функцій належності).

Для аналітичних задач *класифікувального* і *визначального* типів характерні такі предметні галузі:

– *інтерпретація* даних — вибір розв'язку з фіксованої безлічі альтернатив на базі введеної інформації про поточну ситуацію;

– *діагностика* — виявлення причин, що призвели до виникнення ситуації. Необхідна інтерпретація даних із подальшою перевіркою додаткових факторів;

– *корекція* — діагностика, доповнена можливістю оцінки і рекомендації дій щодо виправлення відхилень від нормального стану розглянутих ситуацій.

Трансформувальні ЕС. На відміну від аналітичних статичних ЕС синтезуючі динамічні ЕС припускають повторювані перетворення знань у процесі розв'язування задач, що пов'язано з характером результату, який не можна заздалегідь визначити, а також із динамічністю самої предметної галузі. Як методи розв'язування задач у трансформувальних ЕС використовують різновиди гіпотетичного висновку:

– *генерація* і *тестування*, коли за вихідними даними генеруються гіпотези, а потім перевіряються сформовані гіпотези на підтвердження фактів, що надходять;

– *припущень* і *замовчувань*, коли за неповними даними підбираються знання про аналогічні класи об'єктів, які згодом динамічно адаптуються до конкретної ситуації залежно від її розвитку;

– використання *загальних закономірностей* (метаданих) у разі відомих ситуацій, що дозволяють генерувати відсутні знання.

Багатоагентні ЕС. Для таких динамічних систем характерна інтеграція в базі знань декількох різнорідних джерел знань, які обмінюються між собою одержуваними результатами на динамічній основі (наприклад, дошка оголошень).

Для багатоагентних ЕС характерні такі особливості:

– проведення альтернативних міркувань на підставі використання різних джерел знань та застосування механізму усунення протиріч;

– розподілене вирішення проблем, які розбиваються на паралельно вирішувані підпроблеми, що відповідають самостійним джерелам знань;

– застосування багатьох стратегій роботи механізму виведення залежно від типу вирішуваної проблеми;

– обробка великих масивів даних, що містяться в БД;

– використання різних математичних моделей і зовнішніх процедур, що зберігаються в базі моделі;

– здатність переривати розв'язування задач у зв'язку з необхідністю отримання додаткових даних і знань від користувачів моделей, паралельно вирішуваних підпроблем.

Для синтезованих динамічних ЕС найбільш характерні такі предметні галузі:

– *проектування* — визначення конфігурації об'єктів з точки зору досягнення заданих критеріїв ефективності та обмежень;

– *прогнозування* — передбачення наслідків поточної ситуації на основі математичного та евристичного моделювання;

– *диспетчеризація* — розподіл робіт у часі, складання розкладів;

– *планування* — вибір послідовності дій користувача з досягнення поставленої мети;

– *моніторинг* — спостереження за поточною ситуацією з можливістю подальшої корекції;

– *керування* — моніторинг, доповнений реалізацією дій в автоматичних системах.

Системи, що самостійно навчаються. У їх основі лежать методи автоматичної класифікації ситуацій реальної практики. Приклади реальних ситуацій накопичуються за певний період і становлять навчальну вибірку. Причому навчальна вибірка може бути «з учителем», коли для кожного прикладу задається в явному вигляді значення його належності певному класу ситуацій або класоутворювальні ознаки, і «без вчителя», коли за ступенем близькості значень система сама виділяє класи ситуацій.

У результаті навчання системи автоматично будуються узагальнені правила або функції, що визначають належність ситуацій класам, якими навчена система користується при інтерпретації нових ситуацій.

Недоліки таких систем:

– можлива неповнота або зашумленість (надмірність) навчальної вибірки і, як наслідок, відносна адекватність бази знань проблем, що виникають;

– виникнення проблем пов'язано з поганою змістовою ясністю в залежності ознак;

– обмеження в розмірності однакового простору викликає неглибокий опис предметної галузі та вузьку спрямованість застосувань.

Індуктивні системи характеризуються тим, що узагальнення прикладів за принципом від часткового до загального зводиться до виявлення підмножин прикладів, що відносяться до одних і тих же підкласів, та визначення для них значущих ознак.

Процес класифікації прикладів здійснюється так:

1. Обирається ознака класифікації із заданих або послідовно, або за певним правилом (наприклад, відповідно до максимальної кількості одержуваних підмножин прикладів).

2. За значенням вибраної ознаки множина прикладів розбивається на підмножини.

3. Перевіряється належність кожного прикладу.

4. Якщо будь-яка підмножина прикладів належить одному підкласу, тобто у всіх прикладів підмножини збігається значення класоутворювальних ознак, то процес класифікації закінчується (при цьому інші ознаки класифікації не розглядаються).

5. Для підмножин прикладів із незбіжними значеннями класоутворювальних ознак процес класифікації триває, починаючи з п. 1 (кожна підмножина прикладів стає класифікованою множиною).

4.5. Розробка експертної системи

У розробці ЕС беруть участь представники таких спеціальностей:

– *експерт у предметній галузі* визначає знання (дані та правила), що характеризують предметну галузь, забезпечує повноту та правильність уведених в ЕС знань;

– *інженер зі знань* допомагає експерту виявити і структурувати знання, необхідні для роботи ЕС; здійснювати вибір того інструментального засобу, що найбільш підходить для даної предметної галузі, і визначає спосіб подання знань у цій ЕС; виділяє і програмує (традиційними засобами) стандартні функції (типові для даної предметної галузі), які використовуватимуться в правилах, що вводяться експертом;

– *програміст із розробки інструментальних засобів* поєднує їх із тим середовищем, у якому вони будуть використані.

Необхідно зазначити, що відсутність серед учасників розробки інженерів зі знань (тобто їх заміна програмістами) або приводить до невдачі процес створення ЕС, або значно подовжує його.

Експертна система працює в двох режимах: режимі набуття знань і в режимі розв'язування задачі (режимі консультації або режимі використання ЕС).

У *режимі набуття знань* спілкування з ЕС здійснює експерт (через інженера зі знань). У цьому режимі експерт, використовуючи компонент набуття знань, наповнює систему знаннями, які дозволяють ЕС у режимі розв'язання самостійно (без експерта) розв'язувати задачі з предметної галузі. Експерт описує предметну галузь у вигляді сукупності даних і правил. Дані визначають об'єкти, їх характеристики і значення, що існують в галузі експертизи. Правила визначають способи маніпулювання з даними, характерні для даної галузі.

Відзначимо, що режиму набуття знань у традиційному підході до розробки програм відповідають етапи алгоритмізації, програмування і налагодження, виконувані програмістом. Таким чином, на відміну від традиційного підходу, у випадку ЕС програми розробляє не програміст, а експерт (за допомогою ЕС), який не володіє програмуванням.

У *режимі консультації* спілкування з ЕС здійснює кінцевий користувач, якого цікавить результат і (або) спосіб його одержання. Необхідно зауважити, що залежно від призначення ЕС, користувач може не бути фахівцем у предметній галузі. У такому разі він звертається до ЕС за результатом, не вміючи одержати його сам. Або бути фахівцем, у цьому випадку користувач може сам отримати результат, але він звертається до ЕС, щоб або прискорити процес отримання результату, або покласти на ЕС рутинну роботу.

У режимі консультації дані про задачу користувача після обробки їх діалоговим компонентом надходять до робочої пам'яті. Вирішувач на підставі вхідних даних із робочої пам'яті, загальних даних про предметну галузь і правил із БЗ формує розв'язок задачі. Експертна система при розв'язуванні задачі не лише виконує визначену послідовність операції, а й попередньо формує її. Якщо реакція системи не зрозуміла користувачеві, то він може вимагати пояснення.

4.6. Етапи розробки експертної системи

Розробка ЕС суттєво відрізняється від розробки звичайного програмного продукту.

Використовувати ЕС слід лише тоді, коли розробка ЕС можлива, виправдана, й методи інженерії знань відповідають розв'язуваній задачі.

Щоб розробка ЕС була *можливою* для даного додатка, необхідне одночасне виконання принаймні таких вимог:

– експерти в даній галузі вирішують завдання значно краще, ніж фахівці-початківці;

– експерти сходяться в оцінці запропонованого розв'язку;

– експерти здатні вербалізувати (виразити природною мовою) і пояснити використовувані ними методи;

– вирішення завдання вимагає лише міркувань, а не дій;

– завдання не має бути занадто важким (тобто процес розв'язання повинен займати у експерта декілька годин або днів, а не тижнів);

– завдання має ставитися до структурованої галузі.

Застосування ЕС може бути *виправдане* одним із таких чинників:

– вирішення завдання принесе значний ефект, наприклад економічний;

– використання людини-експерта неможливе;

– при передачі інформації експерту відбувається неприпустима втрата часу або інформації.

Програма *відповідає методам ЕС*, якщо розв'язувана задача характеризується такими рисами:

– завдання може бути природним чином вирішено за допомогою маніпуляції з символами (тобто за допомогою символічних міркувань), а не маніпуляцій із числами, як прийнято в математичних методах і в традиційному програмуванні;

– завдання повинно мати евристичну природу, а не алгоритмічну;

– завдання має бути досить складне, щоб виправдати витрати на розробку ЕС;

– завдання має бути досить вузьким, щоб вирішуватися методами ЕС, і практично значущим.

При розробці ЕС, як правило, використовується концепція ***«швидкого прототипу»***. Її сутність полягає в тому, що розробники не намагаються відразу побудувати кінцевий продукт. На початковому етапі вони створюють прототип (прототипи) ЕС, який може вирішувати типові завдання конкретної програми, при цьому час і трудомісткість його розробки незначні.

У разі успіху експерт за допомогою інженера зі знань розширює знання прототипу про предменту галузь. У разі невдачі може знадобитися розробка нового прототипу або розробники можуть дійти висновку про непридатність методів ЕС для цієї програми. У міру збільшення знань прототип може досягти такого стану, коли він успішно вирішує всі завдання цієї програми. Перетворення прототипу ЕС на кінцевий продукт зазвичай приводить до перепрограмування ЕС мовами низького рівня, що забезпечує як збільшення швидкодії, так і зменшення необхідної пам'яті.

Технологія розробки ЕС включає шість основних етапів (рис. 4.3): 1) ідентифікацію; 2) концептуалізацію; 3) формалізацію; 4) виконання; 5) тестування; 6) дослідну експлуатацію.

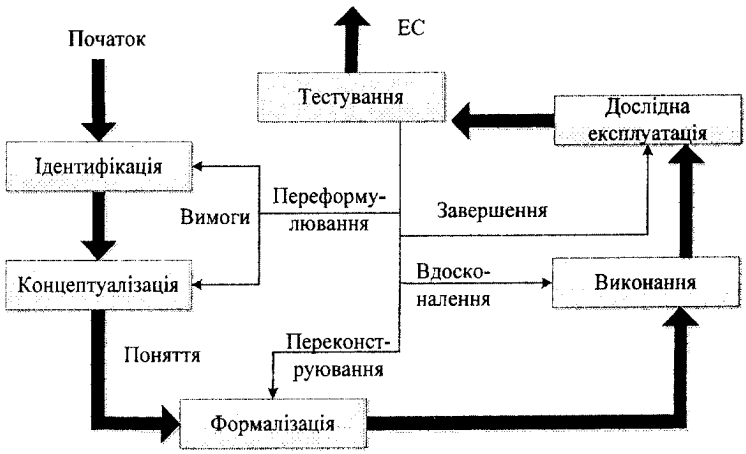


Рис. 4.3. Технологія розробки експертної системи

На етапі ***ідентифікації*** визначаються цілі та завдання розробки, визначаються експерти і типи користувачів.

На етапі ***концептуалізації*** проводиться змістовний аналіз предментої галузі, виявляються використовувані поняття і їх взаємозв'язки, визначаються методи розв'язання задач.

На етапі ***формалізації*** вибираються ЕС і визначаються способи подання всіх видів знань, формалізуються основні поняття, визначаються способи інтерпретації знань, моделюється робота системи, оцінюється адекватність цілям системи зафіксованих понять, методів вирішення, засобів подання й маніпулювання знаннями.

На етапі ***виконання*** експерт наповнює БЗ. У зв'язку з тим, що основою ЕС є знання, цей етап є найбільш важливим і найбільш трудомістким етапом розробки ЕС. У процесі набуття знань виокремлюють *отримання знань від експерта*, *організацію знань*, що забезпечує ефективну роботу системи, і *подання знань* у вигляді, зрозумілому ЕС. Процес набуття знань здійснюється інженером зі знань на підставі аналізу діяльності експерта з вирішення реальних завдань.

На етапі ***тестування*** прототип перевіряється на зручність і адекватність інтерфейсів введення-виведення, ефективність стратегії керування, якість перевіряльних прикладів, коректність БЗ. Тестування — це виявлення помилок у вибраному підході, у реалізації прототипу, а також вироблення рекомендацій із доведення системи до промислового варіанта.

На етапі ***дослідної експлуатації*** перевіряється придатність ЕС для кінцевих користувачів. За результатами цього етапу може знадобитися суттєва модифікація ЕС.

Слід зазначити, що процес розробки ЕС не зводиться до суворої послідовності перерахованих вище етапів. У процесі роботи доводиться неодноразово повертатися на більш ранні етапи й переглядати прийняті там рішення.

4.7. Базові функції експертної системи

Розрізняють чотири базові функції, які реалізуються в ЕС:

- набуття знань;
- подання знань;
- керування процесом пошуку рішення;
- роз'яснення прийнятого рішення.

4.7.1. Набуття знань

Функція ***набуття знань*** у ЕС передбачає передачу потенційного досвіду вирішення проблеми від деякого джерела знань і перетворення його у вигляд, який дозволяє використовувати ці знання в програмі.

Методи отримання знань поділяють на такі групи:

– *пасивні методи*: спостереження, аналіз протоколу «думки вголос», лекції;

– *активні індивідуальні*: анкетування, текстологічні (отримання знань із текстів), інтерв'ю, вільний діалог;

– *активні групові методи*: круглий стіл, дискусія, полеміка, мозковий штурм.

4.7.2. Подання знань

Функція ***подання знань*** у ЕС — це процес пошуку методів формального опису великих масивів корисної інформації для їх подальшої обробки за допомогою символічних обчислень. Цей процес також називають ***інженерією знань***. Моделі подання знань було розглянуто у розд. 3.

Інженером зі знань називається фахівець, який досліджує конкретну предметну галузь, визначає, які поняття в ній важливі, і створює формальне подання об'єктів і відношень у цій предметній галузі.

Проекти в галузі інженерії знань багато в чому відрізняються один від одного за своїм змістом, охопленням та складністю, але всі ці проекти включають перелічені нижче основні етапи процесу інженерії знань.

1. *Ідентифікація завдання*. Інженер зі знань повинен окреслити коло питань, які повинна підтримувати БЗ, і види фактів, які будуть доступними для даного екземпляра завдання.

Наприклад, чи повинна БЗ надавати можливість вибирати дії, чи від неї вимагається лише пошук відповідей на запитання про зміст різних компонентів середовища? Чи повинні факти, отримані від датчиків, включати дані про поточне місцезнаходження? Саме завдання визначає, які знання повинні бути подані в базі, щоб можна було пов'язати екземпляри завдання з відповідями.

2. *Збір необхідних знань*. Інженер зі знань може вже бути експертом у цій предметній галузі або йому може знадобитися спілкуватися зі справжніми експертами для виявлення всього, що вони знають; цей процес називається *набуттям знань*. На цьому етапі знання ще не подані формально. Його призначення полягає в тому, щоб зрозуміти, яким має бути спектр знань у БЗ, який визначається самим завданням, а також розібратися в тому, як фактично функціонує розглянута предметну галузь. Для реальних предметних галузей завдання виявлення релевантних знань (що стосуються справи) може виявитися досить складним.

3. *Визначення словника предикатів, функцій і констант.* У іншому формулюванні цей етап можна визначити як перетворення важливих понять рівня предметної галузі на імена логічного рівня. Від стилю програмування і стилю інженерії знань може істотно залежати остаточний успіх проекту. Результатом вибору найбільш придатних засобів подання стає словник, відомий під назвою *онтології предметної галузі.* Онтологія визначає, які об'єкти існують, але не визначає їх конкретні властивості та взаємозв'язки.

4. *Реєстрація загальних знань про предметну галузь.* Інженер зі знань записує аксіоми для всіх термінів словника. Тим самим він закріплює зміст цих термінів, дозволяючи експерту перевірити їх зміст. На цьому етапі часто виявляються неправильні трактування або пропуски в словнику, який необхідно виправити, повернувшись на етап 3 і знову пройти цю ітерацію в поточному процесі проектування.

5. *Складання опису конкретного екземпляра задачі.* Якщо онтологія добре продумана, цей етап буде нескладним. Він зводиться до написання простих атомарних висловлювань про екземпляри понять, які вже є частиною онтології.

6. *Передача запитів процедурі логічного виведення та отримання відповідей.* На цьому етапі застосовують процедуру логічного виведення до аксіом і фактів про конкретне завдання для отримання фактів, які нам хочеться дізнатися.

7. *Налагодження бази знань.* Відповіді на запити при першій спробі рідко виявляються правильними. Точніше, відповіді будуть правильними для БЗ у тому вигляді, в якому вона написана, за умови, що процедура логічного виведення є несуперечливою, але вони не будуть такими, яких очікує користувач.

Наприклад, якщо бракує якоїсь аксіоми, то на деякі запити з цієї БЗ не можна буде знайти відповідь. У цьому може допомогти продуманий процес налагодження. Відсутні або занадто слабкі аксіоми можна легко помітити виявленням ділянок, на яких несподівано обривається ланцюжок етапів логічного виведення. Неправильні аксіоми можуть бути виявлені на підставі того, що вони становлять собою помилкові твердження про предметну галузь.

4.7.3. Керування процесом пошуку рішення

При проектуванні ЕС значна увага має бути приділена і тому, як здійснюється доступ до знань і як вони використовуються при пошуку рішення.

Знання про те, які знання потрібні в тій чи іншій конкретній ситуації, і вміння ними розпорядитися — важлива частина процесу функціонування ЕС. Такі знання отримали найменування *метазнання* — тобто знання про знання.

Використання різних стратегій перебирання наявних знань, як правило, істотно впливає на характеристики ефективності ЕС. Ці стратегії визначають, яким способом програма знайде вирішення проблеми в деякому просторі альтернатив.

4.7.4. Роз'яснення прийнятого рішення

Подання інформації про поведінку ЕС важливе з багатьох причин.

Користувачі, що працюють із системою, потребують підтвердження того, що в кожному конкретному випадку висновок, якого дійшла програма, здебільшого є коректним.

Інженери, що мають справу з формуванням БЗ, повинні переконатися, що сформульовані ними знання застосовано правильно, у тому числі й у разі, коли існує прототип.

Експертам у предметній галузі бажано простежити хід міркувань і спосіб використання тих відомостей, які з їхніх слів були введені до БЗ.

Це дозволить судити, наскільки коректно вони застосовуються за певної ситуації.

Програмістам, які супроводжують, налагоджують і модернізують систему, потрібно мати у своєму розпорядженні інструмент, що дозволяє зазирнути в «її середину» на рівні більш високому, ніж виклик окремих мовних процедур.

Менеджер системи, що використовує експертну технологію, який врешті-решт відповідає за наслідки рішення, прийнятого програмою, також потребує підтвердження, що ці рішення досить обґрунтовані.

Здатність системи пояснити методику прийняття рішення іноді називають ***прозорістю*** системи.

Під цим розуміється, наскільки просто персоналу з'ясувати, що робить програма і чому.

Цю характеристику системи слід розглядати разом з режимом керування, оскільки послідовність етапів прийняття рішення тісно пов'язана із заданою стратегією поведінки.

Висновки

1. Експертна система — це комп'ютерна програма, здатна частково замінити фахівця-експерта у вирішенні проблемної ситуації і яка є інструментом, що підсилює інтелектуальні здібності експерта.

2. Комп'ютерна програма вважається експертом, якщо вона володіє знаннями в певній предметній галузі, з яких безпосередньо випливає вирішення поставленого завдання.

3. Експертна система відрізняється від інших прикладних програм тим, що моделює механізм мислення людини щодо розв'язування задач у цій предметній галузі, формує певні міркування та висновки, ґрунтуючись на тих знаннях, які має в розпорядженні, при розв'язуванні задач основними є евристичні та наближені методи.

4. Експертна система відрізняється від інших видів програм із галузі ШІ, оскільки оперує предметами реального світу, має високу продуктивність та здатна пояснити, чому запропоновано саме таке рішення і довести його обґрунтованість.

5. Основні галузі, де використовується технологія ЕС, — це діагностика, прогнозування, інтерпретація, планування, проектування, автоматичне керування та навчання.

6. Статична ЕС складається з вирішувача, робочої пам'яті (БД), БЗ, компонента набуття знань, пояснювального та діалогового компонента. У архітектуру динамічної ЕС вводяться додатково підсистеми моделювання зовнішнього світу і зв'язку із зовнішнім оточенням.

7. Розробка ЕС суттєво відрізняється від розробки звичайного програмного продукту. На початковому етапі створюють прототип, який може виконувати елементарні операції. У разі успіху експерт за допомогою інженера зі знань розширює знання прототипу про предметну галузь, досягаючи таким чином необхідного рівня складності ЕС.

8. У розробці ЕС беруть участь представники таких спеціальностей: експерт у предметній галузі, інженер зі знань, програміст із розробки інструментальних засобів.

9. Технологія розробки ЕС включає шість основних етапів: ідентифікацію, концептуалізацію, формалізацію, виконання, тестування, дослідну експлуатацію.

