Комп'ютинг

В.В. Литвин, В.В. Пасічник,
Ю.В. Яцишин

ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ

Підручник

Computing

Міністерство освіти і науки України

В.В. Литвин, В.В. Пасічник, Ю.В. Яцишин

ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ

Підручник

Серія «Комп'ютинґ»

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

Затверджено Міністерством освіти і науки України

Видавництво "Новий Світ – 2000"
Львів – 2009

УДК 004.89 (075.8)
ББК 32.813я73
Л 64

Гриф надано Міністерством освіти і науки України

(Лист № 1.4/18-Г-1079 від 10.07. 2008 р.)

Рецензенти:

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

Б. П. Русин – доктор технічних наук, професор, завідувач відділу фізико-механічного інституту імені В. Г. Карпенка НАН України;

Я.М.Матвійчук – доктор технічних наук, професор, завідувач кафедри інформаційних систем та технологій інституту підприємництва та перспективних технологій при Національному університеті „Львівська політехніка".

Литвин В.В., Пасічник В.В., Яцишин Ю.В.

Л 64 Інтелектуальні системи: Підручник – Львів: "Новий Світ – 2000", 2009 – 406 с.

ISBN 978-966-418-086-0

У підручнику розглядаються основні поняття, методи та моделі побудови інтелектуальних систем. Особливістю викладення є практична спрямованість: оволодіння поданим матеріалом достатнє для самостійного розроблення інтелектуальних систем.

Окрім розділів: фундаментальні проблеми інтелектуальних систем, моделі та методи функціонування інтелектуальних систем, подання знань та доведень, онтологія та онтологічні системи, інтелектуальні агенти та мультиагентні системи, машинне навчання та нейронні мережі, які традиційно належать до курсу з інтелектуальних систем, що є продовженням курсу «Системи штучного інтелекту», розглянуто сучасні теоретичні та практичні аспекти інженерії знань, а також прикладне використання інтелектуальних систем. Підручник створений з урахуванням досвіду, набутого під час опрацювання подібних вітчизняних та іноземних видань.

Рекомендований для бакалаврів, які навчаються за напрямом «Комп'ютерні науки», та магістрів, що освоюють спеціальності, які базуються на такому бакалавраті. Підручник буде корисний також усім, хто цікавиться питаннями розроблення інтелектуальних систем.

УДК 004.89 (075.8)
ББК 32.813я73

ISBN 978-966-418-086-0

Зміст

Передмова наукового редактора серії підручників та навчальних посібників «КОМП'ЮТИНҐ»

Шановний читачу!

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

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

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

Читач, безсумнівно, зверне увагу на нашу послідовну промоцію нового терміну «КОМП'ЮТИНҐ» (computing, англ.), який є вдалим та комплексно уза-гальнювальним для означення галузі знань, науки, виробництва, надання відповідних послуг та сервісів. видається доречним подати ретроспективу як самого терміну комп'ютинг, так і широкої освітньої, наукової, бізнесової та виробничої сфери діяльності, що іменується комп'ютингом.

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

У подальшому сфера використання терміну суттєво розширилась, зокрема, в освітньо-науковій царині його почали використовувати для означення відповідної галузі

знань, для якої періодично (орієнтовно щодесять років) провідними університетами та професійними асоціаціями фахівців розроб-ляються та імплементуються навчальні плани і програми, котрі в подальшо-му набувають статусу міжнародно визнаних освітньо-професійних стандартів. Зокрема, варто акцентувати увагу на версіях підсумкового документу "Computing CURRICULA" 2001 року. За окремими повідомленнями можна стверджувати, що черговий збірник стандартів "Computing CURRICULA" буде поданий професійному загалу до 2011 року. Перше організаційне засідання відповідних фахових робочих груп відбулось у Чікагському університеті влітку 2007 року.

Для формування цілісного однорідного подання суті КОМП'ЮТИНҐУ ми базуємось на сучасних наукових уявленнях з максимально можливим строгим покомпонентним викладенням основних базових означень та понять, які скла-лись історично і є загальновизнаними в професійних колах. Водночас для побудови цілісної зваженої картини ми використали певні узагальнення та загальносистемні класифікаційні підходи.

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

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

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

Третьою прикладною науковою галуззю, яка ґрунтується на фундаментальних законах інформатики, є розділ прикладних наук, основним об'єктом яких є інформаційний ресурс

(у сучасній літературі часто вживається поняття «контент» (content, англ.). У розумінні інформаційного наповнення. Ці прикладні науки одержали назву «інформаційні науки» (information science, англ.).

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

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

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

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

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

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

У зв'язку з поширеним різночитанням та трактуванням поняття інформацій-ної технології (ІТ), видається необхідним детальніше подати сутнісну структуру цього терміну, використовуючи при цьому термінологічні статті популярного інформаційного ресурсу, яким є Wikipedia – [http://www.wikipedia.org/].

Технологія (від грецького techne – мистецтво, майстерність, вміння та грецького logos – знання) – сукупність методів та інструментів для досягнення бажаного результату,

спосіб перетворення чогось заданого в необхідне. Технологія – це наукова дисципліна, в рамках якої розробляються та удоско-налюються способи й інструменти виробництва.

У широкому розумінні – це знання, які можна використати для виробництв-ва продуктів (товарів та послуг) з економічних ресурсів. У вузькому розумінні – технологія подається як спосіб перетворення речовини, енергії, інформації в процесі виготовлення продукції, обробки та переробки матеріалів, складання готових виробів, контроль якості та керування.

Технологія включає в себе методи, прийоми, режими роботи, послідовість операцій та процедур. Вона тісно взаємопов'язана із засобами, що застосовуються, обладнанням, інструментами, використовуваними матеріалами. За методологією ООН – технологія в чистому вигляді охоплює методи та техніку виробництва товарів і послуг (dissembled technology, англ.). Втілена технологія охоплює машини, обладнання, споруди, виробничі системи та продукцію з високими техніко-економічними параметрами (embodied technology, англ.). Матеріальна технологія (МТ) створює матеріальний продукт. Інформаційна технологія (ІТ) створює інформаційний продукт на основі інформаційних ресурсів.

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

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

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

Комплексний виклад структурованого подання галузі КОМП'ЮТИНҐУ дозволяє, загалом, чіткіше уявити проблематику та тематику підручників, котрі будуть виходити в світ у однойменній освітньо-науковій серії в 50-ти книгах. Для кращого розуміння в майбутньому ще раз наведемо означення сфери КОМП'ЮТИНҐУ як галузі знань (науки, виробництва, бізнесу та надання послуг), предметом якої є комплексні дослідження, розроблення, впровадження та використання інформаційних систем, складовими елементами яких є інформаційні технології, що реалізовані на основі сучасних інженерних досягнень комп'ютерної інженерії, інженерії програмного забезпечення, інженерії даних та знань, системної інженерії, котрі базуються на фундаментальних законах та закономірностях інформатики.

Автори підручників і навчальних посібників серії «КОМП'ЮТИНҐ» пропонують значний перелік навчальних дисциплін, котрі, з одного боку, включаються до сфери комп'ютинґу за означенням, а, з іншого боку, їх предмет ще не знайшов якісного

висвітлення у вітчизняній навчальній літературі для вищої школи. Перший крок ми робимо у 2008 році, виданням принаймні десяти книг серії з подальшим її п'ятикратним розширенням до 2011 року. Структурно серія подається узагальненими профілями як то:

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

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

З глибокою повагою науковий редактор серії підручників і навчальних посібників «КОМП'ЮТИНГ», д.т.н., професор Володимир Пасічник.

Передмова

Шановний читачу, подаючи на Твій критичний огляд результати нашої колективної праці сподіваюсь на взаєморозуміння та активну співпрацю. Безсумнівним є той факт, що підгалузь знань, в якій ми презентуємо підручник, а саме – підгалузь інтелектуальних систем є чи не найдинамічнішою в царині наукового та виробничого напряму, який все частіше серед фахівців називається комп'ютингом (computing – англ.).

Хочемо спочатку зафіксувати базові позиції та мотиви, які слугували нам основними засадними принципами під час роботи над рукописом пропонованого підручника.

По-перше, започатковуючи проект, автори в повній мірі усвідомлювали велику складність та надвеликі обсяги інформаційних матеріалів, які видруковуються та з'являються на книжкових полицях маркетів, як то електронних так і традиційно - звиклих книгарень та книгозбірень.

По-друге, ми зразу розуміли, що наш творчий крок слід розуміти, виключно, як чергове наближення до побудови цілісного комплексного казкового уявлення про актуальний стан, що склався в підгалузі знань, котра іменується як "інтелектуальні системи".

По-третє, автори з великою уважністю знайомились з набутим досвідом (відверто, не завжди вдалим та успішним), який зафіксований як результат аналогічних спроб інших авторських колективів та висвітлення результатів досліджень інших наукових шкіл та напрямків.

По-четверте, автори, під час підготовки рукопису, сповідували ідею прикладання теоретичних знань та набутків до вирішення конкретних, прорахованих практикою, завдань. Розглядають вдалі та не зовсім вдалі реалізації інтелектуальних систем. Ми завжди пам'ятали, що критерієм істини є і залишається практика, а тому в підручнику поміщено достатньо прикладів з реального життя та завдань, які при їх вирішенні постійно вимагають зведення до практичного розроблення елементів та підсистем інтелектуальних систем конкретного прикладного спрямування.

Підручник складається з 11-и розділів та 4-х додатків. У кінці кожного розділу наведено перелік запитань для самоконтролю. У першому розділі розглянуті фундаментальні проблеми розроблення інтелектуальних систем. Другий розділ присвячено моделям та методам функціонування інтелектуальних систем. У третому розділі розглянуто методи подання знань, а в четвертому ускладнене подання знань з врахуванням фактору нечіткості. Теоретичні, технологічні та прикладні аспекти інженерії знань описано у наступних трьох розділах. У восьмому розділі розглянуто методи машинного навчання та нейронні мережі. Поняттю онтологій та онтологічних систем присвячено дев'ятий розділ. У десятому розділі розглянуто поняття інтелектуальних агентів та мультиагентних систем. Одинадцятий розділ присвячено опису прикладного застосування інтелектуальних систем під час побудови інтелектуальних пошукових систем та розроблення систем опрацювання природньої мови. У додатках коротко подано описи сучасних програмних засобів побудови інтелектуальних систем (CLIPS, newLISP, Python), і програмний засіб аналізу нейронних мереж – NNML.

За структурою, змістом та формою подання матеріалу курси за підручником можуть викладатися для студентів, котрі навчаються за програмами бакалавратів галузі знань "Інформатика та обчислювальна техніка", а саме "комп'ютерні науки", "комп'ютерна інженерія" та "програмна інженерія".

Безсумнівно, що найоб'єктивнішу оцінку роботи та якості втілення задекларованих нашим авторським колективом базових принципів закладених в основу підручника можеш і маєш право виставити лише Ти, вельмишановний читачу. При цьому зазначимо, що образ читача в нашому уявленні це збірний образ студента, майстра, аспіранта, фахівця в галузі комп'ютингу, який опановує та знайомиться з предметом інтелектуальних систем.

Ми будемо вдячні всім хто знайде за можливе висловити свою думку за структурою змісту та формою подання матеріалу підручника, що в свою чергу слугуватиме добрим підґрунтям подальшої роботи по вдосконаленню, надіємось на співпрацю та активний діалог, тільки такий шлях має перспективу та успіх.

З повагою автори.

РОЗДІЛ 1

ФУНДАМЕНТАЛЬНІ ПРОБЛЕМИ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

- Поняття інтелектуальних систем
- Експертні системи

Розглядається поняття інтелектуальних систем, їх відмінність від традиційних систем. Розглядаються різноманітні задачі, для розв'язування яких необхідна побудува інтелектуальних систем. Детально розглядається підклас таких систем, що часто використовуються в прикладних задачах – експертні системи.

1.1. Поняття інтелектуальних систем

1.1.1. Історія штучного інтелекту

Інтуїтивне розуміння поняття «інтелект»

З давніх-давен людині були необхідні помічники для полегшення виконання тих чи інших операцій. Були винайдені різноманітні механізми, машини тощо. Поява електронно-обчислювальних машин дала змогу автоматизувати виконання трудомістких розрахункових робіт. Згодом стало зрозуміло, що ці машини можна використовувати не тільки для обчислень, але й для керування різними пристроями, складними автоматизованими виробництвами тощо. Значного поширення набули роботи – програмно керовані пристрої, здатні безпосередньо взаємодіяти з фізичним світом та виконувати в ньому певні дії [81]. Такі роботи широко використовуються у виробництві. Вони можуть працювати і в умовах, які людина не в змозі витримати – на дні океану, всередині ядерного реактора тощо. Але такі помічники вимагають автоматизованого керування, тобто їм постійно потрібно «підказувати», що і як вони повинні робити в конкретній ситуації, що склалася на певний момент.

Природна мова на сучасному етапі малопридатна для цього через свою складність та неоднозначність. Одним зі шляхів вирішення цього завдання є формулювання інструкцій мовою, зрозумілою виконавцю, тобто написання програм. Програмування полягає у перекладі інструкцій, написаних мовою, близької до природної, на мову, яку здатна сприйняти обчислювальна система. Відомі складності сучасного програмування, пов'язані з необхідністю надмірної алгоритмізації, тобто детального ретельного розписування інструкцій з урахуванням усіх можливих ситуацій. Із цієї ситуації існує єдиний вихід – підвищення рівня «розумності», інтелектуальності сучасних комп'ютерів та роботів. Постає запитання: що розуміється під такими поняттями, як «інтелект», «інтелектуалізація», «штучний інтелект»?

Термін «інтелект» (intelligence) походить від латинського intellectus – що означає розум, розумові здібності людини. Відповідно штучний інтелект (artificial intelligence)

ШІ (AI) зазвичай визначається як властивість автоматизованих систем брати на себе окремі функції інтелекту людини, наприклад, вибирати і приймати оптимальні рішення на основі раніше отриманого досвіду і раціонального аналізу зовнішніх дій.

Інтелектом називають здатність мозку вирішувати інтелектуальні завдання шляхом придбання, запам'ятовування і цілеспрямованого перетворення знань у процесі навчання на основі отриманого досвіду й адаптації до різноманітних обставин.

У цьому визначенні під терміном «знання» розуміють не тільки ту інформацію, яка надходить у мозок через органи чуття. Такого типу знання надзвичайно важливі, але недостатні для інтелектуальної діяльності. Річ у тому, що об'єкти навколишнього середовища мають властивість не тільки впливати на органи чуття, але й перебувати один з одним у певних відношеннях. Зрозуміло, що для того, щоб здійснювати в навколишньому середовищі інтелектуальну діяльність (або хоч би просто існувати), потрібно мати в системі знань модель цього світу. У цій інформаційній моделі навколишнього середовища реальні об'єкти, їх властивості й відношення між ними не тільки відображаються і запам'ятовуються, але і, як це відзначено в поданому визначенні інтелекту, можуть «цілеспрямовано перетворюватися». При цьому істотно те, що формування моделі зовнішнього середовища відбувається «у процесі навчання на основі отриманого досвіду й адаптації до різноманітних обставин» [58].

Діяльність мозку, якому властивий інтелект, спрямована на розв'язання інтелектуальних задач, ми називатимемо мисленням, або інтелектуальною діяльністю. Інтелект і мислення органічно поєднані в розв'язуванні таких задач, як доведення теорем, логічний аналіз, розпізнавання ситуацій, планування поведінки, ігри і керування в умовах невизначеності. Характерними рисами інтелекту, що виявляються у процесі розв'язування задач, є здібність до навчання, узагальнення, накопичення досвіду (знань і навиків) і адаптації до умов, що змінюються. Завдяки цим якостям інтелекту мозок може розв'язувати різноманітні задачі, а також легко перебудовуватися з розв'язання однієї задачі на іншу. Отже, мозок, наділений інтелектом, є універсальним засобом рішення широкого кола завдань (зокрема неформалізованих), для яких немає стандартних, наперед відомих методів рішення.

Треба мати на увазі, що існують інші, чисто поведінкові (функціональні) визначення. Так, за А. Н. Колмогоровим, будь-яка матеріальна система, з якою можна достатньо довго обговорювати проблеми науки, літератури і мистецтва, має інтелект. Іншим прикладом поведінкового трактування інтелекту може служити відоме визначення А. Тюринґа. Його сенс полягає в наступному. У різних кімнатах розташовані люди і машина. Вони не можуть бачити один одного, але мають можливість 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 такою, що має інтелект.

До речі, цікавий план імітації мислення, запропонований А. Тюрингом. «Намагаючись імітувати інтелект дорослої людини, – пише Тюринґ, – ми вимушені багато роздумувати про той процес, в результаті якого людський мозок досяг свого справжнього стану. Чому б нам замість того, щоб намагатися створити програму, що імітує інтелект дорослої людини, не спробувати створити програму, яка імітувала б інтелект дитини? Адже, якщо інтелект дитини отримує відповідне виховання, він стає інтелектом дорослої людини. Наш розрахунок полягає в тому, що пристрій, подібно до виховання, може бути

легко запрограмований. Таким чином ми розбиваємо нашу проблему на дві частини: на завдання побудови «програми-дитини» і завдання «виховання цієї програми» [215].

Забігаючи вперед, можна сказати, що саме цей шлях використовують практично всі системи ШІ. Адже зрозуміло, що практично неможливо закласти всі знання в достатньо складну систему. Крім того, тільки на цьому шляху виявляться перераховані вище ознаки інтелектуальної діяльності (накопичення досвіду, адаптація тощо).

Можна стверджувати, що «штучний» інтелект у тому чи іншому розумінні повинен наближатися до інтелекту природного і у ряді випадків використовуватися замість нього; так само, як, наприклад, штучні нирки працюють замість природних. Чим більше буде ситуацій, у яких штучні інтелектуальні системи зможуть замінити людей, тим інтелектуальнішими будуть вважатися ці системи.

Навряд чи є сенс протиставляти поняття штучного інтелекту й інтелекту взагалі. Тому треба спробувати визначити поняття інтелекту, незалежно від його походження.

Людина вважається інтелектуальною «від природи», і цей інтелект був вироблений у процесі еволюції протягом мільйонів років. Людина вміє вирішувати багато інтелектуальних задач. Кожна людина вважає, що вона розуміє значення слова «інтелект», але якщо попросити дати визначення цього слова, то переважно чіткої відповіді не буде. І дійсно, дати визначення поняття інтелекту, яке б задовольняло всіх, очевидно, неможливо. Далі будуть проаналізовані деякі спроби визначення цього поняття та їх критика. Але відсутність чіткого визначення не заважає оцінювати інтелектуальність на інтуїтивному рівні. Можна навести як мінімум два методи такого оцінювання: метод експертних оцінок і метод тестування [148].

При застосуванні методу експертних оцінок рішення про ступінь інтелектуальності приймає численна група експертів (незалежно або у взаємодії між собою); відомо багато способів організації взаємодії між експертами.

При застосуванні методу тестування пропонується розв'язати ті чи інші тестові завдання. Існує значна кількість апробованих інтелектуальних тестів для оцінки рівня розумових здібностей людини, що знайшли застосування у психології та психіатрії [3, 62, 77]. Наведемо декілька прикладів.

Приклад 1.1. Вставте число, яке пропущене:
36 30 24 18 6

Приклад 1.2. Викресліть зайве слово:
лев лисиця жираф щука собака

Серед психіатрів інколи можна почути вислів: «він міркує логічно вірно, але неправильно» [152]. Наприклад, дається тестове завдання: «серед слів соловей, чапля, перепілка, стілець, шпак виділити зайве». Більшість людей, не задумуючись, дає відповідь стілець, тому що всі інші слова – це назви птахів. І раптом хтось дає відповідь шпак, пояснюючи це тим, що це єдине слово, в якому відсутня літера «л». Обидві класифікації є логічно вірними і формально рівноправними. Але чому ж перевага віддається одній з них? Тому, що так міркує більшість людей. А чому так міркує більшість людей? Очевидно, тому, що перша класифікація вважається важливішою для практичної діяльності людини. Це положення приймається на аксіоматичному рівні, без доведення (а запропонувати щось інше, очевидно, неможливо).

Наголосимо, що поняття «штучний інтелект» не можна зводити лише до створення пристроїв, які повністю або частково імітують діяльність людини. Не менш важливим

є інше завдання: виявити механізми, які лежать в основі діяльності людини, щоб застосувати їх під час розв'язування конкретних науково-технічних задач. І це лише одна з можливих проблем.

Деякі інтелектуальні задачі

Розглянемо і проаналізуємо в загальних рисах деякі проблеми, які доводиться постійно вирішувати людському розумові: розпізнавання образів, мислення та обчислювальні задачі.

На інтуїтивному рівні можна сформулювати декілька типових задач розпізнавання образів (або просто розпізнавання):

- задача **ідентифікації** полягає в тому, що об'єкт, який спостерігається людиною, потрібно вирізнити серед інших (наприклад, побачивши іншу людину, впізнати в ній свою дружину);
- проблема **розпізнавання** в класичній постановці: визначити належність об'єкту, що спостерігається, до одного із заздалегідь відомих класів об'єктів (наприклад, відрізнити легковий автомобіль від вантажного). Таке застосування терміну «ідентифікація» не є єдино можливим. Цей термін часто застосовується в іншому значенні, а саме: знаючи вхідні та вихідні сигнали деякої системи, визначити її структуру.

Людина робить класифікацію просто. Наприклад, чоловік, повернувшись додому з роботи, відразу ж пізнає свою дружину, але більшість людей в повному обсязі не зможе пояснити, як він це робить. Як правило, раціонального пояснення немає. Теорія розпізнавання, яка інтенсивно розвивається, необхідна для того, щоб навчити штучні інтелектуальні системи розв'язувати задачі розпізнавання на основі досвіду розпізнавання людиною. Зокрема, сформульовано такий ключовий принцип [178]: будь-який об'єкт у природі – унікальний; унікальні об'єкти – типізовані. Відповідно до цього принципу, розпізнавання здійснюється на основі аналізу певних характерних ознак. Вважається, що в природі не існує двох об'єктів, для яких співпадають абсолютно всі ознаки, і це теоретично дозволяє здійснювати ідентифікацію. Якщо ж для деяких об'єктів співпадають деякі ознаки, ці об'єкти теоретично можна об'єднувати в групи або класи за цими спільними ознаками.

Проблема полягає в тому, що різноманітних ознак існує незліченна кількість. Незважаючи на легкість, з якою людина розпізнає, вона дуже рідко в змозі виділити суттєві ознаки. До того ж, об'єкти, як правило, змінюються з часом.

Ми далі спробуємо показати, що розпізнавання об'єктів і ситуацій має виняткове значення для орієнтації людини в навколишньому світі та для прийняття вірних рішень. Розпізнавання, як правило, здійснюється людиною на інтуїтивному, підсвідомому рівні.

Інша інтелектуальна задача – моделювання мислення, зокрема, виведення наслідків із фактів, які безпосередньо спостерігаються або вважаються відомими. Можна виділити два типи процесів мислення:

- підсвідоме, інтуїтивне мислення, механізми якого на сучасному етапі вивчені недостатньо, і яке дуже важко формалізувати та автоматизувати;
- дедуктивні логічні побудови за формалізованими законами логіки.

Дедукцією називається перехід від загального до часткового, виведення часткових наслідків із загальних правил. І тут пересічна людина рідко в змозі пояснити, за якими алгоритмами вона здійснює логічне виведення. Але методики та алгоритми, за якими

можна автоматизувати виведення наслідків із фактів або логічну перевірку тих чи інших фактів, досить відомі.

Перш за все, це формальна логіка Аристотеля на основі конструкцій, які отримали назву силогізмів. Вони практично неподільно панували в логіці аж до початку XIX століття, тобто до появи булевої алгебри. Немає потреби давати якісь формальні визначення силогізмів. Наведемо лише один класичний приклад.

Пе́рше твердження. Усі люди смертні.
Друге твердження. Сократ – людина.
Висновок: Сократ смертний.

Якщо перше та друге твердження у силогізмі істинні та задовольняють певним загальним формальним вимогам, тоді і висновок буде істинним незалежно від змісту тверджень, що входять до силогізму. При порушенні цих формальних вимог легко припуститися логічних помилок, подібних до таких:

Всі студенти інституту А знають англійську мову.
Петров знає англійську мову.
Отже, Петров – студент інституту А.

Або:

Іванов не готувався до іспиту і отримав двійку.
Сидоров не готується до іспиту.
Отже, і Сидоров отримає двійку.

Аристотелем було запропоновано декілька формальних конструкцій силогізмів, які він вважав достатньо універсальними. Лише у XIX столітті почала розвиватися сучасна математична логіка, яка розглядає силогізми Аристотеля як один із часткових випадків. Основою більшості сучасних систем, призначених для автоматизації логічних побудов, є метод резолюцій Робінсона [127]. Але практична реалізація логічних побудов зіткнулася із серйозними проблемами. Головна з них – це феномен, який Річард Беллман назвав прокляттям розмірності. На перший погляд, описати знання про зовнішній світ можна було б, наприклад, таким чином: «об'єкт А має риси X, Y, Z. В відрізняється від А тим, що має рису H, тощо». Але зовнішній світ є винятково складним переплетінням різноманітних об'єктів та зв'язків між ними. Щоб лише ввести всю цю інформацію до пам'яті інтелектуального пристрою, може знадобитися не одна тисяча років. Ще більше часу буде потрібно, щоб врахувати всі необхідні факти при логічному виведенні. Реальні програми, що здійснюють логічне виведення (вони часто називаються експертними системами) мають досить обмежене застосування. Вони мають обмежений набір фактів та правил з певної, більш-менш чітко окресленої та предметної галузі й можуть використовуватися лише у цій галузі.

Що ж стосується людини, то якість її логічного мислення також часто буває далекою від бездоганної. Люди часто роблять логічні помилки, а інколи взагалі керуються принципами, невірними з точки зору нормальної логіки. Дуже часто свідоме логічне виведення на певному етапі обривається, і рішення знову ж таки приймається на підсвідомому, інтуїтивному рівні. Зрозуміло, що таке рішення може бути помилковим. Але, якби в основі поведінки людини лежали спроби здійснювати дедуктивні побудови від логічного початку до логічного кінця, людина була б практично нездатною до будь-якої діяльності: фізичної або розумової – це вимагало б значного часу аналізу.

Спільною рисою згаданих вище проблем була їх погана формалізованість, відсутність або незастосовність чітких алгоритмів розв'язку. Вирішення подібних задач і є основним предметом розгляду в теорії штучного інтелекту.

До зовсім іншого класу відносяться задачі, пов'язані з обчисленнями. Зазвичай важко відповісти на запитання, як саме людина здійснює ті чи інші обчислення. Добре відомими є і низька швидкість, і невисока надійність цього виду людської діяльності. Але були запропоновані ефективні принципи комп'ютерних обчислень, які радикально відрізняються від тих, що застосовуються людиною. Ці добре формалізовані алгоритмічні методики забезпечили рівень розв'язування обчислювальних задач, абсолютно недосяжний для людського інтелекту. Водночас цей високий рівень алгоритмізації значною мірою зумовив слабкість традиційних комп'ютерних систем при розв'язанні тих інтелектуальних задач, з якими людина справляється непогано.

З появою таких обчислювальних потужностей мрійники шістдесятих-сімдесятих років XX століття ставили задачу моделювання в повному обсязі роботи людського мозку. Теоретично цю задачу з певними обмеженнями можна було б вирішити. Але складність потрібних обчислень виявилася такою, що змусила більшість дослідників відійти від поставленої задачі й перейти до простіших задач – моделювання принципів роботи людського мозку при розв'язуванні конкретно визначених типів задач.

Деякі визначення та їх критика

Зроблено чимало спроб дати формальне визначення поняття «інтелекту», зокрема, «штучний інтелект». Найвідомішим є визначення предмету теорії штучного інтелекту, яке введене видатним дослідником у галузі штучного інтелекту Марвіном Мінським [118]. Воно потрапило до багатьох словників та енциклопедій з невеликими змінами і відображає таку основну думку: «штучний інтелект є дисципліна, що вивчає можливість створення програм для вирішення задач, які при вирішенні їх людиною потребують певних інтелектуальних зусиль». Але і це визначення має вади. Головна з них полягала в поганій формалізації поняття «певні інтелектуальні зусилля». «Певних інтелектуальних зусиль» вимагає, наприклад, виконання простих арифметичних операцій, але чи можна вважати інтелектуальною програму, яка здатна до виконання тільки таких операцій? Відтак у ряді книг та енциклопедій до наведеного визначення додається поправка: «сюди не входять задачі, для яких відома процедура їх розв'язування». Важко вважати таке формулювання задовільним. Розвиваючи цю думку далі, можна було б продовжити: отже, якщо я не знаю, як розв'язувати деяку задачу, то вона є інтелектуальною, а якщо знаю – то ні. Наступний крок – це відомі слова Л. Теслера: «Штучний інтелект – це те, чого ще не зроблено» [121]. Цей парадоксальний висновок лише підкреслює дискусійність проблеми.

Є деякі конструктивніші визначення інтелекту. Наприклад, в [189] наводиться одне з них: «інтелект є здатністю правильно реагувати на нову ситуацію». Там же наводиться і критика цього визначення: не завжди зрозуміло, що слід вважати новою ситуацією. Уявіть собі, наприклад, звичайний калькулятор. Цілком імовірно, що на ньому ніколи не обчислювали суму двох нулів. Тоді завдання «обчислити нуль плюс нуль» можна вважати ситуацією, новою для калькулятора. Безумовно, він з нею впорається («правильно відреагує на нову ситуацію»), але чи можна на цій підставі вважати його інтелектуальною системою?