10. Базові функції, які реалізуються в ЕС: набуття та подання знань, керування процесом пошуку рішення, роз'яснення прийнятого рішення.

Вправи

1. Чи є система пошуку в мережі *World Wide Web* експертною? Якщо ні, то яких властивостей їй бракує для того, щоб кваліфікувати її як ЕС пошуку потрібної *Web*-сторінки?

2. Чому завдання набуття знань є вузьким місцем у проектуванні ЕС? Запропонуйте рішення для усунення такої ситуації.

3. Складіть правила для ЕС діагностування несправності комп'ютера.

4. Спроектуйте нескладну консультаційну програму для вибору оптимальної комплектації комп'ютера.

5. Чому як пояснення процесу логічного виведення користувачеві недостатньо представити лише результати трасування використаних правил?

6. Складіть список ознак (у вигляді таблиці) для побудови ЕС, які однозначно характеризують деякі породи собак.

Запитання та завдання для самоперевірки

1. Схарактеризуйте, у чому полягає відмінність ЕС від інших програм?
2. Яке призначення основних компонентів ЕС?
3. Що таке концепція «швидкого прототипу»?
4. Які основні функції має виконувати ЕС?
5. У чому полягає призначення функції роз'яснення прийнятого рішення?
6. У яких двох режимах може працювати ЕС?
7. Що таке «прозорість» ЕС?

РОЗДІЛ 5

НЕЙРОННІ МЕРЕЖІ

5.1. Історія розвитку

Ідея нейронних мереж (НМ) народилася в результаті спроб відтворити здатність біологічних нервових систем навчатися і виправляти помилки, моделюючи низькорівневу структуру мозку.

Основною галуззю досліджень зі ШІ в 60–80-ті рр. XX ст. були ЕС. Такі системи ґрунтувалися на високорівневому моделюванні процесу мислення (зокрема, на його поданні як маніпуляцій із символами). Невдовзі стало зрозуміло, що такі системи, хоч і можуть принести користь у певних галузях, не охоплюють деякі ключові аспекти роботи людського мозку. Згідно з однією з точок зору, причина цього полягає в тому, що вони не можуть відтворити структуру мозку. Щоб створити ШІ, необхідно побудувати систему зі схожою архітектурою.

У 1943 р. вийшла робота Дж. Маккалокома і У. Пітта «Логічне числення ідей, що стосуються нервової діяльності», в якій було побудовано модель нейрона і сформульовано принципи побудови штучних НМ. Вчені вперше дослідили здатність НМ до навчання.

У 1962 р. американський нейрофізіолог Ф. Розенблатт запропонував свою модель НМ — *персептрон*. Тоді персептрон використовувався для класифікації вхідних сигналів у один із двох класів. На жаль, одношаровий персептрон був обмеженим і зазнав критики у 1969 р. у книзі М. Мінскі і С. Пейперта «Персептрони». Значні дослідження з НМ було згорнуто майже на 10 років.

У 70-ті рр. XX ст. було запропоновано багато цікавих розробок, таких, наприклад, як *когнітрон*, здатний добре розпізнавати досить складні образи незалежно від повороту і зміни масштабу зображення. У 1982 р. американський біофізик Дж. Хопфілд запропонував оригінальну модель НМ, названу його ім'ям. У наступні декілька років було знайдено безліч ефективних алгоритмів: мережа зустрічного потоку, двонаправлена асоціативна пам'ять тощо.

5.2. Галузі застосування

Не можна вважати НМ універсальними для вирішення всіх обчислювальних проблем. Традиційні комп'ютери та обчислювальні методи є ідеальними для багатьох застосувань. Сучасні цифрові обчислювальні машини перевершують людину за здатність робити числові й символьні обчислення. Однак людина може без зусиль вирішувати складні завдання сприйняття зовнішніх даних (наприклад, впізнавання людини в юрбі за її обличчям) з такою швидкістю і точністю, що наймогутніший у світі комп'ютер порівняно з нею здається безнадійним. Проте існують галузі, де застосування НМ виявляється досить ефективним. Розглянемо деякі з них.

1. *Класифікація образів.* Завдання полягає у визначенні належності вхідного образу (наприклад, мовного сигналу чи рукописного символу), вираженого вектором ознак, до одного або декількох попередньо визначених класів. До відомих програм відносяться розпізнавання букв, розпізнавання мови, класифікація сигналу електрокардіограми, класифікація клітин крові.

2. *Кластеризація/категоризація.* При розв'язанні задачі кластеризації, що відома також як класифікація образів «без вчителя», навчальна множина з визначеними класами відсутня. Алгоритм кластеризації грунтується на подібності образів і розміщує схожі образи в один кластер. Відомі випадки застосування кластеризації для отримання знань, стиснення даних і дослідження властивостей даних.

3. *Передбачення/прогноз.* Якщо задано n дискретних відліків $\{y(t_1), y(t_2), ..., y(t_n)\}$ у послідовні моменти часу $t_1, t_2, ..., t_n$, завдання полягає в передбаченні значення $y(t_n + 1)$ у наступний момент часу $t_n + 1$. Прогнозування має велике значення для прийняття рішень у бізнесі, науці й техніці (наприклад, передбачення цін на фондовій біржі, прогноз погоди).

4. *Апроксимація функцій.* Припустимо, що є навчальна вибірка $((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$ (пари даних вхід-вихід), яка генерується невідомою функцією F, спотвореною завадами. Завдання апроксимації полягає у визначенні невідомої функції F. Апроксимація функцій необхідна при розв'язуванні численних інженерних і наукових задач моделювання.

5. *Оптимізація.* Численні проблеми в математиці, статистиці, техніці, науці, медицині та економіці можуть розглядатися як проблеми оптимізації. Завданням алгоритму оптимізації є отримання

такого рішення, яке задовольняє системі обмежень і максимізує чи мінімізує цільову функцію.

6. *Асоціативна пам'ять.* У традиційних комп'ютерах звертання до пам'яті доступне лише за допомогою адреси, що не залежить від змісту пам'яті. Більше того, якщо допущено помилку в обчисленні адреси, то може бути знайдено зовсім іншу інформацію. Асоціативна пам'ять, чи пам'ять, що адресується за змістом, доступна за вказівкою заданого змісту. Вміст пам'яті можна викликати навіть за частковим входом чи спотвореним змістом. Асоціативна пам'ять надзвичайно важлива при створенні мультимедійних інформаційних баз даних.

5.3. Біологічний та штучний нейрони

Штучні нейрони і НМ є електронними моделями нейронної структури мозку. Мозок людини складається з дуже великої кількості (приблизно 10^{10}) нейронів, з'єднаних численними зв'язками (у середньому кілька тисяч зв'язків на один нейрон).

Нейрони здатні запам'ятовувати, думати і застосовувати попередній досвід до кожної дії, що відрізняє їх від інших клітин тіла.

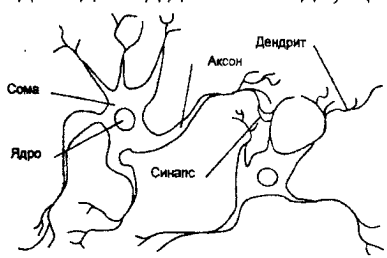


Рис. 5.1. Схема біологічного нейрона

Біологічний нейрон (рис. 5.1) складається з тіла (*cell body*), або соми (*soma*), і відростків нервових волокон двох типів — дендритів (*dendrites*), якими піднімаються імпульси, і єдиного аксона (*axon*), по якому нейрон може передавати імпульс.

Принцип роботи біологічного нейрона моделюється за допомогою ***штучного нейрона*** (рис. 5.2), який становить собою пристрій, що має декілька входів (дендрити), і один вихід (аксон).

Кожному входу ставиться у відповідність певний ваговий коефіцієнт (w), що характеризує пропускну здатність каналу і оцінює ступінь впливу сигналу з цього входу на сигнал на виході. Оброблювані нейроном сигнали можуть бути аналоговими або цифровими (1 або 0). У тілі нейрона відбувається зважене підсумовування вхідних збуджень, і далі це значення є аргументом активаційної функції нейрона. Окремі нейрони взаємодіють один з одним за допомогою великої кількості зв'язків.

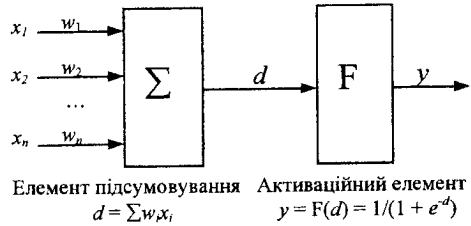


Рис. 5.2. Схема штучного нейрона

5.4. Структура штучної нейронної мережі

Біологічні НМ створені у тривимірному просторі з мікроскопічних компонент і здатні до різноманітних з'єднань. Але для створеної людиною мережі існують фізичні обмеження.

Штучні НМ є групуванням штучних нейронів у вигляді з'єднаних між собою шарів.

Хоча існують мережі, які містять лише один прошарок або навіть один елемент, більшість реалізацій використовують мережі, що містять щонайменше три типи прошарків — вхідний, прихований та вихідний (рис. 5.3).

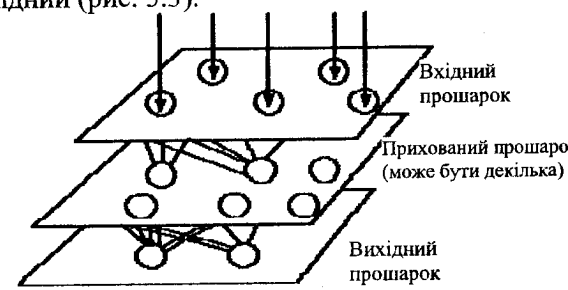


Рис. 5.3. Схема штучної нейромережі

Прошарок *вхідних* нейронів отримує дані або з вхідних файлів, або безпосередньо з електронних датчиків.

Вихідний прошарок пересилає інформацію безпосередньо до зовнішнього середовища, до вторинного комп'ютерного процесу або до іншого пристрою.

Між двома попередніми прошарками може бути декілька *прихованих*, що містять багато різноманітно зв'язаних нейронів. Входи та виходи кожного з прихованих нейронів з'єднані з іншими нейронами.

Напрямок зв'язку від одного нейрона до іншого є важливим аспектом нейромереж. У більшості мереж кожен нейрон прихованого прошарку отримує сигнали від усіх нейронів попереднього прошарку і зазвичай від нейронів вхідного прошарку.

Після виконання операцій над сигналами нейрон передає свій вихід всім нейронам наступних прошарків, забезпечуючи передачу сигналу вперед (*feedforward*) на вихід.

При зворотному зв'язку вихід нейронів прошарку скеровується до нейронів попереднього прошарку (рис. 5.4).

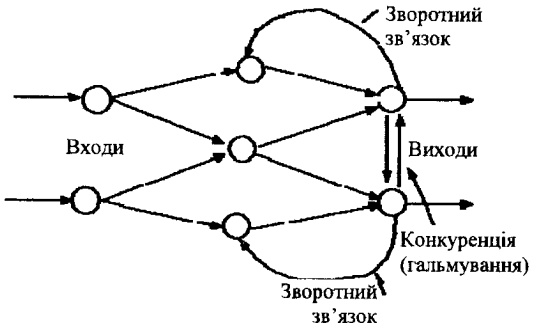


Рис. 5.4. Схема штучної нейромережі зі зворотним зв'язком

5.5. Навчання штучної нейронної мережі

Здатність до навчання є фундаментальною властивістю мозку. Процес навчання може розглядатися як визначення архітектури мережі і налаштування вагових коефіцієнтів для ефективного виконання спеціальної задачі. Нейромережа налаштовує вагові коефіцієнти за наявною навчальною множиною. Властивість мережі навчатися на прикладах робить її більш привабливою порівняно з системами, які функціонують відповідно до визначеної системи правил, сформульованої експертами.

Робота мережі розподіляється на *навчання* та *адаптацію*. Навчання — це процес адаптації мережі до пропонованих еталонних зразків шляхом модифікації (відповідно до тих чи інших алгоритмів) вагових коефіцієнтів зв'язків між нейронами. Для процесу навчання необхідно мати модель зовнішнього середовища, у якій функціонує НМ, потрібну для вирішення завдання інформацію.

Окрім того, необхідно визначити, як модифікувати вагові параметри мережі.

Зауважимо, що цей процес навчання є результатом алгоритму функціонування мережі, а не попередньо закладених у неї знань людини, як це часто буває в СШ. Алгоритм навчання означає процедуру, у якій використовуються правила навчання для налаштування вагових коефіцієнтів.

Існують три види навчання НМ:

– «*із учителем*», коли НМ має в своєму розпорядженні правильні відповіді (виходи мережі) на кожен вхідний приклад. Вагові коефіцієнти налаштовуються так, щоб мережа виробляла відповіді, близькі до відомих правильних відповідей;

– «*без учителя*» (самонавчання) не вимагає знання правильних відповідей на кожний приклад навчальної вибірки. Використовується внутрішня структура даних та кореляція між зразками в навчальній множині для розподілу зразків за категоріями;

– за *змішаного* навчання частина вагових коефіцієнтів визначається за допомогою навчання «з учителем», тоді як інша визначається за допомогою самонавчання.

Розрізняють також контрольоване та неконтрольоване навчання НМ.

За ***контрольованого навчання*** НМ змінний вихід постійно порівнюється з бажаним виходом. Вагові коефіцієнти на початку встановлюються довільно, але під час наступних ітерацій коректуються для досягнення близької відповідності між бажаним та змінним виходом. Створені методи навчання спрямовані на мінімізацію змінних похибок усіх елементів обробки, які виникають за час неперервної зміни синаптичних вагових коефіцієнтів до досягнення прийнятної точності мережі.

Перед використанням НМ із контрольованим навчанням повинна бути навченою. Фаза навчання може тривати досить довго. Навчання вважається закінченим при досягненні НМ визначеного користувачем рівня ефективності. Цей рівень означає, що мережа досягла бажаної статистичної точності, оскільки вона видає бажані виходи для заданої послідовності входів.

Після навчання вагові коефіцієнти з'єднань фіксуються для подальшого застосування. Деякі типи мереж припускають під час використання неперервне навчання з набагато повільнішою оцінкою навчання, що допомагає мережі адаптуватися до умов, що повільно змінюються.

Навчальні множини повинні бути досить великими, щоб містити всю необхідну інформацію для виявлення важливих особливостей і зв'язків. Але й навчальні приклади повинні мати широке розмаїття даних.

Якщо мережа навчається лише для одного прикладу, вагові коефіцієнти, старанно встановлені для цього прикладу, радикально змінюються у навчанні для наступного прикладу. Попередні приклади при навчанні наступних просто забуваються. У результаті система повинна навчатися всьому разом, знаходячи найкращі вагові коефіцієнти для загальної множини прикладів.

Наприклад, у навчанні системи розпізнавання піксельних образів для десяти цифр, які подані двадцятьма прикладами кожної цифри, всі приклади цифри «сім» не доцільно подавати послідовно. Краще надати мережі спочатку один тип подання всіх цифр, потім другий і так далі.

Якщо після контрольованого навчання НМ ефективно опрацьовує дані навчальної множини, важливою є її ефективність при роботі з даними, які не використовувалися для навчання. У разі отримання незадовільних результатів для тестової множини навчання продовжується. Тестування використовується для запам'ятовування не лише даних заданої навчальної множини, але і створення загальних образів, що можуть міститися в даних.

Неконтрольоване навчання передбачає, що комп'ютери можуть самонавчатись у справжньому роботизованому сенсі. На сьогодні неконтрольоване навчання використовується в мережах відомих як самоорганізовані карти (*self organizing maps*), що перебувають у досить обмеженому користуванні, але доводять перспективність такого виду навчання.

За такого навчання мережі не використовують зовнішніх впливів для коректування своїх вагових коефіцієнтів і внутрішньо контролюють свою ефективність, шукаючи регулярність або тенденції у вхідних сигналах та роблять адаптацію згідно з навчальною функцією. Навіть без повідомлення правильності чи неправильності дій мережа повинна мати інформацію щодо власної організації, яка закладена до топології мережі та навчальних правил.

Алгоритм неконтрольованого навчання спрямований на визначення близькості між групами нейронів, які працюють разом. Якщо зовнішній сигнал активує будь-який вузол у групі нейронів, дія всієї

групи в цілому збільшується. Аналогічно, якщо зовнішній сигнал у групі зменшується, це призводить до гальмівного ефекту на всю групу.

Конкуренція між нейронами формує основу для навчання. Навчання конкуруючих нейронів підсилює відгуки певних груп на певні сигнали. Це пов'язує групи між собою та відгуком. При конкуренції змінюються вагові коефіцієнти лише нейрона-переможця.

5.6. Класифікація нейронних мереж

Об'єднуючись у мережі, нейрони утворюють системи обробки інформації, які забезпечують ефективну адаптацію моделі до постійних змін із боку зовнішнього середовища. У процесі функціонування мережі вхідний вектор сигналів перетворюється на вихідний. Архітектура НМ визначає конкретний вид перетворення.

За архітектурою зв'язків більшість відомих НМ можна згрупувати у два великі класи:

1. Мережі *прямого поширення* (із односпрямованими послідовними зв'язками) відносять до статичних. На задані входи нейронів надходить незалежний від попереднього стану мережі вектор вхідних сигналів. До даної групи можна віднести:

– персептрони;

– мережі зі зворотним поширенням помилки (*back propagation*);

– карти Кохонена.

2. Мережі *зворотного поширення* (із рекурентними зв'язками) вважаються динамічними, оскільки за рахунок зворотних зв'язків (петель) входи нейронів модифікуються в часі, що приводить до зміни стану мережі. Сюди відносяться:

– мережа Хопфілда;

– мережа Хеммінга;

– мережа адаптивної резонансної теорії;

– двонапрямлена НМ.

5.7. Персептрон Розенблата

Персептрон Розенблата (рис. 5.5) вважається першою моделлю НМ й класикою для вивчення принципу функціонування.

У найпростішому вигляді персептрон складається з сукупності чутливих (сенсорних) елементів (*S*-елементів), на які надходять вхідні сигнали. *S*-елементи довільно пов'язані з сукупністю асоціа-

тивних елементів (*A*-елементів), вихід яких відрізняється від нуля лише тоді, коли збуджена досить велика кількість *S*-елементів, що впливають на один *A*-елемент. *A*-елементи з'єднані з реагуючими елементами (*R*-елементами) зв'язками, коефіцієнти підсилення (ω) яких змінюються в процесі навчання.

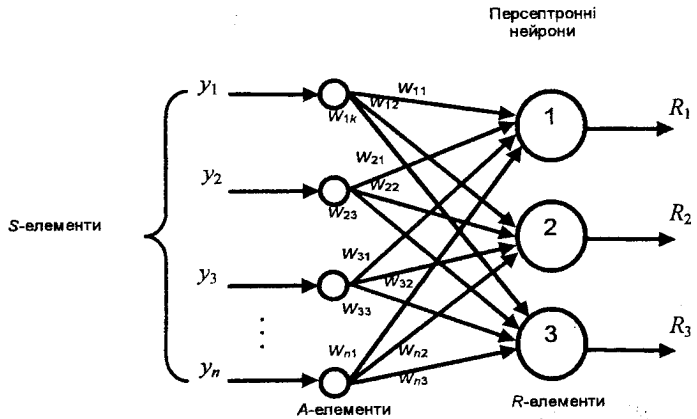


Рис. 5.5. Схема роботи персептрона з декількома входами

Зважені комбінації виходів *R*-елементів становлять реакцію системи, яка вказує на належність об'єкта, що розпізнається, певному образу.

Якщо розпізнаються лише два образи, то в персептроні встановлюється тільки один *R*-елемент, який має дві реакції — позитивну й негативну. Якщо образів більше двох, то для кожного образу встановлюють свій *R*-елемент, а вихід кожного такого елемента становить собою лінійну комбінацію виходів *A*-елементів:

$$R_j = \Theta_j + \sum_{i=1}^{n} w_{ij} x_i ,$$

де R_j — реакція j-го *R*-елемента; x_i — реакція i-го *A*-елемента; w_{ij} — ваговий коефіцієнт зв'язку від i-го *A*-елемента до j-го *R*-елемента; Θ_j — поріг j-го *R*-елемента.

Аналогічно записується рівняння i-го *A*-елемента:

$$x_i = \Theta + \sum_{k=1}^{S} y_n .$$

Тут сигнал y_k може бути неперервним, але найчастіше він набуває лише двох значень: 0 або 1. Сигнали від *S*-елементів подаються на входи *A*-елементів із постійним ваговим коефіцієнтом, який дорівнює одиниці, але кожен *A*-елемент зв'язаний лише з групою випадково вибраних *S*-елементів.

Припустимо, що потрібно навчити персептрон розрізняти два образи V_1 і V_2. Вважатимемо, що в персептроні існує два *R*-елементи, один із яких призначений образу V_1, а інший — образу V_2. Персептрон буде навчений правильно, якщо вихід R_1 перевищує R_2, коли об'єкт, який розпізнається, належить образу V_1, і навпаки. Об'єкти можна розділити на два образи за допомогою лише одного *R*-елемента. Тоді об'єкту образу V_1 повинна відповідати позитивна реакція *R*-елемента, а об'єктам образу V_2 — негативна.

Персептрон *навчається* шляхом висунення навчальної послідовності зображень об'єктів, що належать образам V_1 і V_2. У процесі навчання змінюються вагові коефіцієнти ω *A*-елементів. Зокрема, якщо застосовується система підтримки з корекцією помилок, то насамперед враховується правильність рішення, яке прийняв персептрон. Якщо рішення правильне, то вагові коефіцієнти зв'язків усіх *A*-елементів, які спрацювали і ведуть до *R*-елемента, що видав правильне рішення, збільшується, а вагові коефіцієнти *A*-елементів, які не спрацювали, залишаються незмінними. Можна залишати незмінним ваговий коефіцієнт *A*-елементів, що спрацювали, але зменшувати вагові коефіцієнти тих, що не спрацювали. У деяких випадках вагові коефіцієнти зв'язків, що спрацювали, збільшують, а тих, що не спрацювали, — зменшують. Після процесу навчання персептрон сам, без учителя, починає класифікувати нові об'єкти.

Якщо персептрон діє за описаною схемою і в ньому допускаються лише ті зв'язки, що йдуть від бінарних *S*-елементів до *A*-елементів і від *A*-елементів до єдиного *R*-елемента, то такий персептрон прийнято називати *елементарним α-персептроном*. Зазвичай класифікація задається вчителем. Персептрон повинен виробити в процесі навчання класифікацію, задуману вчителем.

5.8. Нейронна мережа зі зворотним поширенням помилки (*back propagation*)

Серед різних структур НМ однією з найбільш відомих є багатошарова структура, у якій кожний нейрон довільного прошарку зв'язаний з усіма аксонами нейронів попереднього прошарку або, у

випадку першого прошарку, з усіма входами НМ. Такі НМ називаються *повнозв'язними*. Коли в мережі лише один прошарок, алгоритм її навчання вчителем очевидний, тому що правильні вихідні стани нейронів єдиного прошарку наперед відомі, і підлаштування синаптичних зв'язків іде в напрямку, що мінімізує помилку на виході мережі. За цим принципом будується, наприклад, алгоритм навчання одношарового персептрона.

У багатошарових мережах оптимальні вихідні значення нейронів усіх прошарків, крім останнього, як правило, не відомі, і дво- або багатошаровий персептрон уже неможливо навчити, керуючись лише величинами помилок на виходах НМ. Один із варіантів вирішення цієї проблеми — розроблення наборів вихідних сигналів, що відповідають вхідним, для кожного прошарку НМ, що, звичайно, є дуже трудомісткою операцією і не завжди можна зробити.

Другий варіант — динамічне налаштування вагових коефіцієнтів синапсів, у ході якого вибираються, як правило, найбільш слабкі зв'язки й змінюються на малу величину в тий чи інший бік, а зберігаються лише ті зміни, які привели до зменшення помилки на виході всієї мережі. Очевидно, що цей метод, попри свою, на перший погляд, простоту, потребує громіздких рутинних обчислень.

Третій варіант, використаний у НМ зі зворотним поширенням помилки, передбачає поширення сигналів помилки від виходів до входів, у напрямку, зворотному прямому поширенню сигналів у звичайному режимі роботи (рис. 5.6).

Алгоритм навчання мережі.

1. Ініціалізація мережі: вагові коефіцієнти й зміщення мережі набувають малих випадкових значень.

2. Визначення елемента множини, що навчає: (вхід — вихід). Входи (x_1, x_2, ..., x_N) повинні відрізнятися для всіх прикладів множини, що навчає.

3. Обчислення вихідного сигналу:

$$S_{i_m} = \sum_{i_{m-1}}^{N_{m-1}} w_{i_m j_{m-1}} y_{i_{m-1}} - b_{i_m}\ ;\quad y_{i_m} = f(S_{j_m})\ ;$$

$$i_m = 1, 2, ..., N_m; \qquad m = 1, 2, ..., L,$$

де S — вихід суматора; w — ваговий коефіцієнт зв'язку; y — вихід нейрона; b — зміщення; i — номер нейрона; N — кількість нейронів у прошарку; m — номер прошарку; L — кількість прошарків; f — передатна функція.

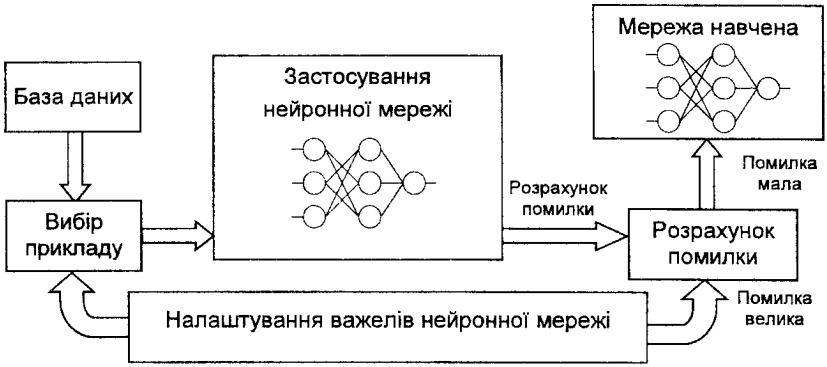


Рис. 5.6. Схема процесу навчання нейронної мережі зі зворотним поширенням помилки

4. Налаштування синаптичниих важелів:

$$w_{ij}(t+1) = w_{ij}(t) + rg_j x'_i,$$

де w_{ij} — ваговий коефіцієнт від нейрона i або від елемента вхідного сигналу i до нейрона j у момент часу t; x'_i — вихід нейрона i; r — швидкість навчання; g_j — значення похибки для нейрона j.

Якщо нейрон із номером j належить до останнього прошарку, то

$$g_j = y_j(1-y_j)(d_j - y_j),$$

де d_j — бажаний вихід нейрона j; y_j — поточний вихід нейрона j.

Якщо нейрон із номером j належить одному із прошарків із першого до передостаннього, тоді

$$g_i = x'_j(1-x')\sum gk^w jk,$$

де k пробігає всі нейрони прошарку з номером на одиницю більше, ніж у того, якому належить нейрон j.

Зовнішні зміщення нейронів b налаштовуються аналогічно.

Галузі застосування таких НМ — переважно розпізнавання образів, прогнозування.

Серед недоліків можна відзначити, що багатокритеріальна задача оптимізації в цьому методі розглядається як набір однокритеріальних задач — на кожній ітерації відбуваються зміни значень параметрів мережі, які поліпшують роботу лише з одним прикладом навчальної вибірки. Такий підхід істотно зменшує швидкість навчання.

5.9. Нейронна мережа Хопфілда

Нейронні мережі Хопфілда і Хеммінга за принципом навчання не підходять ні під навчання «з учителем», ні під навчання «без учителя».

У таких мережах вагові коефіцієнти синапсів розраховуються лише один раз перед початком функціонування мережі на підставі інформації про дані, що обробляються, і все навчання мережі зводиться саме до цього розрахунку. З одного боку, надання апріорної інформації можна розцінювати як допомогу вчителя, але з іншого — мережа фактично просто запам'ятовує зразки до того, як на її вхід надходять реальні дані, і не може змінювати свою поведінку, тому говорити про ланку зворотного зв'язку зі «світом» (учителем) не доводиться.

Із мереж із подібною логікою роботи найбільш відомі мережа Хопфілда й мережа Хеммінга, які зазвичай використовуються для організації асоціативної пам'яті.

Мережа Хопфілда, структурну схему якої зображено на рис. 5.7, складається з єдиного шару нейронів, кількість яких є одночасно кількістю входів і виходів мережі. Кожен нейрон зв'язаний синапсами з усіма іншими нейронами, а також має один вхідний синапс, через який вводиться сигнал. Вихідні сигнали, зазвичай, утворюються на аксонах.

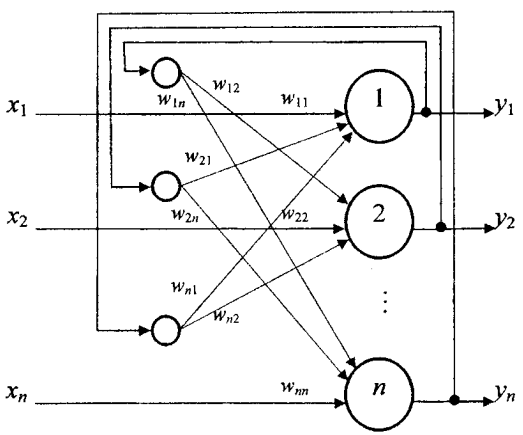


Рис. 5.7. Структурна схема нейронної мережі Хопфілда

Задача, яка розв'язується даною мережею як асоціативна пам'ять, як правило, формулюється так. Відомий деякий набір двійкових сигналів (зображень, звукових оцифровок, інших даних, що описують якісь об'єкта або характеристики процесів), які вважаються зразками. Мережа повинна вміти з довільного неідеального сигналу, поданого на її вхід, виділити («згадати» за частковою інформацією) відповідний зразок (якщо такий є) або «надати висновок» про те, що вхідні дані не відповідають жодному зі зразків.

У загальному випадку, будь-який сигнал може бути описаний вектором

$$\vec{X} = \{ x_i : i = 0, ..., n - 1 \},$$

де n — кількість нейронів у мережі й розмірність вхідних і вихідних векторів.

Кожний елемент x_i дорівнює або +1, або –1. Позначимо вектор, що описує k-й зразок, через X^k, а його компоненти, відповідно, – x_i^k, $k = 0, ..., m - 1$, m — кількість зразків. Коли мережа розпізнає (або «згадає») який-небудь зразок на підставі наданих їй даних, її виходи міститимуть саме його, тобто $\vec{Y} = X^k$, де $\vec{Y}$ — вектор вихідних значень мережі: $\vec{Y} = \{y_i : i = 0, ..., n - 1\}$. У протилежному випадку вихідний вектор не збігається з жодним зразком.

Якщо, наприклад, сигнали становлять собою деякі зображення, то, зобразивши у графічному вигляді дані з виходу мережі, можна буде побачити картинку, яка повністю збігається з однією з тих, що є зразком (у разі успіху), або ж «вільну імпровізацію» мережі (у разі невдачі).

На етапі ініціалізації мережі вагові коефіцієнти синапсів встановлюються так:

$$w_{ij} = \begin{cases} \sum_{k=0}^{m-1} x_i^k x_j^k, & i \neq j; \\ 0, & i = j. \end{cases}$$

тут i та j — індекси, відповідно, передсинаптичного й постсинаптичного нейронів; x_i^k, x_j^k — i-й та j-й елементи вектора k-го зразка.

Іншими словами, створюється матриця вагових коефіцієнтів з діагональними елементами, що дорівнюють нулю (оскільки вони задають автозв'язки, а елементи не зв'язані самі з собою).

Таким чином, вагова матриця, яка відповідає збереженню вектора $\vec{X}$, задається формулою $W = X^T X$.

Алгоритм функціонування мережі, де p — номер ітерації.

1. На входи мережі подається невідомий сигнал. Фактично він вводиться безпосереднім встановленням значень аксонів:

$$y_i(0) = x_i,\ i = 0...n-1,$$

тому позначення на схемі НМ вхідних синапсів у явному вигляді має умовний характер. Нуль у дужці праворуч від y_i означає нульову ітерацію в циклі роботи мережі.

2. Розраховується новий стан нейронів

$$s_j(p+1) = \sum_{i=0}^{n-1} w_{ij} y_i(p),\quad j = 0, \ldots n-1,$$

і нові значення аксонів

$$y_j(p+1) = f\left[s_j(p+1)\right],$$

де f — активаційна функція у вигляді стрибка (рис. 5.8).

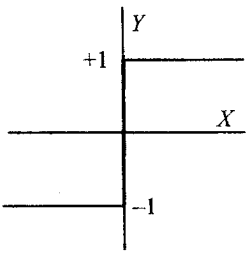


Рис. 5.8. Активаційна функція у вигляді стрибка

3. Перевіряється, чи змінилися вихідні значення аксонів під час останньої ітерації. Якщо так — перехід до пункту 2, інакше (якщо виходи стабілізувалися) — кінець. При цьому вихідний вектор становить собою зразок, що найкраще сполучається із вхідними даними.

Іноді мережа не може розпізнати образ і видає на виході неіснуючий. Це пов'язано з проблемою обмеженості можливостей мережі. Для мережі Хопфілда кількість образів m, що запам'ятовуються, не більше $0{,}15 \cdot n$. Крім того, якщо два образи А і Б дуже схожі, вони, можливо, викликатимуть у мережі перехресні асоціації.

Мережі такого типу застосовують переважно для класифікації та розпізнавання образів. Недоліком є низька швидкість навчання.

5.10. Нейронна мережа Хеммінга

Коли не потрібно, щоб мережа у явному вигляді видавала зразок, тобто досить, скажімо, отримувати номер зразка, асоціативну пам'ять успішно реалізує мережа Хеммінга. Ця мережа характери-

зується, порівняно з мережею Хопфілда, меншими витратами на пам'ять і обсягом обчислень, що стає очевидним із її структури (рис. 5.9).

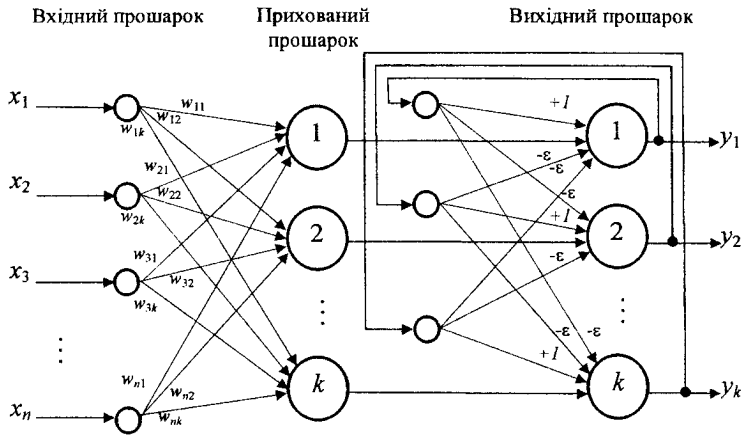


Рис. 5.9. Структурна схема мережі Хеммінга

Мережа складається із двох прошарків. Перший і другий прошарки мають по m нейронів, де m — кількість зразків. Нейрони першого прошарку мають по n синапсів, з'єднаних із входами мережі (які утворюють фіктивний нульовий прошарок). Нейрони другого прошарку зв'язані між собою інгібіторними (негативними зворотними) синаптичними зв'язками. Єдиний синапс із позитивним зворотним зв'язком для кожного нейрона з'єднаний із його ж аксоном.

Ідея роботи мережі полягає в визначенні відстані Хеммінга від зразка, що тестують, до всіх зразків. Відстанню Хеммінга називається кількість біт, що відрізняються, у двох бінарних векторах. Мережа повинна вибрати зразок із мінімальною відстанню Хеммінга до невідомого вхідного сигналу, у результаті чого буде активовано лише один вихід мережі, що відповідає цьому зразку.

На стадії ініціалізації ваговим коефіцієнтам першого прошарку й порогу активаційної функції присвоюються такі значення:

$$w_{ik} = \frac{x_i^k}{2},$$

$$i = 0, ..., n-1, \quad k = 0, ..., m-1, \quad T_k = n/2, \quad k = 0, ..., m-1,$$

тут x_i^k — i-й елемент k-го зразка.

Вагові коефіцієнти гальмівних синапсів у другому прошарку дорівнюють деякій величині $0 < \varepsilon < 1/m$. Синапс нейрона, зв'язаний із його ж аксоном, має ваговий коефіцієнт +1.

Алгоритм функціонування мережі Хеммінга такий:

1. На входи мережі подається невідомий вектор $\vec{X} = \{x_i: i = 0, ..., n - 1\}$, виходячи з якого розраховуються стани нейронів першого прошарку (верхній індекс у дужках вказує номер прошарку):

$$y_j^{(1)} = s_j^{(1)} = \sum_{i=0}^{n-1} w_{ij} x_i + T_j, \quad j = 0, ..., m - 1.$$

Після цього отриманими значеннями ініціалізуються значення аксонів другого прошарку:

$$y_j^{(2)} = y_j^{(1)}, j = 0, ..., m - 1 .$$

2. Обчислити нові стани нейронів другого прошарку:

$$s_j^{(2)}(p+1) = y_j(p) - \varepsilon \sum_{k=0}^{m-1} y_k^{(2)}(p), \quad k \neq j, \quad j = 0...m-1$$

і значення їх аксонів:

$$y_j^{(2)}(p+1) = f\left[s_j^{(2)}(p+1)\right], \; j = 0...m-1 .$$

Активаційна функція f має вигляд порога (рис. 5.10), причому величина F повинна бути достатньо великою, щоб будь-які можливі значення аргументу не приводили до насичення.

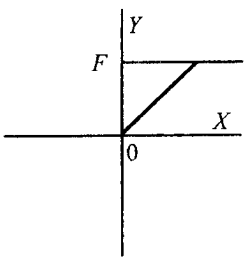


Рис. 5.10. Активаційна функція у вигляді порога

3. Перевірити, чи змінилися виходи нейронів другого шару під час останньої ітерації. Якщо так — перейти до кроку 2. Інакше — кінець.

Із оцінки алгоритму видно, що роль першого прошарку досить умовна: скориставшись один раз на кроці 1 значеннями його вагових коефіцієнтів, мережа більше не звертається до нього, тому перший прошарок може бути взагалі виключений із мережі (замінений на матрицю вагових коефіцієнтів). Застосовують такі мережі для розпізнавання образів. Перевагою є простота та швидкість формування та налаштування вагових коефіцієнтів.

Висновки

1. Ідея НМ народилася в результаті спроб відтворити здатність біологічних нервових систем навчатися і виправляти помилки, моделюючи низькорівневу структуру мозку.

2. Застосування НМ ефективне для задач класифікації образів, кластеризації, прогнозування, апроксимації функцій, оптимізації, створення асоціативної пам'яті.

3. Біологічні нейрони, здатні запам'ятовувати, думати й застосовувати попередній досвід до кожної дії, що відрізняє їх від інших клітин тіла.

4. Штучний нейрон — це пристрій із декількома входами і одним виходом. Кожному входу відповідає ваговий коефіцієнт. У тілі нейрона відбувається зважене підсумовування вхідних збуджень, і далі це значення є аргументом активаційної функції нейрона.

5. Нейрони взаємодіють один з одним за допомогою великої кількості зв'язків і утворюють НМ.

6. Штучна НМ складається переважно з трьох типів прошарків нейронів — вхідного, прихованого та вихідного.

7. Роботу НМ поділяють на навчання та адаптацію. Виокремлюють три види навчання:

– «з учителем», коли НМ має в своєму розпорядженні правильні відповіді;

– «без учителя», коли використовується внутрішня структура даних та кореляція між зразками в навчальній множині для розподілу зразків за категоріями;

– змішане, коли частина вагових коефіцієнтів визначається за допомогою навчання «з учителем», а інша визначається за допомогою самонавчання.

8. За архітектурою зв'язків розрізняють мережі прямого поширення (з односпрямованими послідовними зв'язками) та мережі зворотного поширення (з рекурентними зв'язками).

9. Найпростіша модель НМ — персептрон Розенблатта. Він складається з сенсорних, асоціативних та реагувальних елементів, пов'язаних зв'язками з різними ваговими коефіцієнтами.

10. У нейронних мереж зі зворотним поширенням помилки передбачено поширення сигналів помилки від виходів до входів у напрямку, зворотному прямому поширенню сигналів у звичайному режимі роботи.

11. Мережа Хопфілда складається з єдиного прошарку нейронів, кількість яких є одночасно кількістю входів і виходів мережі. Для мережі Хопфілда кількість образів m, що запам'ятовуються, не більше $0{,}15 \cdot n$.

12. Двошарова мережа Хеммінга, порівняно з мережею Хопфілда, характеризується меншими витратами на пам'ять і обсяг обчислень. Ідея роботи мережі полягає в обчисленні відстані Хеммінга від зразка, що тестують, до всіх зразків.

Вправи

1. На персептронний нейрон j надходить вхідний сигнал від чотирьох інших нейронів, рівні збудження яких дорівнюють 10, –20, 4 і –2. Відповідні вагові коефіцієнти зв'язків цього нейрона дорівнюють 0,8, 0,2, –1,0 і –0,9. Обчисліть вихідне значення нейрона j за умови відсутності порогового значення.

2. Повнозв'язна мережа прямого поширення має десять вхідних вузлів, два приховані прошарки (один із чотирма, а інший із п'ятьма нейронами) і один нейрон у вихідному прошарку. Побудуйте архітектурний граф цієї НМ.

3. Сконструюйте повну рекурентну мережу з п'ятьма нейронами, у якій нейрони не мають зворотних зв'язків самі з собою.

4. Скільки зразків (m) можна викликати з мережі Хопфілда, якщо кожен збережений вектор містить $n = 10$ елементів? Обґрунтуйте відповідь.

5. Визначте вагову матрицю мережі Хопфілда, які відповідають збереженню зразка [1 1 1 –1].

6. Визначте вагові коефіцієнти (матрицю) мережі Хопфілда, які відповідають збереженню вказаних зразків. Візьміть до уваги, що матриці декількох зразків підсумовуються.

[1 –1 1 –1];

[1 –1 –1 1].

7. Визначте вагові коефіцієнти мережі Хопфілда, які відповідають збереженню вказаних зразків.

[–1 1 –1 –1 1 –1 –1 –1].

[–1 –1 1 –1 –1 –1 1 –1].