Тест Тюринґа і фатичний діалог

Відомий англійський учений Алан Тюринг сформулював тезу, спрямовану на визначення моменту, з якого машину можна вважати інтелектуальною [147]. Нехай експерт за допомогою телефону або подібного віддаленого пристрою спілкується з об'єктом, яким може бути як людина, так і машина. Експерт дає певні тести-завдання. За результатами відповідей він повинен визначити, з ким він має справу – з людиною чи з ЕОМ. Якщо він приймає комп'ютер за людину, комп'ютер може вважатися інтелектуальним. Така перевірка дістала назву тесту Тюринґа. Багато спеціалістів вважали, що тест Тюринґа є цілком задовільним для визначення рівня інтелектуальності комп'ютерної системи. Але виявилось, що це не зовсім так.

В основі тесту Тюринґа лежить неявне припущення про те, що необхідною умовою ведення діалогу є розуміння співрозмовника. Але у кінці 60-х рр. минулого сторіччя американський кібернетик Дж. Вейценбаум створив дві програми – ЕЛІЗА і ДОКТОР – для використання у психіатрії. Виявилось, що вони можуть невірно зорієнтувати експерта в умовах тесту Тюринґа. В основі побудови цих програм лежить ідея фатичного діалогу.

Фатичним діалогом називається діалог без розуміння співрозмовника на основі формального перефразування почутого.

Так, наприклад, на висловлення пацієнта «Ви не відверті зі мною» програма може відповісти: «Чому Ви думаєте, що я не відверта з Вами?». У ряді випадків програма може видати заздалегідь заготовлену фразу. Так, на слова пацієнта «Мій батько мене не любив» вона може відповісти: «Розкажіть мені про Вашу сім'ю».

Звичайно, це не має ніякого відношення ні до розуміння, ні до інтелекту, але багато людей, які здійснювали із цими програмами тест Тюринґа, вирішили, що мають справу з людиною.

У роботі [152] наводиться курйозний приклад фатичного діалогу. Віце-президент великої американської компанії захотів зв'язатися з відомим системним програмістом Д. Бобровим. Відбулась така телетайпна розмова:

Віце-президент: Чи можу я зараз скористатись комп'ютером?
Бобров: А чому Ви питаєте?
Віце-президент: Може мені вдасться підписати контракт.
Бобров: Чому Ви в цьому впевнені?
Віце-президент: Мої потенційні покупці ніколи не бачили систему в дії.
Бобров: Це має для Вас велике значення?
Віце-президент: Звичайно.
Бобров: Ви маєте в цьому сумнів?
Віце-президент: Що за безглузде зауваження?
Бобров: Що Ви маєте на увазі, кажучи про безглузде зауваження?
Віце-президент: Чи можу я зараз скористатись комп'ютером?
Бобров: Розкажіть мені ще про Ваших потенційних покупців.
Віце-президент: Зателефонуйте мені 491-1850.

Віце-президент був здивований, тим більше, що дзвінка не було. Непорозуміння з'ясувалося пізніше. Виявилось, що телетайп з'єднав віце-президента не із самим Бобровим, а з програмою, яка реалізувала фатичний діалог.

Звичайно, подібний діалог важко вважати особливо осмисленим, але хто може дати чітке визначення рівня осмисленості? Фатичний діалог може бути і менш примітивним, але системи, які здатні тільки на підтримку такого діалогу, не можна вважати інтелектуальними. Розуміння співрозмовника є абсолютно необхідним для нормального діалогу, а для такого розуміння необхідно мати певну суму знань про світ.

Метод комп'ютерної реалізації фатичного діалогу

В основі псевдоінтелектуальних програм, які можуть підтримувати фатичний діалог з людиною або з іншими аналогічними програмами, можуть лежати дуже прості алгоритми співставлення зі зразками. Нижче описується класичний метод, що дозволяє писати подібні програми [170].

Розглянемо «анатомію» програми ДОКТОР у дещо спрощеному вигляді. Діалог із цією програмою може бути, наприклад, таким:

ДОКТОР: Я Вас слухаю.
ПАЦІЄНТ: Мене бентежить погода.
ДОКТОР: Як давно Вас бентежить погода?
ПАЦІЄНТ: Відтоді, як померла моя мати.
ДОКТОР: Розкажіть про Вашу сім'ю.
ПАЦІЄНТ: Всі вони захоплюються комп'ютерами.
ДОКТОР: Машини Вас лякають.
ПАЦІЄНТ: Ні.
ДОКТОР: Будь ласка, не будьте так небагатослівні... І так далі до нескінченності.

В основі методу, необхідного для реалізації такого діалогу, може лежати механічне порівняння речень, які вводяться людиною, зі зразками (шаблонами) речень, що зберігаються програмою. Якщо речення співпало в повній мірі з одним із зразків, видається одна з відповідей, заздалегідь підготовлених для цього зразка.

Можна розглянути декілька варіантів порівнянь, кожний з них може бути легко запрограмований.

Варіант 1 (повний збіг). Якщо речення, що вводиться, повністю збігається з одним із зразків, може бути відповідь: «Так, Ви маєте рацію», або навпаки: «Ви помиляєтесь, тому що...», і після «рацію» або «тому що» програміст може написати будь-який текст, що імітує глибоке розуміння специфіки предметної області. Наприклад, дуже непогано буде виглядати такий діалог:

Людина: Квадрат гіпотенузи дорівнює сумі квадратів катетів.

Програма: Так, але є подібний результат і для непрямокутних трикутників; це теорема косинусів.

Навряд чи після декількох подібних відповідей у когось залишаться сумніви в інтелектуальних здібностях програми. Але зазвичай повні збіги бувають дуже рідко. Тому доводиться використовувати інші типи порівнянь.

Варіант 2 (використання замінювачів). Типовим є використання замінювачів * і ?. Із замінювачем * співставляється довільний фрагмент тексту, із замінювачем ? співставляється будь-яке окреме слово.

Наприклад, шаблон (* комп'ютери *) успішно співставляється з будь-яким реченням, в якому згадується про комп'ютери; шаблон (Я люблю? яблука) – з такими реченнями, як (Я люблю червоні яблука), (Я люблю солодкі яблука) тощо, але не з реченням (Я люблю їсти зелені яблука).

Варіант 3 (надання значень змінним у процесі співставлення). При цьому можливості програми, що реалізує фатичний діалог, значно розширюються. Вона набуває здібності до генерації відповідей, що залежать від запитань. Так, правило

$$(\text{Я ? *}) \xrightarrow{a:=?} (\text{Що ви ще <a> ?})$$

дозволяє на речення Я люблю яблука відповісти Що Ви ще любите?, а на речення Я ненавиджу дощі – Що Ви ще ненавидите? У цьому прикладі при успішному співставленні змінній а надається значення слова, з яким співставився замінювач? Безумовно, при використанні українських фраз замість англійських потрібно ще стежити за узгодженням суфіксів та закінчень.

Варіант 4 (універсальний зразок). Зі зразком (*) співставляється будь-яке речення. Звичайно, і відповіді, що відповідають цьому зразкові, повинні бути такими ж універсальними, наприклад:

(Я Вас не дуже розумію)
(Не будьте такими небагатослівними)
(Чому це має для Вас значення?) і т. ін.

Варіант 5 (співставлення більш ніж з одним зразком). Речення може співставлятися не з одним зразком, а з декількома. Наприклад, якщо у програмі задані підстановки

(* люблю *) (Що Ви ще любите?) та
(* комп'ютери *) – (Ви маєте здібності до техніки), то речення (Я люблю комп'ютери) співставляється з обома зразками.

1.1.2. Кібернетичні системи

Поняття кібернетичної системи

Системи, що є носіями природного або штучного інтелекту, можуть бути охарактеризовані з точки зору кібернетики – науки, яка вивчає риси, спільні для складних систем будь-якої природи. Зокрема, центральне місце в кібернетиці займає вивчення процесів перетворення інформації у системах та керування ними. Основи цієї науки були викладені у 1948 р. видатним американським ученим Норбертом Вінером у його видатній роботі «Кібернетика, або керування і зв'язок у тварині і машині» [41]. Саме слово «кібернетика» походить від грецького слова "κυβερνητικη" — мистецтво керування.

Ключовим поняттям кібернетики є поняття кібернетичної системи. Кібернетична система визначена в літературі [162] як сукупність пов'язаних один з одним об'єктів (елементів системи), що здатні сприймати, зберігати, перетворювати інформацію, а також обмінюватися інформацією. Так, наприклад, людський організм можна розглядати як систему, що складається з окремих елементів-клітин; органи людського тіла при цьому розглядаються як підсистеми.

Можна дати ще одне визначення системи, в основі якого лежить визначення, сформульоване Холлом і Фейджином.

Системою називається сукупність порівняно простих елементарних структур або процесів, що взаємодіють між собою та об'єднані в єдине ціле для виконання деякої спільної функції, що не зводиться до функцій окремих компонентів. Система має такі ознаки:

- взаємодіє із зовнішнім середовищем та іншими системами як єдине ціле;

- є ієрархією підсистем нижчого рівня;
- є підсистемою для деякої системи вищого рівня;
- зберігає загальну структуру взаємодії елементів при зміні зовнішніх умов та внутрішнього стану.

Опис кібернетичної системи

Складні кібернетичні системи прийнято описувати на двох рівнях.

З одного боку, систему можна визначити як інформаційний перетворювач [52], що сприймає від зовнішнього середовища деякі вхідні величини (подразники, стимули) і залежно від цих подразників та свого внутрішнього стану генерує вихідні величини (реакції). Іншими словами, якщо через $\mathbf{x_1, \dots, x_n}$ позначити вхідні подразники, через $\mathbf{y_1, \dots, y_m}$ – реакції системи, а через $\mathbf{a_1, \dots, a_q}$ – змінні, що визначають поточний стан, то

$$y_j = f(x_1, \dots, x_n, a_1, \dots, a_q), j = 1, \dots, m.$$

Функція f, що задає інформаційне перетворення, може бути задана явно або неявно. Система може описуватися складними математичними рівняннями: диференційними, інтегральними та ін.

Система визначається як пара $\mathbf{S = (M, R)}$, де **M** – сукупність елементів системи, **R** – сукупність зв'язків між цими елементами. Тоді ***підсистемою*** **S′** системи **S** називається пара $\mathbf{S' = (M', R')}$, де $\mathbf{M' \subseteq M, R' \subseteq R}$. Можна окремо виділяти вхідні та вихідні елементи системи, які взаємодіють із зовнішнім світом, та її внутрішні елементи [105].

Класифікація кібернетичних систем

Широко відомою є класифікація кібернетичних систем, яка ґрунтується на двох основних критеріях [51].

1. Ступінь визначеності функціонування. За цим критерієм системи поділяються на:
   - детерміновані системи, елементи якої взаємодіють точно визначеним чином;
   - імовірнісні системи, для яких фактори, що впливають на елементи, або зв'язки між елементами не можуть бути точно описані, і результат функціонування системи стає не повністю визначеним.
2. Ступінь складності системи, за яким системи поділяються на:
   - прості, які можуть бути описані простими математичними моделями з невеликою кількістю рівнянь та змінних;
   - складні, які описуються складними математичними моделями з такою великою кількістю рівнянь та змінних, що їх точне моделювання стає неможливим або недоцільним;
   - дуже складні, для яких немає задовільного математичного опису.

Загальне поняття про керування кібернетичними системами

Проблема керування складними системами є однією з ключових проблем кібернетики. Традиційну задачу керування можна неформально сформулювати так: перевести керований об'єкт до бажаного стану або забезпечити перебування керованого об'єкта у потрібному стані. При цьому важливо мінімізувати затрати, пов'язані з керуванням.

Серед учених, які внесли значний вклад у розвиток теорії керування, можна відзначити Н. Вінера, Р. Беллмана, О. Ляпунова, Р. Калмана, М. Моісеєва, В. Глушкова. Значне

місце посідає принцип оптимального керування, відомий як принцип оптимальності Понтрягіна [142]. Важливий вклад був зроблений київською школою учених (Б. М. Бублик, М. Ф. Кириченко, Ф. Г. Гаращенко, О. Г. Наконечний) [24], [46], [76].

Можна розглядати і задачі структурного керування, які полягають у виборі оптимальної структури системи. Наголосимо, що написання програм, які можуть бути виконані за допомогою універсальних комп'ютерів або спеціалізованих процесорів, можна вважати окремим видом керування.

Треба розрізняти такі важливі випадки:

- якщо вдається побудувати точну математичну модель, іншими словами, точно описати керовану систему за допомогою математичних співвідношень і отримати точну постановку задачі керування, причому існують ефективні точні методи розв'язування цієї задачі, керування може бути точним; звичайно, точні методи керування можна застосовувати тільки для порівняно простих систем;
- для складніших систем доводиться застосовувати наближені методи керування; пов'язано це з тим, що або вдається побудувати лише наближену модель об'єкту, або існують лише наближені методи розв'язування відповідних рівнянь;
- для типових дуже складних систем (таких, як живий організм, фірма, суспільство) побудувати математичну модель у принципі неможливо; можна сподіватися на деякий наближений опис лише окремих сторін функціонування таких систем. Керування такими системами має неформальний характер. Тому задачу такого керування можна вважати однією з основних задач теорії штучного інтелекту. Ці проблеми стали предметом розгляду окремого наукового напряму, який дістав назву ситуаційного керування [150].

1.1.3. Інтелект як високоорганізована кібернетична система

Характеристика алгоритмічного і декларативного підходів до керування

Традиційно виділяють два підходи до керування складними системами та до програмування роботів і комп'ютерів, що могли б розв'язувати ті чи інші задачі. При розв'язуванні задачі на сучасних комп'ютерах в основному реалізований традиційний алгоритмічний підхід, який можна ще назвати імперативним. Цей підхід вимагає заздалегідь продумати та детально розписати, як треба вирішувати певну проблему. Написання програми вимагає задання чітких послідовностей інструкцій. Якщо таку послідовність вдається написати – комп'ютер зможе вирішувати певну задачу, якою б складною вона не була. Але, ясна річ, він буде виконувати інструкції, абсолютно не розуміючи їх змісту.

Інший підхід – декларативний. Інтелектуальному виконавцеві (людині чи комп'ютеру) досить сказати, що треба робити, тобто лише сформулювати завдання, побудувавши всі взаємозв'язки між об'єктами предметної області. Як це завдання буде виконуватися, повинен визначити сам виконавець.

Наведемо для прикладу дві задачі [121]. Різниця у складності їх розв'язку зумовлена тим, що в одному випадку вдається формалізувати постановку і запропонувати чіткі алгоритми розв'язку, в іншому – ні.

Запустити космічний корабель так, щоб він приземлився на Місяць, узяв зразки ґрунту та привіз їх назад – задача дуже складна, але вона піддається точній алгоритмізації. Математичні методи дозволяють точно розрахувати траєкторію руху корабля. Навіть якщо

він випадково відхилиться від цієї траєкторії, відомі методи автоматичного регулювання дозволять ліквідувати це відхилення.

Послати слухняного робота до магазину за пляшкою молока – задача на декілька порядків складніша. Не кажучи вже про сам процес спілкування з продавцем, робот повинен вирішити ряд не зовсім формалізованих підзадач. Заздалегідь розрахувати траєкторію руху неможливо, оскільки робот повинен уникнути зіткнень із людьми та автомобілями. Якщо магазин закритий на переоблік, він повинен знайти інший магазин. Неможливо передбачити всі ситуації, які можуть виникнути, а відтак – алгоритмізувати розв'язок задачі. Тому виконання такого завдання під силу лише інтелектуальній системі, яка вміє орієнтуватися в зовнішньому світі, аналізувати поточні ситуації та коригувати, адаптувати свою поведінку на основі такого аналізу.

Поповнення первинних інструкцій – одна з ключових рис інтелекту

Інструкції, написані природною мовою, можуть бути, з одного боку, досить неоднозначними, а з іншого – вони завжди спираються на те, що виконавець має певний апріорний досвід.

Розглянемо приклад, поданий у літературі [121]. Уявіть собі дві таблички. На першій, яка розташована при вході на будівництво, написано «Обов'язково одягніть на голову каску». На другій, біля входу до лондонського метро – «Обов'язково візьміть на руки собаку».

Ці інструкції однакові за формою, але абсолютно різні за змістом. Людина розуміє не лише те, що написано на табличках, але й те, що «залишається за кадром» і явно не вказується. Людина уявляє собі ситуації, які можуть виникати на будівництві та в метро, і поповнює первинні інструкції, що сприймаються нею безпосередньо. Зокрема, на будівництві зверху можуть падати цеглини, і тому потрібна каска, щоб захистити голову. До такого висновку людина могла б прийти і самостійно, без нагадування. Без каски на будівництво не пустять, і, якщо каски немає, її потрібно придбати. Але зрозуміло, що купувати собаку перед тим, як заходити в метро, зовсім необов'язково.

Людина має певну суму знань про світ, яка дозволяє їй орієнтуватися в життєвих ситуаціях та приймати вірні рішення. Крім того, людина вміє певним чином використовувати ці знання. Цих же рис повинна набути система штучного інтелекту. Можна стверджувати, що здатність до поповнення тих первинних описів ситуацій, що сприймаються безпосередньо, є однією з ключових рис інтелектуальної системи.

Зовнішній світ настільки багатий, що неможливо попередньо описати життєві ситуації з тим степенем деталізації і однозначності, які б дозволили закласти в системи, що проектуються, жорсткі детерміновані алгоритми поведінки. Тому інтелектуальні системи повинні мати механізми адаптації, які б дозволяли їм розв'язувати поставлені перед ними задачі на основі аналізу поточної ситуації.

Формалізація понять алгоритмічності та декларативності

Спробуємо формалізувати деякі з введених раніше понять. Введемо такі позначення:

q – ***первинні інструкції***, записані без будь-яких змін у тому вигляді, в якому їх сформулював автор; аналогічно можна визначити ***первинний опис ситуації*** як результат безпосереднього сприйняття ситуації;

S – ***множина факторів***, які визначають поточний стан виконавців, будемо розглядати **S** як об'єднання двох множин: **S′** та **S″**; тут **S′** – множина ***контрольованих факторів***, відомих авторові процедури і на які він може мати вплив (загалом, контрольованість і керованість – це не одне й те саме: фактор може бути контрольованим, проте некерованим, але для наших цілей це не має суттєвого значення); **S″** – множина ***неконтрольованих факторів***;

Z – знання, які має виконавець;

$\mathbf{r}_A$ – робочий алгоритм, який формується та реалізується виконавцем при алгоритмічному підході; при цьому, як правило, автоматично формується і програма $\mathbf{p}_A$, що відповідає цьому алгоритмові;

$\mathbf{r}_Д$ – робочий алгоритм, який формується та реалізується інтелектуальним виконавцем при декларативному підході. Нагадаємо, що, згідно з фундаментальними тезами Тюринга та Чорча (все, що може бути виконано будь-яким виконавцем, може бути промодельоване на машині Тюринга), сам факт виконання завдання свідчить про існування цього алгоритму. Щоправда, цей алгоритм може і не усвідомлюватися виконавцем, і в цьому випадку не може бути явно сформульований у вигляді алгоритму чи програми. Якщо ж програма $\mathbf{p}_Д$, що відповідає алгоритмові $\mathbf{r}_Д$, повинна бути згенерована, кажуть про ***задачу синтезу програм***.

Тоді можна записати такі співвідношення:

$$\mathbf{r}_A = \mathbf{f}(q, S'),\ \mathbf{r}_Д = \mathbf{h}(q, S'', Z),$$

де **f** і **h** – деякі функції.

Можна вважати, що алгоритм $\mathbf{r}_A$ формується на основі первинних інструкцій $\mathbf{r}_Д$ однозначно. При цьому також може відбуватися зміна і поповнення первинних інструкцій, але цей процес має повністю контрольований і детермінований характер. Як приклад, можна навести компіляцію програм, написаних мовами високого рівня. Тому автор процедури може бути впевнений у гарантованому результаті (якщо, звичайно, були дотримані певні формальні вимоги до первинних інструкцій та забезпечений належний стан виконавця).

На противагу цьому, при декларативному підході такої упевненості немає. Інтелектуальний виконавець певним чином поповнює первинні інструкції, але їх авторові не завжди відомо, як саме це поповнення відбувається. Тому не можна бути впевненим у результаті виконання інструкцій, і ця втрата гарантованості є неминучою платою за відмову від алгоритмічності.

У зв'язку з цим треба розглянути важливе узагальнення поняття алгоритму, яке дістало назву квазіалгоритму.

Квазіалгоритми та джерела квазіалгоритмічності

Ми бачили, що чисто алгоритмічний підхід виявив свою неспроможність при створенні інтелектуальних систем, призначених для розв'язання досить складних неформалізованих задач. Тому для теорії та практики штучного інтелекту важливе значення має узагальнення поняття алгоритму, яке дістало назву «квазіалгоритм» або «квазіалгоритмічна процедура».

Нагадаємо, що алгоритмом називається чітка недвозначна послідовність інструкцій, що зрозуміла виконавцеві й обов'язково призводить до гарантованого результату за скінчений час. На відміну від цього, інструкції квазіалгоритму можуть бути не зовсім чіткими, і результат виконання квазіалгоритмічної процедури не обов'язково є гарантованим.

Можна виділити як мінімум чотири основні джерела квазіалгоритмічності. Вони можуть доповнювати одне одного, у деяких випадках вони можуть бути описані схожими математичними моделями, але ці джерела мають принципово різну природу. Розглянемо їх.

1. **Дія випадкових чинників**, що не залежать від виконавця. Формально, з огляду на цей фактор квазіалгоритмом треба вважати будь-який алгоритм. Алгоритм розрахований на роботу при певних умовах; якщо ці умови зміняться, алгоритм може не привести до потрібного результату. Так, навіть програма на Паскалі, що обчислює суму двох чисел, не буде виконана, якщо під час її виконання в комп'ютері випадково зникне напруга. Якщо ж не вдається чітко окреслити межі застосування алгоритму або забезпечити виконання необхідних умов для його роботи, ми маємо справу зі справжнім квазіалгоритмом.
2. **Недостатнє врахування автором алгоритмічної процедури особливостей виконавця**, і це призводить до того, що виконавець неправильно розуміє, що від нього вимагається. Процедура може бути в принципі алгоритмічною, за нормальних умов приводити до гарантованого результату, але цей результат може бути непередбачений автором і бути для нього несподіванкою. Наприклад, програміст бажає обчислити суму перших 50 натуральних чисел, але не знає, як працюють цикли. Такий програміст може вважати, що цикл виконується, поки лічильник менший від верхньої межі (хоча насправді цикл виконується, поки лічильник менший від верхньої межі або дорівнює їй). Тоді він може написати такий помилковий фрагмент програми на Паскалі:

   S:=0; for i := 1 to 51 do S:= S + i.

3. **Нечіткість формулювань, що фігурують в описі процедури**. Розглянемо будь-який кулінарний рецепт, наприклад: «розтерти тісто, змішати із дрібно порізаними яблуками, додати солі й перцю за смаком і смажити до появи рум'яної скоринки». Це є типовим прикладом нечіткості формулювань, що призводить до невизначеностей. До якої межі треба розтирати тісто? Що означає «дрібно порізані»? Що означає «за смаком»? Як формалізувати момент «появи рум'яної скоринки»? Неінтелектуальна система, орієнтована на чисто алгоритмічне керування, просто не зрозуміє цього опису і тому не зможе його виконати. Інтелектуальна ж система, наприклад, людина, повинна спробувати поповнити і уточнити цей опис на основі наявних знань і досвіду. Сам по собі факт виконання завдання буде свідчити про те, що квазіалгоритмічний первинний опис процедури був зведений до деякого внутрішнього алгоритму, але навряд чи виконавець зможе чітко пояснити і розписати цей алгоритм. Крім того, ці внутрішні алгоритми для кожного виконавця будуть різними, і це зумовить різні результати – у нашому випадку різний смак виробу.
4. **«Свобода волі»**, яку можуть мати високоорганізовані, по-справжньому інтелектуальні системи. Ці системи можуть мати свої цілі і якщо задана процедура суперечить цим цілям, вони можуть відмовитися її виконувати або виконати не так, як це від них вимагається. «Свобода волі» полягає у тому, що інтелектуальна система самостійно приймає рішення про свою поведінку і, в залежності від цих рішень, може виконувати зовнішні розпорядження, не виконувати їх або ж виконувати так, як вона вважає за потрібне.

Зазначимо, що справжньою квазіалгоритмічністю найчастіше вважають лише квазіалгоритмічність, зумовлену третім та четвертим чинниками.

Характеристика інтелектуальних систем із загальнокібернетичних позицій

Природно вважати інтелектуальну систему різновидом кібернетичної системи. Можна розглядати такі випадки:

- **чисто зовнішнє керування**: система сприймає зовнішні інструкції у вигляді речень природної мови, програм, написаних алгоритмічними мовами високого рівня та ін., і намагається їх виконати; ці зовнішні інструкції раніше були названі первинними інструкціями; тоді ці інструкції природно вважати зовнішніми подразниками для системи, і саме вони формують її мету;
- **самокерування**: система на основі аналізу зовнішньої ситуації приймає рішення, спрямовані на досягнення власної мети. Первинною при цьому є зовнішня ситуація у тому вигляді, як вона безпосередньо сприймається системою; результат такого сприйняття раніше був названий первинним описом ситуації;
- **комбіноване керування**, з яким система, у принципі, може бути самокерованою і скеровуватися власною метою. Однак вона може припускати і зовнішнє керування, і тоді вона розглядає мету керівної системи як свою власну. Дуже важливим при цьому є питання пріоритетів, тобто система повинна визначити, яка мета є для неї важливішою: власна чи зовнішня.

Принциповим є те, що інтелектуальна система намагається планувати свої дії для досягнення певної мети на основі первинного опису ситуації.

Розглянемо це питання детальніше. Система має певну мету і прагне так планувати свої дії, щоб досягати цієї мети. Вхідними стимулами системи можна вважати поточну ситуацію, що сприймається і аналізується системою. Результатом реакції системи стає зміна зовнішньої ситуації, і поведінка системи коригується в залежності від того, бажаною чи небажаною є ця зміна. Саме це коригування і є реалізацією одного з основних зворотних зв'язків інтелектуальної системи.

Необхідним компонентом інтелектуальної системи є знання про світ, на основі яких вона поповнює первинний опис зовнішньої ситуації з метою поглибленого розуміння цієї ситуації, аналізує можливі наслідки своїх дій, і, можливо, формує або коригує свою мету. Рис. 1.1 ілюструє вищесказане.

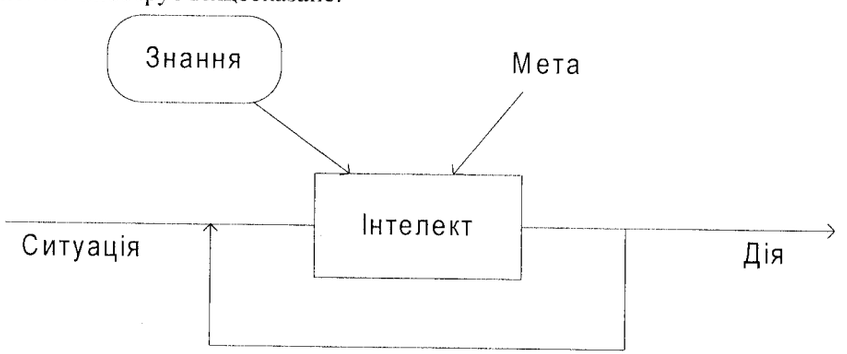


Рис. 1.1. Схема самокерування інтелектуальної системи

Усе це відноситься до будь-якої інтелектуальної системи, незалежно від того, чи мова йде про природний розум людини, чи про природний розум інопланетянина, чи про штучний інтелект. У деякому грубому наближенні можна вважати, що чим інтелектуальнішою є система, тим ефективніше вона може досягати мети.

Людина не отримує всі свої знання і вміння відразу, тим більше неможливо передати всі необхідні знання й уміння системі штучного інтелекту, призначеної для вирішення досить складних завдань. Тому винятково важливе значення має здатність інтелектуальної системи до навчання, тобто до поповнення своїх знань і до набуття нових можливостей. Таке навчання може відбуватися шляхом спілкування інтелектуальної системи з людиною або з іншою інтелектуальною системою. Якщо ж навчання відбувається самостійно, мова йде про самонавчання.

Справді розвинені інтелектуальні системи повинні мати можливість безпосередньо сприймати навколишній світ за допомогою спеціальних органів чуття. Більш того, це сприйняття має бути пов'язаним з тими поняттями, які входять до знань системи, або які формуються в ній. Без такого зв'язку слова залишаються лише словами; людина ж у процесі мислення істотно спирається на чуттєві асоціації, пов'язані з цими словами. Наприклад, слово «пожежа» викликає у свідомості людини цілий пласт уявлень, пов'язаних з такими зовнішніми подразниками, як «яскраво», «гаряче» тощо. На основі цих асоціацій виникають інші асоціації. Все це має рефлекторний характер і зумовлює активну участь підсвідомих інтуїтивних процесів у людському мисленні, прийнятті рішень тощо. Більш того, без набуття таких рис важко говорити про те, що система штучного інтелекту може мати якусь власну мету.

Для людини характерний тісний зв'язок «лівопівкульних» понять, якими оперує людина в процесі логічного мислення, з «правопівкульними» образами, які підкріплюються сигналами, що надходять від органів чуття. У сучасних системах штучного інтелекту такий зв'язок практично відсутній. На цій підставі Дрейфус у 70-ті роки XX ст. взагалі піддав сумніву можливість створення штучних систем, подібних до людини за своїми можливостями [59].