Запитання та завдання для самоперевірки

1. Які є обмеження при застосуванні НМ у різних галузях дослідження?
2. Які структура і принцип роботи біологічного та штучного нейрона?
3. Які особливості архітектури багатошарових НМ?
4. У чому відмінність навчання НМ за принципом «із учителем» та «без учителя»?
5. Розкрийте принцип роботи персептрона Розенблата.
6. Які структура та алгоритм навчання НМ зі зворотним поширенням помилки?
7. Яка архітектура та принцип роботи НМ Хопфілда?
8. Проаналізуйте структуру та принцип роботи НМ Хеммінга.
9. Який тип навчання використовується у НМ Хопфілда і Хеммінга?
10. Назвіть галузі доцільного використання НМ Хопфілда та Хеммінга. Відповідь обґрунтуйте.

РОЗДІЛ 6

ОНТОЛОГІЧНИЙ ПІДХІД ДО ПОДАННЯ ТА ІНТЕГРАЦІЇ ЗНАНЬ

6.1. Онтологічний підхід до подання та інтеграції знань у розподілених інформаційних середовищах типу Інтернет

Середовища типу Інтернет становлять собою велике інформаційне поле, величезну базу знань, яка не містить семантики, і тому пошук інформації, релевантний запитам користувача, а також інтеграція в рамках конкретної предметної галузі ускладнені. Для забезпечення ефективного пошуку необхідно чітко розуміти семантику документів, поданих у мережі.

Для подання предметних галузей в інформаційних системах використовуються онтології і словники. Онтологічний підхід передбачає використання онтології предметної галузі.

Онтологія — це формальний опис результатів концептуального моделювання предметної галузі, поданий у формі, яка сприймається людиною і комп'ютерною системою. У загальному вигляді формальна модель онтології може бути описана так:

$$O = \{L, C, F, G, H, R, A\},$$

де $L = LC \cup LR$ — *словник онтології*, що містить набір лексичних одиниць (знаків) для понять LC і набір знаків для відношень LR; C — набір *понять* онтології, причому для кожного поняття $c \in C$ в онтології існує принаймні одне твердження. Поняття (або класи) є загальними категоріями, які можуть бути впорядковані ієрархічно. Кожен клас описує групу індивідуальних сутностей, об'єднаних загальною властивістю; F і G — *функції посилань*, такі, що F: $FLC \rightarrow 2C$ і G: $FLR \rightarrow 2R$.

Тобто F і G пов'язують набори лексичних одиниць $\{L_j\} \subset L$ із наборами понять і відношень, на які вони відповідно посилаються в цій онтології. При цьому одна лексична одиниця може посилатися на декілька понять або відношень, і одне поняття або відношення може посилатися на декілька лексичних одиниць.

Інверсіями функцій посилань є $F - 1$ і $G - 1$; H — фіксує *таксономічний характер відношень* (зв'язків), за якого поняття онтології пов'язані нерефлексивними, ациклічними, транзитивними відношеннями $H \subset C \times C$. Вираз $H(C_1, C_2)$ означає, що поняття C_1 є підпоняттям C_2; R — позначає *бінарний характер відношень* між поняттями онтології, що фіксують пари галузі застосування (*domain*)/галузі значень (*range*), тобто пари (DR) з D, $R \in C$; A — набір *аксіом* онтології. Аксіоми задають умови співставлення категорій і відношень, вони виражають очевидні твердження, які пов'язують поняття і відношення. Аксіомою можна вважати твердження, що вводиться в онтологію у готовому вигляді, з якого можна виводити інші твердження. Наприклад, аксіомою є твердження: «Якщо X смертний, то X коли-небудь помре». Аксіоми можуть являти собою деякі обмеження. Наприклад, понятійні обмеження вказують на те, який тип понять може виражати таке відношення. Тобто властивість «Колір» може виражатися лише поняттями категорії «Колір».

Запозичуючи принципи об'єктно-орієнтованого підходу, онтологія містить у собі класи та їх екземпляри, яким притаманні деякі властивості (на властивості можуть накладатися логічні обмеження). Основу кожної онтології становить множина термінів, поданих у ній. Крім того, будь-яка онтологія припускає наявність пов'язаних один із одним компонентів. Ними є *таксономії* термінів, описів сенсу термінів, а також правил їх використання та обробки. Онтологія використовується для структурування інформації.

6.2. Класифікація онтологій

Розрізняють три основні принципи класифікації онтологій:

- за ступенем формальності;
- за метою створення;
- за наповненням, вмістом.

6.2.1. Класифікація за ступенем формальності

Зазвичай люди й комп'ютерні агенти (програми) мають певне уявлення про значення термінів. Онтології можуть бути застосовані, щоб надати конкретну специфікацію імен термінів і значень термінів.

Спектр онтологій за ступенем формальності подання, використання тих чи інших синтаксичних конструкцій можна розділити на два класи:

1. Системи, що надають «людино-зрозумілий» опис:

– *каталог на основі ID* — контрольований словник, тобто кінцевий список термінів на основі ідентифікаторів. Каталоги надають точну (небагатозначну) інтерпретацію термінів. Наприклад, щоразу, посилаючись на термін «машина», ми використовуватимемо одне й те саме значення (що, відповідає деякому *ID* у словнику), незалежно від того, про що йдеться в контексті: про «пральну машину», «автомобіль» або «державну машину»;

– *словник термінів* (глосарій) — список термінів з їх значеннями природною мовою. Це дає більше інформації, оскільки люди можуть прочитати такий коментар і зрозуміти сенс терміна. Інтерпретації термінів можуть бути багатозначними. Глосарії непридатні для автоматичної обробки програмними агентами, але можна, як і раніше, присвоїти термінам *ID*;

– *тезауруси* несуть додаткову семантику, визначаючи зв'язки між термінами. Відношення: синонімія, ієрархічне відношення і асоціація. Ранні ієрархії термінів, що з'явилися в мережі Інтернет, визначали терміни через операції узагальнення і уточнення. *Yahoo*, наприклад, ввела невелику кількість категорій верхнього рівня, таких як «предмети одягу». Потім «сукня» визначалась як вид (жіночого) одягу. Явна ієрархія *Yahoo* не відповідала — формальним властивостям ієрархічного відношення підклас–клас. У таких ієрархіях може трапитися ситуація, у якій екземпляр класу-нащадка не є екземпляром класу-предка. Наприклад, загальна категорія «предмети одягу» має підкатегорію «жіночі» (яка повинна була б більш точно називатися «жіночі предмети одягу»), а ця категорія, своєю чергою, включає підкатегорії «аксесуари» та «сукня». Ясно, що аксесуари, наприклад «брошки», не є предметами одягу. Тут не виконується важлива властивість відносин підклас–клас — транзитивність.

2. Системи з «машинно-зрозумілим» описом:

– *формальні таксономії* — включає точне визначення відношення підклас–клас. Зберігається транзитивність відношення: якщо *A* є підкласом класу *B*, то кожен підклас класу *A* також є підкласом класу *B*. Сувора ієрархія класів необхідна для процедури логічного виведення;

– *формальні екземпляри* — відношення екземпляр–клас, для яких виконується «успадкування» вздовж відношення: якщо A є підкласом класу B, то кожен екземпляр класу A також є екземпляром класу B. Тому в наведеному вище прикладі «брошки» не можуть розміщуватися в ієрархії нижче «предмета одягу», навіть у підкатегорії «жіночі предмети одягу», або стати екземпляром цієї категорії;

– *властивості на основі фреймів* — слоти. Якщо вони визначені на верхніх рівнях ієрархії, то успадковуються підкласами. Так, у споживчій ієрархії клас «продукт» може мати властивість «ціна», яку отримають усі його підкласи;

– онтології, що включають *обмеження на область значень* властивостей. Значення властивостей беруться з деякої визначеної множини (цілі числа, символьні константи) або з підмножини концептів онтології (множини екземплярів цього класу, множини класів). Можна ввести додаткові обмеження на те, що може заповнювати властивість. Наприклад, для властивості «зроблений з» класу «предмет одягу» значення можуть бути обмежені екземплярами класу «матеріал». Легко побачити проблеми, які можуть виникнути за використання несуворої таксономії. Якщо «парфуми» — нащадок класу «предмет одягу», то він успадкує властивість «зроблений з» разом із обмеженням на його значення («матеріал»);

– *диз'юнктивні класи* — онтології, що дозволяють визначати два і більше класів диз'юнктивними (такими, що не перетинаються). Це означає, що у даних класів не існує спільних екземплярів;

– *логічні обмеження* — мови опису онтологій дозволяють робити довільні логічні твердження про концепти — аксіоми.

6.2.2. Класифікація за метою створення

За метою створення розрізняють чотири рівні онтологій.

1. *Подання* — концептуалізація формалізмів подання знань. Мета — опис предметної галузі, створення умов для специфікації інших онтологій більш низьких рівнів.

2. *Верхнього рівня* — повторно використовувана онтологія в різних предметних галузях. Призначення — створення єдиної «правильної» онтології, що фіксує знання, загальні для декількох предметних галузей, і в багаторазовому використанні цієї онтології. Але в цілому спроби створити онтологію верхнього рівня на всі випадки життя поки не привели до очікуваних результатів. Багато онто-

логій верхнього рівня схожі одна на одну. Вони містять одні й ті самі концепти: сутність, явище, процес, об'єкт, роль, простір, час, матерія, подія, дія тощо.

3. *Предметної галузі* (онтологія домену) — повторно використовувана в одній предметній галузі. Призначення подібне до призначенням онтології верхнього рівня, але сфера застосування обмежена однією предметною галуззю (так званим *доменом*).

4. *Прикладна* — неможливо використовувати повторно. Призначення — опис концептуальної моделі конкретного завдання або програми. Прикладні онтології описують концепти, які залежать як від онтології задач, так і від онтології предметної галузі. Прикладом може бути онтологія для автомобілів, будівельних матеріалів, обчислювальної техніки. Такі онтології містять найбільш специфічну інформацію.

6.2.3. Класифікація онтологій за наповненням

Ця класифікація схожа на попередню, але акцент зміщується на реальний вміст онтології, а не на абстрактну мету.

1. *Загальні онтології* — описують найбільш загальні концепти (простір, час, матерія, об'єкт, подія, дія тощо), які не залежать від конкретної проблеми чи галузі. Сюди входять онтології подання і верхнього рівня.

2. *Онтології задач* — використовується конкретною прикладною програмою і містить терміни, які застосовуються при розробці програмного забезпечення, що виконує конкретне завдання. Вона відображає специфіку застосування, але може також містити деякі загальні терміни (наприклад, у графічному редакторі будуть і специфічні терміни — палітра, тип заливки, накладення прошарків тощо, і загальні — зберегти і завантажити файл). Завдання, яким може бути присвячена онтологія, найрізноманітніші: складання розкладу, визначення цілей, діагностика, продаж, розробка програмного забезпечення, побудова класифікації. При цьому онтологія задач використовує спеціалізацію термінів, поданих в онтологіях верхнього рівня (загальних онтологіях).

3. *Предметна онтологія* (або онтологія предметів) описує реальні предмети, що беруть участь у будь-якій діяльності (виробництві). Наприклад, це може бути онтологія всіх частин і компонентів літаків певної марки (наприклад, *Boeing*) і відомості про їх постачальників, характеристики, способи з'єднання один із одним тощо.

6.3. Методи побудови онтологій

При формуванні онтологій можуть залучатися фахівці різних галузей, і для кожної галузі є свої базові методи роботи. Філософи використовують абстракцію і комбінування властивостей, когнітивісти схильні покладатися на інтуїтивні відмінності, фахівці в комп'ютерній галузі оперують логічними теоріями і використовують структури, побудовані на умовиводах, а лінгвісти приділяють велику увагу описам міжмовних відповідників. Загальним же методом для всіх є використання класифікацій.

Варто згадати, що побудова онтології не завжди очевидна, не завжди легко зібрати поняття, виділити диференціальні ознаки. Існує декілька варіантів дій, що залежать від конкретних завдань і вихідного матеріалу.

1. *Безпосередній збір* елементів для онтології. За такого підходу спочатку збирають і класифікують поняття, підбирають слова, потім проводять відповідність між поняттями і лексиконом. Проблемою є близький, але не ідентичний перетин значень у словах різних мов.

2. *Використання мікротеорій.* За такого підходу спершу треба зрозуміти явище, потім формуються примітиви теорії, елементи лексикону визначаються з точки зору примітивів, після цього лексикон ускладнюється. Проблема цього методу полягає у виборі мікротеорії, а також у необхідності окремої теорії для кожного комплексу значень.

В основі онтології повинні лежати поняття, але виникає запитання: як при побудові визначити поняття, ґрунтуючись на словах? Зазвичай сполучною ланкою стає категорія значення (лексико-семантичного варіанта). Слова існують у рамках однієї мови, значення ж незалежні від конкретної мови.

Тоді використовують формальну процедуру виявлення понять на базі сукупності слів. Алгоритм переходу від слів до значень, а потім до понять викладено нижче.

1. *Ініціалізація.* Для певного слова необхідно зібрати декілька десятків речень, що його містять. Підібрати визначення з різних словників.

2. Розподілити значення слова в попередні, приблизно схожі групи.

3. *Диференціація.* Почати будувати дерево, розташувавши всі групи в корені.

4. Розглядаючи всі групи, визначити групу, найбільш відмінну від інших:

- якщо можна знайти одну групу, що чітко виділяється, потрібно виписати її найбільш яскраву відмінність у явній формі — вона слугуватиме відмінною ознакою і буде формалізована у вигляді аксіоми.
- якщо відмінності, за якими можна далі поділити групу, не виявляються, перейти на іншу гілку.
- якщо буде виявлено декілька відмінностей, що дозволяють розділити групу декількома рівнозначними способами, роботу з цією гілкою також слід припинити і перейти на іншу гілку.

5. У структурі дерева створити дві нові гілки та розташувати нову групу під однією гілкою, а решту — під іншою.

6. Повторити дії з кроку 4, досліджуючи окремо групу/групи під кожною гілкою.

7. *Формування понять.* Коли розгалуження закінчується, кінцевим результатом є дерево дедалі більш дрібних відмінних ознак, які в явному вигляді перераховані на кожному рівні дерева. Кожен лист стає окремим поняттям, яке в цій задачі (додатку, предметній галузі) далі не ділиться. Кожна відмінність має бути формалізована у вигляді аксіоми, яка спрацьовує для гілки, з якою асоціюється.

8. *Додавання понять в онтологію.* Починаючи з вершини, слід пройти кожен вузол із розгалуженням. Перевірити, чи мають створена гілка і гілка, що додається, приблизно однакове значення:

- якщо так, об'єднати їх в онтології у відповідному вузлі і закінчити проходження цієї гілки;
- якщо ні, розділити дерево і повторити крок 8 до кінця кожної гілки.

Зазвичай понять виявляється менше, ніж значень-змістів.

Однак існують й інші джерела знаходження нових понять. Для цього можна використовувати вже існуючі онтології і різні списки, словники і тезауруси, а також автоматичне виявлення понять шляхом кластеризації слів.

Основоположні правила розробки онтології:

1.Не існує єдиного правильного способу моделювання предметної галузі — завжди існують життєздатні альтернативи. Кращий розв'язок майже завжди залежить від передбачуваного додатку та очікуваних розширень.

2. Розробка онтології — це обов'язково ітеративний процес. Під ітеративним процесом розуміють неодноразовий прохід по онтології для її уточнення, тобто на початковому етапі будують чорновий варіант. Потім перевіряють і уточнюють складену онтологію, додаючи деталі, частково або навіть повністю переглядаючи початкову онтологію.

3. Елементи онтології повинні бути близькі до об'єктів (фізичних або логічних) і відношень у певній предметній галузі. Найімовірніше, вони відповідають іменникам (об'єкти) або дієсловам (відношення) у реченнях, які описують предметну галузь.

Вимоги до онтології можна визначити так:

- *чіткість:* онтологія має бути чіткою і легко передавати зміст. Повинна бути об'єктивною.
- *послідовність*: у ній мають міститися твердження, які не суперечать одне одному, ієрархії понять, відношенням, що їх зв'язують, екземплярам.
- *можливість розширення*: наявність можливості введення нових елементів без перегляду інших елементів.
- *мінімальна ступінь спеціалізації*: небажаність повного підпорядкування онтології конкретній задачі, що може ускладнити її подальше використання в інших задачах.

Цей список вимог до онтологій не є вичерпним, але він може допомогти при створенні структури онтології.

6.3.1. Автоматичні методи побудови онтологій

Існує низка методів розширення онтологій, які специфічні для онтологій різних зон. Для розширення верхньої, найбільш загальної зони, необхідне детальне теоретизування, після чого можна визначати поняття і аксіоми.

Для онтологій середньої зони, які відрізняються великою кількістю понять, поняття збирають автоматично за допомогою кластеризації. У процесі обробки великої кількості інформації поняття збирають і розбивають їх на класи на підставі загальних характеристик.

Існують методи збільшення точності вилучення семантично пов'язаних сімей понять. За такого аналізу згодом можна також встановлювати перехресні посилання всередині онтології.

Однак для онтології важливо знати не лише те, що поняття взаємопов'язані, але й те, як саме вони взаємопов'язані. Для вияв-

лення таких відношень між поняттями можуть бути використані автоматичні методи перегляду й аналізу різних текстів, наприклад, можна витягати дану інформацію зі словникових визначень. Це зумовлено тим, що існує обмежений набір фразових моделей, що вводять визначальне, за характером яких можна сформулювати тип зв'язків між поняттями і ввести ці дані в онтологію. Таким чином, можна вибрати словникові статті, розібрати їх, виявити семантичні тлумачення для слів і початкових конструкцій визначень.

Виявлення ж аксіоматичних знань, правил може бути вироблено на підставі Інтернету. Окремі приклади збирають шляхом вивчення великої бази ресурсів, однак спочатку формулюють параметри, що вказують, які саме екземпляри потрібні. Це можна пояснити на конкретному прикладі: якщо потрібно зібрати назви столиць, то можна задати умови пошуку з моделлю «*X* — столиця ...». При цьому можна використовувати не одну таку модель, а декілька. Таким чином, знаходження окремих екземплярів зводитиметься до аналізу текстів для виявлення прикладів із цими структурами.

Такі шаблони можна формуватися як вручну, так і автоматично за допомогою програм, які самонавчаються.

Поряд із пошуком окремих екземплярів важливим є встановлення різних зв'язків між елементами онтології, класами. Можна розділити всі методи отримання відношень між елементами онтології на два класи: підходи, що ґрунтується на використанні шаблонів, і методи, які використовують кластеризацію. При застовуванні методів на основі шаблонів дослідники шукають мовні моделі, які вказують на будь-який тип відношень між класами. Здебільшого шукають родо-видові відношення і відношення «частина-ціле».

За наявності базового переліку категорій можна нарощувати їх кількість, звертаючись до корпусу текстів. У корпусі виділяють кластери близьких за значенням елементів, потім оцінюють тісноту зв'язку між елементами, далі кожному кластеру приписують ім'я, яке асоціюється з категорією, що включається в онтологію. Таким чином, онтологія поповнюється новими елементами автоматично і також автоматично визначається місце нового елемента в ієрархії категорій.

У онтології є не лише класи й окремі екземпляри, але й аксіоми. Вони роблять великий внесок в удосконалення аналізу інформації, дають можливість комп'ютеру доповнити текст додатковими знаннями, які для нас здаються тривіальними. Аксіоми повідомляють

комп'ютер, наприклад, про те, що два висловлювання мають один і той самий зміст або ж один факт тягне за собою наявність іншого. Наприклад, метод автоматичного вилучення аксіом із текстів і їх подальша класифікація і перевірка на релевантність. Основною проблемою автоматичного виявлення аксіом є побудова помилкових припущень або занадто загальних, а також проблема визначення симетричності аксіом. За основу гіпотези беруть схожість значень при схожості контекстів зустрічальності, доповнюючи їх своїми обмеженнями. Тут ключовим стає визначення методів обчислення близькості контекстів і близькості понять. Така методика виявлення аксіом дозволяє витягувати релевантні аксіоми з маленьким відсотком помилок.

Здебільшого проблемою автоматичного вилучення стає велика кількість «завад», які треба ефективно відсіювати. У зв'язку з цим іноді поряд із автоматичними методами використовують подальшу ручну обробку отриманого матеріалу для одержання даних більшої точності.

Далі наведено загальні вимоги до систем автоматичного одержання даних для онтологій:

- мінімальний контроль — зведення до мінімуму або виключення взагалі участі людини;
- універсальність — застосовність до різних джерел, залежно від їх розміру, галузі знання тощо;
- точність — витягнута інформація повинна містити якомога менше помилок.

6.4. Сфери застосування онтологій

Переважні сфери застосування онтологій та способи використання онтологій:

- для спільного використання людьми або програмними агентами загального розуміння структури інформації;
- для можливості повторного використання знань у предметній галузі;
- для можливості зробити припущення у предметній галузі явними;
- для відокремлення знань у предметній галузі від оперативних знань;
- для аналізу знань у предметній галузі.

Побудова онтології часто не є сама по собі кінцевою метою, зазвичай онтології далі використовуються іншими програмами для вирішення практичних цілей. На сучасному етапі розвитку науки існує низка задач, де застосування онтологій може дати хороші результати. Однак зараз лише мала кількість додатків природною мовою включає у себе онтологічні бази, звідки черпаються знання про навколишній світ.

Онтології ефективно використовуються:

- у машинному перекладі;
- у системах «запитання-відповідь»;
- при інформаційному пошуку;
- у системах добування знань;
- у загальних системах ведення діалогу між комп'ютером і людиною;
- у системах розуміння мови (автоматичне реферування тексту, рубрикація тощо).

6.5. Лексичні онтології для обробки текстів природною мовою

Щоб застосувати онтологію для автоматичної обробки текстів, зокрема для вирішення завдань інформаційного пошуку, необхідно поняттям онтології зіставити набір мовних виразів (слів і словосполучень), якими поняття можуть виражатися в тексті.

Процедура зіставлення понять онтологій і мовних виразів може бути здійснена різними способами.

1. *Логічна класифікація* — створення онтології шляхом логічного аналізу, «згори-вниз». Імена відображають ознаки, закладені в основу класифікації. Недоліками є те, що одне слово може відповідати декільком поняттям залежно від контексту. Крім того, спостерігається значне ускладнення при розробці докладних онтологій для реальних програм.

2. *Лексичні відношення* — встановлення відповідностей між ієрархічними лексичними ресурсами і деякою онтологією, тобто лексичні відношення між значеннями слів, подані у вигляді окремих одиниць в ієрархічній мережі — сінсетів. Головна характеристика лінгвістичних онтологій — прив'язка до значень мовних виразів (слів, іменних груп тощо). Лінгвістичні онтології охоплюють більшість слів мови і разом з тим мають онтологічну структуру, яка

виявляється у відношеннях між поняттями. Тому лінгвістичні онтології можуть розглядатися як особливий вид лексичної бази даних і особливий тип онтології.

3. *Об'єднання систем понять і лексичних значень* дозволяє в створюваному ресурсі розподілити ці одиниці й докладно описати їх взаємозв'язки.

6.5.1. Автоматичне реферування та отримання інформації у середовищах типу Інтернет

Одним із важливих завдань онтології є автоматичне реферування інформації у середовищах типу Інтернет. При знаходженні інформації, наприклад, за ключовими словами або за допомогою інших більш складних механізмів пошуку, система отримання інформації має отримати на вхід цей текст і реферувати його з урахуванням заданої наперед предметної галузі.

Система має знайти корисну інформацію, пов'язану із зазначеною предметною галуззю, і закодувати її у вигляді, зручному для кінцевого користувача або збереження у структурованій базі даних.

На відміну від системи «глибокого» розуміння природної мови, система отримання інформації проглядає текст у пошуку відповідних розділів і концентрує свою увагу лише на обробці цих розділів. Наприклад, система отримання інформації може реферувати інформацію з пропозицій роботодавців у галузі комп'ютерних наук.

Приклад. Оголошення про вакансію:

«Факультет інформатики та обчислюваної техніки НТУУ «КПІ»... проводить конкурс на заміщення двох посад доцента. Необхідні спеціалісти в таких галузях: програмне забезпечення, в тому числі аналіз, проектування та засоби розробки...; системи, включаючи архітектуру, компілятори, мережі...

Кандидати повинні мати ступінь канд. техн. наук за спеціальністю...

На факультеті виконують міжнародні проекти в галузі адаптивних обчислень, штучного інтелекту...».

Система реферування інформації може отримувати відомості за допомогою подібних речень і заповнювати шаблон певної структури.

Приклад частково заповненого шаблону.

Організація: факультет інформатики та обчислюваної техніки НТУУ «КПІ».

Місто: Київ.

Опис вакансії: посада доцента.

Необхідна кваліфікація: ступінь кандидата технічних наук за спеціальністю... .

Необхідні навички: вміння використовувати програмне забезпечення та засоби розробки, у тому числі для аналізу та проектування... .

Досвід практичної діяльності: ...

Відомості про організацію: (текст додається).

У перших системах обробки даних природною мовою для отримання інформації використовувалися різні підходи, від традиційних засобів, які виконували детальний семантичний аналіз та забезпечували повний синтаксичний опис кожного речення, до систем, які використовували методи порівняння ключових слів, що не містять у складі лінгвістичний аналіз та БЗ. Більш сучасним вважається такий метод отримання інформації.

1. *Текст*: факультет інформатики та обчислювальної техніки НТУУ «КПІ»... проводить конкурс на заміщення двох посад доцента. Необхідні спеціалісти в таких галузях...
2. *Розмітка та тегування*: факультет/*noun* інформатики/*adj* та/*adj* обчислювальної/*adj* техніки/*noun*...
3. *Аналіз речення*: факультет/*subj* проводить/*verb* конкурс/*obj*...
4. *Отримання*: Організація: факультет інформатики та обчислювальної техніки
   Опис вакансії: посада доцента
5. *Об'єднання*: НТУУ «КПІ»=КПІ...
6. *Генерація шаблону*: результат наведено вище.

Хоча деталі архітектурної реалізації системи отримання інформації можуть розрізнятися залежно від додатку, послідовність виконання основних функцій при цьому не змінюється.

Спочатку кожне речення, знайдене в Інтернеті, підлягає розмітці та тегуванню.

Для цього можна використовувати стохастичний підхід. Потім аналізується речення, де виконується граматичний розбір та виділяються дієслівні групи, іменники та інші граматичні конструкції. Після цього на етапі отримання інформації визначаються семантичні сутності, що відповідають цій предметній галузі.

У прикладі на цьому етапі визначається назва організації, вакансії, вимоги до кандидата тощо.

Стадія отримання інформації — це перший етап процесу, який повністю визначається специфікою предметної галузі. На цій стадії система виявляє специфічні відношення між відповідними компонентами тексту. У прикладі в ролі організації виступає факультет інформатики та обчислювальної техніки, а його місцезнаходженням є університет «КПІ». На стадії об'єднання будується список синонімів та визначаються дозволені анафори. Зокрема, символи НТУУ «КПІ» та КПІ визначаються як синоніми, а в процесі резолюції анафор факультет інформатики та обчислювальної техніки з першого речення зв'язується зі словом «факультет» у наступному тексті.

На етапі об'єднання застосовується виведення на рівні міркувань. Рівень міркувань робить внесок у генерацію шаблону. У процесі такого виведення визначається кількість різних відношень у тексті, фрагменти інформації зв'язуються з полями шаблону, а потім будується і сам шаблон.

Серед недоліків сучасних систем отримання інформації слід відзначити такі. По-перше, це точність та робастність систем. Помилки в процесі отримання інформації пов'язані з недостатнім розумінням вихідного тексту. По-друге, побудова систем отримання інформації для нової предметної галузі може виявитися достатньо складним та ресурсомістким процесом. Ці проблеми пов'язані з орієнтованою на предметну галузь природою задач отримання інформації. Процес отримання інформації можна покращити шляхом налаштування лінгвістичних джерел знань на конкретну предметну галузь, однак модифікація лінгвістичних знань вручну — досить складний та трудомісткий процес.

Висновки

1. Середовища типу Інтернет становлять собою велике інформаційне поле, величезну БЗ, яка не містить семантики. Тому пошук інформації, релевантний запитам користувача, а також інтеграція у рамках конкретної предметної галузі ускладнені.

2. Для подання предметних галузей в інформаційних системах використовуються онтології і словники.

3. Онтологія — це формальний опис результатів концептуального моделювання предметної галузі, поданий у формі, яка сприймається людиною і комп'ютерною системою.

4. Базуючись на принципі об'єктно-орієнтованого підходу, онтологія містить у собі класи та їх екземпляри, яким притаманні деякі властивості.

5. Основу кожної онтології формує множина термінів, поданих у ній. Крім того, будь-яка онтологія припускає наявність пов'язаних один із одним компонентів.

6. Розрізняють три основні принципи класифікації онтологій:

– за ступенем формальності. До цієї групи належать каталог на основі *ID*, словник термінів, тезауруси, формальні таксономії, формальні екземпляри, слоти, онтології, що включають обмеження на галузь значень властивостей, диз'юнктивні класи та логічні обмеження;

– за метою створення розрізняють онтології верхнього рівня, подання, предметної галузі та прикладні;

– за наповненням, вмістом розрізняють загальні онтології, онтології задач та предметні онтології.

7. Щоб застосувати онтологію для автоматичної обробки текстів, зокрема для вирішення завдань інформаційного пошуку, необхідно поняттям онтології зіставити набір мовних виразів (слів і словосполучень), якими поняття можуть виражатися в тексті.

8. Процедура зіставлення понять онтологій і мовних виразів здійснюється за допомогою логічної класифікації, лексичних відношень та об'єднання системи понять і лексичних значень.

9. Побудова онтології не завжди очевидна, складно зібрати поняття, виділити диференціальні ознаки. Основні вимоги до онтології включають чіткість, послідовність, можливість розширення та мінімальну ступінь спеціалізації.

10. Основоположні правила розробки онтології свідчать про те, що не існує єдиного правильного способу моделювання предметної галузі, розробка онтології — це обов'язково ітеративний процес, елементи онтології повинні бути близькі до об'єктів і відношень у певній предметній галузі.

11. Існує низка методів розширення онтологій, які специфічні для онтологій різних зон. Для розширення верхньої, найбільш загальної зони, необхідне детальне теоретизування, після чого можна приступати до побудови понять і аксіом. Для онтологій середньої зони, які відрізняються великою кількістю понять, поняття можна збирати автоматично за допомогою кластеризації. У процесі обробки великої кількості інформації збирають поняття і розбивають їх за класами на підставі загальних характеристик.

Вправи

1. Для обраної предметної галузі розробіть онтологію в термінах: ключові сутності і відношення, предикати предметної галузі, операції в предметній галузі.

2. Наведіть приклад отримання інформації аналогічно наведеному у п. 6.5.1.

3. Проаналізуйте (на прикладі) основні етапи автоматичного реферування інформації.

Запитання та завдання для самоперевірки

1. Які компоненти входять до формальної моделі онтології?
2. На які два класи розділяють онтології за ступенем формальності?
3. Системи на основі яких онтологій можуть вважатися такими, що мають «людино-зрозумілий» інтерфейс?
4. Системи на основі яких онтологій мають «машино-зрозумілий» опис?
5. Як класифікують онтології за метою створення?
6. Що таке онтології верхнього рівня?
7. Які особливості прикладних онтологій?
8. Наведіть класифікацію онтологій за наповненням.
9. Чим відрізняються загальні онтології від предметних та онтологій задач?
10. Як можна застосовувати онтологію для автоматичної обробки текстів?
11. Що включає в себе формальна процедура виявлення понять на базі сукупності слів?
12. Які основоположні правила розробки онтології?
13. Які основні вимоги до онтології?
14. Назвіть сфери ефективного застосування онтологій.
15. Які методи використовуються для розширення онтологій?

РОЗДІЛ 7

СУЧАСНІ ПРОГРАМНІ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ СТВОРЕННЯ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ

7.1. Мови функціонального та логічного програмування

Метою логічного та функціонального програмування є виведення розв'язків, і вони тісно пов'язані із завданнями, які розв'язуються в СШІ.

Процедурна програма складається з послідовності операторів та речень, які керують послідовністю їх виконання. Основою такого програмування є дії над якоюсь змінною і збереження нового значення за допомогою оператора присвоювання. Так відбувається доти, доки не буде отримано бажане остаточне значення.

Функціональна програма складається з сукупності визначень функцій. Функції, своєю чергою, становлять собою виклики інших функцій і речень, які керують послідовністю викликів. Кожен виклик повертає деяке значення і функцію, яка його викликала, обчислення якої після цього триває. Цей процес повторюється доти, доки функція, що запустила процес, не поверне результат користувачеві.

У *логічних мовах програмування* для розв'язання задачі досить опису структури та умов цього завдання. Оскільки послідовність та спосіб виконання програми не фіксується, як при описі алгоритму, програми можуть працювати в обох напрямках, тобто програма може як на підставі вихідних даних обчислити результати, так і за результатами — вихідні дані.

7.2. Lisp, Allegro Common Lisp, CLOS, CLIPS, JESS

Lisp (LISP, від англ. LISt *Processing language* — «мова обробки списків») — сімейство мов програмування, програми та дані у яких подані у вигляді систем лінійних списків символів. *Lisp* є мовою низького рівня, її можна розглядати як асемблер, орієнтований на роботу зі списковими структурами.

Мова *Lisp* розроблена в Стенфорді під керівництвом Дж. Маккарті на початку 60-х рр. XX ст. Основні принципи: використання єдиного спискового подання для програм і даних; застосування виразів для визначення функцій; дужковий синтаксис мови.

Традиційна мова *Lisp* має динамічну систему типів. Мова є функціональною, але пізнішим версіям притаманні також риси імперативності. Мова має повноцінні засоби символьної обробки, тому можна реалізувати об'єктно-орієнтованість. Прикладом такої реалізації є платформа *CLOS*.

Однією з базових ідей мови *Lisp* є подання кожного символу як вузла багатокоординаційно символьної мережі; при цьому координати, властивості, рівні мережі записані в так званих слотах символу. Основні слоти:

– ім'я символу;

– функціональний слот;

– слот-значення;

– розширюваний список властивостей (можна вільно розширити слотову систему зручним для розв'язання задачі способом).

Завдяки швидкості й легкості, з якою можна писати, виконувати і видозмінювати програми, мова *Lisp* сьогодні в США залишається основною мовою програмування в галузі ШІ.

Як більшість популярних мов, *Lisp* має численні діалекти: *Common Lisp*, *Mac Lisp*, *Inter Lisp*, *Standard Lisp* тощо. Відмінності між ними не принципові і зводяться до різниці у наборі вбудованих функцій та формі запису програм.

Common Lisp (скор. *CL*) — діалект мови *Lisp*, стандартизований *ANSI*. Був розроблений для об'єднання розрізнених на той момент діалектів *Lisp*. Доступно декілька реалізацій *Common Lisp* як комерційних, так і вільно розповсюджуваних. Діалект підтримує комбінацію процедурного, функціонального та об'єктно-орієнтованого програмування. Включає в себе *CLOS*, систему *Lisp*-макросів, що дозволяє вводити в мову нові синтаксичні конструкції, використовувати техніки метапрограмування і узагальненого програмування.

Allegro Common Lisp — комерційна реалізація мови програмування *Common Lisp*. До її складу входить також кросплатформенне інтегроване середовище розробки і налагоджувач. *Allegro Common Lisp* включається в себе всю функціональність *ANSI Common Lisp*, а

також деякі розширення, такі як *OpenGL*-інтерфейс, розгалуження, *CLOS*-потоки, *CLOS MOP*, юнікод, *SSL*-потоки і реалізацію деяких *TCP*-протоколів.

CLOS (*Common Lisp Object System* — «об'єктна система *Common Lisp*») — система об'єктно-орієнтованого програмування, що є частиною *Common Lisp* — стандарту мови *Lisp*. Крім того, її вбудовують в інші діалекти, такі як *EuLisp* або *Emacs Lisp*. Спочатку запропонована як доповнення, *CLOS* була прийнята як частина стандарту *ANSI CommonLisp*.

CLOS має такі особливості:

– множинна диспетчеризація (тобто викликаний метод визначається усіма аргументами, а не лише першим), або «мультиметоди»;

– методи не визначаються всередині класів. Вони концептуально групуються в «узагальнені функції»;

– не забезпечує приховування, яке створюється іншою частиною *Common Lisp* — пакетами.

– наслідування може приводити до того, що методи суперкласів комбінуються різними способами на вибір програміста, а не лише простим перевизначенням;

– є динамічним, тобто не лише вміст, але й структура об'єктів може змінюватися під час роботи програми. *CLOS* підтримує зміну структури класу на льоту (навіть якщо екземпляри цього класу вже існують), так само, як і зміну класу цього примірника за допомогою методу *CHANGE-CLASS*;

– множинне наслідування.

CLIPS (*C Language Integrated Production System*) — програмне середовище для розробки ЕС. Перші версії розроблялися з 1984 р.

CLIPS є продукційної системою. Таке подання близьке до людського мислення і відрізняється від програм, написаних традиційними алгоритмічними мовами, де дії впорядковані і виконуються з суворим дотриманням алгоритму.

CLIPS є одним із найбільш широко використовуваних інструментальних середовищ для розробки ЕС завдяки своїй швидкості, ефективності й безоплатності.

CLIPS включає повноцінну об'єктно-орієнтовану мову *COOL* для написання ЕС. Хоча вона написана мовою *C++*, її інтерфейс набагато ближче до мови програмування *Lisp*. Розширення можна створювати мовою *C++*, крім того, можна інтегрувати *CLIPS* у програми мовою *C++*.

CLIPS розроблене для застосування як мова прямого логічного виведення. Як і інші ЕС, має справу з правилами та фактами.

Нащадками *CLIPS* є мови програмування *JESS* (частина *CLIPS*, що працює з правилами і переписана на *Java*, пізніше розвинулася в іншому напрямку).

7.3. Створення СШІ за допомогою *Visual Prolog*

Виокремлюють два взаємопов'язані способи подання знань:

– *процедурний* — визначення алгоритму обробки даних;

– *декларативний* — визначення окремих понять, їхнього стану в конкретні моменти часу і зв'язків між ними.

Традиційні алгоритмічні мови (*Visual Basic*, *C++*, *Fortran*) є процедурними, оскільки їх мета — опис алгоритму. Але вони містять і декларативні компоненти (опис змінних). У мові *Prolog* (*Pro* — програмування, *log* — логічне) основною є декларативна компонента, тому вона призначена більше для обробки фактів і декларативних правил, ніж для обробки даних.

Сукупність фактів, що описують визначену предметну галузь, формують БЗ. Після запиту логічні методи забезпечують одержання нових фактів із фактів, поданих у БЗ, тобто намагаються довести істинність логічного виразу.

Мова *Prolog* з'явилась у 1973 р., у 1997 р. випустили *Visual Prolog* 5.0, а з 1999 р. розповсюджується варіант *Personal Edition*, призначений для некомерційного використання.

Мова *Prolog* може застосовуватися для:

– доведення теорем і виведення розв'язків у задачах;

– створення пакетів символьної обробки при розв'язуванні рівнянь, диференціюванні, інтегруванні тощо;

– розробки спрощених версій СШІ;

– створення природно-мовних інтерфейсів для наявних програм;

– перекладу текстів з однієї мови іншою, в тому числі — з однієї мови програмування іншою.

7.3.1. Середовище програмування Visual Prolog

Середовище програмування *Visual Prolog* запускається так: *Пуск* ⇨ *Програми* ⇨ *Visual Prolog* 5.2 ⇨ *Visual Prolog* 32. Вигляд вікна середовища *Visual Prolog* зображено на рис. 7.1.

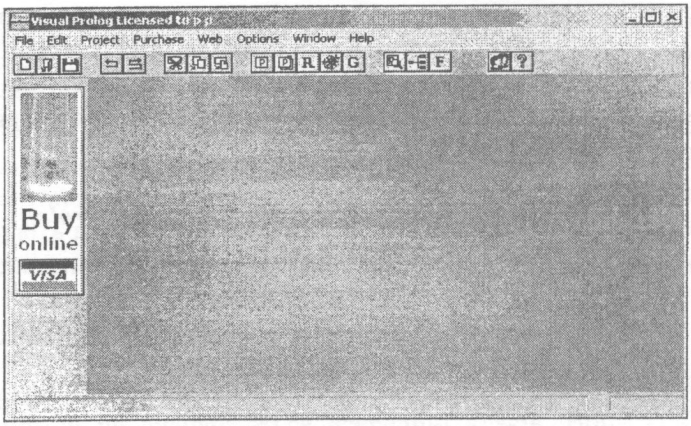


Рис. 7.1. Вигляд основного вікна середовища *Visual Prolog 5.2*

Запуск і тестування програми. За допомогою команди меню *File | New* (*F7*) створюється новий файл *noname.pro*. Для виконання програми достатньо у вікні надрукувати текст

```
GOAL Write("Привіт!"), nl.
```

Активацією команди *Project | Test Goal* (*Ctrl+G*) виконується GOAL. Результат виконання команди GOAL наведено на рис. 7.2.

Результат виконання програми буде виведено в окремому вікні (*Inactive ...*), яке необхідно закрити перед тим, як тестувати наступну програму.

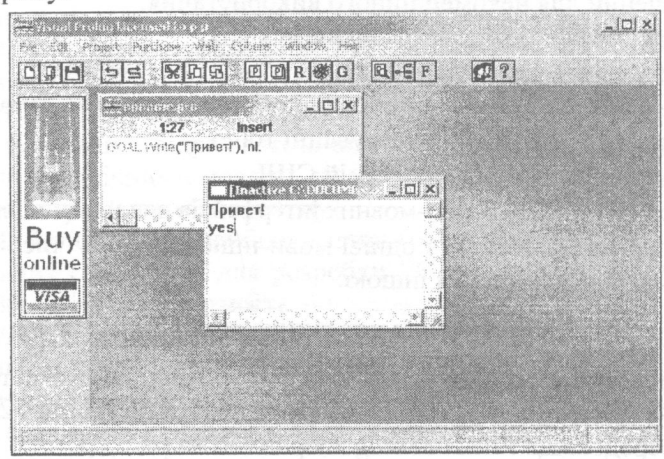


Рис. 7.2. Вигляд тестового вікна виконання команди GOAL

Обробка помилок. У разі виявлення помилок у програмі середовище *Visual Prolog* відобразить вікно *Errors* (*Warnings*). Вигляд списку виявлених помилок наведено на рис. 7.3.

Подвійне натискання на напис помилки дозволяє переходити на її місце у вихідному коді програми.