Відомо, що мозок людини складається з двох півкуль, ліва півкуля відповідає за процеси логічного мислення, а права – за виникнення зорових, слухових та інших образів, чуттєвих асоціацій тощо.

У зв'язку з цим треба знову згадати про інтелектуальні роботи. У принципі можна говорити про їх здатність безпосередньо сприймати зовнішній світ за допомогою органів чуття, але можливості використання цього світовідчуття для активізації підсвідомих рефлекторних процесів у роботів поки що залишаються дуже слабкими.

Отже, до переліку інтелектуальних задач можна віднести такі задачі:

- аналіз ситуацій;
- розпізнавання образів;
- логічне мислення;
- розуміння нової інформації;
- навчання і самонавчання;
- планування цілеспрямованих дій.

Інтелектуальною системою називається самокерована кібернетична система, яка має певну суму знань про світ і здатна на основі безпосереднього сприйняття і подальшого аналізу поточної ситуації планувати дії, спрямовані на досягнення певної мети, а також поповнювати свої знання.

Це визначення не може претендувати на повноту й універсальність. Воно фокусує увагу на основних завданнях, які повинна вміти вирішувати будь-яка інтелектуальна природна, технічна або програмна система. Іншими словами: інтелектуальна система – програмний комплекс, який оперує знаннями у певній предметній області (ПО) з метою вироблення рекомендацій або розв'язання задачі.

Типова схема функціонування інтелектуальної системи

Ми бачили, що функціонування інтелектуальної системи можна описати як постійне прийняття рішень на основі аналізу поточних ситуацій для досягнення певної мети. Природно виділити окремі етапи, які утворюють типову схему функціонування інтелектуальної системи.

Отже, інтелектуальна система в процесі своєї роботи вирішує ряд задач:

1. Безпосереднє сприйняття зовнішньої ситуації; результатом є формування первинного опису ситуації.
2. Співставлення первинного опису зі знаннями системи і поповнення цього опису; результатом є формування вторинного опису ситуації в термінах знань системи. Цей процес можна розглядати як процес розуміння ситуації або як процес перекладу первинного опису на внутрішню мову системи. При цьому можуть змінюватися внутрішній стан системи та її знання.

Вторинний опис може бути не єдиним, і система може вибирати між різними вторинними описами. Крім того, система в процесі роботи може переходити від одного вторинного опису до іншого.

Якщо ми можемо формально задати форми внутрішнього подання описів ситуацій та операції над ними, ми можемо сподіватися на певний автоматизований аналіз цих описів.

3. Планування цілеспрямованих дій та прийняття рішень, тобто аналіз можливих дій та їх наслідків і вибір тієї дії, яка найкраще узгоджується з метою системи. Це рішення, взагалі кажучи, формулюється деякою внутрішньою мовою (свідомо або підсвідомо).
4. Зворотна інтерпретація прийнятого рішення, тобто формування робочого алгоритму для здійснення реакції системи.
5. Реалізація реакції системи; наслідком є зміна зовнішньої ситуації та внутрішнього стану системи і т.ін.

Не слід вважати, що вказані етапи є повністю розділеними у тому розумінні, що наступний етап починається тільки після того, як повністю закінчиться попередній. Навпаки, для функціонування інтелектуальної системи характерним є взаємне проникнення цих етапів. Наприклад, ті чи інші рішення можуть прийматися уже на етапі безпосереднього сприйняття ситуації. Насамперед, це рішення про те, на які зовнішні подразники треба звертати увагу, а на які не обов'язково. Зовнішніх подразників так багато, що їх сприйняття має бути вибірковим.

Класифікація основних напрямів досліджень

Однією з найбільш традиційних класифікацій напрямів досліджень у галузі штучного інтелекту є класифікація Ханта [178].

Перший напрям носить назву біологічного. У його основі лежать спроби вирішення інтелектуальних завдань шляхом безпосереднього моделювання психофізіологічних особливостей мозку людини засобами електронно-обчислювальних машин. З часом стало зрозумілим, що розв'язати цю проблему в повному обсязі неможливо і недоцільно. Моделювання особливостей людського розуму в повному обсязі призведе до копіювання не тільки позитивних якостей, але й негативних. Стало зрозумілим, що таке моделювання, хоч і має певне значення для створення штучного інтелекту, більше підходить для вивчення інтелекту природного. Таке вивчення оформилося у вигляді самостійної науки, яка дістала назву когнітивної психології.

Інший підхід має назву прагматичного. У ньому майже не розглядається психофізіологічна діяльність людського мозку. Розвиваються підходи до розв'язання інтелектуальних задач незалежно від того, як співвідносяться ці підходи з тими процесами, що відбуваються в мозку людини.

Існує інша класифікація систем штучного інтелекту залежно від моделей та методів, що використовуються для вирішення практичних задач. Йдеться про символьний та конекціоністський підходи. Ці підходи будуть детально розглянутими в ході вивчення курсу; зараз можна відзначити, що символьний підхід орієнтований на моделювання процесів свідомого логічного мислення, а конекціоністський – підсвідомих рефлекторних процесів. Підходи можуть застосовуватися як роздільно, так і в комбінації. Найбільших успіхів можна досягти, лише комбінуючи ці підходи. Зрозуміло, що без використання рефлекторних методів жодна серйозна інтелектуальна проблема не може бути вирішена через «прокляття розмірності». З іншого боку, рефлекторні рішення далеко не завжди правильні та оптимальні, і тому людина не може обійтись без логічного мислення, як не можуть без нього обійтись і системи штучного інтелекту.

Характерною особливістю сучасних досліджень є зближення та взаємопроникнення символьного та конекціоністського підходів.

І, нарешті, дослідження в галузі штучного інтелекту можна класифікувати за типами задач. Серед практичних інтелектуальних задач, у розв'язку яких можна відзначити реальні здобутки – розпізнавання образів, автоматизація логічних побудов, створення діалогових систем типу «людина-машина» та систем автоматизованого перекладу текстів, робототехніка. Останнім часом бурхливо розвиваються інтелектуальні агенти – автономні програмні системи, призначені для роботи в Інтернеті (див. розділи 9 і 10).

Відзначимо, що жодна з наявних систем штучного інтелекту, які працюють у перелічених галузях, не може реалізовувати перелічені вище функції інтелектуальної системи в достатньому обсязі. Тому про них краще говорити як про інтелектуалізовані системи, які мають деякі риси, що наближають їх до справді інтелектуальних систем.

У роботі [56] відзначається десять типів предметних областей (ПО), в яких інтелектуальні системи знайшли своє застосування. Перелік ПО та задачі, які в них необхідно розв'язати, наведені у табл. 1.1.

Таблиця 1.1. Перелік предметних областей

Тип предметної області	Опис задачі
Інтерпретація	Побудова описів ситуацій за даними, що спостерігаються
Прогнозування	Виведення імовірнісних висновків із заданих ситуацій

продовження табл 1.1.

Діагностика	Твердження про порушення в системі, виходячи зі спостережень
Проектування	Побудова конфігурації об'єктів за певних обмежень
Планування	Проектування плану дій
Моніторинг	Порівняння спостережень із критичними точками плану
Налагодження	Вироблення рекомендацій для запобігання неполадок
Ремонт	Виконання плану застосування виробленої рекомендації
Навчання	Діагностика, налагодження і виправлення поведінки учня
Керування	Інтерпретація, прогнозування, ремонт і моніторинг поведінки системи

Ці ПО були визначені в кінці 80-х років минулого століття. На нашу думку, тепер сюди ще треба додати пошук релевантної інформації за допомогою інтелектуальних пошукових систем, тобто пошук не тільки за ключовими словами, а й за контекстом. Цими питаннями зараз займаються багато вчених в галузі штучного інтелекту. Ця проблема, а також перелічені в табл. 1 задачі, належать до класу слабко-структурованих (неструктурованих), неформалізованих задач. Ці задачі володіють однією або декількома наступними характеристиками:

- задачі не можуть формулюватися в числовій формі;
- цілі не можуть задаватися в термінах чітко визначеної цільової функції;
- не існує алгоритмічного рішення задач;
- алгоритм пошуку розв'язку задачі відомий, однак його не можна використати через обмеженість ресурсів часу і/або пам'яті.

Неформалізовані задачі, зазвичай, мають такі особливості:

- хибність, неоднозначність, неповнота і суперечливість початкових даних;
- хибність, неоднозначність, неповнота і суперечливість знань про предметну область і задачі, які необхідно в ній розв'язати;
- велика розмірність простору пошуку розв'язку задачі;
- дані та знання динамічно змінюються.

Відзначимо, що неформалізовані задачі складають дуже важливий клас. Багато спеціалістів вважають, що вони є наймасовішим класом задач [148]. Саме необхідність їх розв'язання обумовила виникнення особливого класу автоматизованих інформаційних систем (АІС), які отримали назву інтелектуальні тому, що на відміну від традиційних АІС та інших програмних систем, вони мають такі ознаки:

- моделюють не лише фізичну (чи іншу) природу деякої ПО, а і механізм логічного виведення для розв'язування задач у цій ПО;
- формують виведення розв'язків, грунтуючись на тих знаннях про ПО і методи розв'язування задач, якими інтелектуальна система наділена; знання подаються

у деякому спеціальному формалізмі, який забезпечує можливість їх розуміння і використання комп'ютерним комплексом;

- під час розв'язування задач основними є евристичні й наближені методи, які, на відміну від алгоритмічних, не завжди гарантують успіх у знаходженні розв'язку; евристика – знання, набуті людиною в часі накопичення практичного досвіду розв'язування аналогічних задач.

Соціальні наслідки інтелектуалізації комп'ютерних технологій

Нарешті, можна коротко зупинитися на соціальних наслідках інтелектуалізації інформаційних технологій. Позитивні наслідки зрозумілі та загальновідомі. З наслідків, не дуже сприятливих для людини, найочевидніший – це поступове витіснення людей машинами як у процесі виробництва, так і в керуванні суспільством. Подібні тенденції почали проявлятися ще задовго до початку комп'ютерної ери. Що стосується майбутнього, то тут письменники-фантасти малюють похмурі картини повного захоплення влади комп'ютерами і навіть фізичного винищення людства.

Запропоновано ряд принципів, орієнтованих на зменшення ризиків, пов'язаних з комп'ютеризацією та інтелектуалізацією. Найбільш відомими є принципи, сформульовані американським фантастом Айзеком Азімовим. Ось ці принципи:

- комп'ютер або робот не повинен заподіяти шкоди людині;
- він повинен виконувати накази людини, якщо це не суперечить першому принципу;
- він повинен прагнути до самозбереження, якщо це не суперечить першим двом принципам.

Інша проблема – як ці або подібні їм принципи втілювати в життя. Зазначимо, що час, коли штучний інтелект розвинеться настільки, що зможе реально загрожувати людському, настане ще дуже не скоро. Цей висновок, який був зроблений у 60-70-ті роки XX століття, залишається незмінним і сьогодні (якщо, звичайно, не розглядати можливості, пов'язані з «доступом до ядерної кнопки» і подібні їм).

Серед інших негативних аспектів інтелектуалізації можна відзначити, що сучасні комп'ютерні технології полегшують здійснення не тільки корисних справ, але й різних зловживань, злочинів тощо.

Як відзначають провідні спеціалісти в області ШІ – «показником інтелектуальності системи є її вміння використовувати у потрібний момент необхідні (релевантні) знання» [147]. Системи, що не мають засобів для визначення релевантних знань, зіштовхуються з проблемою «комбінаторного вибуху». Ця проблема є однією з основних причин, через яку обмежується сфера застосування інтелектуальних систем [246].

Очевидно, що вирішення цієї проблеми полягає у підвищенні рівня інтелектуальності цих систем. Це веде до необхідності розв'язання таких задач:

- розширити сферу застосування інтелектуальних систем;
- видавати інтелектуальною системою точніші та надійніші розв'язки;
- узагальнити та модифікувати знання (індукція);
- підвищити ефективність функціонування.

У процесі моделювання та побудови інтелектуальних систем використовується поняття моделі ПО.

Модель предметної області (МПО) – система знань, яка необхідна для автоматичного синтезу алгоритму розв'язування задачі в деякій ПО. Ці знання складаються з

об'єктів, які називають поняттями [108]. Вони, зазвичай, мають назву (можливо, кілька назв-синонімів), визначення, структуру, взаємопов'язані з іншими поняттями, і входять у систему понять. Також сюди відносяться зв'язки між поняттями або твердження про властивості понять і зв'язків між ними.

Інтелектуальні системи мають такі характерні риси, що є важливими з точки зору її моделювання:

- містить систему знань про ПО, які подані у вигляді МПО;
- володіє механізмами міркувань, якими виступають метапроцедури, що використовують знання з метою вироблення розв'язку;
- володіє природно-мовним інтерфейсом, що забезпечує інтерактивну взаємодію користувача з інтелектуальною системою.

Ці три компоненти складають сутність розгляду інтелектуальних систем як об'єкту моделювання.

Серед напрямів ШІ є декілька передових, які зараз викликають найбільше зацікавлення в дослідників і на практиці. Розглянемо їх детальніше.

1. Подання знань і розроблення систем, що ґрунтуються на знаннях. Це основний напрям в області розроблення СШІ. Він пов'язаний з розробленням моделей подання знань, створенням баз знань, онтологій, що формують ядро інтелектуальної системи. Останнім часом сюди включають моделі й методи видобування і структурування знань, що зливається з інженерією знань. Детальніше дивіться розділи 2, 3, 4 цього підручника.
2. Програмне забезпечення інтелектуальних систем. Тут розробляються спеціальні мови для розв'язування інтелектуальних задач, в яких традиційно наголос робиться на логічне і символьне опрацювання інформації, а не на обчислювальні процедури. Ці мови орієнтовані на символьне опрацювання інформації – Lisp, Prolog, Smalltalk, Clips, NewLisp, Python та інші. Трохи детальніше із трьома останніми мовами можна ознайомитись в додатках.
3. Розроблення природно-мовних інтерфейсів і машинний переклад. Одним з найпопулярніших напрямків досліджень є комп'ютерна лінгвістика, і, як складова, машинний переклад (МП). На сьогодні МП використовують такі моделі:
   - використання «мов-посередників», за допомогою яких відбувається додаткова трансляція «мова оригіналу-мова змісту-мова перекладу»;
   - асоціативний пошук аналогічних фрагментів тексту та їх перекладів у спеціальних текстових репозиторіях або базах даних;
   - структурний підхід, який передбачає послідовний аналіз і синтез природно-мовних повідомлень; традиційно такий підхід містить наступні етапи аналізу:
     а) морфологічний аналіз – аналіз слів у тексті;
     б) синтаксичний аналіз – складовий аналіз речень і граматичних зв'язків між словами;
     в) семантичний аналіз – аналіз змісту складових частин кожного речення на основі деякої предметно-орієнтованої бази знань;
     г) прагматичний аналіз – аналіз змісту речень в реальному контексті на основі власної бази знань.

4. Інтелектуальні роботи. Робот – електротехнічний прилад, який призначений для автоматизації людської праці. На сьогодні у світі виробляється понад 60 000 роботів на рік. Фактично робототехніка зараз – це інженерна наука, яка використовує технології інтелектуальних систем, але наразі не готова до впровадження їх через різні причини. У нашому підручнику цей напрям не розглядається.
5. Навчання і самонавчання. Область ШІ, яка активно розвивається. Містить моделі, методи й алгоритми, орієнтовані на автоматизоване накопичення і формування знань на основі аналізу й узагальнення даних [58]. Містить навчання на основі прикладів (або індуктивне), а також традиційні підходи з теорії розпізнавання образів. Останніми роками з цим напрямом тісно пов'язані системи аналізу даних (data mining) і пошуку закономірностей в базах даних (knowledge discovery) [255]. Детальніше розглядаються у восьмому розділі.
6. Розпізнавання образів. Це один із напрямів ШІ, який зараз вже виділяється як самостійна наука. Тому нами у цьому підручнику не розглядається.
7. Інші напрями. До інших напрямів відносяться:
   - нові архітектури комп'ютерів;
   - ігри та машинна творчість;
   - когнітивне моделювання;
   - інтелектуальні інтерфейси;
   - розпізнавання та синтез мови;
   - менеджмент знань та інші.

Очевидно, що в межах одного підручника неможливо розглянути всі підходи та ідеї. Найрозповсюдженішим видом інтелектуальних систем є експертні системи. Тому їм присвятимо окремий підрозділ.

1.2. Експертні системи

Експертні системи є підкласом інтелектуальних систем.

1.2.1. Визначення і класифікація

Одним з найзначніших досягнень штучного інтелекту та інтелектуальних систем стало розроблення потужних комп'ютерних систем, що отримали назву «експертних» або базованих на «знаннях» систем. У сучасному суспільстві під час розв'язування задач керування складними багатопараметричними і сильнозв'язаними системами, об'єктами, виробничими і технологічними процесами доводиться стикатися з рішеннями, що не формалізуються або важко формалізуються з точки зору опису завдань. Такі завдання часто виникають у таких областях: авіація, космос і оборона, нафтопереробна промисловість і транспортування нафтопродуктів, хімія, енергетика, металургія, целюлозно-паперова промисловість, телекомунікації та зв'язок, харчова промисловість, машинобудування, виробництво цементу, бетону тощо, транспорт, медицина і фармацевтичне виробництво, адміністративне керівництво, прогнозування і моніторинг. Найзначнішими досягненнями в цій області стало створення систем, які встановлюють діагноз захворювання, відкривають родовища корисних копалин, допомагають у проектуванні електронних пристроїв, машин і механізмів, вирішують завдання керування реакторами та інші завдання.

Під експертною системою (ЕС) розуміють програму, яка використовує знання фахівців (експертів) про деяку конкретну вузькоспеціалізовану предметну область і в межах цієї області здатна приймати рішення на рівні експерта-професіонала.

Усвідомлення корисності систем, які можуть копіювати «дорогі» або такі, що рідко зустрічаються, людські знання, привело до широкого впровадження і розквіту цієї технології у 80-ті, 90-ті роки минулого століття. Основу успіху ЕС склали дві важливі властивості, що відзначаються основними фахівцями в цій області:

- у ЕС знання відокремлені від даних, і потужність експертної системи обумовлена в першу чергу, потужністю бази знань і лише в другу чергу – використовуваними методами розв'язування задач;
- задачі, які розв'язуються ЕС є неформалізованими або слабоформалізованими і використовують евристичні, експериментальні, суб'єктивні знання експертів у певній предметній області.

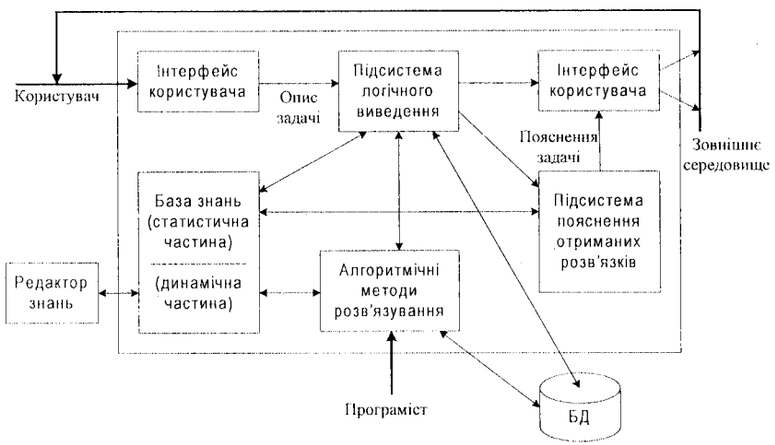


Рис. 1.2. Структура експертної системи

Узагальнена схема ЕС наведена на рис. 1.2. Основу ЕС складає підсистема логічного виведення, яка використовує інформацію з бази знань (БЗ), генерує рекомендації за рішенням шуканого завдання. Найчастіше для подання знань в ЕС використовуються системи продукцій і семантичні мережі (див. розділ 3 «Подання знань») [293].

Обов'язковими частинами будь-якої ЕС є також модуль придбання знань і модуль відображення і пояснення рішень. У більшості випадків реальні ЕС в промисловій експлуатації працюють також на основі баз даних (БД). Тільки одночасна робота із знаннями і великими об'ємами інформації з БД дозволяє ЕС отримати неординарні результати, наприклад, поставити складний діагноз (медичний або технічний), відкрити родовище корисних копалини, керувати ядерним реактором у реальному часі.

Важливу роль під час створення ЕС відіграють інструментальні засоби. Серед інструментальних засобів для створення ЕС найпопулярніші такі мови програмування, як LISP і PROLOG, а також експертні системи-оболонки (ЕСО): KEE, CENTAUR, G2 і GDA, CLIPS, АТТЕХНОЛОГІЯ, що надають в розпорядження розробника – інженера

й знань широкий набір для комбінування систем подання знань, мов програмування, об'єктів і процедур [294].

Розглянемо різні способи класифікації ЕС.

За призначенням ЕС поділяються:

- ЕС загального призначення;
- спеціалізовані ЕС:
  - проблемно-орієнтовані для завдань діагностики, проектування, прогнозування;
  - наочно-орієнтовані для специфічних завдань, наприклад, контролю ситуацій на атомних електростанціях.

За ступенем залежності від зовнішнього середовища виділяють:

- статичні ЕС, незалежні від зовнішнього середовища;
- динамічні, які враховують динаміку зовнішнього середовища і призначені для вирішення завдань в реальному часі; час реакції в таких системах може задаватися в мілісекундах, і ці системи реалізуються, як правило, на мові С++.

За типом використання розрізняють:

- ізольовані ЕС;
- ЕС на вході/виході інших систем;
- гібридні ЕС або, інакше кажучи, ЕС інтегровані з базами даних та іншими програмними продуктами (застосуваннями).

За складністю задач, які розв'язує ЕС, розрізняють:

- прості ЕС – до 1000 простих правил;
- середні ЕС – від 1000 до 10000 структурованих правил;
- складні ЕС – понад 10000 структурованих правил.

За стадією створення виділяють:

- дослідницький зразок ЕС, розроблений за 1-2 місяці з мінімальною БЗ;
- демонстраційний зразок ЕС, розроблений за 2-4 місяці, наприклад, на мові типу LISP, PROLOG, CLIPS;
- промисловий зразок ЕС, розроблений за 4-8 місяців, наприклад, на мові типу CLIPS з повною БЗ;
- комерційний зразок ЕС, розроблений за 1,5-2 роки, наприклад, на мові типа С++, Java з повною БЗ.

1.2.2. Труднощі під час розроблення експертних систем

Розроблення ЕС пов'язана з певними труднощами, які потрібно добре знати, так само як і способи їх подолання. Розглянемо докладніше ці проблеми.

1. Проблема добування знань від експертів. Жоден фахівець ніколи просто так не розкриє секрети своєї професійної майстерності, свої фахові знання в професійній області. Він має бути зацікавлений матеріально або морально, причому добре зацікавлений. Ніхто не хоче рубати сук, на якому сидить. Часто такий фахівець побоюється, що, розкривши всі свої секрети, він не буде потрібний компанії. Замість нього працюватиме експертна система. Уникнути цієї проблеми допоможе вибір висококваліфікованого експерта, зацікавленого у співпраці.

2. Проблема формалізації знань експертів. Експерти-фахівці в певній області, як правило, не в змозі формалізувати свої знання. Часто вони ухвалюють правильні рішення на інтуїтивному рівні та не можуть аргументовано пояснити, чому прийняте те або інше рішення. Іноді експерти не можуть прийти до взаєморозуміння (фраза «зустрілися два геологи, у них були три думки» – не жарт, а реальне життя). У таких ситуаціях допоможе вибір експерта, що вміє чітко формулювати свої думки й легко пояснювати іншим свої ідеї.
3. Проблема браку часу в експерта. Вибраний для розроблення ЕС експерт не може знайти достатньо часу для виконання проекту. Він дуже зайнятий. Він всім потрібний. У нього є проблеми. Щоб уникнути цієї ситуації, необхідно отримати від експерта, перш ніж почнеться проект, згоду витрачати на проект час в певному фіксованому обсязі.
4. Правила, формалізовані експертом, не дають необхідної точності. Проблеми можна уникнути, якщо розв'язувати разом з експертом реальні задачі. Не треба придумувати «іграшкових» ситуацій або завдань. В умовах завдань потрібно використовувати реальні дані, такі як лабораторні дані, звіти, щоденники й іншу інформацію, взяту з практичних завдань. Намагайтеся говорити з експертом на одній мові, використовуючи єдину термінологію. Експерт, як правило, легше розуміє правила, записані на мові, близькій до природної, а не на мові типу LISP або PROLOG.
5. Недостатність ресурсів. Як ресурси виступають персонал (інженери знань, розробники інструментальних засобів, експерти) і засоби побудови ЕС (засоби розроблення і засоби підтримки). Недостатність грамотних адміністраторів породжує скептицизм і нетерпіння у керівників.
6. Неадекватність інструментальних засобів задачі, яка розв'язується. Часто певні типи знань (наприклад, тимчасові або просторові) не можуть бути легко подані на одній мові подання знань (МПЗ), так само як і різні схеми подання (наприклад, фрейми і продукції) не можуть бути достатньо ефективно реалізовані на одній МПЗ. Деякі завдання можуть бути непридатними для вирішення за технологією ЕС (наприклад, окремі завдання аналізу станів). Необхідний ретельний аналіз задач, щоб визначити придатність пропонованих інструментальних засобів і зробити правильний вибір.

Підвищену увагу і перебільшення викликали нереалістичні очікування, які приводять до розчарування відносно експертних систем. ЕС можуть давати не найкращі рішення на межі їх застосовності, при роботі із суперечливими знаннями і в міркуваннях на основі здорового глузду. Потрібні значні зусилля, щоб добитися невеликого збільшення якості роботи ЕС. Експертні системи вимагають значного часу на розроблення. Так, створення системи PUFF для інтерпретації функціональних тестів легенів зайняло 5 людино-років, на розроблення системи PROCPECTOR для розвідки рудних родовищ пішло 30 людино-років, система XCON для розрахунку конфігурації комп'ютерних систем на основі VAX 11/780 зажадала 8 людино-років. ЕС останніх років розробляються швидшими темпами за рахунок розвитку технологій ЕС, але проблеми залишилися. Подвоєння персоналу не скорочує час розроблення наполовину, тому що процес створення ЕС – це процес із безліччю зворотних зв'язків. Все це потрібно враховувати під час планування створення ЕС.

Про інші труднощі під час створення ЕС детальніше можна прочитати у книзі Д. Уотермана [293] і підручнику [295].

1.2.3. Методологія побудови експертних систем

Розглянемо методику формалізації експертних знань на прикладі створення експертних діагностичних систем (ЕДС).

Метою створення ЕДС є визначення стану об'єкта діагностування (ОД) і наявних в ньому несправностей.

Станами ОД можуть бути: справний, несправний, працездатний. Несправностями, наприклад, радіоелектронних ОД є обрив зв'язку, замикання провідників, неправильне функціонування елементів тощо.

Кількість несправностей може бути достатньо великою (декілька тисяч). У ОД може бути одночасно декілька несправностей. У цьому випадку говорять, що несправності кратні.

Введемо наступні визначення. Різні несправності ОД проявляються у зовнішньому середовищі інформаційними параметрами. Сукупність значень інформаційних параметрів визначає «інформаційний образ» (ІО) несправності ОД. ІО може бути повним, тобто містити всю необхідну інформацію для встановлення діагнозу, або, відповідно, неповним. У разі неповного ІО встановлення діагнозу має імовірнісний характер.

Основою для побудови ефективних ЕДС є знання експерта для встановлення діагнозу, записані у вигляді інформаційних образів, і система подання знань, що вбудовується в інформаційні системи забезпечення функціонування і контролю ОД, інтегровані з відповідною технічною апаратурою.

Для опису своїх знань експерт за допомогою інженера зі знань повинен виконати наступне.

1. Виділити, за можливістю, всі несправності ОД, які повинна розрізняти ЕДС.
2. Виділити інформативні (істотні) параметри, значення яких дозволяють розрізнити кожну несправність ОД і встановити діагноз з деякою ймовірністю.
3. Для вибраних параметрів потрібно виділити інформативні значення або інформативні діапазони значень, які можуть бути як кількісними, так і якісними. Наприклад, точні кількісні значення можуть бути записані: затримка 25 нсек, затримка 30 нсек тощо. Кількісний діапазон значень може бути записаний: затримка 25-40 нсек, 40-50 нсек, 50 нсек і вище. Якісний діапазон значень може бути записаний: індикаторна лампа світиться яскраво, світиться слабо, не світиться.

Для зручнішого подальшого використання якісний діапазон значень може бути закодований, наприклад, таким чином:

- світиться яскраво P1 = +++ (або P1 = 3);
- світиться слабо P1 = ++ (або P1 = 2);
- не світиться P1 = + (або P1 = 1).

Процедура отримання інформації по кожному з параметрів визначається індивідуально в кожній конкретній системі діагностування. Ця процедура може полягати в автоматичному вимірюванні параметрів в ЕДС, в ручному вимірюванні параметра за допомогою приладів, якісному визначенні параметра, наприклад, «світиться слабо» і т. ін.

Процедура створення повних або неповних Ю кожної несправності в алфавіті значень інформаційних параметрів може бути визначена таким чином. Складаються діагностичні правила, що визначають вірогідний діагноз на основі різних поєднань діапазонів значень вибраних параметрів ОД. Правила можуть бути записані в різній формі.