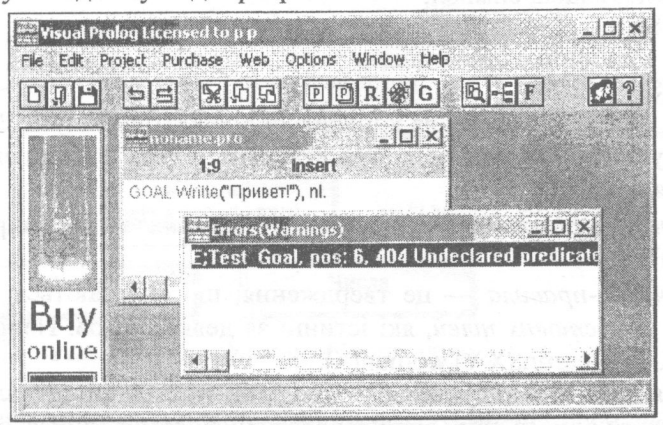


Рис. 7.3. Вигляд вікна з виявленими помилками

7.3.2. Структура програми на Visual Prolog

Prolog є мовою числення предикатів. ***Предикат*** — це логічна формула одного або декількох аргументів. Наприклад, висловлювання «Роза червона» записується як red(rose).

Програма може включати такі розділи:

CONSTANTS

опис констант;

DOMAINS

опис імен та структур об'єктів;

PREDICATES

опис предикатів, тобто назв відносин, що існують між об'єктами (ім'я, тип, кількість аргументів). Необхідно також вказати один із режимів детермінізму фактів:

– *nondeterm* (режим за замовчуванням) — фактів може бути скільки завгодно, навіть жодного;

– *determ* — факт може бути в одному екземплярі або не бути зовсім;

– *single* — факт лише один. Як правило, використовується для лічильників, при оголошенні лічильнику присвоюється початкове значення. *Single*-факти, або ще їх називають факти-змінні, допус-

кають руйнівне присвоювання і відіграють роль змінних із «глобальною» областю видимості зі справжніх імперативних мов. Використовуючи їх, можна максимально пришвидшити роботу програми з підрахунку чого-небудь. Тип факту-змінної може бути не лише число, рядок, символ.

DATABASE

опис зовнішньої бази даних;

CLAUSES

опис фактів і правил для конкретної предметної галузі. Використовуються речення двох типів, кожне з яких завжди закінчується крапкою:

– *речення-факти* — це одиночна істинна ціль, наприклад, man(ivan);

– *речення-правила* — це твердження, що складаються з *головної цілі* і *хвостових цілей*, які істинні за деяких умов. Наприклад: mother(X,Y):—parent(X,Y),woman(X);

Предикати записуються у довільній послідовності, але предикати одного типу обов'язково записуються один за одним. Наприклад:

man(ivan). man(tom). man(bob).

woman(lisa). woman(pam).

GOAL

– опис цілі (запиту). Включається безпосередньо в програму. Речення завершується крапкою.

Обов'язковими є три секції — predicates, clauses, goal.

Для коментування використовують такі конструкції:

% рядковий коментар; /* блочний коментар */.

7.3.3. Об'єкти даних у Visual Prolog

Visual Prolog розрізняє тип об'єкта за його синтаксичною формою в тексті програми: *змінні* починаються з великої літери, *константи* — з малої. На рис. 7.4 наведено класифікацію об'єктів даних.

Атоми і ***змінні*** можуть набувати складних форм, а саме становити собою ланцюжки великих і малих літер, цифри та спеціальні символи (наприклад, + - * / = : . & _ ~).

Атоми можна створювати трьома способами:

1) з ланцюжка літер, цифр і символу підкреслення _, починаючи такий ланцюжок із малої літери. Наприклад: анна, x_25, Петро_Іванов;

2) зі спеціальних символів, окрім символа :-. Наприклад: <---> або ====>;

3) з ланцюжка символів, в одинарних лапках. Наприклад: 'Південна_Америка' або 'Петро Іванов'.

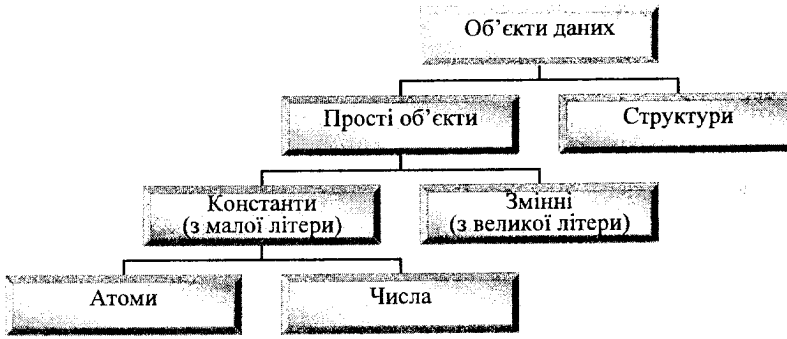


Рис. 7.4. Класифікація об'єктів даних у мові *Prolog*

Якщо змінна трапляється один раз, то можна використовувати *анонімну змінну* (символ нижнього підкреслення).

Числа у мові *Prolog* бувають цілими й дійсними. При звичайному програмуванні мовою *Prolog* дійсні числа використовуються рідко, оскільки ця мова призначена передусім для обробки символьної, а не числової інформації.

Структури — це об'єкти, які складаються з кількох компонент. Ці компоненти, в свою чергу, можуть бути структурами. Наприклад, дату можна розглядати як структуру, що складається з трьох компонент, — день, місяць, рік: дата (1, травень, 2017).

Для об'єднання використовують *функтор* дата (рис. 7.5). Усі компоненти в цьому прикладі є константами (дві компоненти — цілі числа і одна — атом). Компоненти можуть бути також змінними або структурами. Довільний день у травні можна зобразити структурою: дата (День, травень, 2017).

Рис. 7.5. Приклад структурованого об'єкта

У цьому випадку День є змінною і їй можна приписати довільне значення при подальших обчисленнях. Такий метод структуризації даних простий і ефективний. Це є однією з причин того, чому

Prolog доцільно використовувати для розв'язування задач обробки символьної інформації.

Усі структурні об'єкти можна зображати у вигляді дерев. Коренем дерева є функтор, гілками, що виходять з нього, — компоненти. Якщо деяка компонента теж є структурою, тоді їй відповідає піддерево в дереві, яке зображує весь структурний об'єкт. Кількість компонент в структурі називається ***арністю***.

Приклад написання програми у середовищі *Visual Prolog* наведено у програмі 7.1.

Програма 7.1

```
DOMAINS
        name=symbol
        age=integer
PREDICATES
        nondeterm  parent(name, name)
                /*якщо не використовувати секцію domains, то можна записати: person(symbol, integer)*/
        nondeterm  man(name, age)
        nondeterm  woman(name, age)
CLAUSES
        parent(pam, bob).
        parent(tom, bob).
        man(bob, 18).
        man(tom, 45).
        woman(pam, 40).
```

7.3.4. Складання запитів мовою Prolog

Запит у *Prolog* називається ***ціллю*** (goal), яку необхідно вивести з наявних фактів. Цілі бувають прості, складні, константні, зі змінними.

Прості запити. У *простій константній цілі* аргументи предикатів задано у вигляді констант, тому відповідь може бути лише «так» або «ні». Наприклад, запит для програми 7.1:

```
GOAL
   parent(pam, bob).          Yes
Чи є Пам родичем Боба?        Відповідь — так.
```

У *простих запитах зі змінними* замість аргументів можуть стояти змінні, які обов'язково починаються з великої літери. Наприклад, запит для програми 7.1:

```
GOAL
   parent(Who, bob).
```
Хто батьки Боба?

```
Who=pam
Who=tom
2 Solution
```
Дві відповіді: Пам і Том.

```
GOAL
   parent(X, Y).
```
Хто кому родич?

Будуть виведені всі батьки та їх діти.

Коли необхідно вивести лише одну з декількох змінних, використовується *анонімна змінна*, яка позначається символом нижнього підкреслення «_». Наприклад:

```
GOAL
   parent(_,Child).
```
Вивести лише імена дітей.

Складні запити містять декілька цілей, розділених комою. Можуть бути константними або зі змінними. Наприклад:

```
GOAL
   parent(Who,bob),
man(Who,_).
```
Хто родич Боба?
Чи є цей родич чоловіком?
Тобто: Хто батько Боба?

7.3.5. Складання правил мовою Prolog

Для зменшення коду програми та спрощення опису предметної галузі застосовують також речення-правила, які дозволяють визначати нові відношення через уже існуючі. Такі речення складаються з головної (ліворуч) і хвостової (праворуч) частин. У хвостовій частині відношення записують через кому (,) — «І» або крапку з комою (;) — «АБО».

Запишемо правило для відношення *батько* (*ім'я батька, ім'я дитини*) через існуючі відношення *родич* (*ім'я родича, ім'я дитини*) та *чоловік* (*ім'я, вік*).

Правило читається так: «*X* буде батьком *Y*, якщо *X* — родич *Y*, та *X* — чоловік» і записується: father(X,Y) :— parent(X,Y), man(X,_).

На рис. 7.6 зображено граф для цього правила. Використовуються позначення: квадрат — особа чоловічої статі, трикутник — особа жіночої статі, коло — стать не вказана.

Доповнюємо програму новим предикатом father.

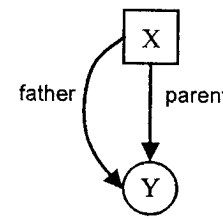


Рис. 7.6. Граф для відношення «батько»

```
PREDICATES
    nondeterm  father(symbol, symbol)
```
та правилом
```
CLAUSES
    father(X, Y):—parent(X, Y), man(X, _).
```

При виконанні програми на запит: father(F, Ch) — «Хто кому батько?» буде видано одну відповідь.

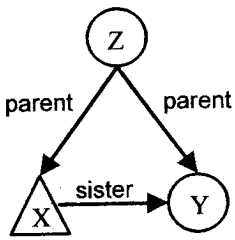


Рис. 7.7. Граф для відношення «сестра»

При складанні правил слід брати до уваги особливості предметної галузі. Наприклад, при складанні правила для відношення *сестра* (*ім'я сестри, ім'я родича*) слід пам'ятати, що на місці першого і другого аргументу не може бути одна й та сама людина, тобто не можна бути сестрою самій собі — це помилка (рис. 7.7). Тому треба явно вказати, що це різні об'єкти: $X \diamond Y$: sister(X,Y):-parent(Z,X), parent(Z,Y), woman(X,_), X<>Y.

7.3.6. Механізм роботи Visual Prolog

Для доведення вивідності цілі з сукупності фактів і правил *Prolog* використовує *метод резолюцій* і спосіб доведення «*від протилежного*». Тоді задача подається у вигляді сукупності тверджень (аксіом), цілі завдання також записуються у вигляді твердження. Розв'язок задачі становить з'ясування логічного слідування з аксіом

$$(A_1 \wedge A_2 \wedge A_3 \wedge \ldots A_n) \rightarrow B.$$

Тобто, якщо $A \rightarrow B$ істинне, то $\overline{A \rightarrow B}$ хибне і $A \vee \overline{B}$ хибне. Для доведення необхідно отримати пустий диз'юнкт. Для окремого випадку (хорнівських диз'юнктів) ця задача завжди розв'язна. Будь-яку програму мовою *Prolog* можна розглядати як БЗ. Механізм обробки запитів називається ***уніфікацією***. Після того, як користувач вводить запит, інтерпретатор приступає до аналізу вмісту БЗ, виконуючи допустимі підстановки фактів у цільове твердження, щоб обґрунтувати його істинність.

У *Prolog* реалізовано стратегію пошуку *по дереву в глибину*. Зіставлення з цілями проводиться шляхом обходу дерева згори вниз і зліва направо. Якщо процес зіставлення невдалий, програма по-вертається до найближчого альтернативного варіанта, при цьому зв'язані змінні звільняються.

Ціль (*goal*) у *Prolog* — це речення, яке необхідно довести, тобто вивести його з тих речень-фактів, які є в розділі CLAUSES програми. Цілі бувають прості, складні, константні, зі змінними.

При пошуку розв'язку *Prolog* може виконувати такі операції:

– зіставлення зі зразком;

– зв'язування змінних;

– відкат і звільнення змінних.

Зіставлення зі зразком. Розглянемо цю операцію на прикладі *простої константної цілі*. Метод розв'язування простий — потрібно переглянути всі предикати man(symbol) і зіставити аргументи речення-цілі і речення-факту.

Для прикладу, наведеного у програмі 7.2, якщо вони збігаються, то ціль доведено, і *Prolog* підтверджує YES, якщо такого факту немає, то NO. Але це не означає, що Джон не чоловік. Просто такого факту в БЗ немає.

Програма 7.2

```
CLAUSES
    man(tom).
    man(jon).
    man(bob.)
```

```
GOAL
    man(jon).
```

Необхідно довести, що Джон — чоловік

Операція зіставлення цілі з реченнями програми називається ***операцією зіставлення зі зразком*** і починається згори вниз, а порівняння параметрів — зліва направо. Пошук *складної константної цілі* відрізняється тим, що операція зіставлення проводиться необхідну кількість разів. Щоб довести складну ціль, необхідно довести всі підцілі зліва направо. У даному випадку у програмі 7.3 перша ціль збігається з третім реченням, а друга — з восьмим. Тобто ціль доведено.

Програма 7.3

```
CLAUSES
    father(jon, lisa).
    father(bob, tom).
    father(bob, pat).
    man(jon).
    man(tom).
    man(bob).
    woman(lisa).
    woman(pat).
```

```
GOAL
    father(bob,pat), woman(pat).
```

Чи є Пат дочкою Боба?

Зв'язування змінних. Розглянемо цю операцію на прикладі *простої цілі зі змінними.*

Для програми 7.4 *Prolog* починає операцію зіставлення зі зразком (згори вниз). Зіставляються назва предиката і його аргументи. Ціль (зразок) father (Who, tom). Першим реченням, що відповідає вимогам, є father (bob, tom).

Програма 7.4

```
CLAUSES
  father(jon, lisa).
  father(bob, tom).
  father(bob, pat).
```

```
GOAL
    father(Who, tom).
```

Хто батько Тома?

Далі *Prolog* робить операцію ***зв'язування вільної змінної*** Who з константою, що стоїть на тому ж місці в предикаті. Вільна змінна Who стає зв'язаною змінною зі значенням bob.

Prolog вдалося знайти факт, що не суперечить цілі, зв'язавши змінну Who. Ціль доведено. На екран виводиться значення зв'язаної змінної Who = bob.

Операцію зв'язування називають також операцією ***уніфікації***, а програми, які її здійснюють, — програмами уніфікації. Програми уніфікації шукають усі можливі розв'язки.

При пошуку *складної цілі зі змінними* також проводиться зв'язування змінних, а в разі невдачі — ***відкат, звільнення змінних*** і повернення на попередню позицію.

Для програми 7.5 *Prolog* починає доводити першу підціль. Зразок father (bob, X) вдається зіставити з другим реченням father (bob,tom). При цьому вільна змінна X зв'язується зі значенням tom. Підціль доведено.

Програма 7.5

```
CLAUSES
    father(jon, lisa).
    father(bob, tom).
    father(bob, pat).
    man(jon).
    man(tom).
    man(bob).
    woman(sui).
    woman(pat).
```

```
GOAL
    father(bob,X), woman(X).
```

Чи є у Боба дочка?

Prolog переходить до доведення другої підцілі — woman (X). Значення зв'язаної змінної X=tom підставляється у предикат woman. Тоді зразок для пошуку буде woman (tom). Друга підціль не може бути доведена, тому що такого факту немає. Перше зв'язування виявилося невдалим.

Prolog звільняє зв'язану змінну і повертається (відкочується) до доведення першої підцілі, оскільки ще є речення, з якими можна зіставляти.

Друге зіставлення зразка father (bob, X) з реченням father (bob, pat) зв'язує X значенням pat. Доведенням другої підцілі — woman (pat) є восьме речення. *Prolog* знайшов розв'язок:

X = pat

1 Solution.

Складемо дерево пошуку розв'язку для цього прикладу (рис. 7.8). При доведенні першої підцілі father(bob, X) можливі три варіанти. *Prolog* переглядає їх по черзі: перший відкидає, другий — підходить, тому «розгортає» цю гілку до кінця. При доведенні другої підцілі переглядає два варіанти. Таким чином, виникає шість можливих варіантів розв'язку. Хрестиком позначені помилкові шляхи, тобто відкат.

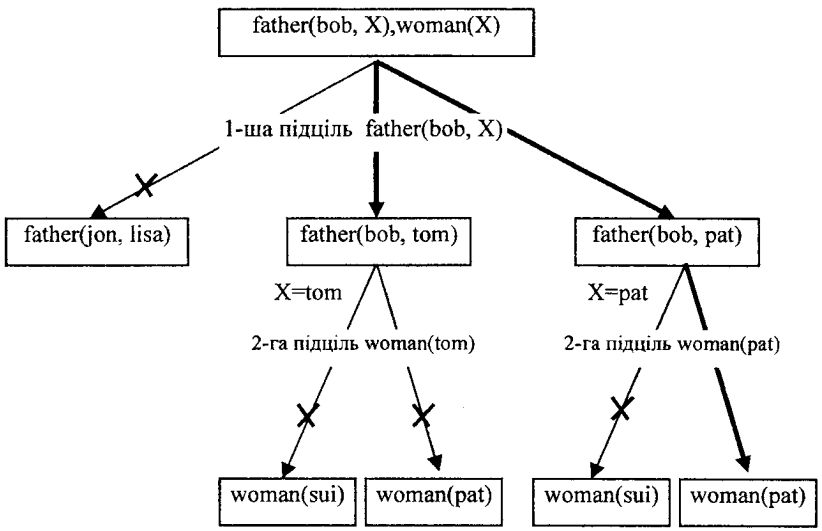


Рис. 7.8. Приклад дерева пошуку

Розглянемо приклад пошуку у випадку подання *цілі реченням-правилом* (програма 7.6).

Речення-правило складається з двох частин: голови і хвоста (заголовка і тіла). У цьому разі ціль зіставляється з заголовком правила. Наявні вільні змінні зв'язуються з відповідними значеннями. Якщо правило, своєю чергою, містить у заголовку змінні, вільні змінні цілі зв'язуються з вільними змінними заголовка без значення.

Потім *Prolog* переходить до доведення підцілей правила.

Програма 7.6

```
CLAUSES
    likes(bob, nut).
    likes(jim, apple).
    likes(tom, X):-likes(jim,Y).
```

```
GOAL
    likes(tom,Y).
```

Що любить Том?

Для програми 7.6. підходить третій предикат. Але він є правилом. Тому змінна Y зв'язується зі змінною X правила. Потім *Prolog* почне доводити підціль likes(jim,Y). Такий факт є: Y=apple. Тоді X=apple. Том любить яблука.

7.3.7. Керування пошуком розв'язків у Visual Prolog

Керування пошуком розв'язків у *Visual Prolog* реалізовано за допомогою механізму пошуку з поверненням (*backtracking*), за якого система намагається відшукати всі можливі варіанти розв'язку задачі. Механізм виведення програми запам'ятовує ті точки процесу уніфікації, у яких не були використані всі альтернативні розв'язки, а потім повертається в ці точки і шукає розв'язки іншим шляхом.

Предикат fail називають ***відкатом*** після невдачі. Він викликає штучне неуспішне завершення пошуку, що дозволяє отримати всі можливі розв'язки задачі.

Предикат ***відсікання*** cut позначається за допомогою символу !. Він дозволяє отримати доступ лише до частини даних, усуваючи подальші пошукові дії. У загальному випадку можна записати:

$$R :- A, B, !, C.$$

Це означає, що якщо для цілей A і B знайдено хоча б один розв'язок, то подальше перебирання можливих варіантів значень A і B не потрібне.

Приклад наведено у програмі 7.7.

Програма 7.7

```
PREDICATES
    nondeterm child(symbol).
    show.
GOAL
    write(«Це хлопчики:»),nl, show.
CLAUSES
    child(«Петро»).child(«Іван»).child(«Олег»).
    child(«Маша»).child(«Оля»).child(«Катя»).
    show :- child(X), write(X), nl, X=«Олег»,!.
```

Програма надрукує лише імена хлопчиків.

7.3.8. Арифметичні операції у Visual Prolog

Visual Prolog використовує низку вбудованих функцій для обчислення арифметичних виразів, деякі з яких перелічені в табл. 7.1.

Таблиця 7.1

Основні арифметичні операції *Visual Prolog*

Позначення	Тип операції
>,<,=,>=,<=,<>	Операції порівняння
+, -, *, /	Арифметичні операції
X mod *Y*	Остача від ділення *X* на *Y*
X div *Y*	Частка від ділення *X* на *Y*
abs(*X*)	Абсолютна величина числа *X*
sqrt(*X*)	Квадратний корінь із *X*
sin(*X*),cos(*X*), tan(*X*),arctan(*X*)	Тригонометричні функції
exp(*X*)	Піднесення до степеня *X*
log(*X*), ln(*X*)	Логарифми числа *X*

Для опису будь-яких операцій арифметики можна також використовувати власні предикати.

Приклад використання власних предикатів для здійснення арифметичних операцій наведено у програмі 7.8.

Програма 7.8

```
PREDICATES
    nondeterm add(integer,integer)
    nondeterm fadd(real,real)
    nondeterm maximum(real,real,real)
```

```
CLAUSES
    add(X,Y):- Z=X+Y, write("Sum= ",Z),nl.
    fadd(X,Y):- Z=X+Y, write("FSum= ",Z),nl.
    maximum(X,X,X).
    maximum(X,Y,X):- X>Y.
    maximum(X,Y,Y):- X<Y.
```

Предикати *Prolog* не можуть з'являтися в арифметичних виразах. Якщо необхідно, наприклад, змінній *R* присвоїти значення, яке дорівнює більшому з двох виразів *X* і *Y*, помноженому на 3, то, використавши предикат maximum, це можна записати так:

```
maximum(X,Y,Z), R= 3*Z.
```

Обчислити довжину гіпотенузи за довжинами катетів прямокутного трикутника можна за допомогою конструкції типу:

```
gipotenuza(X,Y,Z):- Z = sqrt(X*X + Y*Y).
```

7.3.9. Рекурсія в Visual Prolog

Рекурсія в *Visual Prolog* важлива, оскільки в ньому немає циклічних конструкції. ***Рекурсія*** — це зведення розв'язку задачі до неї ж, але меншої розмірності. Процес продовжується поки її можна буде розв'язати безпосередньо.