Для запису послідовності здійснення тестових процедур і завдання обмежень (якщо вони є) на їх здійснення може бути запропонований аналогічний механізм. Механізм запису послідовності проведення тестових процедур у вигляді правил реалізується, наприклад, таким чином:

ЯКЩО: P2 = 1

ТО: тест = T1, T3, T7,

де T1, T3, T7 – тестові процедури, що подаються на ОД при активізації (спрацьовуванні) відповідної продукції.

У сучасних ЕДС застосовуються різні стратегії пошуку рішення і постановки діагнозу, які дозволяють визначити необхідні послідовності тестових процедур. Проте пріоритет в ЕС віддається перш за все знанням і досвіду, а лише потім логічному виведенню.

1.2.4. Приклади експертних систем

Спершу зробимо короткий екскурс в історію створення ранніх і найбільш відомих ЕС. У більшості цих ЕС як СПЗ використовувалися системи продукцій (правила) і прямий ланцюжок міркувань. Медична ЕС MYCIN розроблена в Стенфордському університеті в середині 70-х років минулого сторіччя для діагностики і лікування інфекційних захворювань крові. MYCIN на сьогодні використовується для навчання лікарів.

ЕС DENDRAL розроблена в Стенфордському університеті в середині 60-х років XX ст. для визначення топологічних структур органічних молекул. Система виводить молекулярну структуру хімічних речовин за даними мас-спектрометрії і ядерного магнітного резонансу.

ЕС PROSPECTOR розроблена в Стенфордському університеті в 1974-1983 роках для оцінки геологами потенційної рудоносності району. Система містить понад 1 000 правил і реалізована на INTERLISP. Програма порівнює спостереження геологів з моделями різного роду покладів руд. Програма залучає геолога до діалогу для отримання додаткової інформації. 1984 року вона точно передбачила існування молібденового родовища, оціненого в багатомільйонну суму.

Розглянемо експертну систему діагностування (ЕСД) цифрових і цифроаналогових пристроїв, в якій використовувалися системи продукцій і фрейми, а також прямий і зворотний ланцюжок міркувань одночасно. Як об'єкт діагностування (ОД) в ЕСД можуть використовуватися цифрові пристрої (ЦП), цифро-аналогові пристрої. Така ЕСД працює спільно з автоматизованою системою контролю і діагностування (АКД), яка подає в динаміці дії на ОД (десятки, сотні й тисячі дій в секунду), аналізує вихідні реакції та робить висновок: придатний або не придатний. У випадку, якщо реакція, що перевіряється ОД не відповідає еталонним значенням, то підключається заснована на знаннях підсистема діагностування. ЕСД запитує значення сигналів в певних контрольних точках і веде оператора по схемі ОД, рекомендуючи йому здійснити вимірювання в певних контрольних точках або підтвердити проміжний діагноз, і в результаті приводить його до місця несправності. Початковими даними для роботи ЕСД є результати машинного моделювання ОД на етапі проектування. Ці результати моделювання передаються в ЕСД на магнітних носіях у вигляді тисяч продукційних правил. Рух по контрольних точках здійснюється на основі моделі, записаної у вигляді мережі фреймів для ОД.

Така ЕСД не була б інтелектуальною системою, якби вона не накопичувала досвід. Вона запам'ятовує знайдену несправність для певного типу ОД. Наступного разу при

діагностиці несправності ОД цього типу вона пропонує відразу ж перевірити цю несправність, якщо реакція ОД говорить про те, що така несправність можлива. Так чинять досвідчені майстри радіоелектронної апаратури (РЕА), що знають «слабкі» місця в конкретних типах РЕА і перевіряють їх в першу чергу. ЕСД накопичує імовірнісні знання про конкретні несправності з метою їх використання під час логічного виведення. При русі по дереву пошуку рішень на черговому кроці використовується критерій – максимум відношення вірогідності (коефіцієнта упевненості) встановлення діагнозу до трудомісткості розпізнавання несправності. Коефіцієнти упевненості автоматично коректуються в часі роботи ЕСД під час кожного підтвердження або не підтвердження діагнозу для конкретних ситуацій діагностування. Трудомісткості елементарних перевірок спочатку задаються експертом, а потім автоматично коректуються в процесі роботи ЕСД.

Серед сучасних комерційних систем варто виділити експертну систему – оболонку G2 американської фірми Gensym (USA) [296] як неперевершену експертну комерційну систему для роботи з динамічними об'єктами. Робота в реальному часі з малим часом відповіді часто необхідна при аналізі ситуацій в корпоративних інформаційних мережах, на атомних реакторах, в космічних польотах і безлічі інших завдань. У цих завданнях необхідно ухвалювати рішення протягом мілісекунд з моменту виникнення критичної ситуації. ЕС G2, призначена для вирішення таких завдань, відрізняється від більшості динамічних ЕС такими характерними властивостями, як:

- робота в реальному часі з розпаралелюванням процесів міркувань;
- структурований природно-мовний інтерфейс із керуванням по меню і автоматичною перевіркою синтаксису;
- зворотний і прямий висновок, використання метазнань, сканування і фокусування;
- інтеграція підсистеми моделювання з динамічними моделями для різних класів об'єктів;
- структуризація БЗ, наслідування властивостей, розуміння зв'язків між об'єктами;
- бібліотеками знань є ASCII-файлами і легко переносяться на будь-які платформи і типи ЕОМ;
- розвинений редактор для супроводу БЗ без програмування, засобу трасування і відлагодження БЗ;
- керування доступом за допомогою механізмів авторизації користувача і забезпечення бажаного погляду на застосування;
- гнучкий інтерфейс оператора, що включає графіки, діаграми, кнопки, піктограми тощо;
- інтеграція з іншими застосуваннями (по TCP/IP) і базами даних, можливість віддаленої та розрахованої на багато користувачів роботи.

Як приклад швидкодіючої системи для відстежування стану корпоративної інформаційної мережі (КГМ) можна привести засновану на знаннях систему моніторингу OMEGAMON фірми Candle (IBM з 2004 р.). OMEGAMON – типовий представник сучасних експертних мультиагентних динамічних систем, що працюють у реальному часі. OMEGAMON дозволяє за лічені хвилини ввести і відлагодити правила моніторингу позаштатних ситуацій для об'єктів. Правило (sitation) записується як продукція. Логічне виведення в такій ЕС реалізоване за допомогою механізму policy, що забезпечує

побудову ланцюжків логічного виведення на основі situations. На рис. 1.3 наведений один з інтерфейсів для заповнення БЗ в ЕС OMEGAMON. На цьому рисунку показана ситуація, що визначає критичну кількість повідомлень у чергах транспортної системи IBM WebSphere MQ (MQSeries).

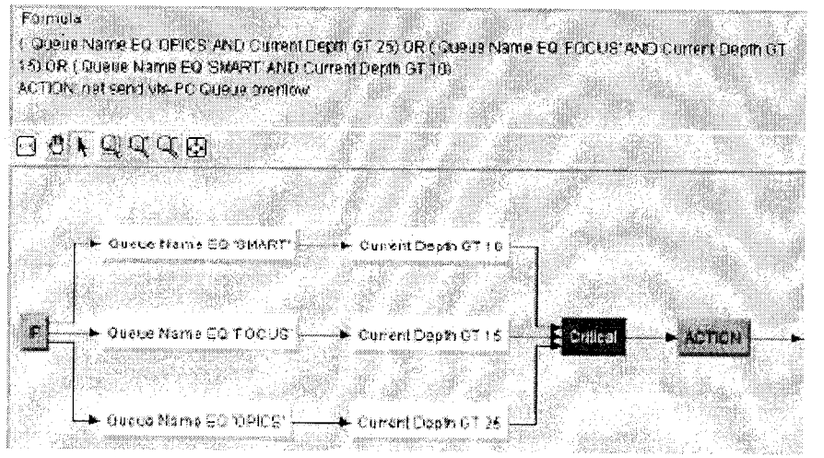


Рис. 1.3. Інтерфейс OMEGAMON для заповнення БЗ.

На рис. 1.4 показані основні компоненти системи OMEGAMON:

- сервер збирання інформації від агентів CandleMana-gementServer (CMS);
- сервер відображення результатів, сповіщення користувачів і налагодження моніторингу CandleNetPortal Server (CNP) зі своїми клієнтами;
- Candle Management Workstation (CMW) – робоча станція адміністратора OMEGAMON;
- Managed Systems – комп'ютери КІМ, на яких працюють агенти.

Агенти OMEGAMON працюють на контрольованих системах (Managed Systems), як першокласні шпигуни: вони непомітні з точки зору використання CPU і оперативні при моніторингу огляду на час постачання своїх донесень в центр (CMS). Вони фіксують критичну ситуацію і забезпечують реакцію (ACTION) менш, ніж за 1 секунду. Все визначається тим інтервалом моніторингу, який задається експертом на основі своїх інтуїтивних знань. Як ACTION при визначенні ситуацій можна використовувати різні типи дій: посилання поштових повідомлень і sms фахівцям супроводу, посилання інформації в інші системи, виконання системних команд тощо. Кількість об'єктів моніторингу (комп'ютерів КІМ) може досягати декількох сотень, і на кожному об'єкті може бути декілька сотень контрольованих параметрів. Кількість платформ (типів операційних систем), на яких працюють агенти, перевищує 30, починаючи від OS/390, OS/400, різні UNIX-платформи (HP_UX, AIX, Solaris) і закінчуючи Windows. На одному сервері можуть працювати декілька агентів, наприклад, для моніторингу WebSphere MQ (MQSeries), WebSphere Application Server, DB-2 і HPUNIX одночасно.

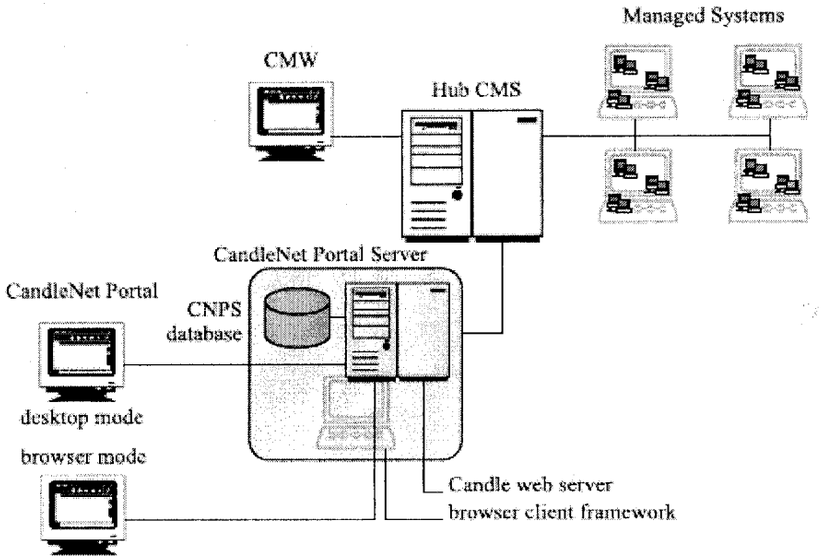


*Рис. **1.4.** Структурна схема ЕС OMEGAMON.*

Сервери CMS і CNP-servers можуть працювати на одному виділеному сервері, як правило, на базі операційної системи Windows. Налагодження ситуацій (situations) і механізмів логічного виведення (policy) здійснюється на робочому комп'ютері адміністратора через CNP-client. Для тільки що створеної ситуації ви натискаєте кнопку Apply і вмить бачите відображення ACTION через CNP-client, через пошту тощо.

Підкреслимо, що заснована на знаннях система моніторингу OMEGAMON – це вельми ефективна система керування обчислювальними ресурсами, надійний і незамінний помічник у пошуках рішень з оперативного усунення критичних і важких для діагностування ситуацій, при аналізі інформаційних потоків, аналізі продуктивності і налагодженні КІМ.

Запитання для повторення та контролю знань

1. Що розуміється під такими поняттями, як «інтелект», «інтелектуалізація», «штучний інтелект»?
2. Які системи називаються інтелектуальними?
3. Які Ви знаєте методи визначення інтелектуальності?
4. Наведіть приклади інтелектуальних задач.
5. Які бувають типи процесів мислення?
6. Що таке силогізми? Наведіть приклади.
7. Які властивості інтелектуальних задач?
8. У чому полягає тест Тюринга?
9. Який діалог називається фатичним? Наведіть приклад.

10. Які Ви знаєте методи побудови псевдоінтелектуальних програм?
11. Які системи називаються кібернетичними?
12. Які Ви знаєте рівні опису кібернетичними системами?
13. Як класифікуються кібернетичні системи?
14. Які існують підходи до керування кібернетичних систем?
15. Які ключові риси інтелекту?
16. Дайте формалізацію понять алгоритмічності та декларативності.
17. Які алгоритми називають квазіалгоритмами?
18. Дайте характеристику інтелектуальних систем із загально-кібернетичних позицій.
19. Як виглядає схема самокерування інтелектуальної системи?
20. Які задачі відносяться до інтелектуальних?
21. Дайте визначення інтелектуальної системи.
22. Як виглядає типова схема функціонування інтелектуальної системи?
23. Які характеристики мають слабкоструктуровані (неструктуровані), неформалізовані задачі?
24. Що є показником інтелектуальності системи?
25. Як формально задається модель предметної області?
26. Який клас інтелектуальних систем складають експертні системи?
27. Як визначається структура експертної системи?
28. За якими ознаками класифікуються експертні системи?
29. Які виникають труднощі під час розроблення експертних систем?
30. Які ви знаєте методології побудови експертних систем?
31. Наведіть приклади експертних систем, що вже існують.

Завдання для самостійного розв'язування

1. Чи є експертною система, програма якої прогнозує погоду на заході України та виводить повідомлення такого виду: „Завтра погода не буде відрізнятись від сьогоднішньої". Будемо вважати, що вона представляє сьогоднішню погоду у символьному вигляді, легко модифікується і розширюється, може пояснити чому вона прийшла до певного висновку, виводячи, наприклад таке повідомлення: „Добові зміни кліматичних умов в цей період року малоймовірні".
2. Чи є експертною система програма, яка формує прогнозування погоди на певну дату (наприклад, 17 січня), середню температуру повітря, кількість опадів?
3. Чи є система пошуку інформації в мережі World Wide Web експертною? Якщо ні, то яких властивостей їй не вистачає для того, щоб класифікуватись як експертна система пошуку потрібної web-сторінки?
4. Придумайте базу знань для експертної системи діагностування захворювань.

РОЗДІЛ 2

МОДЕЛІ ТА МЕТОДИ ФУНКЦІОНУВАННЯ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

- Основні поняття формальної моделі
- Формальне означення інтелектуальної системи
- Функціональна модель інтелектуальної системи
- Пошук та задоволення обмежень
- Приклади побудови моделі керування процесом пошуку розв'язку

Розглядається структурна та функціональна моделі інтелектуальної системи. Сформована задача пошуку та задоволення обмежень. Наводяться приклади побудови моделі керування процесом пошуку розв'язку.

2.1. Основні поняття формальної моделі

Відповідно до основних принципів опису системи, наведених у роботі [117], виділимо основні компоненти системи – її входи та виходи. Входом системи є множина запитів клієнтів системи, виходами – множина відповідей системи на запити.

У загальному випадку реакція інтелектуальної системи визначається не лише станами предметної області (ПО), а й станами самої системи. Стан інтелектуальної системи визначається станом ПО, а також вмістом знань, визначених у системі на заданий момент. Модифікація знань здійснюється за допомогою індуктивної компоненти (досвіду набутого системою у процесі свого функціонування).

Введемо такі позначення:

$X = \{x_1, x_2, ..., x_n\}$ – множина імен об'єктів (предметів, сутностей і т.ін.) деякої ПО;

$C = \{c_1, c_2, ..., c_m\}$ – множина імен властивостей об'єктів;

$R = \{r_1, r_2, ..., r_k\}$ – множина імен відношень, які утворюють об'єкти заданої ПО;

$B = \{b_1, b_2, ..., b_l\}$ – множина імен операторів, які можна виконувати над об'єктами ПО, змінюючи їх властивості та відношення між ними;

$H = \{h_1, h_2, ..., h_s\}$ – множина імен евристичних функцій (метазнань), визначених у заданій ПО;

D – індуктивна компонента (досвід).

Стан ПО не є постійним, а змінюється в часі. Тому визначимо стан ПО в момент часу t як кортеж

$$E(t) = \langle X(t), C(t), R(t) \rangle. \quad (2.1)$$

Формула (2.1) описує ситуацію, у якій перебуває ПО в момент часу t. Залежно від стану ПО та наявних метазнань інтелектуальна система вибирає з множини B можливих операторів той, який, на її думку, необхідно застосувати, щоб знайти розв'язок задачі.

Позначимо через $A = E \times B \times H$ декартовий добуток множин даних, знань та метазнань.

Позначимо через u відображення стану ПО $E(t)$ на множину B. Тоді

$$u : \langle X(t), C(t), R(t) \rangle \xrightarrow{H} B. \quad (2.2)$$

Формула (2.2) відображає взаємозв'язок між станами ПО та операторами. Можна сказати, що стани ПО проектуються на множину операторів.

Процедура розв'язування задачі

Нехай $E(0)$ початковий стан ПО. Задача інтелектуальної системи полягає в тому, щоб перевести ПО із стану $E(0)$ в деякий кінцевий стан, який називають станом мети і позначають $E(z)$. Це можливо зробити, якщо застосовувати оператори з множини B до відповідних станів ПО. Необхідно з множини B вибрати оператор b_j, який буде застосовано до поточного стану ПО. Схематично процес розв'язування задачі виразимо формулою

$$P : E(0) \xrightarrow{B} E(z). \quad (2.3)$$

Якщо для переходу з початкового стану в кінцевий було використано послідовність операторів $(b_1, b_2, ..., b_j)$, то ця послідовність є алгоритмом розв'язування задачі.

Задачу модифікації знань та метазнань за допомогою індуктивної компоненти D розіб'ємо на дві окремі задачі, які стосуються окремо знань та метазнань. Подамо це формально у вигляді:

$$D_1 : A \xrightarrow{D} B, \quad (2.4)$$

$$D_2 : A \xrightarrow{D} H, \quad (2.5)$$

де формула (2.4) формалізує постановку задачі модифікації множини знань, а (2.5) – модифікацію множини метазнань.

Для модифікації множин знань та метазнань відомі такі алгоритми:

- **асоціативні правила** (алгоритми AIS, SETM, Apriori, AprioriTid, AprioriHybrid, Basic, Cumulate, Stratify, Estimate, EstMerge, Max-Miner, ID3, C4.5 тощо);
- **послідовні зв'язки** (алгоритми AprioriAll, AprioriSome, DynamicSome, GSP, Minepi, Winepi, FreeSpan, PrefixSpan);
- **класифікація** (алгоритми IC, SLIQ, SPRINT, LQCLR, Case-based reasoning, Rough sets тощо);
- **прогнозування** (алгоритми лінійної регресії, поліноміальної регресії, пуасонівської регресії, логлінійної регресії);
- **кластеризація** (алгоритми розділення (EM, PAM, CLARA), ієрархічні алгоритми (AGNES, DIANA, BIRCH, CURE, ROCK, CHAMELEON), алгоритми густини (DBSCAN, OPTICS, DENCLUE), алгоритми ґратки (STING, WaveCluster, CLIQUE), алгоритми моделювання (DBCLASS, COBWEB, CLASSIT, AutoClass), нейронні мережі).

2.2. Формальне означення інтелектуальної системи

Формальна математична модель інтелектуальної системи має характеризувати її як структуру, що базується на множинах вхідних сигналів, відповідей та станів, і має властивість поповнення знань (модифікує множину операцій та множину евристичних функцій).

Інтелектуальна система S – кортеж чотирьох елементів

$$S = \langle Q, V, St, D \rangle, \tag{2.6}$$

де $Q \subseteq A$ – **вхідна** сукупність інформації, що мінімально необхідна для початку функціонування інтелектуальної системи; $V \subseteq A$ – **вихідна** інформація, яку генерує інтелектуальна система у відповідь на вхідну інформацію, яку надає клієнт; це є можливі відповіді, що видає інтелектуальна система; St – **стан інтелектуальної системи**: унікальний набір характеристик, який повністю описує поточний стан інтелектуальної системи; стан інтелектуальної системи визначається станом ПО та наявними знаннями в системі, тобто

$$St = \langle E, B, H \rangle; \tag{2.7}$$

D – **індуктивна компонента,** означена формулами (2.4) та (2.5).

Графічно структурна модель наведена на рис. 2.1.

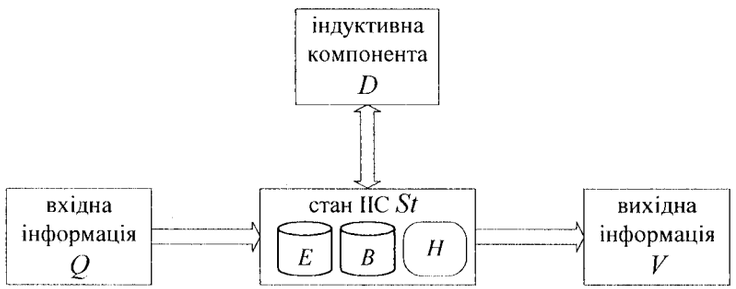


Рис. 2.1. Структурна модель інтелектуальної системи.

2.3. Функціональна модель інтелектуальної системи

Процес функціонування інтелектуальної системи відбувається циклічно. У циклі розрізняють два етапи:

1) вибір релевантного оператора;

2) виконання оператора.

2.3.1. Процес вибору релевантного оператора

Основною задачею функціонування інтелектуальної системи є задача вибору з множини операторів того оператора, який, на думку системи, має виконатись у черговому циклі. Такий оператор називають релевантним. Нехай інтелектуальна система має розв'язати задачу $P: E(0) \xrightarrow{B} E(z)$. Розглянемо $(t+1)$-цикл. Процес вибору релевантного оператора розіб'ємо на три етапи:

- вибір стану ПО,
- співставлення,
- розв'язання конфліктів.

Вибір стану ПО

Нехай під час розв'язування задачі ПО перебувала у станах $E(0), E(1), ..., E(t)$. Множину таких станів позначимо $\overline{E} = \{E(0), E(1), ..., E(t)\}$. Тоді задача першого етапу полягає у виборі із множини станів ПО $\overline{E}$ стану $E(i) \in \overline{E}$, з якого продовжується функціонування інтелектуальної системи в $(t+1)$-циклі. Для вибору стану використовують евристичні функції, якщо такі визначені в ПО. Формально перший етап буде мати вигляд:

$$q: \overline{E} \xrightarrow{H, D_2} E(i). \tag{2.8}$$

Беручи до уваги (2.5) отримаємо

$$E(i) = q(E, H, D_2). \tag{2.9}$$

Співставлення

Етап співставлення визначає множину операторів, які можна застосувати до вибраного стану ПО $E(i)$. Множину таких операторів називають **конфліктним набором** [0]. Позначимо його $B^{t+1}(i) = \{b_{i_1}^{t+1}, b_{i_2}^{t+1}, ..., b_{i_l}^{t+1}\}$. Верхній індекс вказує, що оператори вибираються у момент $(t+1)$-циклу. Згідно з (2.4) множина операторів могла змінитись, на відміну від тієї, якою вона була на початку функціонування системи. Необхідність модифікації множини операторів буде розглянута у другому розділі. Елементи множини $B^{t+1}(i)$ називають **елементами конфліктного набору**, підкреслюючи те, що до роботи готовий набір операторів, але механізм виведення ще не знає, якому із них надати перевагу. Формально етап співставлення набуде вигляд:

$$w: E(i) \xrightarrow{D_1} B^{t+1}(i). \tag{2.10}$$

Розв'язання конфліктів

З конфліктного набору обирається релевантний оператор, який буде виконуватися в $(t+1)$-циклі. Тобто, необхідно оцінити елементи множини $B^{t+1}(i)$ щодо їхньої корисності для досягнення кінцевого стану ПО $E(z)$. Введемо функцію u – оцінки корисності вибору оператора, яку називають **керівною**. Це "вільна функція", яка є в нашому розпорядженні. Вважатимемо, що із системою пов'язаний деякий суб'єкт, який має право приймати рішення, тобто здійснювати вибір керівної функції. Релевантний оператор позначають b_p^{t+1}.

Для вибору релевантного оператора використовуємо евристичні знання, якщо такі визначені в ПО. Тоді етап вирішення конфліктів подамо у вигляді:

$$u: B^{t+1}(i) \xrightarrow{H, D_2} b_p^{t+1}. \tag{2.11}$$

Беручи до уваги (2.4), отримаємо

$$b_p^{t+1} = u(B^{t+1}(i), H, D_2). \tag{2.12}$$

Подамо етап розв'язання конфліктів (вибір релевантного оператора) як розв'язок системи булевих рівнянь (СБР). СБР будують для поточного циклу, тому надалі верхній індекс використовувати не будемо.

Введемо таку булеву функцію:

$$u\left(b_j\right)=u_j=\begin{cases}1, & b_p=b_j\\ 0, & b_p\neq b_j\end{cases}. \qquad (2.13)$$

Тоді процес вибору релевантного оператора можна подати у вигляді СБР, яка у загальному матиме вигляд:

$$u_1\vee u_2\vee\ldots\vee u_{i_t}=1. \qquad (2.14)$$

Така система має i_t розв'язків. У складних системах, де міститься велика кількість знань, значення i_t є доволі велике, що збільшує простір пошуку розв'язку, тобто, веде до комбінаторного вибуху. Тому необхідно зменшити кількість можливих розв'язків, тобто, ввести додаткові обмеження. Такими додатковими обмеженнями виступає множина евристичних функцій H. Евристичну функцію $h\in H$ будемо розглядати як булеве рівняння:

$$h_j\left(B(i)\right)=u_{j_1}\vee u_{j_2}\vee\ldots\vee u_{j_k}=1. \qquad (2.15)$$

Тоді процес вирішення конфліктів будемо розглядати як розв'язання наступної СБР:

$$\begin{cases}u_1\vee u_2\vee\ldots\vee u_{i_t}=1\\ h_1\left(B(i)\right)=1\\ h_2\left(B(i)\right)=1\\ \ldots\\ h_{s_i^{t+1}}\left(B(i)\right)=1\end{cases}, \qquad (2.16)$$

де s_i^{t+1} – кількість евристичних функцій, які можна використати для стану ПО $E(i)$, множини операторів $B(i)$ під час $(t+1)$-циклу.

Потужність множини операторів $B(i)$ можна зменшити за рахунок індуктивної компоненти D. Таке скорочення кількості елементів множини $B(i)$ може призвести до того, що система (2.16) не матиме розв'язку. Тому необхідно зменшити і кількість евристичних функцій.

2.3.2. Виконання оператора

За допомогою релевантного оператора, визначеного на етапі вирішення конфліктів, переводимо ПО у новий стан. Формально це подамо у вигляді:

$$y: E(i)\xrightarrow{b_p^{t+1}}E(t+1). \qquad (2.17)$$

Після виконання релевантного оператора перевіряється умова закінчення задачі і, якщо вона не задовольняється, то виконується черговий цикл. Вигляд цих умов розглянемо нижче.

Повертаючись поетапно назад, цикл інтерпретатора формально запишеться у вигляді

$$\begin{aligned}E(t+1)&=y\left(E(i),b_p^{t+1}\right)=y\left(E(i),u\left(B^{t+1}(i)\right),H,D_2\right)=\\ &=y\left(E(i),u\left(w\left(E(i),D_1\right)\right),H,D_2\right)=\\ &y\left(q\left(\bar{E},H,D_2\right),u\left(w\left(q\left(\bar{E},H,D_2\right),D_1\right)\right),H,D_2\right)=i\left(\bar{E},H,D\right)\end{aligned}. \qquad (2.18)$$

Механізм, який здійснює цикл, називають **інтерпретатором**. Функцію $i\left(\bar{E},H,D\right)$, яка визначає цикл, називають **моделлю керування процесом пошуку розв'язку**.

Отже, цикл інтерпретатора складається з чотирьох етапів: ***вибору стану, співставлення, вирішення конфліктів, виконання*** (рис. 2.2). Ці етапи слідують один за одним у строго визначеному порядку.

Отриманий за формулою (2.18) стан ПО $E(t+1)$ повинен визначати її новий стан. Випадок, в якому ПО повертається у деякий минулий стан, розглядатись не може, бо це задача першого етапу, а не всього циклу. Тому введемо наступне обмеження:

$$E(t+1)\notin\bar{E}. \qquad (2.19)$$

Отже, цикл інтерпретатора треба реалізувати так, щоб виконувалась умова (2.19). Тепер новий стан ПО заносимо у множину її станів:

$$\bar{\bar{E}}=\left\{E(0),E(1),\ldots,E(t)\right\}\cup\left\{E(t+1)\right\}. \qquad (2.20)$$

Послідовність релевантних операторів позначатимемо $B^{t}=\left(b_p^{(1)},b_p^{(2)},\ldots,b_p^{(t)}\right)$. Після чергового циклу інтерпретатора, ця множина також поповнюється новим елементом, визначеним згідно з (2.12) або як розв'язок СБР (2.16).