Спосіб, за якого розв'язування складається з двох фаз: перша фаза — розбиття на підзадачі, друга фаза — розв'язування підзадач, називається ***низхідною стратегією***. Він простий, але не завжди ефективний. Головний недолік — велика витрата пам'яті. При кожному рекурсивному виклику динамічно виділяється ділянка пам'яті для всіх змінних.

Стратегія розв'язування простого випадку, піднімаючись вгору по ієрархії підзадач, поки не буде розв'язано початкову задачу, називається ***висхідною***. Тоді рекурсивний виклик — останній оператор у правилі, і запам'ятовувати значення змінних немає потреби. Всі відомості передаються як параметри.

Приклад 1. Розглянемо задачу про факторіал $n! = (n-1)!*n$ та її розв'язання за допомогою низхідної рекурсивної стратегії. У *Visual Prolog* програма матиме такий вигляд (програма 7.9).

Програма 7.9

```
PREDICATES
    nondeterm fact(integer, integer)
```

```
CLAUSES
    fact(1, 1).
    fact(N, P):-N1=N-1, fact(N1, P1), P=P1*N, N1>0.
GOAL
    fact(N, P).
```

Предикат fact включає два аргументи:

– перший аргумент — розмір задачі,

– другий — результат.

У *Prolog* першим завжди пишеться предикат для розв'язування простого випадку. У даному разі при аргументі N=1 результат теж дорівнює одиниці.

Друге правило складається з чотирьох підцілей:

– перша — це зменшення розмірності задачі (оскільки *Prolog* забороняє писати як параметри предиката арифметичні вирази);

– друга — рекурсивне звернення зі зменшеним значенням аргументу N;

– третя — формування спільного розв'язування зі щойно отриманих;

– четверта — перевірка умови на закінчення виконання.

Приклад 2. Приклад висхідної (хвостової рекурсії) для розв'язання задачі про факторіал.

Для обчислення факторіала $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$ початкове значення (позначимо його як R0=1) треба множити на число I0=0, яке збільшується на 1 при кожному рекурсивному виклику. Коли значення I0 буде дорівнюватиме N, рекурсивні виклики закінчуються. Вводитися допоміжна процедура fv з двома додатковими параметрами R0=1 — початкове значення факторіала, I0=0 — лічильник кількості рекурсивних викликів. Тоді: f(N, R):-fv(N, R, 1, 0).

Далі записується основна частина програми.

```
fv(N, R, R0, I0) :- I0<N, !,    % перевірка на закінчення
I1=I0+1,            % модифікація лічильника
R1=R0*I1,           % обчислення поточного значення
fv(N,R,R1,I1).      % хвостова рекурсія
```

Після закінчення рекурсивних викликів у R1 буде останнє обчислене значення факторіала. Для пересилання його в змінну R використовується така конструкція предиката:

```
fv( _, R, R, _).
```

Приклад 3. Задача про суму натуральних чисел.

Виведемо рекурентну формулу для обчислення суми. Для цього віднімемо S_{n-1} від S_n.

$$S_n = 1 + 2 + \ldots + (n-1) + n$$

—

$$S_{n-1} = 1 + 2 + \ldots + (n-1)$$

$$S_n - S_{n-1} = n$$

$$S_n = S_{n-1} + n$$

$$S_0 = 0$$

Фрагменти програми мовою *Prolog* для обчислення суми методом загальної рекурсії (програма 7.10) і методом хвостової рекурсії (програма 7.11) наведено нижче.

Програма 7.10

```
CLAUSES
    Sum(0,0).
    Sum(N,S): -
        N1 = N – 1,
            Sum(N1,S1),
            S = S1+N,
            N1>=0.
```

Програма 7.11

```
CLAUSES
    Sum(N,S): - Sm(N,S,1,0).
    Sm(N,S,N0,S0): -
        N0 <= N,!,
        N1=N0 + 1,
        S1= N0 + S0,
        Sm(N,S,N1,S1),
        Sum(_,S,_,S).
```

На рис. 7.9 зображено схему розв'язування задачі про знаходження суми для випадку Sum(3, S) методом загальної рекурсії.

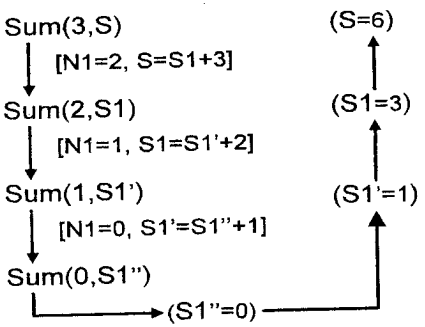


Рис. 7.9. Етапи розв'язуваання задачі про суму чисел методом низхідної рекурсії

7.3.10. Списки в Visual Prolog

Списки в *Visual Prolog* — це множина елементів одного типу (одного домену), які записуються у вигляді послідовності елементів, розділених знаком кома і записаних у квадратні дужки: [1,3,5], [a,b,e,f], [jam, apple, jin].

Списки є динамічними структурами, тобто можуть містити довільну кількість елементів, і їх кількість може змінюватися в процесі роботи програми. Частковим випадком списку є список, що складається з одного елемента — [x] і порожній список — [].

Кожен непорожній список може бути розділений на *голову* — перший елемент списку і *хвіст* — інші елементи списку (рис. 7.10). Це дозволяє зобразити список у вигляді бінарного дерева.

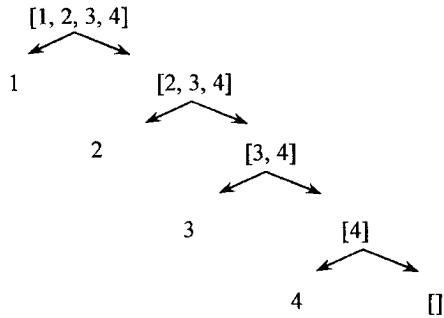


Рис. 7.10. Структура списку у *Visual Prolog*

Список є рекурсивною складною структурою даних, головний спосіб обробки списку — це перегляд та обробка кожного його елемента, поки не буде досягнуто кінця.

Операції зі списками у *Prolog* позначаються одним символом — вертикальна риска (|):

```
L = [Head | Tail].
```

Залежно від ситуації, програми зіставлення та уніфікації самостійно виконують операції поділу списку на голову Head і хвіст Tail або додавання елемента Head до списку Tail.

Алгоритм складається з двох речень: перше вказує, що робити зі звичайним списком (списком, який можна розділити на голову і хвіст), друге — що робити з порожнім списком.

Розглянемо декілька прикладів роботи зі списками.

Приклад друкування списку наведено у програмі 7.12.

Програма 7.12

```
DOMAINS
    list = integer*   % позначення списку
PREDICATES
    nondeterm print(list)
CLAUSES
    print([]).     % якщо список порожній, то нічого не робити
    print(H|T):- write(H), % надрукувати значення голови списку
    nl,            % перехід на наступний рядок
    print(T).      % надрукувати хвіст списку
```

Приклад визначення довжини списку наведено у програмі 7.13. При складанні програми взято до уваги, що довжина списку дорівнює довжині хвоста списка плюс 1, а довжина порожнього списку ([]) дорівнює нулю.

Програма 7.13

```
DOMAINS
    list = Integer*
PREDICATES
    nondeterm len(List, Integer)
CLAUSES
    len([], 0).              %довжина порожнього списку
len([ _|Tail], N):- len(Tail, N1), %обчислення довжини хвоста
    N=N 1+1.                 % обчислення загальної довжини
```

Приклад визначення належності елемента до списку наведено у програмі 7.14.

Програма 7.14

```
DOMAINS
   list = Integer*
PREDICATES
   nondeterm member(integer, list)
CLAUSES
   member(X, [X | _ ]).
   member(X, [ _ | T ]) :- member(X, T).
```

Порівнюємо перший елемент списку з шуканим елементом X. Якщо збігається, то припиняємо виконання — відповідь «так».

Якщо ні, перевіряємо наявність елемента X у хвості списку. Хвіст T рекурсивно передаємо у процедуру member(X, T) поки не буде досягнуто кінця списку. Якщо елемент не знайдено — відповідь «ні».

Приклад видалення елемента X зі списку у програмі 7.15.

Якщо X є першим елементом списку, то розв'язком є хвіст списку. У протилежному випадку треба видалити елемент X з хвоста і приєднати до отриманого списку голову. Тобто переносимо голову списку H у результуючий список, а хвіст T рекурсивно передаємо у процедуру delete(X, T, L) поки не буде досягнуто кінця списку.

Якщо у списку декілька однакових елементів, то при даному випадку програми буде видалено лише перший з таких елементів.

Програма 7.15

```
DOMAINS
    list = integer*
PREDICATES
    nondeterm delete(integer, list, list).
CLAUSES
    delete(X, [X | T ], T).
    delete(X, [ H | T ], [H | L]) :- delete(X, T, L).
```

Приклад злиття двох списків наведно у програмі 7.16.

Якщо перший список порожній, то результат — другий список. В іншому випадку необхідно злиття хвоста першого списку з другим списком та додавання до отриманого списку голови першого списку.

Програма 7.16

```
DOMAINS
    list = integer*
PREDICATES
    nondeterm merge(list, list, list).
CLAUSES
    merge([], L2, L2).
    merge([H | T], L2, [H | L3]) :- merge(T, L2, L3).
```

Приклад об'єднання двох списків наведено у програмі 7.17.

Об'єднання відрізняється від злиття тим, що елементи списку не мають повторюватись.

Тому якщо голова першої множини належить до другої множини, то її треба додавати до отриманого списку.

Програма 7.17

```
DOMAINS
    list = integer*
PREDICATES
    nondeterm union(list, list, list).
```

```
CLAUSES
    union([], L2, L2).
    union([H | T], L2, L3):-
    member(H, L2),          /*перевіряємо належність голови H до
                                другого списку L2*/
    union(T, L2, L3), !.
    union([H | T], L2, [H | L3]) :-
    union(T, L2, L3).
```

7.3.11. Бази даних та знань у Visual Prolog

Факти, описані в розділі clauses, можна розглядати як статичну БД. Ці факти є частиною коду програми і не можуть бути оперативно змінені. Для створення динамічної БД у *Visual Prolog* передбачений спеціальний розділ database.

Предикати в цьому розділі можуть мати таку ж форму подання, що і в статичній частині програми, але повинні мати інше ім'я.

У програмі 7.18 наведено приклад опису БД.

Програма 7.18

```
DOMAINS
    name = symbol.
    rost, ves = integer.
DATABASE
    nondeterm dplayer(name, rost, ves).
PREDICATES
    player(name, rost, ves).
CLAUSES
    player(«Михайлов», 180, 87).
    player(«Петров», 187, 93).
    player(«Харламов», 177, 80).
```

Припустимо, що необхідно після запуску програми перемістити дані зі статичної БД у динамічну. Для цього можна описати таке правило:

```
    assert_database:-
    player(N,R,V),
    assertz(dplayer(N,R,V)),fail.
```

У цьому правилі використано вбудований предикат assertz, який поміщає твердження в БД після всіх тверджень, які там уже є. Є також вбудовані предикати для видалення тверджень (retract), зчитування з диска (consult), запису БД на диск (save) і збору даних із БД у список (findall).

Головна перевага БД у *Prolog*, як і будь-якій іншій БД, полягає в можливості швидкого вибіркового доступу до інформації. Наприклад, у наведеному вище прикладі це може бути запит:

```
GOAL
    player(N, R, V), R>180.
```

Основна відмінність *Prolog*-програм від реляційних БД у тому, що внаслідок використання правил і логічного виведення можна отримувати нові знання, а не лише отримувати наявні відомості або змінювати дані за певним законом. Тому *Prolog*-програма може розглядатися як БЗ (експертна система).

У програмі 7.19 наведено приклад простої ЕС на правилах, яка розв'язує задачу визначення виду спійманої риби. Загалом ЕС складається із п'яти основних модулів: БЗ, БД, блока інтерфейсу з користувачем, механізму логічного виведення, пояснювального компонента.

У наведеному прикладі динамічна БД, у яку записуються відповіді користувача, отримані при роботі ЕС, міститься у розділі DATABASE *Prolog*-програми. БЗ, побудована на правилах, містить знання про предметну галузь і описується предикатом fish_is. Блок інтерфейсу з користувачем реалізований предикатом vopros. Механізм логічного виведення забезпечується предикатом expertiza. Пояснювальний компонент у наведеному прикладі відсутній.

Програма 7.19

```
DATABASE
    xpositive(symbol, symbol)
    xnegative(symbol, symbol)
PREDICATES
    expertiza
    nondeterm vopros(symbol, symbol)
    nondeterm fish_is(symbol)
    nondeterm positive(symbol, symbol)
    nondeterm negative(symbol, symbol)
    nondeterm remember(symbol, symbol, symbol)
    clear_facts
GOAL
    expertiza.
CLAUSES
    expertiza:-
        fish_is(X), !, nl,
        write(«ваша риба це », X, « »), nl,  clear_facts.
```

```
expertiza:-
    nl, write(«це невідома риба!»), clear_facts.
vopros(X,Y):-
    write(«запитання — », X, « », Y, «? (так/ні)»),
    readln(R), remember(X, Y, R).
positive(X, Y):-
    xpositive(X, Y), !.
positive(X,Y):-
    not(negative(X, Y)), !,
    vopros(X, Y).
negative(X, Y):-
    xnegative(X, Y), !.
remember(X,Y, «так»):-
    assertz(xpositive(X, Y)).
remember(X, Y, «ні»):-
    assertz(xnegative(X, Y)), fail.
clear_facts:-
    retract(xpositive(_, _)), fail.
clear_facts:-
    retract(xnegative(_, _)), fail.
fish_is(«сом»):-
    positive(«у риби»,«вага >40 кг»).
fish_is(«сом»):-
    positive(«у риби»,«вага <40 кг»),
    positive(«у риби»,«є вуса»).
fish_is(«щука»):-
    positive(«у риби»,«вага <20 кг»),
    positive(«у риби»,«довге вузьке тіло»).
fish_is(«окунь»):-
    positive(«у риби»,«вага <20 кг»),
    positive(«у риби»,«широке тіло»),
    positive(«у риби»,«темні смуги»).
fish_is(«плітка»):-
    positive(«у риби»,«вага <20 кг»),
    positive(«у риби»,«широке тіло»),
    positive(«у риби»,«сріблясta луска»).
```

Із коду програми 7.19 можна побачити, що програма реалізує задане дерево пошуку розв'язку.

Відповіді на поставлені запитання дозволяють просуватися по гілках цього дерева до одного з варіантів розв'язку.

7.3.12. Наслідування на семантичних сітках у Visual Prolog

Механізм наслідування — це метод подання багатьох станів знань, за якого конкретний стан може наслідувати інформацію від загального стану. Фраза ставиться у відповідність стану шляхом додавання імені стану в заголовок фрази (у вигляді першого її аргументу), всі інші аргументи не змінюються.

Головним завданням при використанні механізму наслідування є встановлення ієрархії станів знань. Взаємозв'язок станів можна розглядати як дерево, у якому кожен стан зображується вузлом. Нехай є два стани, А і Б. Якщо стан Б — це вузол дерева, породжений станом А, то Б успадкує всі фрази, пов'язані з А, за винятком тих, які явно спростовуються у Б.

Розглянемо реалізацію механізму наслідування у *Prolog* для простої мови *семантичних сіток*. Для спрощення не враховуватимемо важливу різницю між класами та їх екземплярами.

У семантичній сітці, зображеній на рис. 7.11, вузли представляють такі об'єкти, як конкретна канарка tweety, та класи ostrich (страус), penguin (пінгвін), robin (дрізд), bird (птах) та vertebrate (хребетне).

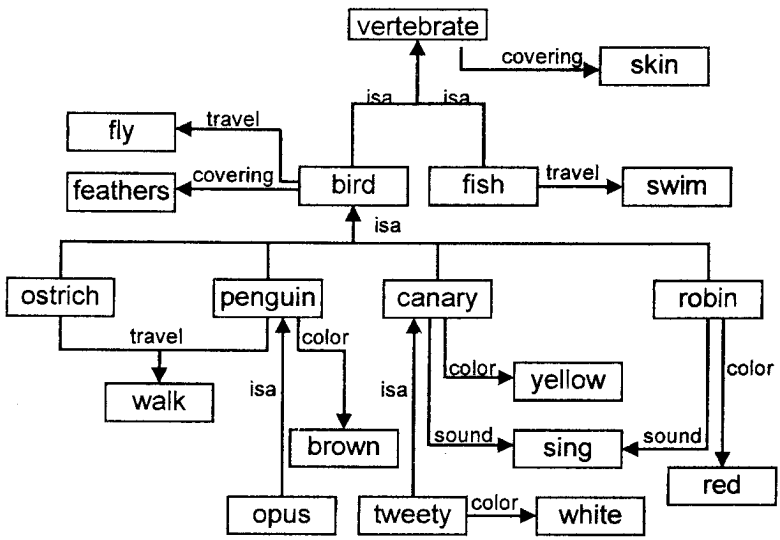


Рис. 7.11. Фрагмент семантичної сітки, що описує деяких птахів та риб

Відношення isa зв'язує класи різних рівнів ієрархії наслідування. У цій сітці реалізовані канонічні форми подання даних.

Предикат isa (Type, Parent) вказує на те, що об'єкт Type є підтипом Parent, а предикат hasprop(Object, Property, Value) описує властивість об'єктів.

Предикат hasprop вказує на те, що об'єкт Object має властивість Property зі значенням Value. При цьому Object та Value — це вузли, а Property — ім'я зв'язку, що їх об'єднує.

Фрагмент списку предикатів, які описують ієрархію деяких птахів, зображену на рис. 7.11, наведено у програмі 7.20.

Програма 7.20

```
CLAUSES
    isa(canary, bird).
    isa(ostrich, bird).
    isa(bird, vertebrate).
    isa(opus, penguin).
    isa(robin, bird).
    isa(penguin, bird).
    isa(fish, vertebrate).
    isa(tweety, canary).
    hasprop(tweety, color, white).
    hasprop(canary, color, yellow).
    hasprop(bird, travel, fly).
    hasprop(ostrich, travel, walk).
    hasprop(robin, sound, sing).
    hasprop(bird, cover, feathers).
    hasprop(robin, color, red).
    hasprop(penguin, color, brown).
    hasprop(fish, travel, swim).
    hasprop (penguin, travel, walk).
    hasprop(canary, sound, sing).
    hasprop(vertebrate, cover, skin).
```

Створимо рекурсивний алгоритм пошуку, що дозволяє визначити, чи притаманні деякому об'єкту семантичної сітки вказані властивості.

Властивості зберігаються в сітці на найвищому рівні, на якому вони істинні. За допомогою наслідування об'єкт або підклас набуває властивостей свого суперкласу.

Так, наприклад, властивість fly відноситься до об'єкта bird і всіх його підкласів.

Винятки розміщуються на окремому рівні винятків. Так, страус ostrich і пінгвін penguin ходять (walk), але не літають (fly).

Предикат hasproperty починає пошук із конкретного об'єкта. Якщо інформація явно не описана для цього об'єкта, то відбувається перехід по зв'язку isa до суперкласів. Якщо суперкласу вже не існує, і предикат hasproperty не знайшов потрібну властивість, пошук завершується невдачею. Предикат hasproperty виконує пошук в глибину в ієрархії наслідування.

```
hasproperty(Object, Property, Value):-
        hasprop(Object, Property, Value).
hasproperty(Object, Property, Value):-
        isa(Object, Parent),
        hasproperty(Parent, Property, Value).
```

7.3.13. Наслідування на фреймах у Visual Prolog

Семантичні сітки можна розбивати на частини, додаючи до опису вузлів додаткову інформацію і забезпечуючи тим самим фреймову структуру. Запишемо розглянутий у попередньому пункті приклад (рис. 7.11) за допомогою фреймів та дослідимо механізм наслідування.

Кожен фрейм — це набір відношень, а комірка isa визначає ієрархію фреймів (рис. 7.12).

name: bird
isa: vertebrate
properties: travel(flies)
default:

name: penguin
isa: bird
properties: color(brown)
default: travel(walks)

name: canary
isa: bird
properties: color(yellow) sound(sing)
default: size(small)

name: tweety
isa: canary
properties:
default: color(white)

Рис. 7.12. Фрейм для бази знань про птахів

У першій комірці кожного фрейму міститься ім'я вузла, наприклад, name(tweety) або name(bird).

У другій комірці визначається відношення наслідування між даним вузлом та його родичами.

У третій комірці міститься список властивостей, які описують цей вузол. У цьому списку можна використовувати будь-які предикати. В останній комірці міститься список винятків та прийнятих за замовчуванням значень даного вузла. Його елементи також можуть бути словами або предикатами, що описують властивості.

Кожен предикат frame зв'язує імена комірок, списки властивостей та прийняті за замовчуванням значення. Це дозволяє розрізняти типи знань та приписувати їм різну поведінку в ієрархії наслідування. Хоча за наявної реалізації підкласи можуть наслідувати властивості обох списків, для конкретних додатків можуть виявитися корисними й інші подання. Наприклад, можна наслідувати лише значення, що використовуються за замовчуванням, або будувати третій список, який містить властивості самого класу, а не його екземплярів. Такі значення називають ***значеннями класу***. Наприклад, можливо, щоб клас canary визначав вид співочого птаха.