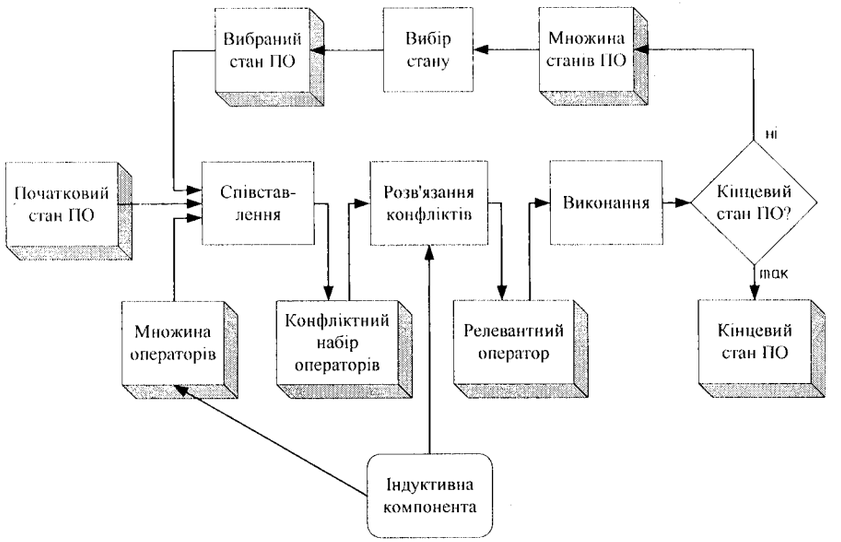


Рис. 2.2. Цикл роботи інтерпретатора.

2.3.3. Обмеження функціонування інтелектуальної системи

Розглянемо такі поняття.

Стан інтелектуальної системи називається **безвихідним**, якщо на етапі співставлення отримуємо пусту множину операторів.

Множину безвихідних станів інтелектуальної системи позначимо St'. Множину станів ПО, що входять у елементи множини St', позначимо відповідно E'.

Стан інтелектуальної системи називається **зацикленим**, якщо виконання будь-якого оператора веде у попередній стан системи.

Множину зациклених станів інтелектуальної системи позначимо St''. Множину станів ПО, що входять у елементи множини St'', позначимо відповідно E''.

Згідно з цими означеннями та (4.19) отримаємо наступне твердження.

На першому етапі циклу інтерпретатора (вибір стану ПО) не може вибратися стан, що входить у множину E' або E''. Тобто

$$q\left(\overline{E},H,D\right)=E\left(i\right)\notin E'\cup E''. \tag{2.21}$$

Зацикленням інтелектуальної системи називається такий момент функціонування ІС, коли всі стани системи є зацикленими або безвихідними.

Зациклення інтелектуальної системи означає невиконання умови (2.21). Це призводить до припинення процесу розв'язування задачі, тобто, задача не має розв'язку. Момент зациклення інтелектуальної системи має повідомлятися користувачеві.

2.4. Пошук та задоволення обмежень

2.4.1. Задача знаходження задовільних розв'язків

Мета функціонування інтелектуальної системи полягає у знаходженні такого стану ПО $E\left(z\right)$, об'єкти, властивості об'єктів та відношення між об'єктами якого задовольняють користувача. З кожною предметною областю та користувачем пов'язана множина невизначеності Ω (множина всіх можливих факторів, що впливають на результат). Тому **функцію допустимості розв'язку** запишемо у вигляді $\tau(\omega)$, де $\omega\in\Omega$,

$$\tau:\Omega\to\Upsilon, \tag{2.22}$$

де Υ – частково або повністю впорядковане відношення $\leq$.

Введемо функцію оцінки (якості) роботи інтелектуальної системи як відображення:

$$g:B'\left(i\right)\times\Omega\times E\left(t+1\right)\to\Upsilon. \tag{2.23}$$

Згідно з (2.23) отримаємо

$$g=g\left(B,\omega,i\right). \tag{2.24}$$

Тоді **критерій задовільності розв'язку** набуде такого вигляду: необхідно знайти стан ПО $E\left(z\right)$, що

$$g\left(B,\omega,i\right)\leq\tau\left(\omega\right). \tag{2.25}$$

Нерівність (2.25) визначає умову закінчення процесу пошуку розв'язку.

Наведемо приклади критеріїв задовільності.

Приклад 1. Розглянемо інтелектуальну пошукову систему, яка здійснює пошук інформації за деяким взірцем. Знайдений документ у кінцевому варіанті $E\left(z\right)$ має задовольняти деяку множину вимог Ω. Визначимо функцію τ як функцію, що задає 10 % невиконання цих умов. Функція g показує, на скільки процентів умови не задовольняються.

Приклад 2. У багатьох прикладних задачах, для яких використовується апарат ІС, необхідно отримати логічну відповідь у формі "Так" або "Ні" на деяке запитання. Визначимо функцію τ, як функцію, що приймає лише два значення: 0 або 1, де 0 – відповідь хибна, а 1 – істинна. Визначимо функцію g як відображення процесу отримання даних $E\left(z\right)$, про які йшлося у питанні, на множину $\{0,1\}$. Вважатимемо, що $g=1$, якщо $E\left(z\right)$ досягнуто. Тоді критерій задовільності запишеться у вигляді: $g\left(z,i\right)=1$.

Можливий варіант, що $E\left(z\right)$ не досягається і тоді процес пошуку розв'язку буде нескінченним. Щоб позбутись таких наслідків, необхідно ввести додаткову умову на процес пошуку розв'язку. Отже, необхідно ввести ще одну додаткову умову на витрати ресурсів інтелектуальною системою (кількість циклів інтерпретатора, час розв'язування задачі тощо).

Позначимо через Ξ множину ресурсів інтелектуальної системи. Тоді критерій витрат ресурсів інтелектуальною системою подамо у вигляді:

$$\psi\left(\xi\right)\leq\upsilon, \tag{2.26}$$

де $\xi\in\Xi$, ψ – функція оцінки витрат ресурсів, $\upsilon\in\Upsilon$.

Загалом, модель функціонування інтелектуальної системи зводиться до знаходження такої моделі керування процесом пошуку розв'язку (2.18), щоб виконувалась умова (2.19) та задовольнялись умови (2.25) і (2.26). Умови (2.19) і (2.26) є жорсткими умовами. Коли ці умови не задовольняються, процес розв'язування задачі припиняється, про що повідомляється користувачеві. Особливо складною для правильного визначення є умова (2.26). Її вибір залежить від показників програмного комплексу. Для багатьох простих задачах її можна взагалі не вказувати.

Модель керування процесом пошуку розв'язку залежить від вибору функцій на першому та третьому етапах циклу інтерпретатора. Тобто вільними є функції q та u. Саме від їх правильного визначення залежить процес пошуку розв'язку задачі.

2.4.2. Ланцюжки виведення

Ланцюжок виведення – послідовність операторів із множини релевантних операторів B^{p} таких, що вибраний стан ПО на першому етапі циклу інтерпретатора, збігається із кінцевим станом попереднього циклу.

Ланцюжок виведення до стану ПО $E\left(k\right)$ від початкового стану $E\left(0\right)$ позначають $L_k=\left(b_p^{(k_1)}b_p^{(k_2)}\ldots b_p^{(k_n)}\right)$.

Перехід з $E\left(0\right)$ в $E\left(k\right)$ можна відобразити таким ланцюжком:

$$E\left(0\right)\xrightarrow{b_p^{(k_1)}}E\left(k_1\right)\xrightarrow{b_p^{(k_2)}}E\left(k_2\right)\xrightarrow{b_p^{(k_3)}}\ldots\xrightarrow{b_p^{(k_n)}}E\left(k\right).$$

Число k_n називають відстанню між початковим станом $E\left(0\right)$ і станом $E\left(k\right)$, та позначають $d_k=n$.

Ланцюжок виведення до кінцевого стану ПО $E\left(z\right)$ від початкового стану $E\left(0\right)$ матиме вигляд $L_z=\left(b_p^{(z_1)}b_p^{(z_2)}\ldots b_p^{(z_n)}\right)$.

2.4.3. Взаємозалежність між етапами циклу інтерпретатора

Як було сказано, модель керування процесом пошуку розв'язку залежить від вибору функцій на першому та третьому етапах циклу інтерпретатора. Тобто, вільними є функції

q та u. Саме від їх правильного визначення залежить процес пошуку розв'язку задачі.

Насправді функції q та u є взаємопов'язані. Якщо функція u визначає оператор, що входить у ланцюжок виведення L_z, то функція q для наступного циклу інтерпретатора вибирає стан ПО, який є останнім станом, до якого перейшла система. Тобто,

$$E(i)=q(\overline{E},H,D)=E(t). \tag{2.27}$$

Підсумовуючи наведене, отримаємо висновок, що основною задачею циклу інтерпретатора є етап вирішення конфліктів. Тому при дослідженні задач підвищення ефективності функціонування інтелектуальної системи, основну увагу приділяють проблемі прийняття рішення щодо вибору релевантного оператора.

2.5. Приклади побудови моделі керування процесом пошуку розв'язку

Методи повного перебору

Повний перебір отримуємо, коли в інтелектуальній системі не визначені евристичні функції та не враховується досвід. Тобто, модель керування процесом пошуку розв'язку має вигляд:

$$E(t+1)=i(\overline{E}). \tag{2.28}$$

Тоді стан інтелектуальної системи збігається зі станом ПО.

Розглянемо процес розв'язування задачі. Нехай ПО перебуває у стані $E(t)$. Визначимо функції q, u наступним чином.

$$E(i)=q(\overline{E})=\tilde{q}(e_t)=\begin{cases}\varnothing, & t=-1\\ E(t), & E(t)\notin E'\cup E''\\ \tilde{q}(e_{t-1}), & e_t\in E'\cup E''\end{cases}. \tag{2.29}$$

Функція $\tilde{q}$ є рекурсивною. Як стан ПО для циклу інтерпретатора обирається останній стан, який не належить множині безвихідних чи зациклених станів. Якщо таких станів немає, то задача не має розв'язку.

Нехай $B^{t+1}(i)=\{b_{i_1}^{t+1},b_{i_2}^{t+1},...,b_{i_t}^{t+1}\}$ множина операторів, отримана на етапі співставлення. Через те, що у ПО не визначені евристичні функції та не враховується досвід, етап вирішення конфліктів запишеться у вигляді:

$$b_p^{t+1}=u(B^{t+1}(i)),$$

а СБР матиме вигляд (2.14). Це означає, що будь-який оператор може бути релевантним.

Визначимо функцію u у вигляді:

$$b_p^{t+1}=u(B^{t+1}(i))=\tilde{u}(b_j)=\begin{cases}b_j, & b_j\notin B^p\\ \tilde{u}(b_{j+1}), & b_j\in B^p\end{cases}, \tag{2.30}$$

де індекс j на початку рівний 1.

Функція $\tilde{u}$ також є рекурсивною. Оператор b_p^{t+1} визначається так, щоб після етапу виконання задовольнялась умова (2.19).

Модель керування процесом пошуку розв'язку, отримана за формулами (2.29), (2.30), у літературі відома, як пошук у глибину [22].

Визначимо функцію q іншим способом. Вибраним буде той стан, відстань від якого до початкового стану системи є найменшою та не належить до множин E' та E''.

$$E(i)=q(\overline{E})=\underset{E(j)\notin E'\cup E''}{\arg\min} d_j. \tag{2.31}$$

Тоді модель керування процесом пошуком розв'язку, отримана за формулами (2.30), (2.31), у літературі відома, як пошук у ширину [22].

Приклад (Задача про вісім ферзів)

Існує багато різних підходів до проблеми пошуку шляху для задач, сформованих у термінах простору станів.

Розглянемо стратегію пошуку в глибину та задачу про 8 ферзів як ілюстрацію її застосування.

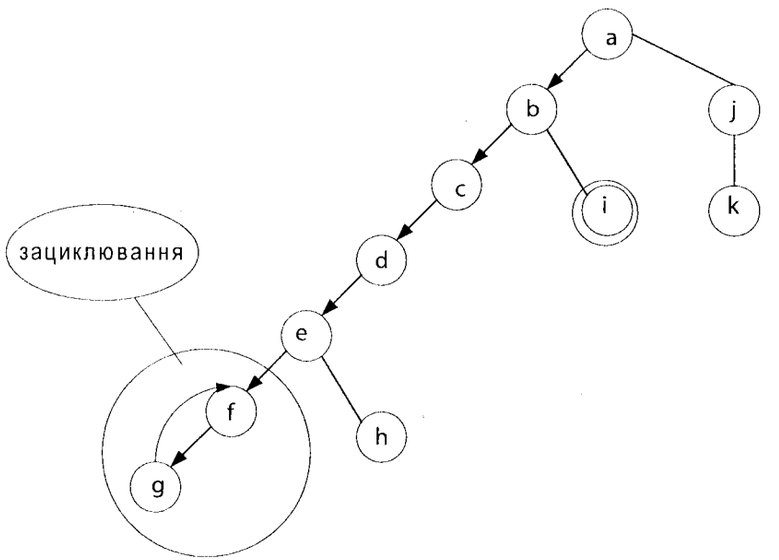


Рис. 2.3. Демонстрація пошуку в глибину

В основі пошуку в глибину (рис. 2.3) лежить ідея вибору «найглибшої» вершини, тобто вершини, яка найбільше віддалена від стартової. Якщо досягнуто кінцевої вершини і потрібного шляху не знайдено, то відбувається повернення на рівень вище та робота алгоритму продовжується доти, поки такий шлях не буде знайдено або не буде переглянуто усі вершини та встановлено факт відсутності потрібного шляху.

У чистому вигляді пошук у глибину має значний недолік: у ході роботи програми можливі зациклювання, якщо предок має лише одного нащадка і цей нащадок не є цільовою вершиною. Одним із методів розв'язання цієї проблеми є введення заборони двічі розглядати одну і ту ж вершину.

Пошук у глибину є найадекватнішим для рекурсивного стилю програмування. Рекурсивне визначення означає, що якесь поняття використовується для його ж визначення. Наприклад, двійкове дерево оголошується як:

- порожнє дерево;
- складене з кореня, лівого піддерева, правого піддерева. Одним із способів застосування методології пошуку в глибину є задача про вісім ферзів.

Задача полягає у розміщенні восьми ферзів на шаховій дошці таким чином, щоб жоден з них «не бив» іншого. Вважатимемо дошку звичайних розмірів (8 х 8). Кожна позиція характеризується координатою X та координатою Y. Нагадаємо, що ферзь може бити по горизонталі, по вертикалі та по діагоналі у різні сторони.

Як спосіб подання позиції на дошці оберемо подання позиції у вигляді списку з восьми елементів з координатами X та Y та номером вертикалі та горизонталі и та V відповідно, причому:

$$U = X - Y,\ V = X + Y,$$

тобто усі характеристики місця розміщення ферзя залежні між собою.

Звідси області визначення

Dx = [1, 2, 3, 4, 5, 6, 7, 8];

Dy = [1, 2, 3, 4, 5, 6, 7, 8];

Du = [-7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7];

Dv = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16].

Задача пошуку розміщення ферзів зводиться до вибору восьми четвірок X, Y, и, V, що входять у відповідні області визначення, так, щоб жоден з елементів не обирався двічі з одної області визначення.

Використання евристичних функцій

Нехай в ПО визначена множина евристичних функцій $H = \{h_1, h_2, ..., h_s\}$. Тоді модель керування пошуку розв'язку в такій системі залежить від станів ПО та евристичних функцій. Тобто,

$$E(t+1) = i(\overline{E}, H).$$

Нехай $E(z)$ – кінцевий стан.

Евристичну оцінку близькості вибраного стану $E(i)$ і кінцевого $E(z)$ задамо евристичною функцією $\tilde{h}(E(i), E(z))$. При такій мірі близькості релевантний оператор вибирається так, щоб оцінка $\tilde{h}(E(t+1), E(z))$ була найменшою. Серед можливих оцінок $\tilde{h}$ існує така оцінка $\tilde{h}^*(E(i), E(z))$, при якій довжина виведення з $E(i)$ до $E(z)$ буде мінімальною. У теорії оптимального керування вона має назву функції Беллмана і визначається з наступних співвідношень:

$$\tilde{h}^*(E(i), E(z)) = \begin{cases} 1 + \min \tilde{h}^*(E(t+1), E(z)), \text{ якщо } E(t+1) \neq E(z), \\ 0, \text{ якщо } E(t+1) \equiv E(z), \\ \infty, \text{ якщо } E(z) \text{ недосяжне з } E(i). \end{cases} \quad (2.32)$$

Функція Беллмана визначає не тільки найкоротшу довжину виведення, але й найоптимальніший шлях виведення.

Знайти і запам'ятати точний розв'язок (2.32) для складних систем є неможливим. Це, як у шахах, у принципі існує оптимальна стратегія, яка визначається функцією Беллмана, але ні саму функцію, ні стратегію реально обчислити неможливо.

Як частковий випадок, евристичну функцію для визначення вибраного стану можна визначити у вигляді суми двох складових. Перша з них – оцінка оптимального шляху з початкового стану ПО $E(0)$ у вибраний стан $E(i)$ $h_1(E(0), E(i))$. Друга – оцінка оптимального шляху із вибраного стану в кінцевий $h_2(E(i), E(z))$. Функцію q визначимо у вигляді:

$$q(\overline{E}, H) = E(i) = \arg\min_{E(j)} (h_1(E(0), E(j)) + h_2(E(j), E(z))). \quad (2.33)$$

Вибраним стає такий стан, для якого значення функції $h_1(E(0), E(j)) + h_2(E(j), E(z))$ є мінімальним. У такому випадку, ми отримаємо алгоритм А* .

Методи пошуку у графах, зокрема, у деревовидних структурах, широко використовуються як у дослідженні операцій, так і в побудові інтелектуальних систем. Пошук у графах для розв'язання задач, як правило, неможливий без вирішення проблеми комбінаторної складності, яка виникає через швидкий ріст альтернатив.

Один з шляхів – це використання евристичної інформації про задачу – одержання евристичних оцінок для вершин простору станів.

2.5.1. Опис А-алгоритму як евристичного пошуку

Розв'язок довільної задачі розглядається як пошук шляху у графі від деякої фіксованої вершини (початкового стану) до однієї з вершин, які належать до раніше вибраної підмножини – множини початкових станів. Цей граф, що характеризує простір станів, містить ще множину операторів, які дозволяють породжувати нові стани із заданих.

Застосування дозволеного оператора до деякого стану («батьківського») подається у вигляді дуги, скерованої від батьківського стану до вершини-нащадка, яка називається «дочірнім» станом. Кожній дузі призначається деяка вартість c (n,n'), подана додатнім числом. Мета пошуку полягає у побудові шляху між початковим станом та одним з кінцевих станів з мінімальною вартістю, причому вартість шляху розраховується як сума вартостей дуг, що входять у нього. Методологія пошуку полягає у послідовному спусканні по дереву від початкового стану доти, поки не буде досягнутий деякий кінцевий стан. Якщо провести порівняння з пошуком у ширину, то можна помітити, що пошук у глибину обирає того кандидата, глибина якого найменша, а пошук за допомогою А-алгоритму для кожного кандидата визначає оцінку, і обирається кандидат з найкращою оцінкою. Граф, що відповідає заданому етапу пошуку, називається графом пошуку. Коли не вдається досягнути кінцевого стану у графі пошуку, виникає задача вибору деякого стану, до якого буде застосовуватись оператор з метою породження нових станів («розкриття» станів). Цей вибір керується оцінювальною функцією, яка кожному здатному до відкриття стану n ставить у відповідність додатне число f(n), яке визначається у вигляді суми:

$$f(n) = g(n) + h(n). \quad (2.34)$$

Функція g характеризує вартість найкоротшого шляху між початковим станом та станом n, відому на поточному етапі виконання алгоритму, а функція h – вартість, яка залежить від стану n; її зазвичай інтерпретують як оцінку мінімальної вартості шляху між станом n і якимось із кінцевих станів. Стан для подальшого відкриття вибирається з числа тих, у яких оцінювальна функція є мінімальною.

2.5.2. Опис A*-алгоритму

Алгоритм пошуку назвемо допустимим, якщо він завжди знаходить оптимальний розв'язок.

Якщо граф пошуку скінчений, а евристичний член h(n) є нижньою оцінкою мінімальної вартості шляху між станом n і якимось із кінцевих станів (h*(n)), то має місце застосування алгоритму пошуку типу A* і тоді забезпечується виконання наступних умов:

- виконується зупинка програми, що реалізує цей алгоритм;
- така зупинка дозволяє або знайти деякий шлях до кінцевого стану (якщо такий шлях існує), або підтвердити факти відсутності шляху;
- коли шлях до кінцевого стану знайдений, то він має мінімальну вартість.

У загальному випадку – якщо не забезпечується наявність такої нижньої оцінки h(n), алгоритм має тип A (а не A*) і виконуються тільки властивості 1 та 2.

Зазначимо, що

$$h(n) \leq h^*(n).$$

Якщо h(n) = 0 і c (n, n') = 1 для всіх дуг простору станів, то ми отримуємо пошук у ширину.

2.5.3. Методологія розв'язування гри у вісімки

Як приклад використання евристичних функцій розглянемо відому гру у вісімки. Необхідно з деякого початкового стану перевести гру в кінцевий стан (як приклад див. рис. 2.4).

1. Вершина простору станів – деяка конфігурація з фішок на гральній дошці, наприклад, список поточного положення фішок.
2. Кожне розміщення визначається парою координат (x, y).
3. Елементи списку розташовані у такому порядку: поточне розміщення порожньої клітинки, поточне розміщення фішки 1, поточне розміщення фішки 2 і т. д.
4. Довжина усіх дуг – 1.
5. Евристична функція h визначається як кількість фішок не на своєму місці.
6. Функція g визначається як довжина шляху від поточної вершини до початкової вершини.

Ця евристична функція добре працює у тому розумінні, що ефективно скеровує пошук до кінцевої мети. У більшості випадків, якщо початкова комбінація фішок має розв`язок, кінцева комбінація знаходиться з мінімальною кількістю повернень. Проте введена евристична функція не є допустимою, тобто нема гарантії, що коротший шлях знайдеться швидше, ніж довший. Справа у тому, що для функції h умова $h \leq h^*$ виконується не для всіх вершин простору станів.

На рис. 2.4 наведено приклад простору станів гри у вісімку. Зліва біля кожного стану наведено значення функції f, над деякими станами зазначено порядок їх проходження відповідно до вищенаведеного алгоритму. Так значення функції f для початкового стану рівне 6, оскільки g=0, а h=6 (не на своєму місці 1, 2, 4, 5, 6, 8). З початкового стану можливі 3 ходи, обирається середній, бо для нього значення f найменше. І так далі. Кінцевий стан ми досягаємо за 10 ходів (не рахуючи ходів повернення у попередні стани). Тобто зроблено 3 зайві ходи (їх номери 3, 4, 5).

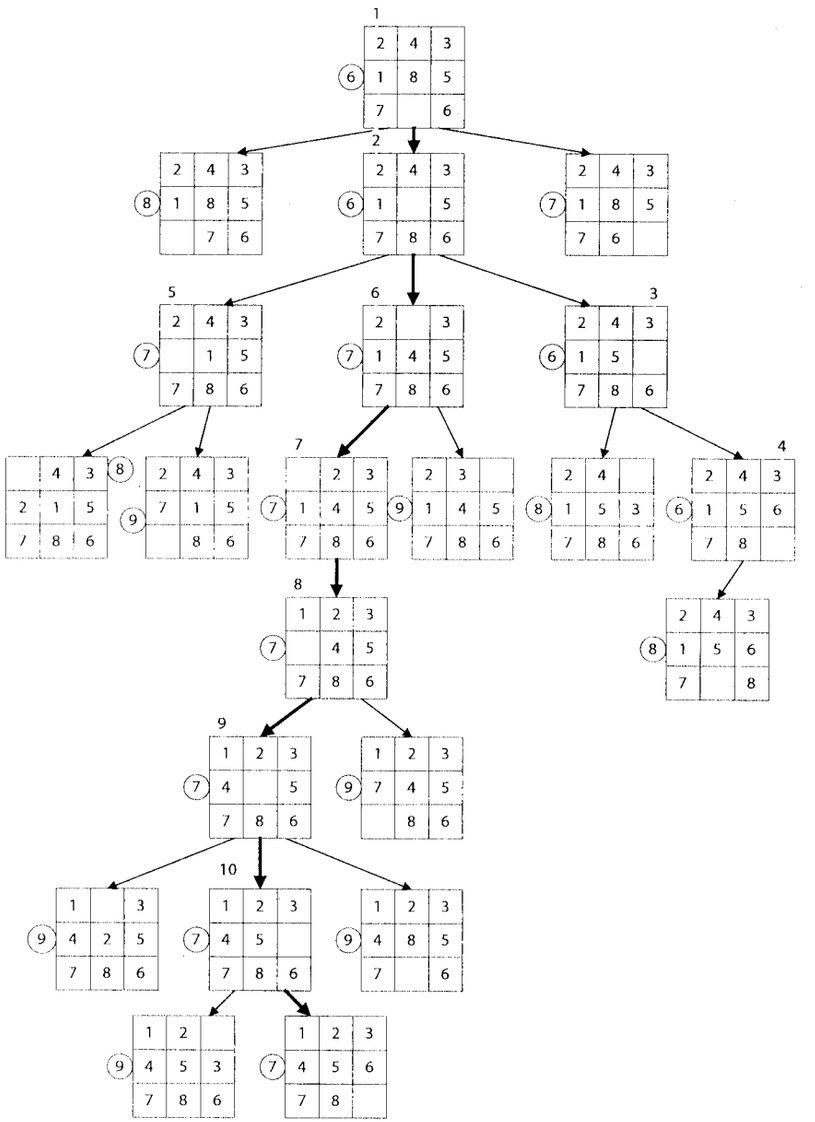


Рис. 2.4. Приклад простору станів для гри у вісімку

Запитання для повторення та контролю знань

1. Як визначається стан інтелектуальної системи?
2. У чому полягає процедура розв'язування задачі інтелектуальною системою? Дати широкий та схематичний зміст.
3. Які алгоритми відомі для модифікації множин знань та метазнань?
4. Як визначається математична формальна модель інтелектуальної системи?

5. Як визначається структурна модель інтелектуальної системи?
6. Які етапи функціонування інтелектуальної системи?
7. Яка основна задача функціонування інтелектуальної системи?
8. Які є етапи процесу вибору релевантного оператора?
9. Який механізм називають інтерпретатором?
10. Дайте означення моделі керування процесом пошуку розв'язку.
11. Як виглядає цикл роботи інтерпретатора?
12. Які накладаються обмеження на функціонування інтелектуальної системи?
13. Який стан інтелектуальної системи називається безвихідним?
14. Який стан інтелектуальної системи називається зацикленим?
15. Який момент функціонування інтелектуальної системи називається зацикленням?
16. У чому полягає мета функціонування інтелектуальної системи?
17. Сформуйте критерій задовільності розв'язку.
18. Для чого вводиться умова на витрати ресурсів інтелектуальною системою?
19. Як виглядає модель функціонування інтелектуальної системи?
20. Яка послідовність операторів називається ланцюжком виведення?
21. Яка взаємозалежність між етапами циклу інтерпретатора?
22. Які методи повного перебору Ви знаєте?
23. Яка модель керування процесом пошуку розв'язку називається пошуком у глибину?
24. Яка модель керування процесом пошуку розв'язку називається пошуком у ширину?
25. Наведіть приклади евристичних функцій.
26. Яка функція називається функцією Беллмана?
27. Що таке А-алгоритм?
28. Що таке А*-алгоритм?

Завдання для самостійного розв'язування

1. Задача „8 ферзів": потрібно так розмістити вісім ферзів на шаховій дошці, щоб жоден з них не знаходився під ударом іншого. Для представлення ситуації на шаховій дошці можна використати масив бітів розміром 8*8, де кожний елемент відповідає одній клітинці дошки. Якщо поле знаходиться під ударом, відповідному елементу масиву присвоюємо значення 1, інакше – 0. Однак існує краще представлення, в якому використовується тільки один вектор довжиною 8 елементів. Використання такого представлення суттєво зменшує розмірність, а відповідно і складність задачі. Яке це представлення?
2. Як ви розумієте термін „простір пошуку"? Що представляє собою простір пошуку у грі в шахи?

3. Побудуйте простір пошуку розв'язку для такої задачі: з лівого берегу ріки на правий дідусю треба перевезти човном вовка, козу і капусту. У човен він може взяти лише щось одне. Необхідно знайти шлях з початкового стану в кінцевий, якщо відомо що вовк з'їсть козу, а коза капусту якщо їх залишити самих.
4. Напишіть програму на будь-якій мові програмування, яка реалізує гру у вісімки та попередню задачу.
5. Побудуйте модель предметної області діагностування захворювань.

РОЗДІЛ 3

ПОДАННЯ ЗНАНЬ ТА МОДЕЛІ МІРКУВАНЬ

- Моделі подання знань
- Семантичні мережі
- Моделі міркувань

Наведено моделі подання знань, детальніше розглянуто продукційну модель та семантичні мережі. Описано дедуктивну та індуктивну моделі міркувань.

3.1. Моделі подання знань

Модель подання знань – це множина синтаксичних та семантичних погодженостей, що робить можливим описання предмету. Під «предметом» тут розуміється стан ПО, тобто об'єктів певної області, їх властивостей, відношень, які існують між об'єктами в деякий фіксований момент часу.

Розрізняють два типи моделей подання знань: декларативні (знати, що подавати) і процедурні (знати, як подавати) [170]. Кожна з цих двох моделей має свої переваги та недоліки.

Процедурна модель подання базується на тому, що інтелектуальна діяльність розглядається як знання ПО, вкладене у програми. Тобто знання про те, як можна використати ті чи інші сутності ПО.

Декларативна модель базується на тому, що знання сутностей не має глибоких зв'язків із процедурами, що використовуються для опрацювання цих сутностей. Модифікація знань здійснюється простим додаванням або видаленням тверджень про сутності ПО.

Важливим для моделювання інтелектуальних систем є питання, чим відрізняються знання від даних. У роботі [147] Поспелов відзначає чотири ознаки що відрізняють знання від даних:

- внутрішня інтерпретованість знань;
- структурованість знань;
- зв'язність компонентів знань;
- активність знань.

Внутрішня інтерпретованість знань означає, що людина, яка вперше зіштовхується із новими знаннями, може їх зрозуміти і засвоїти.

Структурованість розглядають як властивість декомпозиції складних об'єктів на простіші та встановлення зв'язків між простими об'єктами.

Зв'язність практично неможливо знайти в базах даних. Знання зв'язані не тільки у сенсі структури, вони відображають закономірності щодо фактів, процесів, подій і причинно-наслідкові відношення між ними.

Активність – нові знання є програмами, а дані пасивно зберігаються у пам'яті машини. Наприклад, стимулом активності є неповнота знань, що змушує до необхідності їх поповнення.

Відмінність між даними та знаннями привели до появи спеціальних формалізмів у вигляді моделей подання знань, що відображають всі чотири ознаки. На сьогодні найпоширенішими є такі моделі подання знань [154]:

- семантичні мережі;
- системи фреймів;
- логічні моделі;
- продукційні системи;
- об'єктно-орієнтована модель.

Методи подання знань детально описані в роботах [18], [154]. У нашій роботі ми розглянемо лише продукційні правила та семантичні мережі як такі, що будуть нам потрібні для подальшого викладу матеріалу.

3.1.1. Продукційна система подання знань

Значна кількість інтелектуальних систем ґрунтується на понятті «формальна продукційна система» [18]. Продукційна система PS визначається таким чином:

$$PS = \langle E, B, I \rangle,$$

де E – база даних, яка містить біжучі дані; B – база знань, яка містить множину продукцій (правила вигляду "умова → дія"); іншими словами це інформація про можливі способи перетворення даних; I – інтерпретатор (машина логічного виведення), що реалізує процес виведення; в циклі виконуються такі дії: визначається множина пар (правило (b_i), набір поточних даних (E_i), на якій цей модуль (правило) задовольняється); цей модуль виконує відповідні дії над даними, тим самим здійснюючи зміни в базі даних.

Процес інтерпретації задається як пошук розв'язку в просторі станів ПО, що визначається вмістом бази даних. Оскільки не можна побудувати відразу всі шляхи розв'язування задачі, то вибір стану здійснюється відповідно до деяких евристичних функцій [5].

До факторів, які визначають переваги продукційних моделей, належать [65]:

- значна частина знань може бути записана у вигляді продукцій;
- системи продукцій є модульними;
- системи продукцій реалізують будь-які алгоритми, тобто відображають будь-яке процедурне знання;
- можливість паралельного виконання продукцій.

До факторів, які визначають недоліки продукційних моделей, належать:

- складність перевірки протиріччя системи продукцій, що змушує при додаванні нових продукцій витрачати багато часу на перевірку несуперечності нової системи;
- складність перевірки коректності роботи системи: якщо одночасно активізуються біля тисячі продукцій, то мало шансів, що система буде нормально функціонувати.

Продукційним моделям бракує формальної теорії. При заданні моделі ПО у вигляді набору продукцій не можна бути впевненим в її повноті і суперечності. Перехід до ал-

горитмічної схеми мало що дає, оскільки при цьому втрачається головна перевага продукційних систем – їх модульність, з якої випливає синхронізація і розпаралелювання виконання операцій у системі.

3.2. Семантичні мережі

Семантичну мережу можна визначити як множину об'єктів, на якій задано множину відношень. Об'єктами можуть бути як одиничні дані (Петро, дім № 5), так і класи даних (літальний апарат, меблі). Аналогічно відношеннями можуть бути як одиничні зв'язки між об'єктами (об'єкт х летить до об'єкта у), так і узагальнені (об'єкт х рухається до об'єкта у).

Переведення речень з природної мови у семантичні мережі є на сьогодні однією з актуальних проблем. Отримані результати в цьому напрямку належать до синтаксично і семантично обмежених речень. Замість поняття «одиниця природної мови» будемо надалі використовувати поняття «об'єкт» або «лінгвістична змінна».

Семантична мережа, об'єкти і відношення якої мають індивідуальний характер, називається мікромережею. Аналогічно макромережею називається мережа, об'єкти і відношення якої мають узагальнений характер.

Розглядаються також змішані мережі, які є композицією мікро- і макромереж. При визначенні операції вважається заданою загальна для всіх операцій база знань, яка подається у вигляді макромережі, що називається базовою макромережею. Усі семантичні мережі, над якими виконуються операції, вважаються такими, що належать до базової макромережі. Поняття належності тут використовується в теоритично-множинному і логічному поняттях. У першому функція належності визначається або через перечислення елементів класу, або через вказування обмеженої якості (предикату), а в другому – за допомогою правил виведення. Семантична мережа є істинною, якщо вона належить до базової макромережі.

Операції визначаються тільки над множиною істинних семантичних мереж.

Перемінний характер базової мікромережі великої системи обумовлює можливість переходу раніше істинної мережі у клас неправдивих, що тягне за собою зміну операцій, які здійснюються над мережею.

Операції над семантичними мережами зручно виконувати, оперуючи з їх аналітичними виглядами.

Розглянемо такі непусті множини:

$X = \{X_1, X_2, \ldots, X_n\}$; $R = \{R_1, R_2, \ldots, R_m\}$;

$x = \{x_1, x_2, \ldots, x_j\}$;

$r = \{r_1, r_2, \ldots r_q\}$; $Z = \{z_1, z_2, \ldots, z_a\}$;

$Q = \{q_1, q_2, \ldots, q_b\}$.

Домовимось позначати об'єкти і відношення: макромережі відповідно елементами множин X, R; мікромережі x, r і будь-якої мережі Z, Q. Тоді будь-яку семантичну мережу можна подати як

$$\{ z_1, z_2, \ldots, z_i, \ldots, z_n, Fz_1, Fz_2, \ldots, Fz_i, \ldots, Fz_n\},$$

де Fz_i – множина об'єктів, з якими об'єкт z_i пов'язаний відношеннями із Q.

Побудуємо аналітичний приклад розглянутої семантичної мережі: "Літак ІЛ-18 перебуває на смузі аеропорту "Львів" о 15 годині". Позначимо: z_1 – літак; z_2 – ІЛ-18; z_3 – смуга; z_4 – аеропорт; z_5 – "Львів"; z_6 – 15год.; q_1 – об'єкт х має ім'я у; q_2 – об'єкт х перебуває на об'єкті у; q_3 – об'єкт х є частиною об'єкта у; q_4 – об'єкт х має час у. Тоді мережу запишемо:

$$\{z_1, z_2, z_3, z_4, z_5, z_6, Fz_1, Fz_2, Fz_3, Fz_4, Fz_5, Fz_6\},$$

де $Fz_1 = \{q_1z_2, q_2z_3, q_4z_6\}$, $Fz_2 = \{\varnothing\}$, $\varnothing$ – символ пустої множини; $Fz_3 = \{q_3z_4\}$; $Fz_4 = \{q_1z_5\}$; $Fz_5 = \{\varnothing\}$; $Fz_6 = \{\varnothing\}$.

У загальному вигляді семантичну мережу можна розглядати як пару (Z, F), де F – відображення Z в Z, зважене відношеннями із Q. Цей скорочений запис семантичної мережі будемо використовувати при визначенні операції над мережами в загальному вигляді.

Перейдемо до визначення основних елементарних операцій семіотичної мови, використовуючи для позначення мікро- і макромереж символи s і S з відповідними індексами. Будь-яка мережа позначається символом L з індексами.

3.2.1. Об'єднання мереж

Нехай задані мережі $L_1 = (Z_1, F_1)$, $L_2 = (Z_2, F_2)$, …, $L_n = (Z_n, F_n)$. Мережа $L = (Z, F)$ називається об'єднанням мереж L_i, i =1, n і позначається $L = U_{1,n} L_i$, якщо $Z = U_{1,n} Z_i$ і для $\forall z \in Z$, $Fz = U_{1,n} F_i z$. Розглянемо дві мережі: L_1 – "Петро іде в школу № 5"; L_2 – "Школа №5 розташована поруч з басейном". Подамо мережі в аналітичній формі:

$$L_1 = \{z_1, z_2, z_3, F_1z_1, F_1z_2, F_1z_3\},$$
$$L_2 = \{z_2, z_3, z_4, F_2z_2, F_2z_3, F_2z_4\},$$

де – z_1 – Петро; z_2 – школа; z_3 – №5 z_4 – басейн; q_1 – об'єкт х іде в об'єкт у; q_2 – об'єкт х має ім'я у; q_3 – об'єкт х розташований поруч з об'єктом у;

$$F_1z_1 = \{q_1z_2\}; F_1z_2 = \{q_2z_3\};$$
$$F_1z_3 = \{\varnothing\}; F_2z_2 = \{q_2z_3, q_3z_4\};$$
$$F_2z_3 = \{\varnothing\}; F_2z_4 = \{\varnothing\}.$$

Знайдемо об'єднання мереж $L_3 = L_1 \cup L_2 = (Z_3, F_3)$.

Відповідно до формули об'єднання

$$Z_3 = Z_1 \cup Z_2 = \{z_1, z_2, z_3, z_4\}; F_3z_1 = F_1z_1 \cup F_2z_1 = \{q_1z_2\}$$
$$F_3z_2 = F_1z_2 \cup F_2z_2 = \{q_2z_3, q_3z_4\}; F_3z_3 = F_1z_3 \cup F_2z_3 = \{\varnothing\};$$
$$F_3z_4 = F_1z_4 \cup F_2z_4 = \{\varnothing\}.$$

У результаті об'єднання отримали мережу L_3 – "Петро іде в школу №5, яка розташована поруч з басейном".

3.2.2. Перетин мереж

Нехай задані мережі $L_1 = (Z_1, F_1)$, $L_2 = (Z_2, F_2)$, …, $L_n = (Z_n, F_n)$.

Мережа $L = (Z, F)$ називається перетином мереж L_i, i =1, n та позначається $L = \cap_{1,n} L_i$, якщо $Z = \cap_{1,n} Z_i$ і для $\forall z \in Z$, $Fz = \cap_{1,n} F_i z$. Якщо застосувати операцію перетину з мережами L_1 і L_2, які розглядались в попередньому прикладі, то отримаємо мережу $L_3 = \{z_2, z_3, F'_3z_2, F'_3z_3\}$, де $F'_3z_2 = F_1z_2 \cap F_2z_2 = \{q_2z_3\}$; $F'_3z_3 = F_1z_3 \cap F_2z_3 = \{\varnothing\}$.

Цій мережі відповідає вираз: "Школа №5".

3.2.3. Доповнення мережі

Нехай задані мережі $L_1 = (Z_1, F_1)$ і $L_2 = (Z_2, F_2)$, що L_2 є підмережею L_1, тобто $L_2 \subseteq L_1$. Мережа $L_3 = (Z_3, F_3)$ називається доповненням мережі L_2 до мережі L_1, якщо $L_3 = L'_1 \cup L'_2$, де $L'_1 = (Z'_1, F'_1)$, $L'_2 = (Z'_2, F'_2)$, і виконані наступні умови:

$$Z'_1 = Z_1/Z_2;\ \forall z \in Z'_1,\ F'_1 z \subseteq F_1 z;$$

$Z'_2 = Z''_1 \cup Z''_2$, де $Z''_1 \subseteq Z'_1$, $Z''_2 \subseteq Z_2$, причому кожний об'єкт $z \in Z''_1$ пов'язаний принаймні одним відношенням з об'єктами із Z_2; $\forall z \in Z'_2$, $F'_2 z \subseteq F_1 z$. як L_1 розглянемо вже наведену вище мережу: "Літак ІЛ-18 перебуває на смузі аеропорту "Львів" о 15 годині". Відповідно до введених позначень $L_1 = \{z_1, \ldots, z_6, F_1 z_1, \ldots, F_1 z_6\}$. Нехай L_2 буде: "смуга аеропорту "Львів", тобто $L_2 = \{z_3, z_4, z_5, F_2 z_3, F_2 z_4, F_2 z_5\}$.

Знайдемо L_3 – доповнення L_2 до L_1. Нехай $L_3 = L'_1 \cup L'_2$.

Спочатку визначимо $L'_1 = (Z'_1, F'_1)$. Відповідно до вказаного правила

$$Z'_1 = Z_1/Z_2 = \{z_1, \ldots, z_6\}/\{z_3, z_4, z_5\} = \{z_1, z_2, z_6\};$$
$$F'_1 z_1 = \{q_1 z_2, q_4 z_6\};\ F'_1 z_2 = \{\varnothing\};\ F'_1 z_6 = \{\varnothing\}.$$

Знайдемо

$$L'_2 = (Z'_2, F'_2);\ Z'_2 = Z''_1 \cup Z''_2,$$

де $Z''_1 = \{z_1\}$, $Z''_2 = \{z_3\}$, тобто $Z'_2 = \{z_1, z_3\}$; $F'_2 z_1 = \{q_2 z_3\}$, $F'_2 z_3 = \{\varnothing\}$. Таким чином,

$$L_3 = \{z_1, z_2, z_3, z_6, F'_1 z_1, F'_1 z_2, F'_1 z_6, F'_2 z_1, F'_2 z_3\},$$

тобто L_3: "Літак ІЛ – 18 перебуває на смузі о 15 годині".

3.2.4. Включення (кореляція) мереж

Спочатку введемо поняття структурно-еквівалентних мереж. Дві мережі (Z_j, F_j) називаються структурно-еквівалентними, якщо вони відрізняються одна від одної тільки перепозначенням об'єктів або відношень. Мережа "Літак летить у Київ" структурно-еквівалентна мережі: "Петро іде до школи", оскільки остання виходить з першої заміною "літак" на "Петро", "Київ" на "школу", відношення "летить в" на "іде до". При цьому зберігається як кількість об'єктів, так і орієнтація відношень між ними. Але мережа: "На місто насувається хмара" не є структурно-еквівалентною попереднім мережам, оскільки вона відрізняється від них орієнтацією відношень. Далі всюди, якщо не буде спеціально обумовлено, розглядатимемо структурно-еквівалентні мережі.

Перейдемо до визначення операції включення мікромережі в макромережу. Спочатку визначимо цю операцію для простих мереж, які складаються з двох об'єктів і одного відношення між ними. Нехай $S_0 = (X_1, X_2, F_0X_1, F_0X_2)$, де $F_0X_1 = \{R_1X_2\}$; $F_0X_2 = \{\varnothing\}$ є макромережею. Мікромережа $s_1 = (x_1, x_2, F_1x_1, F_1x_2)$, де $F_1x_1 = \{r_1x_2\}$, $F_1x_2 = \{\varnothing\}$, називається включеною в S_0, якщо $x_1 \in X_1$, $x_2 \in X_2$, $F_1x_1 \subseteq F_0X_1$, $F_1x_2 \subseteq F_0X_2$.

Наприклад, мережа: "Пасажирський літак ІЛ-18 летить у місто-курорт Сочі", де x_1 – пасажирський літак ІЛ-18, x_2 – місто-курорт Сочі, r_1 – об'єкт х летить в об'єкт у, включена в базову макромережу: "Літальний апарат рухається до мети", де X_1 – літальний апарат, X_2 – мета, R_1 – об'єкт X рухається до об'єкту Y, оскільки виконані умови включення.

Сформулюємо правило включення будь-якої мікромережі в макромережу. Мікромережа s називається включеною в базову макромережу S_0, якщо будь-яка підмережа мікромережі, яка складається з двох об'єктів і одного відношення між ними, включена в макромережу. В результаті виконання операції включення відношення, які існують між об'єктами макромережі, можуть бути перенесені з конкретизацією на об'єкти мікромережі.

Розглянемо макроситуацію: "Літальний апарат рухається до мети .Літальний апарат є далеко від мети".

Якщо в класі літальних апаратів є об'єкт "пасажирський літак ІЛ – 18", у класі мети – "місто Київ", у класі відношень "є далеко" – "є на відстані понад 500км", то мікроситуація: "Пасажирський літак ІЛ – 18 летить у місто Київ" – включається в макроситуацію. При цьому її можна розширити шляхом конкретизації другого відношення до мікроситуації: "Пасажирський літак ІЛ – 18 летить у місто Київ і є від нього на відстані понад 500км".

Правило включення мікромережі в макромережу дозволяє сформулювати більш загальне правило включення мікромереж.

Мікромережі s_i, s_j називаються включеними одна в одну, якщо кожна з них включена в базову макромережу і на дві мікромережі перенесені відношення макромережі. При цьому можливі три основні типи включення: безпосереднє, пряме і без встановлення відношень.

Безпосереднє включення відбувається тоді, коли відношення встановлюються безпосередньо між об'єктами мікромереж. Нехай є макромережа: "Літальний апарат рухається до мети" – і дві макромережі: "літак" і "місто Київ". Обидві мікроситуації включаються у макромережу, оскільки "літак" належить до класу літальних апаратів, а "місто Київ" – до класу мети. При цьому між ними встановлюється одне з відношень, яке входить у клас відношення руху. Якщо таким буде "об'єкт х що летить до об'єкту у", то будується наступна мережа: "Літак летить у Київ"

Пряме включення мікроситуацій відбувається тоді, коли вони включаються у фрагменти макроситуації, не пов'язані один з одним безпосередніми відношеннями. Розглянемо макромережу: "Літальний апарат рухається до мети, яка є в гірській місцевості", в якій вказуються наступні відношення: об'єкт (літальний апарат) рухається до об'єкту (мета); об'єкт (мета) знаходиться є об'єкті (гірська місцевість). Нехай мікроситуаціями будуть: "Літак ІЛ – 18" і "Карпати". Оскільки мікроситуації включаються в макроситуацію, то на їх об'єкти можуть переноситись відношення, які існують між включеними об'єктами і іншими об'єктами макромережі. У нашому випадку роль зв'язної ланки відіграє об'єкт "ціль". Якщо серед елементів цього класу є об'єкт "місто Сколе", то виникає наступна мікроситуація: "Літак ІЛ – 18 летить до міста Сколе, яке є в Карпатах". Вона є результатом включення заданих мікромереж.

Включення мікромереж без встановлення відношень є частковим випадком розглянутих двох видів включення. Цей випадок можливий тоді, коли мікроситуації включаються в непов'язані один з одним фрагменти макроситуації. Тоді говорять про існування пустих відношень між включеними мікроситуаціями.

3.2.5. Трансформація мереж

Операція трансформації семантичних мереж задається за допомогою макропідстановок, які не змінюють істинність причино-наслідкових і інших відношень між подіями, що подаються семантичними мережами. Макропідстановками називаються підстановки $P: S_1 \rightarrow S_2$, ліва і права частина, якої є макроситуаціями, які входять до складу базової макроситуації. Розглянемо правило застосування підстановки Р до мікроситуації s_k.

Сформулюємо умову застосування підстановки. Підстановка Р вважається застосовною до s_k, якщо s_k : 1) структурно-еквівалентна S_1; 2) s_k включена в S_1. Застосування підстановки є в ґенерації будь-якої мікроконструкції s_q, структурно-еквівалентної S_j і включеної в неї. При практичному застосуванні цієї операції в реальних задачах зазви-

чай накладаються обмеження, наприклад, дозволяється ґенерувати тільки такі мікроситуації, які складаються з наявних на цей момент об'єктів і відношень між ними.

Нехай S_1 – "Літальний апарат перебуває на смузі аеропорту", S_j – "Літальний апарат рухається до мети", s_k – "Пасажирський літак ІЛ-18 перебуває на смузі №1 аеропорту "Львів". Підстановка $S_i \to S_j$ застосовується до s_k, оскільки s_k структурно-еквівалентна S_i і включена в неї. Результатом застосування може бути будь-яка мікроситуація $s_l \in S_j$, до складу якої входить будь-який літак і будь-який об'єкт з класу мети. Якщо існує обмеження вказаного характеру, то найбільш вдалою буде s_l: "Пасажирський літак ІЛ-18 летить у Київ".

З поняттям макропідстановки пов'язано ще одне важливе поняття трасформаційної макромережі. Розглянемо множину макропідстановок $P = \{P_1, P_2, \ldots, P_n\}$.

Макромережа $S_T = U_{1,n}P_j$, отримана в результаті теоретико-множинного об'єднання макропідстановок, називається трасформаційною. Трансформаційну макромережу можна подати геометрично у вигляді графа, вершинами якого є макромережі, а дуги вказують напрямок переходу від однієї макромережі до іншої. Трансформаційна макромережа використовується зазвичай для подання макрорівня керування. Вершини мережі в цьому випадку відповідають макростану керівного об'єкту, а дуги – всеможливим шляхам переходу між макростанами. Трансформаційна макромережа є частковим випадком макромережі, а відповідно, по відношенню до неї залишаються справедливими всі твердження, наведені при розгляді операцій. Введемо поняття трансформаційної мікромережі – мікромережі, яка включається в трансформаційну макромережу. Трансформаційні мікромережі використовуються для подання мікрорівнів керування. У загальному випадку одному переходу в макромережі може відповідати ланцюжок переходів в мікромережі.

Прийняття рішень в багаторівневих ієрархічних системах викликає необхідність розглядати багаторівневі трансформаційні семантичні мережі. Вищі рівні таких мереж використовуються для подання вищих рівнів керування. У сучасних системах часто розглядаються трьохрівневі системи, які складаються з мета-, макро- і мікрорівнів. Метарівень відповідає, як правило, класу проблемних середовищ, макрорівень – макростана проблемного середовища. У таких системах необхідно розглядати три типи трансформаційних підстановок – мета-, макро- і мікропідстанови, які подають закони керування різного ступені складності. На метарівні формулюються, як правило, глобальні цілі поведінки проблемного середовища, загальні для класу керівних об'єктів. Цей рівень задається наперед. Мікро- і макрорівні формулюються кожний раз, коли задається нове середовище. Це відбувається в процесі навчання за допомогою операції узагальнення.

3.2.6. Узагальнення мереж

Ця операція визначається на багатьох структурно-еквівалентних мережах, які входять в базову макромережу S_0. Вважається, що в S_0 є інформація про :1) класифікацію об'єктів і відношень, які використовуються в узагальнених мережах; 2) оцінку потужності класів об'єктів, які формуються в результаті узагальнення. Оцінку можна виражати або кількісно, або якісно: багато, мало, дуже багато, декілька і ін.

Фрагмент базової макромережі, структурно-еквівалентний узагальненим мережам, відіграє роль мети узагальнення. Нехай таким буде

$$S = \{X_1, X_2, \ldots, X_i, \ldots, X_n, FX_1, FX_2, \ldots, FX_i, \ldots, FX_n\}.$$

Розглянемо m узагальнених мереж, структурно-еквівалентних S і таких, що входять в S:

$s_1 = \{x_{11}, x_{12}, \ldots, x_{1i}, \ldots, x_{1n},$
$F_1x_{11}, F_1x_{12}, \ldots, F_1x_{1i}, \ldots, F_1x_{1n}\};$
$s_2 = \{x_{21}, x_{22}, \ldots, x_{2i}, \ldots, x_{2n},$
$F_2x_{21}, F_2x_{22}, \ldots, F_2x_{2i}, \ldots, F_2x_{2n}\};$
. . . . . .
$s_m = \{x_{m1}, x_{m2}, \ldots, x_{mi}, \ldots, x_{mn},$
$F_mx_{m1}, F_mx_{m2}, \ldots, F_mx_{mi}, \ldots, F_mx_{mn}\}.$

Мережа $\underline{S} = \{\underline{X}_1, \underline{X}_2, \ldots, \underline{X}_i, \ldots, \underline{X}_n, \underline{FX}_1, \underline{FX}_2, \ldots, \underline{FX}_i, \ldots, \underline{FX}_n\}$ називається результатом узагальнення мереж s_j, j=1, m, якщо 1) $\underline{X}_i = (a_i, \{x_{1i}, x_{2i}, \ldots, x_{mi}\})$, де a_i – оцінка i-ї множини об'єктів; 2) $\underline{FX}_i = \{F_1x_{1i}, F_2x_{2i}, \ldots, F_mx_{mi}\}$, i=1, n.

Розглянемо змістовний приклад узагальнення мереж. Нехай роль цільового фрагменту S_0 відіграє наступна макромережа: "Кожний день робочого тижня учні встають зранку, роблять зарядку, снідають, йдуть до зупинки транспорту, їдуть в навчальний заклад". Як узагальнені вибираємо три мікромережі, структурно-еквівалентні цільовій і такі, що входять у неї.

s_1: "У понеділок учень 1-го класу Петро Іваненко встає о 7 годині ранку, робить одну вправу ранкової зарядки, випиває склянку чаю з канапкою, біжить до зупинки трамваю №6, їде до школи №4, розташованої неподалік від його дому".

s_2: "У вівторок учень 2-го класу Іван Петренко встає о 7 год. 10хв. ранку, робить три вправи ранкової зарядки, випиває філіжанку кави з тістечком, іде до зупинки тролейбуса №15, їде до школи №2, розташованої неподалік від його дому".

s_3: "У середу учень 3-го класу Василь Климчук встає о 7год. 30хв. ранку, робить 5 вправ ранкової зарядки, випиває склянку молока з булочкою, іде швидко до зупинки таксі, їде до школи №18, розташованої неподалік від його дому".

Нехай у базі знань є наступна класифікаційна інформація: всі учні з 1-го по 4-й класи є учнями початкової школи; час до 7год. 30хв. ранку належить до раннього ранку, будь-які гімнастичні вправи належать до ранкової зарядки; все, що людина їсть зранку, відноситься до сніданку; всі технічні засоби переміщення людей є транспортом; всі школи, розташовані неподалік від дому, є поблизу; йти швидко, бігти – означає поспішати. Нехай також описаний зміст оцінок: деякі, принаймні, в основному, майже.

Застосувавши до s_1, s_2, s_3 сформульоване вище правило узагальнення, отримаємо наступну узагальнену мережу (спеціально опускаємо громіздкі аналітичні викладки): "Принаймні в першій половині робочого тижня деякі учні початкової школи встають, в основному, рано-вранці, роблять декілька вправ ранкової зарядки, майже снідають. Поспішають до зупинки транспорту та їдуть до найближчої школи".

Як бачимо, змінюючи інформацію в базі знань, ми будемо отримувати інші узагальнені мережі. Якщо в базі знань вказано, що до ранку належить проміжок часу до 6год. 30хв, а проміжок від 6год. 30хв. до 7год. 30хв належить до пізнього ранку, то в узагальненій мережі з'явиться: "встають пізно вранці".

3.2.7. Конкретизація мереж

Ця операція за своїм характером близька до операції включення мереж у переносному значенні. В її основі лежить механізм ґенерації мікро мереж, структурно-еквівалентних

базовій макромережі (або її фрагменту) і таких, що включаються в неї. У простішому випадку генерація полягає у виділені елементів у багатьох об'єктах і відношень макромережі та побудові з них мікромережі, структурно-еквівалентної макромережі, що включається в неї. Аналітично ця операція визначається наступним чином. Нехай $S_0 = \{X_1, X_2, ..., X_i, ..., X_n, FX_1, FX_2, ..., FX_i, ..., FX_n\}$–базова макромережа. Мікромережа $s = \{x_1, x_2, ..., x_i, ..., x_n, Fx_1, Fx_2, ..., Fx_i, ..., Fx_n\}$ називається результатом конкретизації S_0, якщо $x_i \in X_i$, $Fx_i \subseteq FX_i$, $i = 1,n$.

Приклад найпростішої конкретизації ми навели при розгляді операції трансформації. У складнішому випадку конкретизація відбувається шляхом визначення одних значень через інші, при цьому використовується інформація із бази знань.

Нехай роль об'єкту X_1 в S_0 відіграє "літальний апарат". Всі елементи цього класу можна визначити через кінцеву множину базових елементів і відношень між ними. Наприклад, поняття "літак" можна визначити як: "Атмосферний літальний апарат, важчий за повітря, має в своєму складі крила і силову установку, яка створює тягу". Поняття вертоліт визначається як "Атмосферний літальний апарат важчий за повітря, який має в своєму складі повітряний гвинт, що створює підйомну силу і тягу". Аналогічне визначення можна дати і іншим елементам класу X_1.

Однак не важко побачити, що, якщо ми далі продовжимо процес визначення понять, які входять до складу вже визначених, то врешті-решт отримаємо поняття, визначення змісту яких не складає інтересу для вирішення поданих задач. Ці поняття називаються базовими. З їхньою допомогою і деякого кінцевого набору відношень можна визначити нескінчену множину понять, які входять в X_1. Операція конкретизації, заснована на механізмі визначення понять, широко застосовується в механізмах цілеутворення і побудови мови опису проблемного середовища.

Елементарні операції лежать в основі більш складних механізмів мови, за допомогою яких відбувається побудова середовища мети, формування мікро- і макромоделей проблемного середовища, побудова моделей цілеспрямованої поведінки в середовищі і ін.

3.3. Моделі міркувань

3.3.1. Дедуктивні моделі міркувань

Дедуктивні моделі міркувань націлені на отримання логічно правильних тверджень з множини аксіом $B = \{b_1, b_2, ..., b_n\}$, які є правильно побудованими формулами. Формула b' виводиться з формул $b_1, b_2, ..., b_n$, якщо b' є істинною тоді і лише тоді, коли істинною є формула $b_1 \wedge b_2 \wedge ... \wedge b_n$. Доведення формули b' з множини аксіом $B = \{b_1, b_2, ..., b_n\}$ зводиться до доведення хибності формули $b_1 \wedge b_2 \wedge ... \wedge b_n \wedge \overline{b'}$. Методи доведення, базовані на цьому, називаються методами заперечення. Найвідомішим серед них є метод резолюцій [127]. Дедуктивні моделі міркувань реалізовані у продукційних системах подання знань.