Ця властивість не має наслідуватися підкласами або екземплярами: tweety не є видом співочих птахів.

За допомогою предиката frame з чотирма аргументами запишемо відношення, показані на рис. 7.11 (програма 7.21). Для перевірки відповідності типів предиката frame, зокрема для перевірки того, що в третій комірці фрейма міститься список, елементи якого набувають значення з даного діапазону властивостей, можна використовувати вбудовані можливості мови *Prolog*.

Програма 7.21

```
CLAUSES
    frame(name(bird),
          isa(vertebrate),
          [travel(flies)],
          []).
     frame(name(penguin),
          isa(bird),
          [color(brown)],
          [travel(walks)]).
    frame(name(canary),
           isa(bird),
           [color(yellow), sound(sing)],
           [size(small)]).
    frame(name(tweety),
          isa(canary),
          [],
          [color (white)]).
```

Визначивши для фрейму на рис. 7.12 повну множину описів і відношень наслідування, розробимо процедури для отримання властивостей з цього подання (програма 7.22).

Програма 7.22

```
CLAUSES
    get(Prop, Object):-
        frame(name(Object), _, List_of_properties, _),
        member(Prop, List_of_properties).
    get(Prop, Object):-
        frame(name(Object), _, _, List_of_defaults),
        member(Prop, List_of_defaults).
    get(Prop, Object):-
        frame(name(Object), isa(Parent),_,_),
        get(Prop, Parent).
```

Якщо структура фреймів допускає множинне наслідування властивостей, то в подання та стратегію пошуку необхідно внести такі зміни. По-перше, в поданні фрейму аргумент, зв'язаний із предикатом isa, має містити список суперкласів цього об'єкта. Кожен суперклас цього списку — це батьківський клас об'єкта, вказаного в першому аргументі предиката frame. Якщо opus відноситься до класу пінгвінів penguin і є персонажем мультфільму (cartoon_char), то це можна подати як показано у програмі 7.23:

Програма 7.23

```
CLAUSES
    frame(name(opus),
          isa([penguin, cartoon_char]),
          [color(black)],
          []).
```

Перевірити властивості об'єкта opus можна за допомогою повернення по ієрархії isa до обох суперкласів penguin та cartoon_char. У попередньому прикладі необхідно замінити останній предикат get (програма 7.24).

Програма 7.24

```
CLAUSES
    get(Prop,Object):-
        frame(name(Object), isa(List), _, _),
        get_multiple(Prop, List).
```

Предикат get_multiple можна визначити як показано у програмі 7.25:

Програма 7.25

```
CLAUSES
    get_multiple(Prop, [Parent|_]):-
```

```
        get(Prop, Parent).
get_multiple(Prop, [_|Rest]):-
        get_multiple(Prop, Rest).
```

У цій ієрархії властивості класу penguin і його суперкласів будуть перевірені до початку перевірки властивостей класу cartoon_char.

Із кожною коміркою фрейму можна зв'язати будь-яку процедуру. Як параметр предиката frame можна додати правило *Prolog* або список таких правил. Для цього все правило необхідно помістити в дужки та включити цю структуру до списку аргументів предиката frame. Цей список правил, де кожне правило розміщено в дужках, має стати параметром предиката frame, який визначає відповідь залежно від значення, що передається фрейму opus.

Висновки

1. Виділяють два взаємопов'язані способи подання знань: процедурний — визначення алгоритму обробки даних і декларативний — визначення окремих понять, їхнього стану в конкретні моменти часу і зв'язків між ними.

2. Для розробки ЕС найчастіше застосовують програмне середовище *CLIPS*. Його перевагами є те, що воно включає повноцінну об'єктно-орієнтовану мову *COOL* для написання ЕС. Крім того, *CLIPS* можна інтегрувати в програми мовою *C++*. *CLIPS* розроблене для застосування як мову прямого логічного виведення і, як і інші ЕС, має справу з правилами та фактами.

3. *Prolog* є мовою числення предикатів. Предикат — це логічна формула одного або декількох аргументів.

4. Для доведення вивідності цілі із сукупності фактів і правил *Visual Prolog* використовує метод резолюцій і спосіб доведення «від протилежного». Програма у *Visual Prolog* може включати розділи constants, domains, predicates, database, clauses, goal. Обов'язковими є три секції — predicates, clauses і goal.

5. Запит у *Prolog* називається ціллю (goal), яку необхідно вивести з наявних фактів. Ціль буває простою, складною, константною, зі змінними.

6. Окрім речень-фактів використовують речення-правила, які складаються з головної (ліворуч) і хвостової (праворуч) частин. У хвостовій частині відношення записують через кому — «І» або крапку з комою — «АБО».

7. Механізм обробки запитів називається уніфікацією. Після того, як користувач вводить запит, інтерпретатор приступає до аналізу вмісту БЗ, виконуючи допустимі підстановки фактів у цільове твердження, щоб обґрунтувати його істинність. При пошуку розв'язку виконуються операції зіставлення зі зразком, зв'язування змінних, відкат і звільнення змінних.

8. Керування пошуком розв'язків у *Prolog* реалізовано за допомогою механізму пошуку з поверненням (*backtracking*). Використовується предикат fail — відкат після невдачі та предикат відсікання cut, що дозволяє отримати доступ лише до частини даних.

9. У *Visual Prolog* відсутні циклічні конструкції. Тому використовується рекурсія, тобто зведення розв'язування задачі до неї ж, але меншої розмірності доти, поки її можна буде розв'язати безпосередньо. Розрізняють висхідну і нисхідну стратегії.

10. Списки у *Prolog* — це множина елементів одного типу, які записуються у вигляді послідовності елементів, розділених знаком кома і записаних у квадратні дужки. Кожен непорожній список може бути розділений на голову — перший елемент списку і хвіст — інші елементи списку. Головний спосіб обробки списку — це перегляд та обробка кожного його елемента, поки не буде досягнуто кінця.

11. Для створення БД у *Visual Prolog* використовується спеціальний розділ database. У БД можна лише отримувати наявні відомості або змінювати дані за певним законом. Оскільки в *Prolog* унаслідок використання правил і логічного виведення можна отримувати нові знання, то його можна розглядати як БЗ (ЕС).

12. Наслідування в *Prolog* — це метод подання багатьох станів знань, за якого конкретний стан може наслідувати інформацію від загального стану. Фраза ставиться у відповідність стану шляхом додавання імені стану в заголовок фрази (у вигляді першого її аргументу), всі інші аргументи не змінюються. Головним завданням є встановлення ієрархії станів знань.

Вправи

1. Складіть програму у середовищі *Visual Prolog*, яка описує родинні відношення. Складіть правила та графи для відношень тітка, дитина, онучка, племінник. Складіть запити різних типів.

2. Складіть мовою *Prolog* програму для обчислення суми парних чисел, використовуючи висхідну стратегію рекурсії.

3. Складіть мовою *Prolog* програму для обчислення суми чисел Фібоначі, використовуючи низхідну стратегію рекурсії.

4. Складіть мовою *Prolog* програму для розділення списку на два списки за компаратором *K*: перший список містить числа, менші за *K*, другий список — числа, більші за *K*.

5. Складіть мовою *Prolog* програму для видалення зі списку елементів із парними номерами.

6. Складіть мовою *Prolog* програму для обчислення кількості елементів у списку.

7. Складіть мовою *Prolog* програму для вставляння елемента *X* у список після елемента *Y*.

8. Складіть програму мовою *Prolog*, яка описує певну предметну галузь за допомогою семантичної сітки.

9. Реалізуйте систему мовою *Prolog*, яка описує обрану предметну галузь за допомогою фрейму та використовує механізм наслідування властивостей класу підкласами.

10. Складіть програму для ЕС певної предметної галузі, яка міститиме БЗ з предметної галузі, БД, яка містить факти, введені користувачем, блок інтерфейсу, що забезпечує діалог із користувачем, та механізм логічного виведення, який забезпечує пошук необхідних знань і фактів.

Запитання та завдання для самоперевірки

1. Чим відрізняються процедурний та декларативний методи програмування?
2. Яке основне призначення програмного середовища *CLIPS*?
3. Які переваги *CLIPS* спричинили його широке застосування для розробки ЕС?
4. Охарактеризуйте структуру програми мовою *Prolog*. Які секції є обов'язковими?
5. Як реалізується пошук цілі в *Visual Prolog*? Які операції при пошуку виконує *Prolog*?
6. Які предикати використовуються для керування пошуком?
7. Яке призначення рекурсії у *Prolog*? Яка стратегія рекурсивного пошуку є більш ефективною?
8. Як обробляються списки в *Prolog*?
9. Як реалізують БД та БЗ у *Prolog*?
10. Що таке наслідування при опису предметної галузі у *Prolog*?

ПІСЛЯМОВА

Програмні засоби, що базуються на технологіях і методах ШІ, набули значного поширення. Їх значущість, передусім, ЕС і НМ, полягає в тому, що вони суттєво розширюють коло практичних задач, розв'язання яких можна автоматизувати, і це приносить економічний ефект.

Об'єднання методів ШІ з технологією традиційного програмування додає нові якості до комерційних продуктів внаслідок забезпечення динамічної модифікації додатків користувачем, яке робить їх більш «прозорими» (наприклад, знання зберігаються обмеженою природною мовою, що не потребує коментарів до них, спрощує навчання й супровід).

Нижче наведено короткий огляд сучасних розробок із застосуванням технологій ШІ. Одним із перспективних напрямів є використання нейрокомп'ютерів для розв'язування задач із обробки зображень, наприклад, обробка аерокосмічних зображень (стиснення із відновленням, сегментація, обробка зображень), пошук, виділення й розпізнавання на зображенні рухомих об'єктів заданої форми, обробка потоків зображень, обробка інформації у високопродуктивних сканерах.

Учені університету в Гонконзі винайшли технологію за назвою *lip motion password*, яка використовує рухи губ людини, щоб створити пароль. Оскільки ніхто не може імітувати рух губ користувача, коли він вимовляє пароль, технологія допоможе забезпечити подвійну безпеку в ідентифікації та аутентифікації. Система перевіряє особу людини шляхом одночасного зіставлення змісту пароля з базовими поведінковими характеристиками руху губ. Для підвищення рівня безпеки систем можна використовувати пароль для губ у поєднанні з функцією розпізнавання особи.

Для автономного планування й складання розкладів розроблено програму *Remote Agent* агентства NASA, що працює на відстані в сотні мільйонів кілометрів від Землі. Вона стала першою бортовою автономною програмою планування, призначеною для керування процесами складання розкладу операцій для космічного апарата. Ця програма розробляла плани на основі цілей високого рівня, що задаються із Землі, а також контролювала роботу космічного апарата в ході виконання планів: виявляла, діагностувала й усувала несправності в ході їх виникнення.

Іншою галуззю застосування ШІ є ведення ігор. Програма *Deep Blue* компанії *IBM* стала першою комп'ютерною програмою, якій вдалося перемогти чемпіона світу в матчі з шахів, після того як вона обіграла Гаррі Каспарова з рахунком 3.5:2.5 у показовому матчі. Г. Каспаров заявив, що відчував напроти себе за шаховою дошкою присутність «інтелекту нового типу».

Використання ШІ для автономного керування також досить актуальне. Систему комп'ютерного зору *Alvinn* навчили кермувати автомобілем,

дотримуючись визначеної смуги руху. В університеті CMU ця система була розміщена в мікроавтобусі, яким керував комп'ютер *Navlab*, і використовувалася для проїзду Сполученими Штатами. Протягом 4586,6 км система забезпечувала кермування автомобілем під час 98 % часу. Людина брала на себе кермування лише під час інших 2 %, переважно на виїзних пандусах. Комп'ютер *Navlab* був обладнаний відеокамерами, які передавали зображення дороги в систему *Alvinn*, яка обчислювала найкращий напрямок руху, спираючись на досвід, отриманий у попередніх навчальних пробігах.

Не менш успішні розробки в галузі ЕС реального часу, у тому числі з алгоритмом нейромереж. Необхідність їх використання виникає при значному збільшенні кількості правил і висновків. Прикладами реалізації конкретних нейромережевих ЕС можуть бути система вибору повітряних маневрів під час повітряного бою й медична діагностична ЕС для оцінки стану льотчика. Серед спеціалізованих систем, що базуються на знаннях, найбільш значущі ЕС реального часу (динамічні ЕС). На їх частку припадає 70 % розробок у галузі ШІ.

Значущість інструментальних засобів реального часу визначається передусім тим, що лише за допомогою таких засобів створюються стратегічно важливі додатки в таких галузях, як керування безперервними виробничими процесами в хімії, фармакології, виробництві цементу, продуктів харчування тощо, аерокосмічні дослідження, транспортування й переробка нафти та газу, керування атомними й тепловими електростанціями, фінансові операції, зв'язок і багато інших.

Класи задач, які розв'язуються ЕС реального часу, такі:

– моніторинг у реальному часі;
– системи керування верхнього рівня;
– системи виявлення несправностей;
– діагностика;
– складання розкладів, планування;
– оптимізація;
– системи-радники оператора;
– системи проектування.

Серед ЕС поширені медичні діагностичні програми, що ґрунтується на ймовірнісному аналізі. Вони зуміли досягти рівня досвідченого лікаря в декількох галузях медицини. Відомий випадок, коли провідний спеціаліст у галузі патології лімфатичних вузлів не погодився з діагнозом програми в особливо складному випадку. Творці програми запропонували, щоб цей лікар попросив у комп'ютера пояснення із приводу такого діагнозу. Машина вказала основні фактори, що вплинули на її рішення, і пояснила нюанси взаємодії декількох симптомів, що спостерігалися в цьому випадку. Врешті-решт експерт погодився з рішенням програми.

Багато хірургів тепер використовують роботів-асистентів у мікрохірургії. Наприклад, *Hipnav* — це система, у якій застосовуються методи комп'ютерного зору для створення тривимірної моделі анатомії внутрішніх органів пацієнта, а потім застосовується робототехнічне керування для керівництва процесом вставляння протеза, що заміняє тазостегновий суглоб.

Дослідники з університету Бостона і Масачусетського технологічного інституту навчили робота *Baxter*, розробленого в 2012 р., інтерпретувати мозкові хвилі людини в реальному часі, щоб машина могла коректувати свою поведінку при виконанні різних завдань. Дії робота можна змінювати, використовуючи сигнали людського мозку, передані за допомогою електроенцефалографії (ЕЕГ). Щоб керувати роботом, людині потрібно надіти шолом ЕЕГ. Робот *Baxter* може розпізнавати електричні сигнали мозку, що виникають при здійсненні помилкових дій. Під час взаємодії з роботом людина повинна лише в думках погоджуватися або не погоджуватися з виконуваними ним діями — цього досить, щоб робот «усвідомив» помилку.

Розуміння природної мови, реферування текстів та «м'який пошук» — також перспективні напрями розвитку.

Один із найбільш популярних напрямів розвитку систем ШІ останніх років пов'язаний із поняттям автономних агентів. Їх не можна розглядати як «підпрограми», оскільки однією з найважливіших їхніх рис є автономність, незалежність від користувача. Ідея агентів спирається на поняття делегування своїх функцій. Інакше кажучи, користувач повинен довіритися агентові у виконанні певної задачі або класу задач. Автономні агенти дозволяють суттєво підвищити продуктивність роботи при розв'язуванні тих задач, у яких на людину покладається основне навантаження з координації різних дій. Прикладом можуть бути розробки агентів, відповідальних за автоматичне генерування технічної документації.

Підсумовуючи вище викладене, можна зробити висновок, що досягнення в галузі ШІ вже давно мають практичне значення, а їх використання приносить значний економічний ефект. Однак навряд чи у найближчому майбутньому працю людини вдасться замінити роботою ШІ. На сьогодні СШІ досягають якнайкращих результатів, функціонуючи спільно з людиною, оскільки саме людина, на відміну від ШІ, уміє мислити нестандартно і творчо, що дозволяє їй розвиватися. Тому якісна підготовка фахівців у сфері комп'ютерних наук неможлива без вивчення дисципліни «Методи та системи штучного інтелекту». Знання, здобуті в ході вивчення цієї дисципліни, зокрема про моделі подання знань, методи пошуку розв'язків у системах ШІ, технології розробки ЕС, принципи функціонування НМ, можуть використовуватися надалі при вивченні таких дисциплін, як «Проектування баз даних та експертних систем», «Корпоративні інформаційні системи» тощо.

СПИСОК ЛІТЕРАТУРИ

1. *Адаменко А.* Логическое программирование и Visual Prolog / А. Адаменко, А. Кучуков. — СПб. : БХВ-Петербург, 2003. — 982 с.

2. *Борисов В. В.* Нечёткие модели и сети / В. В. Борисов, В. В. Круглов, А. С. Федулов. — М. : Горячая линия — Телеком, 2007. — 284 с.

3. *Боровиков В.* Нейронные сети. Statistica Neural Networks. Методология и технологии современного анализа данных / В. Боровиков. — М. : Горячая линия — Телеком, 2008. — 392 с.

4. *Братко И.* Язык *Prolog* (Пролог): алгоритмы искусственного интеллекта: пер. с англ. / И. Братко. — 3-е изд. — М. : Вильямс, 2004. — 640 с.

5. *Бураков М. В.* Интеллектуальные системы управления: учеб. пособие / М. В. Бураков, О. С. Попов. — СПб. : ГААП., 1997. — 108 с.

6. *Гаврилова Т. А.* Базы знаний интеллектуальных систем / Т. А. Гаврилова, В. Ф. Хорошевский. — СПб. : Питер, 2000. — 384 с.

7. *Джарратано Дж.* Экспертные системы: принципы разработки и программирование / Дж. Джарратано, Г. Райли. — 4-е изд. — М. : Вильямс, 2006. — 1152 с.

8. *Джексон П.* Введение в экспертные системы, 3-е изд. : пер. с англ. / П. Джексон. — М. : Вильямс, 2001. — 624 с.

9. *Искусственный* интеллект: справочник: в 3 кн. / под ред. Э. В. Попова. Кн. 1: Системы общения и экспертные системы. — М. : Радио и связь, 1990. — 462 с.

10. *Каллан Р.* Основные концепции нейронных сетей: пер. с англ. / Р. Каллан. — М. : Вильямс, 2001. — 287 с.

11. *Коста Э.* Visual Prolog 7.1 для начинающих: пер. с англ. / Э. Коста. — М., 2008. — 210 с.

12. *Круглов В. В.* Нечёткая логика и искусственные нейронные сети / В. В. Круглов, М. И. Дли, Р. Ю. Голунов. — М. : ФИЗМАТЛИТ, 2001. — 224 с.

13. *Круглов В. В.* Искусственные нейронные сети. Теория и практика / В. В. Круглов, В. В. Борисов. — 2-е изд., стер. — М. : Горячая линия — Телеком, 2002. — 382 с.

14. *Люгер Джордж Ф.* Искусственный интеллект: стратегии и методы решения сложных проблем. — 4-е изд. : пер. с англ. / Джордж Ф. Люгер. — М. : Вильямс, 2005. — 864 с.

15. *Поспелов Г. С.* Искусственный интеллект — основа новой информационной технологии / Г. С. Поспелов. — М. : Наука, 1988. — 289 с.

16. *Рассел С.* Искусственный интеллект: современный поход. — 2-е изд. : пер. с англ. / С. Рассел, П. Норвиг. — М. : Вильямс, 2016. — 1408 с.

17. *Рутковская Д.* Нейронные сети, генетические алгоритмы и нечёткие системы: пер. с польск. / Д. Рутковская, М. Пилиньский, Л. Рутковский. — М. : Горячая линия — Телеком, 2006. — 452 с.

18. *Рутковский Л.* Методы и технологии искусственного интеллекта / Л. Рутковский. — М. : Горячая линия — Телеком, 2010. — 520 с.

19. *Хайкин С.* Нейронные сети: полный курс: пер. с англ. / С. Хайкин. — 2-е изд. — М. : Вильямс, 2008. — 1103 с.

20. *Цуканова Н. И.* Логическое программирование на языке Visual Prolog: учеб. пособие для вузов / Н. И. Цуканова, Т. А. Дмитриева. — М. : Горячая линия — Телеком, 2008. — 144 с.

21. *Частиков А. П.* Разработка экспертных систем. Среда CLIPS / А. П. Частиков, Д. Л. Белов, Т. А. Гаврилова. — СПб. : БХВ-Петербург, 2003. — 608 с.

22. Штовба С. Д. Проектирование систем средствами MATLAB / С. Д. Штовба. — М. : Горячая линия — Телеком, 2007. — 288 с.

ЗМІСТ

Навчальне видання

САВЧЕНКО Аліна Станіславівна
СИНЕЛЬНІКОВ Олексій Олексійович

МЕТОДИ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Навчальний посібник

Редактор *Л. М. Дудченко*
Технічний редактор *А. І. Лавринович*
Комп'ютерна верстка *Л. Т. Колодіної*

Підп. до друку 02.06.2017. Формат 60×84/16. Папір офс.
Офс. друк. Ум. друк. арк. 10,23. Обл.-вид. арк. 11,0.
Тираж 100 пр. Замовлення № 81-1.

Видавець і виготівник
Національний авіаційний університет
03680. Київ-58, проспект Космонавта Комарова, 1

Свідоцтво про внесення до Державного реєстру ДК № 977 від 05.07.2002