3.3.2. Індуктивні моделі міркувань

Індуктивне виведення дозволяє отримати загальні закономірності на основі експериментальних спостережень. Іншими словами, застосовуються міркування від часткового до загального. Програмам демонструються приклади. Ставиться задача аналізу набору

властивостей цих прикладів та ідентифікації відповідних концептів. Загальна форма задач, що розв'язуються в такій системі навчання, отримала назву індукції. Отже, індуктивна програма навчання – це програма, яка навчається на основі узагальнення властивостей прикладів, що їй надаються.

Оскільки індуктивне виведення здійснює виведення від часткового до загального, то формалізовані індуктивні методи відкривають можливості автоматизованого поповнення баз знань із експериментальних даних. Якщо в інтелектуальній системі реалізована підсистема автоматизованого поповнення знань, то говорять, що інтелектуальна система містить індуктивну компоненту (ІК).

Підсистеми поповнення знань дозволяють в автоматизованому або напівавтоматизованому режимі видобувати нові знання. Ідея такого підходу, ґрунтується на принципах індуктивного узагальнення у машинному навчанні.

Як відзначає Джексон [58]: «навчання – властивість адаптивної системи вдосконалювати свою поведінку, накопичуючи досвід, наприклад, досвід розв'язування аналогічних задач».

Велика кількість спеціалістів вважає задачу видобування знань однією з головних проблем технології інтелектуальних систем [188]. Тому виникли системи автоматизації процесу передавання знань від спеціаліста до машини. Виділяють два напрями:

1) автоматизоване видобування знань (виникло як розвиток систем людино-машинного діалогу);
2) машинне навчання (ідея полягає в тому, щоб машина вчилася вирішувати проблеми приблизно так, як вчиться людина).

Існують три варіанти видобування знань, що дозволяють обійтися без спільних зусиль людини-експерта та інженера зі знань:

- використовувати інтерактивні програми, які б добували знання від людини-експерта у процесі діалогу за терміналом;
- використовувати програми, що можуть навчатися, читаючи тексти, аналогічно до того, як навчається людина у процесі опрацювання технічної літератури;
- використовувати програми, які навчаються під керівництвом людини-вчителя; один з підходів полягає в тому, що вчитель надає програмі приклади реалізації деякого концепту, а задача програми полягає в тому, щоб отримати з наявних прикладів набір атрибутів і значень, що визначають цей концепт.

Запитання для повторення та контролю знань

1. Що таке модель подання знань?
2. Які існують типи моделей подання знань?
3. Яка відмінність між даними та знаннями?
4. Які моделі подання знань Ви знаєте?
5. Наведіть означення продукційної системи.
6. Які переваги продукційної моделі подання знань?
7. Які недоліки продукційної моделі подання знань?

8. Яка мережі називаються семантичними?
9. Які існують операції над семантичними мережами?
10. Як задається аналітична модель семантичних мереж?
11. Як задається операція об'єднання семантичних мереж?
12. Як задається операція перетину семантичних мереж?
13. Як задається операція доповнення семантичних мереж?
14. Як задається операція включення (кореляції) семантичних мереж?
15. Які мережі називаються структурно-еквівалентними?
16. Як задається операція трансформації семантичних мереж?
17. Яка семантична мережа називається трансформаційною?
18. Як задається операція узагальнення семантичних мереж?
19. Як задається операція конкретизації семантичних мереж?
20. Яка модель міркувань називається дедуктивною?
21. Яка модель міркувань називається індуктивною?
22. Що таке індуктивне узагальнення у машинному навчанні?
23. Які Ви знаєте напрямки передавання знань від спеціаліста до машини?
24. Які існують варіанти видобування знань, що дозволяють обійтися без спільних зусиль людини-експерта та інженера зі знань?

Завдання для самостійного розв'язування

1. Подайте розроблену модель предметної області діагностування захворювань (див. попередній розділ) за допомогою продукційних правил та семантичної мережі.
2. Побудувати семантичну мережу для такого речення: *Ми ходили на літературні зустрічі, в театри, і вештались у пошуках вина, і йшли на каву, і рушали в мандри, збирались в гості, цмулили коньяк, влаштовували сцени, ревнували самих себе, спостерігали, як летять у прірву наші ідеали.* Розписати множини Z, Q та Fz_i.
3. Який зв'язок існує між операторами та правилами у продукційній системі. Чи еквівалентні вони. Чи можна одні виразити в термінах інших?

РОЗДІЛ 4

УСКЛАДНЕНЕ ПОДАННЯ ЗНАНЬ ІЗ ВРАХУВАННЯМ ФАКТОРУ НЕВИЗНАЧЕНОСТІ

- Методи задання невизначеностей в інтелектуальних системах
- Нечіткі множини
- Використання коефіцієнтів впевненості та теорії ймовірності для моделювання нечіткостей в інтелектуальних системах

Розглянуто способи задання невизначеностей в інтелектуальних системах за допомогою нечітких множин, нечіткої логіки, нечітких відношень, коефіцієнтів впевненості та теорії ймовірності. Наведено відповідні приклади.

4.1. Методи задання невизначеностей в інтелектуальних системах

Одна із проблем, з якою стикаються експерти, полягає в тому, що у реальному житті ніщо не має абсолютно визначеного характеру, виключаючи смерть, хоча і вона може наступити несподівано. Мабуть, найбільш разючою властивістю людського інтелекту є здатність приймати правильні рішення в обстановці неповної й нечіткої інформації. Побудова моделей наближених міркувань людини і використання їх у комп'ютерних системах майбутніх поколінь є сьогодні однією з найважливіших проблем науки.

Сучасні бази даних та знань можуть містити значну кількість невизначеної інформації. Такий процес є закономірним, враховуючи динаміку світу та швидкість його розвитку. Тим не менше, нечітка чи неповна інформація теж повинні бути належним чином подані та опрацьовані, особливо це стосується предметних областей з явно вираженою динамікою розвитку (сфери, які обліковують фактор часу – музеї, картотеки; сфери, які працюють із неструктурованими даними – біржі праці, соціальні фонди тощо; сфери, діяльність яких пов'язана з ризиком у разі неточної інформації – планові відділи, ринки; виробництво та сфера послуг).

Тому рішення проблем у реальному житті потребує врахування фактора невизначеності. Для цього були запропоновані різні схеми, в яких фрагментарна і ненадійна інформація використовувалась для отримання оцінки істини [61]. Для опису даних та роботи з ними в умовах невизначеності та слабкої структурованості використовують:

- теорію ймовірностей;
- нечітку логіку (нечіткі множини, лінгвістичні змінні, інтервальні оцінки, нечіткі числа, емпіричні оцінки).

Апарат нечітких (розмитих) множин і нечіткої логіки понад 10 років з успіхом застосовується для рішення завдань, в яких вихідні дані є ненадійними й слабко формалізованими. Сильні сторони такого підходу:

- опис умов і методу рішення завдання мовою, близькою до природної;
- універсальність: відповідно до знаменитої теореми FAT (Fuzzy Approximation Theorem), доведеної Б. Коско (B. Kosko) в 1993 р., будь-яка математична система може бути апроксимованою системою, заснованою на нечіткій логіці;
- ефективність (пов'язана з універсальністю), що пояснюється множиною теорем, аналогічних теоремам про повноту для штучних нейронних мереж.

Одним із плюсів інтелектуальних систем є створення методів, які дозволяють бути точними щодо неточностей. Розглянемо декілька способів роздумів під час існування невизначеності, коли дані, які відносяться до задачі, або правила виведення (можливо одне і друге) не є на 100% надійними.

Для нечітких інтелектуальних (експертних і керівних) систем характерні й певні недоліки:

- вихідний набір нечітких правил формулюється експертом-людиною й може виявитися неповним або суперечливим;
- вид і параметри функцій залежності, що описують вхідні й вихідні змінні системи, вибираються суб'єктивно й можуть не повністю відображати реальну дійсність.

4.2. Нечіткі множини

Теорія нечітких множин – це етап на шляху до зближення точності класичної математики і всепроникної неточності навколишнього світу. Раніше поява класичної логіки була кроком вперед у боротьбі з нечіткістю, розпливчастістю подання людських знань. Тепер виникає необхідність створення теорії, яка б дозволяла формально описувати нестрогі, нечіткі поняття, і давала б можливість зрозуміти процес міркувань, що використовують такі поняття. Поштовхом до народження і бурхливого розвитку такої теорії стала стаття американського математика Л. А. Заде, яка називалася «Fuzzy Sets». Основна ідея Заде полягала в тому, що спосіб мислення, притаманний людині, не може бути описаний в рамках традиційних математичних формалізмів, яким притаманна однозначна інтерпретація тоді, як використання природних мов допускає багатозначну інтерпретацію. Мета, поставлена Заде, – створити нову математичну дисципліну, яка основана не на класичній теорії множин, а на теорії нечітких множин; на основі ідеї нечіткості побудувати нечіткі аналоги всіх основних математичних понять і створити формальний апарат для моделювання людських міркувань і людських прийомів розв'язування задач. Цей підхід дає строгий математичний опис в реальності розпливчастих тверджень, дозволяючи тим самим спробу подолати лінгвістичний бар'єр між людиною і машиною.

Теорія нечітких множин виникла в результаті узагальнення і переосмислення досягнень важливих напрямів класичної математики: багатозначної логіки, що дала можливість теорії ймовірності породити велику кількість способів статичного опрацювання експериментальних даних і відкрила шляхи визначення й інтерпретації функції належності; дискретної математики, яка запропонувала інструмент для побудови моделей багаторівневих систем.

На сьогодні відомі декілька способів формалізації нечітких понять.

Перший спосіб, оснований на працях Заде, знімає обмеження класичної теорії множин про те, що елемент може або належати, або не належати множині. Вводиться характерна функція належності, що приймає свої значення на інтервалі [0, 1]. Це так звані нечіткі множини Заде.

Другий спосіб формалізації нечіткості є більш загальним порівняно з першим і припускає, що характеристична функція належності приймає свої значення у скінченній чи нескінченній дистрибутивній решітці. Таке узагальнення називається нечіткою множиною в розумінні Голена. Більшість основних операцій нечіткої логіки зі значеннями істотності з інтервалу [0,1] можна розповсюдити на логіку зі значеннями істотності з дистрибутивної решітки.

Третій спосіб формалізації нечіткості – це Р-нечіткі множини. У цьому випадку кожне значення функції належності є не точкою в інтервалі [0, 1], а його підмножиною чи частиною. Алгебра Р-множин може бути зведена до алгебри класів.

Четвертий спосіб – гетерогенні нечіткі множини. Тут різним елементам множини відповідають значення в різних дистрибутивних решітках. Кожний елемент зв'язується з найбільш підхожою для нього оцінкою. Самі оцінки можуть бути нечіткими і задаватися у вигляді функції. Гетерогенні нечіткі множини і зв'язані з ними зіставлені лінгвістичні змінні високого порядку дозволяють моделювати ситуації багатокритеріального прийняття рішень, коли є ознаки як з кількісними, так і з порядковими шкалами.

Розглянуті способи формалізації нечітких понять дають можливість описувати системи настільки складні та погано визначені, що вони не піддаються точному математичному аналізу. І такий опис часто виявляється єдино можливим.

Область застосування нечітких множин дуже широка і різноманітна. Щодо філософії теорія нечітких множин відкриває новий підхід до вирішення проблеми абстракції та утворення понять, що мають велику кількість змістових відтінків. В області аналізу великих систем, таких як керування економікою країни, галузі, з'являється можливість моделювання невизначеності, вираженої, наприклад, у градаціях поінформованості центру про організовані нижче рівні. В області психології – це моделювання властивостей цілісності, дифузності психічних образів і уявлень, гнучності мислення, багатозначності елементів мови. В області лінгвістики – моделювання змісту речень і текстів за допомогою розподілення можливостей, які описуються функціями належності. В області техніки теорії нечітких алгоритмів стимулюють розвиток гнучких автоматизованих виробництв і робототехнічних комплексів, частково роботів, здатних виконувати окремі інтелектуальні дії людини. Теорія нечітких множин корисна при створенні діалогових систем з мовою спілкування, близькою до природної.

Між штучним інтелектом і теорією нечітких множин існує тісний взаємозв'язок, який стає очевидним, якщо прийняти тезу Л. Заде про те, що «людина мислить не числами, а нечіткими поняттями».

Множина – це одне з базових понять у математиці. Однак багато понять людських знань і зв'язків з навколишнім світом є структурами, які не можна назвати множинами в класичному розумінні. Це сукупності з нечіткими границями, для яких перехід від належності до неналежності не різкий, а поступовий. Викликає сумніви гіпотеза про те, що людські судження ґрунтуються на класичній двозначній логіці й навіть на багатозначній логіці. Швидше за все ці судження базуються на логіці з нечіткими значен-

нями істинності, нечіткими зв'язками і нечітким висновком. Існує два основних підходи до формалізації нечіткості. Перший ґрунтується на узагальненому понятті належності, другий – на поданні нечітких множин у вигляді послідовності чітких множин. Обидва підходи будуть розглянуті детальніше.

Л. Заде визначив також ряд операцій над нечіткими множинами й запропонував узагальнення відомих методів логічного виведення modus ponens і modus tollens. Увівши потім поняття лінгвістичної змінної й допустивши, що як значення виступають нечіткі множини, Л. Заде створив апарат для опису процесів інтелектуальної діяльності, включаючи нечіткість і невизначеність виразів.

Подальші роботи професора Л. Заде і його послідовників заклали міцний фундамент нової теорії та створили передумови для впровадження методів нечіткого керування в інженерну практику.

Вже до 1990 р. за цією проблематикою опубліковано понад 10 000 робіт, а число дослідників досягло 10 000, причому лише в США. В останні 5-7 років почалося використання нових методів і моделей у промисловості й у військовій справі. Спектр застосування їх широкий: від керування процесом відправлення й зупинки поїзда метрополітену, керування вантажними ліфтами й доменною піччю до пральних машин, порохотягів. При цьому нечіткі системи дозволяють підвищити якість продукції при зменшенні ресурсів і енерговитрат, і забезпечують вищу стійкість до впливу факторів, що заважають, у порівнянні із традиційними системами автоматичного керування.

Інакше кажучи, нові підходи дозволяють розширити сферу використання систем автоматизації за межі застосовності класичної теорії. У цьому плані цікава точка зору Л. Заде: «Я вважаю, що зайве прагнення до точності викликає дію, що зводить нанівець теорію керування й теорію систем, тому що вона приводить до того, що дослідження в цій області зосереджуються на тих і тільки тих проблем, які піддаються точному рішенню. У результаті багато класів важливих проблем, в яких дані, цілі й обмеження є занадто складними або непевними для того, щоб допустити точний математичний аналіз, залишалися й залишаються осторонь із тієї причини, що вони не піддаються математичному трактуванню. Для того, щоб сказати щось істотне для проблем подібного роду, ми повинні відмовитися від наших вимог точності й допустити результати, які є трохи розмитими або невизначеними».

Найбільшого апогею розвитку ця тема досягає з кінця 80-х років XX ст. і дотепер. Цей період характеризується бумом практичного застосування теорії нечіткої логіки в різних сферах науки і техніки. До 1990-го року з'явилося близько 40 патентів, що відносяться до нечіткої логіки (30 – японських). Сорок вісім японських компаній утворили спільну лабораторію LIFE (Laboratory for International Fuzzy Engineering), японський уряд фінансував 5-річну програму з нечіткої логіки, що включає 19 різних проектів – від систем оцінки глобального забруднення атмосфери і передбачення землетрусів до АСУ заводських цехів і складів. Результатом виконання цієї програми була поява цілого ряду нових масових мікрочіпів, заснованих на нечіткій логіці. Сьогодні їх можна знайти у пральних машинах і відеокамерах, цехах заводів і моторних відсіках автомобілів, у системах керування складськими роботами і бойовими вертольотами.

У США розвиток нечіткої логіки йде шляхом створення систем, що потрібні великому бізнесу і військовим. Нечітка логіка застосовується при аналізі нових ринків, біржовій грі, оцінці політичних рейтингів, виборі оптимальної цінової стратегії. Виникли і комерційні системи масового застосування.

У праці [82] розглянуті теоретичні аспекти складових нейронних мереж, апарат нечіткої логіки, основи теорії штучних нейронних мереж і гібридних мереж щодо завдань керування й прийняття рішень в умовах невизначеності.

Перш ніж дати визначення нечіткості множин, необхідно узагальнити поняття належності. У класичній теорії множин кожний елемент універсальної множини може або належати, або не належати конкретній підмножині заданої універсальної множини. Нехай U – універсальна множина, A – підмножина U, u – елемент множини U. Введемо характеристичну функцію належності $\mu_A(u)$ таку, що

$$\mu_A(u)=\begin{cases}0, u\notin A\\ 1, u\in A\end{cases}.$$

Вона вказує, чи є u елементом A. Областю значень такої функції буде множина [0,1].

Припустимо тепер, що функція належності може приймати свої значення з відрізку [0,1], або ще загальніший випадок – на цілком впорядкованій множині M.

Нехай U – зчисленна універсальна множина, u – елемент U. Тоді нечітка підмножина A множини U є множиною впорядкованих пар

$$\{(u,\mu_A(u))\}\forall u\in U,$$

де $\mu_A(u)$ – характеристична функція належності, яка приймає свої значення в цілком впорядкованій множині M і вказує на ступінь належності елемента u підмножині A. Множину M називають множиною належності. A означає нечітку підмножину.

Як повністю впорядковану множину належності M для простоти і наглядності використовують інтервал [0,1].

Носієм нечіткої підмножини A універсальної множини U називається множина всіх таких $u\in U$, для яких $\mu_A(u)>0$. Будемо описувати нечітку підмножину її функцією належності або множиною пар з елементів U з відповідними їй ступенями належності, відділеними від елементів рискою.

Нехай U={1,2,3,4,5,6,7,8,9,10}, "декілька"={(1/0), (2/0), (3/0,5), (4/0,8), (5/1), (6/1), (7/0,8), (8/0,5), (9/0), (10/0)}.

Звичайною підмножиною, найближчою до нечіткої підмножини A, називається така підмножина A, що

$$\mu_A(u)=\begin{cases}0, \mu_A(u)<0.5\\ 1, \mu_A(u)>0.5\\ 0\vee 1, \mu_A(u)=0.5\end{cases}.$$

Нехай $\alpha\in[0,1]$. Підмножиною α-рівня нечіткої підмножини A називається звичайна підмножина

$$A_\alpha=\{u\mid \mu_A(u)\geq\alpha\}.$$

Із визначення випливає очевидна властивість

$$\alpha_2\geq\alpha_1\Rightarrow A_{\alpha_2}\subset A_{\alpha_1}.$$

4.2.1. Основні характеристики нечітких множин

Нехай M=[0,1] і A – нечітка множина з елементами з універсальної множини U та множиною залежностей M.

- Величина $\sup \mu_A(u)$ називається *висотою* нечіткої множини A, $u \in U$. Нечітка множина A *є нормальною*, якщо її висота дорівнює 1, тобто верхня межа її функції залежності дорівнює 1 ($\sup \mu_A(u) = 1$), $u \in U$. При $\sup \mu_A(u) < 1$ нечітка множина називається *субнормальною*.
- Нечітка множина *порожня*, якщо $\forall u \in U \mid \mu_A(u) = 0$. Непорожню субнормальну множину можна нормалізувати за формулою:

$$\mu_A(u) = \frac{\mu_A(u)}{\sup \mu_A(u)}; u \in U.$$

- Нечітка множина унімодальна, якщо $\mu_A(u) = 1$ тільки на одному u з U.
- Носієм нечіткої множини A є звичайна підмножина з властивістю $\mu_A(u) > 0$, тобто носій $A = \{u / u \in U, \mu_A(u) > 0\}$.
- Елементи $u \in U$, для яких $\mu_A(u) = 0,5$, називаються точками переходу множини A.

Приклад.

Нехай U = {Таврія, Славута, Ланос,...} – множина марок автомобілів, а $U' = [0, \infty)$ – універсальна множина «Вартість», тоді на U' ми можемо визначити нечіткі множини типу: "для бідних", "для середнього класу", "престижні" з функціями належності, вигляд яких наведено на рис. 4.1.

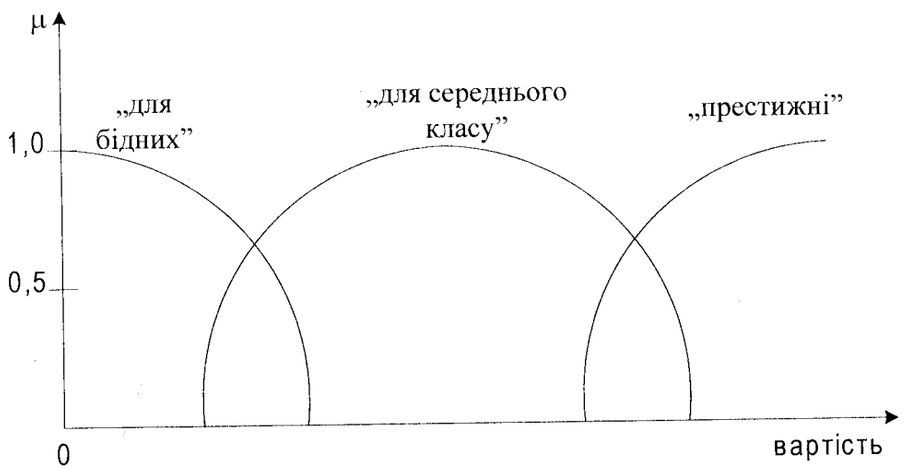


Рис. 4.1. Приклади функцій належності.

Маючи ці функції й знаючи вартості автомобілів з U в певний момент часу, ми тим самим визначимо на U' нечіткі множини із цими ж назвами.

Так, наприклад, розмита множина «Для бідних», задана на універсальній множині U = {Таврія, Славута, Сенс,...}, виглядає так, як показано на рис. 4.2.

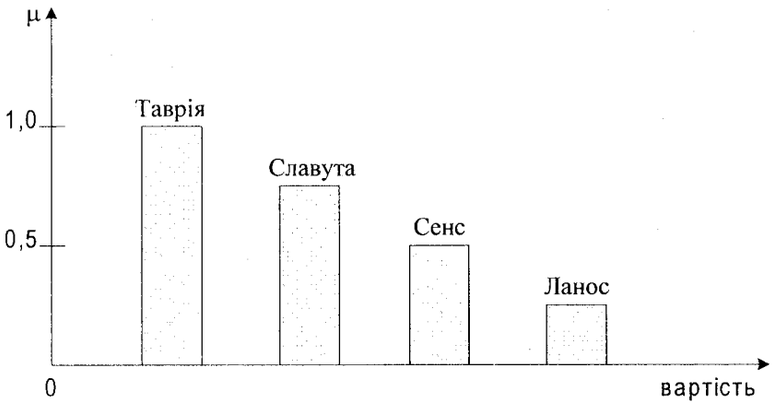


Рис. 4.2. Приклад задання нечіткої множини.

Властивості нечіткої множини:

- нормальність: $\exists$ u$\in$[0,1], $\mu_A(u) = 1$,
- опуклість: $\forall$ x,y,z$\in$[0,1]^3, $x \le y \le z \Rightarrow \mu_A(y) \ge \min(\mu_A(x), \mu_A(z))$.

4.2.2. Операції над нечіткими множинами

Логічні операції

Включення. Нехай A і B – нечіткі множини на універсальній множині U. Говорять, що A міститься у B, якщо $\forall u \in U \mid \mu_A(u) \le \mu_B(u)$.

Позначення: $A \subset B$.

Іноді використовують термін *домінування*, тобто у разі, коли $A \subset B$, говорять, що B домінує над A.

Рівність. A і B рівні, якщо $\forall u \in U \mid \mu_A(u) = \mu_B(u)$.

Позначення: $A = B$.

Доповнення. Нехай $M = [0, 1]$, A і B – нечіткі множини, задані на U. A і B доповнюють один одного, якщо $\forall u \in U \mid \mu_A(u) = 1 - \mu_B(u)$.

Позначення: $B = \overline{A}$ або $A = \overline{B}$.

Очевидно, що $\overline{\overline{A}} = A$ (доповнення визначене для M = [0, 1], але очевидно, що його можна визначити для будь-якого впорядкованого M).

Перетин. $A \cap B$ – найбільша нечітка підмножина, що міститься одночасно в A і B з функцією належності:

$$\mu_{A \cap B}(u) = \min(\mu_A(u), \mu_B(u)).$$

Об'єднання. $A \cup B$ – найменша нечітка підмножина, що включає як A, так і B, з функцією належності:

$$\mu_{A \cup B}(u) = \max(\mu_A(u), \mu_B(u)).$$

Різниця. $A - B = A \cap \overline{B}$ з функцією належності:

$$\mu_{A-B}(u) = \mu_{A \cap \overline{B}}(u) = \min(\mu_A(u), 1 - \mu_B(u)).$$

Диз'юнктивна сума

$$A \oplus B = (A - B) \cup (B - A) = (A \cap \overline{B}) \cup (\overline{A} \cap B)$$

з функцією належності:

$$\mu_{A \oplus B}(u) = \max\left(\min(\mu_A(u), 1 - \mu_B(u)); \min(1 - \mu_A(u), \mu_B(u))\right).$$

Приклади. Нехай

A={(x1/0,4),(x2/0,2),(x3/0),(x4/1)};
B={(x1/0,7),(x2/0,9),(x3/0,1),(x4/1)};
C={(x1/0,1),(x2/1),(x3/0,2),(x4/0,9)}.

Тут:

1. $A \subset B$, тобто A міститься у B або B домінує над A; C непорівняне ні з A, ні з B, тобто пари {A,C} і {B,C} – пари нечітких множин, що не домінують.
2. $A \neq B \neq C$.
3. $\overline{A}$={(x1/0,6),(x2/0,8),(x3/1),(x4/0)}; $\overline{B}$={(x1/0,3),(x2/0,1),(x3/0,9),(x4/0)}.
4. $A \cap B$={(x1/0,4),(x2/0,2),(x3/0),(x4/1)}.
5. $A \cup B$={(x1/0,7),(x2/0,9),(x3/0,1),(x4/1)}.
6. $A - B = A \cap \overline{B}$={(x1/0,3),(x2/0,1),(x3/0),(x4/0)};
   $B - A = B \cap \overline{A}$={(x1/0,6),(x2/0,8),(x3/0,1),(x4/0)}.
7. $A \oplus B$={(x1/0,6),(x2/0,8),(x3/0,1),(x4/0)}.

Наочне зображення логічних операцій над нечіткими множинами

Для нечітких множин можна будувати візуальне подання. Розглянемо прямокутну систему координат, на осі ординат якої відкладаються значення $\mu_A(u)$, на осі абсцис в довільному порядку розташовані елементи U. Якщо U за своєю природою впорядкована, то цей порядок бажано зберегти в розташуванні елементів на осі абсцис. Таке подання робить наочними прості логічні операції над нечіткими множинами (див. рис. 4.3).

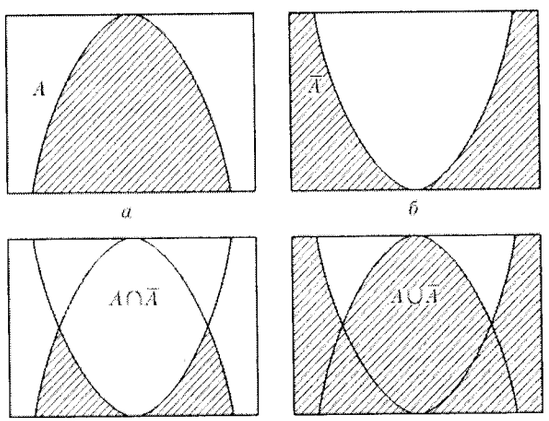


Рис. 4.3. Графічна інтерпретація логічних операцій: а – нечітка множина A, б – нечітка множина $\overline{A}$, в – $A \cap \overline{A}$, г – $A \cup \overline{A}$.

На рис. 4.3а заштрихована частина відповідає нечіткій множині A і, якщо говорити точно, зображає область значень A і всіх нечітких множин, що містяться в A. На рис. 4.3 б, в, г дані $\overline{A}$, $A \cap \overline{A}$, $A \cup \overline{A}$.

Властивості операцій $\cup$ і $\cap$.

Нехай A, B, C – нечіткі множини, тоді справедливі наступні властивості:

$$\left.\begin{aligned} A \cap B &= B \cap A \\ A \cup B &= B \cup A \end{aligned}\right\} \text{ – } комутативність;$$

$$\left.\begin{aligned} (A \cap B) \cap C &= A \cap (B \cap C) \\ (A \cup B) \cup C &= A \cup (B \cup C) \end{aligned}\right\} \text{ – } асоціативність;$$

$$\left.\begin{aligned} A \cap A &= A \\ A \cup A &= A \end{aligned}\right\} \text{ – } ідемпотентність;$$

$$\left.\begin{aligned} A \cap (B \cup C) &= (A \cap B) \cup (A \cap C) \\ A \cup (B \cap C) &= (A \cup B) \cap (A \cup C) \end{aligned}\right\} \text{ – } дистрибутивність:$$

$A \cup \varnothing = A$, де $\varnothing$ – *порожня множина*, тобто $\mu_\varnothing(u) = 0 \quad \forall u \in U$;

$A \cap \varnothing = \varnothing$;

$A \cap U = A$, де U – *універсальна множина;*

$A \cup U = U$

$$\left.\begin{aligned} \overline{A \cap B} &= \overline{A} \cup \overline{B} \\ \overline{A \cup B} &= \overline{A} \cap \overline{B} \end{aligned}\right\} \text{ – } теореми\ де\ Моргана.$$

На відміну від чітких множин, для нечітких множин в загальному випадку:

$A \cap \overline{A} \neq \varnothing$, $A \cup \overline{A} \neq U$

(що, зокрема, проілюстровано вище в прикладі наочного подання нечітких множин).

Зауваження. Введені вище операції над нечіткими множинами засновані на використанні операцій max і min. У теорії нечітких множин розробляються питання побудови узагальнених операторів перетину, об'єднання і доповнення, що дозволяють врахувати різноманітні сенсові відмінки відповідних їм зв'язкам «і», що параметризуються, «або», «ні».

Один з підходів до операторів перетину і об'єднання полягає в їх визначенні у *класі трикутних норм і конорм.*

Трикутною нормою (t-нормою) називається двозначна дійсна функція, що задовольняє наступні умови:

$T(0,0) = 0$; $T(\mu_A, 1) = \mu_A$; $T(1, \mu_A) = \mu_A$ – *обмеженість;*

$T(\mu_A, \mu_B) \leq T(\mu_C, \mu_D)$, якщо $\mu_B \leq \mu_D$ – *монотонність;*

$T(\mu_A, \mu_B) = T(\mu_B, \mu_A)$ – *комутативність;*

$T(\mu_A, T(\mu_B, \mu_C)) = T(T(\mu_A, \mu_B), \mu_C)$ – *асоціативність.*

Приклади трикутних норм

$\min(\mu_A, \mu_B)$,

добуток $\mu_A \cdot \mu_B$,

$\max(0, \mu_A + \mu_B - 1)$.

Трикутною конормою (t-конормою) називається двозначна дійсна функція $S : [0,1] \times [0,1] \to [0,1]$ з властивостями:

$S(1,1) = 1$; $S(\mu_A, 0) = \mu_A$; $S(0, \mu_A) = \mu_A$ – *обмеженість;*

$S(\mu_A, \mu_B) \geq S(\mu_C, \mu_D)$, якщо $\mu_B \geq \mu_D$ – *монотонність;*

$S(\mu_A, \mu_B) = S(\mu_B, \mu_A)$ – *комутативність;*

4) $S(\mu_A, S(\mu_B, \mu_C)) = S(S(\mu_A, \mu_B), \mu_C)$ – *асоціативність.*

Приклади t-конорм

$\max(\mu_A, \mu_B)$,

$\mu_A + \mu_B - \mu_A \cdot \mu_B$,

$\min(1, \mu_A + \mu_B)$.

Операції алґебри над нечіткими множинами

Алгебраїчний добуток множин A і B позначається $A \cdot B$ і визначається так: $\forall u \in U$ $\mu_{A \cdot B}(u) = \mu_A(u) \cdot \mu_B(u)$.

Алгебраїчна сума цих множин позначається $A \oplus B$ і визначається так: $\forall u \in U$ $\mu_{A \oplus B}(u) = \mu_A(u) + \mu_B(u) - \mu_A(u) \cdot \mu_B(u)$.

Для операцій $\{\cdot, \oplus\}$ справедливі властивості:

1) $\left.\begin{array}{l} A \cdot B = B \cdot A \\ A \oplus B = B \oplus A \end{array}\right\}$ – *комутативність;*

2) $\left.\begin{array}{l} (A \cdot B) \cdot C = A \cdot (B \cdot C) \\ (A \oplus B) \oplus C = A \oplus (B \oplus C) \end{array}\right\}$ – *асоціативність;*

3) $A \cdot \varnothing = \varnothing$, $A \oplus \varnothing = A$, $A \cdot U = A$, $A \oplus U = U$;

4) $\left.\begin{array}{l} \overline{A \cdot B} = \overline{A} \oplus \overline{B} \\ \overline{A \oplus B} = \overline{A} \cdot \overline{B} \end{array}\right\}$ – *теореми де Моргана.*

Не виконуються співвідношення:

1) $\left.\begin{array}{l} A \cdot A = A \\ A \oplus A = A \end{array}\right\}$ – *ідемпотентність;*

2) $\left.\begin{array}{l} A \cdot (B \oplus C) = (A \cdot B) \oplus (A \cdot C) \\ A \oplus (B \cdot C) = (A \oplus B) \cdot (A \oplus C) \end{array}\right\}$ – *дистрибутивність;*

3) а також $A \oplus \overline{A} = U$.

Зауваження. При одночасному використанні операцій $\{\cup, \cap, \oplus, \cdot\}$ справедливі наступні властивості:

$A \cdot (B \cup C) = (A \cdot B) \cup (A \cdot C)$;

$A \cdot (B \cap C) = (A \cdot B) \cap (A \cdot C)$;

$A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$;

$A \oplus (B \cap C) = (A \oplus B) \cap (A \oplus C)$.

На основі операції алгебраїчного добутку визначається операція *піднесення до степеня* α нечіткої множини A, де α – *додатнє число. Нечітка множина* A^α визначається функцією належності $\mu_A^\alpha = \mu_A^\alpha(u)$. Окремим випадком піднесення до степеня є:

$\mathrm{CON}(A) = A^2$ – операція *концентрації (ущільнення);*

$\mathrm{DIL}(A) = A^{0,5}$ – операція *розтягування,*

які використовуються під час роботи з лінгвістичними невизначеностями (рис. 4.4).

Множення на число. Якщо α – *додатнє число,* то нечітка множина $\alpha \cdot A$ має функцію належності:

$$\mu_{\alpha A}(u) = \alpha \cdot \mu_A(u).$$

Опукла комбінація нечітких множин. Нехай $A_1, A_2, \ldots, A_n$ – нечіткі множини універсальної множини U, а $\omega_1, \omega_2, \ldots, \omega_n$ – невід'ємні числа, сума яких рівна 1.

Опуклою комбінацією $A_1, A_2, \ldots, A_n$ називається нечітка множина A з функцією належності:

$$\forall u \in U \mid \mu_A(u_1, u_2, \ldots, u_n) = \omega_1 \mu_{A_1}(u) + \omega_2 \mu_{A_2}(u) + \ldots + \omega_n \mu_{A_n}(u).$$

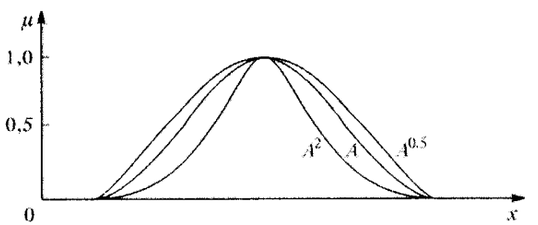


Рис. 4.4. Ілюстрація понять операцій концентрації (ущільнення) і розтягування.

Декартовий (прямий) добуток нечітких множин. Нехай $A_1, A_2, \ldots, A_n$ – нечіткі підмножини універсальних множин $U_1, U_2, \ldots, U_n$ відповідно. Декартовий, або прямий добуток $A = A_1 \times A_2 \times \ldots \times A_n$ є нечіткою підмножиною множини $U = U_1 \times U_2 \times \ldots \times U_n$ з функцією наналежності:

$$\mu_A(u_1,u_2,...,u_n)=\min\left(\mu_{A_1}(u_1),\mu_{A_2}(u_2),...,\mu_{A_n}(u_n)\right).$$

Оператор збільшення нечіткості використовується для перетворення чітких множин в нечіткі і для збільшення нечіткості нечіткої множини.

Нехай A – нечітка множина, U – універсальна множина і для всіх $u \in U$ визначені нечіткі множини $K(u)$. Сукупність всіх $K(u)$ називається ядром оператора збільшення нечіткості Φ. Результатом дії оператора Φ на нечітку множину A є нечітка множина вигляду

$$\Phi(A,K)=\bigcup_{u\in U}\mu_A(u)K(u),$$

де $\mu_A(u)K(u)$ – добуток числа на нечітку множину.

Приклад. Нехай

$U=\{1,2,3,4\}$; $A=\{(1/0{,}8),(2/0{,}6),(3/0),(4/0)\}$; $K(1)=\{(1/1),(2/0{,}4)\}$; $K(2)=\{(1/0{,}4),(2/1),(3/0{,}4)\}$; $K(3)=\{(3/1),(4/0{,}5)\}$; $K(4)=\{(4/1)\}$.

Тоді

$$\Phi(A,K)=\mu_A(1)K(1)\cup\mu_A(2)K(2)\cup\mu_A(3)K(3)\cup\mu_A(4)K(4)=$$

$$=0{,}8\{(1/1),(2/0{,}4)\}\cup\cup\,0{,}6\{(1/0{,}4),(2/1),(3/0{,}4)\}=\{(1/0{,}8),(2/0{,}6),(3/0{,}24)\}.$$

4.2.3. Нечітка і лінґвістична змінні

Поняття нечіткої і лінґвістичної змінних використовується під час опису об'єктів і явищ за допомогою нечітких множин.

Нечітка змінна характеризується трійкою $\langle\alpha,X,A\rangle$, де α – найменування змінної; X – універсальна множина (область визначення α); A – нечітка множина на X, що описує обмеження (тобто $\mu_A(x)$) на значення нечіткої змінної α.

Лінґвістичною змінною (ЛЗ) називається набір $\langle\beta,T,X,G,M\rangle$, де β – найменування лінгвістичної змінної; T – множина її значень (терм-множина), що є найменуваннями нечітких змінних, областю визначення кожної з яких є множина X. Множина T називається базовою *терм-множиною* лінґвістичної змінної; G – синтаксична процедура, що дозволяє оперувати елементами терм-множини T, зокрема, ґенерувати нові терми (значення). Множина $T\cup G(T)$, де $G(T)$ – множина термів, що зґенерувалися, називається розширеною терм-множиною лінґвістичної змінної; M – семантична процедура, що дозволяє перетворити кожне нове значення лінґвістичної змінної, що утворюється процедурою G, в нечітку змінну, тобто сформувати відповідну нечітку множину.

Зауваження. Щоб уникнути великої кількості символів:

1) символ β використовують як для назви самої змінної, так і для всіх її значень;
2) користуються одним і тим самим символом для позначення нечіткої множини і його назви, наприклад терм «Молодий», такий, що є значенням лінґвістичної змінної β = «вік», одночасно є і нечіткою множиною M («Молодий»).

Присвоєння декількох значень символам допускає, що контекст дозволяє вирішити можливі невизначеності.

Приклад. Хай експерт визначає товщину виробу, що випускається, за допомогою понять «Мала товщина», «Середня товщина» і «Велика товщина», при цьому мінімальна товщина рівна 10 мм, а максимальна – 80 мм.

Формалізація такого опису може бути здійснена за допомогою наступної лінґвістичної змінної $\langle\beta,T,X,G,M\rangle$, де β – товщина виробу; T – { «Мала товщина», «Середня товщина», «Велика товщина»}; X – [10, 80]; G – процедура утворення нових термів за допомогою зв'язок «і», «або» і модифікаторів типу «дуже», «не», «злегка» і т.ін. Наприклад: «Мала або середня товщина», «Дуже мала товщина» і т.ін.; M – процедура задання на X = [10, 80] нечітких підмножин A_1=«Мала товщина», A_2=«Середня товщина», A_3=«Велика товщина», а також нечітких множин для термів з $G(T)$ відповідно до правил трансляції нечітких зв'язок і модифікаторів «і», «або», «не», «дуже», «злегка» та інших операцій над нечіткою множиною вигляду: $A\cap B$, $A\cup B$, $\bar{A}$, $\mathrm{CON}A=A^2$, $\mathrm{DIL}A=A^{0,5}$ і т.ін.

Зауваження. Разом з розглянутими вище базовими значеннями лінґвістичної змінної «Товщина» (T = { «Мала товщина», «Середня товщина», «Велика товщина»}) можливі значення, що залежать від області визначення X. У такому випадку значення лінґвістичної змінної «Товщина виробу» можуть бути визначені як «біля 20 мм», «біля 50 мм», «біля 70 мм», тобто у вигляді нечітких чисел.

Терм-множина і розширений терм-множина в умовах прикладу може характеризуватися функціями належності, наведеними на рис. 4.5 і рис. 4.6.

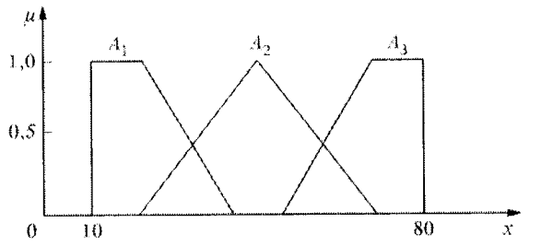


Рис. 4.5. Функції належності нечітких множин: «Мала товщина» = A_1 «Середня товщина» = A_2, «Велика товщина» = A_3.

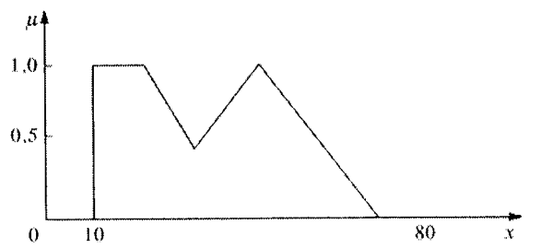


Рис. 4.6. Функція належності нечіткої множини «Мала або середня товщина» = A_1UA_2.

4.2.4. Нечіткі відношення

Поняття відношення відіграє фундаментальну роль в математиці. Його також можна узагальнити на випадок нечітких підмножин. При цьому виникають деякі цікаві властивості. Наприклад, поняття класу еквівалентності замінюється поняттям подібності. Передпорядок і порядок узагальнюється аналогічним до класичного чином, визначаються деякі інші відношення, наприклад, схожості і несхожості. Створення нових теорій, ґрунтованих на нечітких відношеннях, дасть змогу знайти компроміс вирішення несумісності високої точності опису систем з великою їх складністю.

На принципі роботи нечітких відношень будується нечітка логіка, а звідси і нечітке логічне виведення.

Нехай P – прямий добуток n множин і M – його множина належності. Нечітка підмножина P, яка приймає свої значення в M, називається n-арним відношенням.

Нечіткі відношення подаються у вигляді множини пар – впорядкованої n-ки і ступеня належності цієї n-ки певному нечіткому відношенню. Нечіткі відношення позначаються також як звичайні відношення.

Нехай $U_1=U_2=\{1,2,3,4\}$.

R ="приблизно рівні" = {(1,1)/1; (1,2)/0,5; (1,3)/0; (1,4)/0; (2,1)/0,5; (2,2)/1; (2,3)/0,5; (2,4)/0; (3,1)/0; (3,2)/0,5; (3,3)/1; (3,4)/0,5; (4,1)/0; (4,2)/0; (4,3)/0,5; (4,4)/1}.

Нечітке бінарне відношення R – називається симетричним, якщо

$$\forall(x,y)\in U\otimes U\,|\,(\mu_R(x,y)=\mu)\Rightarrow(\mu_R(y,x)=\mu).$$

Нечітке бінарне відношення R називається рефлексивним, якщо

$$\forall(x,x)\in U\otimes U\,|\,\mu_R(x,x)=1.$$

Нехай $x, y, z \in U$, тоді нечітке бінарне відношення R називається транзитивним, якщо

$$\forall(x,y),(y,z),(x,z)\in U\otimes U\,|\,\mu_R(x,z)=\mu\geq\max_y\left[\min(\mu_R(x,y),\mu_R(y,z))\right].$$

Прикладом симетричного і рефлексивного нечіткого відношення є відношення "y близьке до x". Нечіткі відношення "y набагато більше від x", "x – далекий родич y" є транзитивними, а відношення "x схоже на y" не є таким, оскільки з того, що x схоже на y і y схоже на z може не випливати, що x схоже на z.

Нехай $U=U_1\times U_2\times\ldots\times U_n$ – прямий добуток універсальних множин і M – деяка множина залежностей (наприклад, $M=[0,1]$). Нечітке n-арне відношення визначається як нечітка підмножина R на U, що приймає свої значення в M. У випадку $n=2$ і $M=[0,1]$ нечітке відношення R між множинами $X=U_1$ і $Y=U_2$ буде називатися функція $R:(X,Y)\to[0,1]$, що ставить у відповідність кожній парі елементів $(x,y)\in X\times Y$ величину $\mu_R(x,y)\in[0,1]$.

Приклади.

1. Нехай $X=\{x_1,x_2,x_3\}$ $Y=\{y_1,y_2,y_3,y_4\}$ $M=[0,1]$. Нечітке відношення R може бути задано у вигляді таблиці:

	y_1	y_2	y_3	y_4
x_1	0	0	0,1	0,3
x_2	0	0,8	1	0,7
x_3	1	0,5	0,6	1

2. Нехай $X=Y=(-\infty,\infty)$, тобто множина всіх дійсних чисел. Відношення $x\succ y$ (x набагато більше за y) можна задати функцією належності:

$$\mu_R=\begin{cases}0, & якщо\ x\leq y\\ \dfrac{1}{1+(1/(x-y)^2)}, & якщо\ x>y\end{cases}.$$

3. Відношення R, для якого $\mu_R(x,y)=e^{-k(x-y)^2}$, при досить великих k можна інтерпретувати так: «x і y близькі за значенням числа».

4.2.5. Нечітка логіка

У булевій алгебрі 1 відображає істину, а 0 – хибність. Те ж саме має місце в нечіткій логіці, але, крім того, використовуються також всі дроби між 0 і 1, щоб вказати на часткову істину. Так, запис

$$p(\text{високий }(X))=0{,}75$$

говорить про те, що твердження "X – високий", в деякому сенсі на три четвертини істинне, так само як і на одну четверту хибне. Для комбінування нецілочисельних значень істинності в нечіткій логіці визначаються еквіваленти операцій "і", "або", "не".

p_1 і p_2 = min (p_1,p_2) (тобто менше)

p_1 або p_2 = max (p_1,p_2) (тобто більше)

не p_1 = 1–p_1 (тобто "зворотне значення").

Таким чином інформацію можна комбінувати на основі строгих і погоджених методів, тому нечітка логіка з успіхом використовується, наприклад, в системі забезпечення прийняття рішень REVEAL.

4.2.6. Нечітке логічне виведення

Для нечітких множин така процедура відповідає композиції нечіткої множини A з нечітким відношенням F, тобто

$$B=A\circ F\ \text{ або }\ \mu_B(y)=\max_x\min\left[\mu_A(x),\mu_F(x,y)\right].$$

У цьому полягає композиційне правило виведення. З вихідних нечіткої підмножини X і нечіткого відношення $X\otimes Y$ ми робимо висновок про відповідні їм обмеження множини Y.

Нехай $X=Y=\{1,2,3,4\}$ і:

$A(X)$ = "малий" = {(1/1), (2/0,6), (3/0,2), (4/0)};

$F(X,Y)$ = "приблизно рівні" = {((1,1)/1) , ((1,2)/0,5), ((1,3)/0), ((1,4)/0), ((2,1)/0,5), ((2,2)/1), ((2,3)/0,5), ((2,4)/0), ((3,1)/0), ((3,2)/0,5), ((3,3)/1), ((3,4)/0,5), ((4,1)/0), ((4,2)/0), ((4,3)/0,5), ((4,4)/1)}.

Це відношення можна подати у вигляді таблиці:

	(Y_1)	(Y_2)	(Y_3)	(Y_4)
F	1	2	3	4
(X_1)1	1	0,5	0	0
(X_2)2	0,5	1	0,5	0
(X_3)3	0	0,5	1	0,5
(X_4)4	0	0	0,5	1

Тоді:

$$B(Y)=A(X)\circ F(X,Y)=[1;0{,}6;0{,}2;0]^\circ\begin{bmatrix}1 & 0.5 & 0 & 0\\ 0.5 & 1 & 0.5 & 0\\ 0 & 0.5 & 1 & 0.5\\ 0 & 0 & 0.5 & 1\end{bmatrix}=[1;0{,}6;0{,}5;0{,}2]=$$

$=$ {(1/1), (2/0,6), (3/0,5), (4/0,2)} = "більш менш малий".

Використовуваний у різних експертних і керівних системах механізм нечітких виведень у своїй основі має базу знань, сформовану фахівцями предметної області у вигляді сукупності нечітких предикатних правил вигляду:

$П_1$: якщо $x \in A_1$, то $y \in B_1$,

$П_2$: якщо $x \in A_2$, то $y \in B_2$,

...

$П_n$: якщо $x \in A_n$, тоді $y \in B_n$,

де x – вхідна змінна, y – змінна виведення; A і B – функції належності, відповідно на x і y.

Приклад подібного правила: якщо x – низько, то у – високо.

Знання експерта $A \to B$ відбиває нечітке причинне відношення передумови й висновку, тому його можна назвати нечітким відношенням і позначити через R:

$$R = A \to B,$$

де "$\to$" називають нечіткою імплікацією.

Відношення R можна розглядати як нечітку підмножину прямого добутку $X \times Y$ повної множини передумов X і висновків Y. Отже, процес одержання (розмитого) результату виведення B' з використанням певного спостереження A' і знання $A \to B$ можна зобразити у вигляді формули

$$B' = A' \circ R = A' \circ (A \to B).$$

Як операцію композиції, так і операцію імплікації в алгебрі нечітких множин можна реалізовувати по-різному, але в кожному разі загальне логічне виведення здійснюється в наступні чотири етапи.

1. **Нечіткість** (введення нечіткості, фазифікація, fuzzification). Функції залежності, певні на вхідних змінних, застосовуються до їхніх фактичних значень для визначення ступеня істинності кожної передумови кожного правила.
2. **Логічне виведення.** Обчислене значення істинності для передумов кожного правила застосовується до висновків кожного правила. Це призводить до однієї нечіткої підмножини, що буде призначена кожній змінній виведення для кожного правила. Як правила логічного виведення зазвичай використовуються тільки операції min (МІНІМУМ) або prod (МНОЖЕННЯ).
3. **Композиція.** Всі нечіткі підмножини, призначені для кожної змінної виведення, поєднуються разом, щоб формувати одну нечітку підмножину. При подібному об'єднанні зазвичай використовуються операції max (МАКСИМУМ) або sum (СУМА). При композиції МАКСИМУМУ комбіноване виведення нечіткої підмножини конструюється як поточний максимум всіх нечітких підмножин, визначений зміною виведення правилами логічного виведення.
4. **Приведення до чіткості** (дефазифікація, defuzzification), що використовується, коли корисно перетворити нечіткий набір виведень у чітке число. Є велика кількість методів приведення до чіткості.

Приклад. Нехай деяка система описується наступними нечіткими правилами:

$П_1$: якщо $x \in A$, то $w \in D$,
$П_2$: якщо $y \in B$, то $w \in E$,

$П_3$: якщо $z \in C$, то $w \in F$,

де x, y і z – імена вхідних змінних, w – ім'я змінної виведення, а A, B, C, D, E, F – задані функції належності.

Передбачається, що вхідні змінні прийняли деякі конкретні (чіткі) значення – $x_0 y_0$ і z_0. Відповідно до наведених етапів, на етапі 1 для заданих значень, виходячи з функцій належності A, B, C, перебувають ступені істинності $\alpha(x_0)$, $\alpha(y_0)$, $\alpha(z_0)$ для передумов кожного із трьох наведених правил.

На етапі 2 відбувається «відсікання» функцій приналежності висновків правил (тобто D, E, F) на рівнях $\alpha(x_0)$, $\alpha(y_0)$, $\alpha(z_0)$.

На етапі 3 розглядаються усічені на другому етапі функції належності й виробляється їхнє об'єднання з використанням операції max, у результаті чого отримується комбінована нечітка підмножина, яка описується функцією належності $\mu_\Sigma(\omega)$ і відповідає логічному виведенню для вихідної змінної w.

Нарешті, на 4-му етапі, якщо необхідно, знаходимо чітке значення вихідної змінної, наприклад, із застосуванням центрового методу: чітке значення вихідної змінної визначається як центр ваги для кривої $\mu_\Sigma(\omega)$, тобто

$$\omega_0 = \frac{\int_\Omega \omega \mu_\Sigma(\omega) d\omega}{\int_\Omega \mu_\Sigma(\omega) d\omega}.$$

Наступні досить часто використовувані модифікації алгоритму нечіткого виведення:

$П_1$: якщо $x \in A_1$ і $y \in B_1$, то $z \in C_1$,

$П_2$: якщо $x \in A_2$ і $y \in B_2$, то $z \in C_2$,

де x і y – імена вхідних змінних, z – ім'я змінної виведення, A_1, A_2, B_1, B_2, C_1, C_2 – деякі задані функції залежності, при цьому чітке значення z_0 необхідно визначити на основі наведеної інформації й чітких значень x_0 і y_0.

Алгоритм Mamdani [82]. Даний алгоритм математично може бути описаний у такий спосіб.

1. Нечіткість: знаходяться ступені істинності для передумов кожного правила: $A_1(x_0)$, $A_2(x_0)$, $B_1(y_0)$, $B_2(y_0)$.
2. Нечітке виведення: знаходяться рівні «відсікання» для передумов кожного із правил:

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0),$$
$$\alpha_2 = A_2(x_0) \wedge B_2(y_0),$$

де через "$\wedge$" позначена операція логічного мінімуму (min), потім знаходяться «усічені» функції належності

$$C_1'(z) = (a_1 \wedge C_1(z)),$$
$$C_2'(z) = (a_2 \wedge C_2(z)),$$

3. Композиція: з використання операції МАКСИМУМ (max, далі позначуваної як "$\vee$") виробляється об'єднання знайдених усічених функцій, що приводить до одержання підсумкової нечіткої підмножини для функції належності:

$$\mu_{\Sigma}(z) = C(z) = C_1'(z) \vee C_2'(z) = (a_1 \wedge C_1(z)) \vee (a_2 \wedge C_2(z)).$$

4. Нарешті, приведення до чіткості (для знаходження z_0) здійснюється, наприклад, центровим методом.

Алгоритм Tsukamoto. [82] Вихідні дані – як у попереднього алгоритму, але в цьому випадку передбачається, що функції $C_1(z), C_2(z)$ є монотонними.

1. Перший етап – такий же, як в алгоритмі Mamdani.
2. На другому етапі спочатку знаходяться рівні «відсікання» α_1 і α_2, а потім розв'язуючи рівняння: $\alpha_1 = C_1(z_1)$, $\alpha_2 = C_2(z_2)$, знаходимо чіткі значення (z_1 і z_2) для кожного з вихідних правил.
3. Визначається чітке значення змінної виведення (як зважене середнє z_1 й z_2):

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2};$$

у загальному випадку (дискретний варіант центрового методу)

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i z_i}{\sum_{i=1}^{n} \alpha_i};$$

Приклад. Нехай маємо $A_1(x_0) = 0{,}7$, $A_2(x_0) = 0{,}6$, $B_1(y_0) = 0{,}3$, $B_2(y_0) = 0{,}8$, що відповідають рівням відсікання

$$\alpha_1 = \min(A_1(x_0), B_1(y_0)) = \min(0{,}7; 0{,}3) = 0{,}3,$$

$$\alpha_2 = \min(A_2(x_0), B_2(y_0)) = \min(0{,}6; 0{,}8) = 0{,}6$$

і значення $z_1 = 8$ і $z_2 = 4$, знайдені в результаті рішення рівнянь $C_1(z_1) = 0{,}3$, $C_2(z_2) = 0{,}6$.

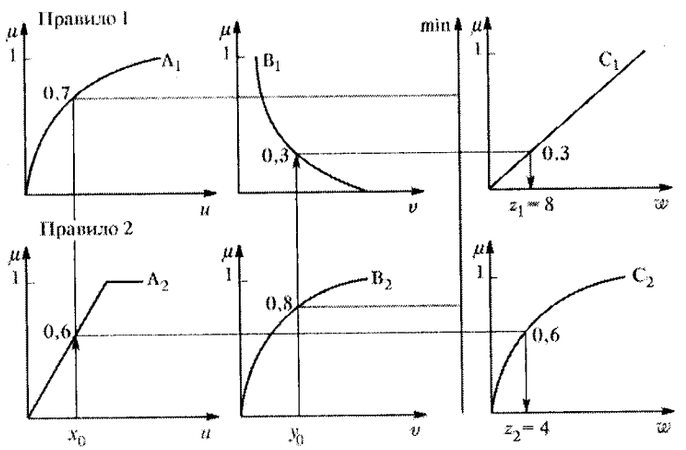


Рис. 4.7. Ілюстрації до алгоритму Tsukamoto.

При цьому чітке значення змінної виведення (див. рис. 4.7.): $z_0 = (8 \cdot 0{,}3 + 4 \cdot 0{,}6)\,(0{,}3 + 0{,}6) = 6$.

Алгоритм Sugeno. [82] Sugeno і Takagi використовували набір правил у наступній формі:

$П_1$: якщо $x \in A_1$ і $y \in B_1$, то $z_1 = a_1 x + b_1 y$,

$П_2$: якщо $x \in A_2$ і $y \in B_2$, то $z_2 = a_2 x + b_2 y$.

Подання алгоритму

1. Перший етап – як в алгоритмі Mamdani.
2. На другому етапі знаходимо $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$, $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$ і індивідуальні виходи правил:

$$z_1^* = a_1 x_0 + b_1 y_0,$$

$$z_2^* = a_2 x_0 + b_2 y_0.$$

На третьому етапі визначається чітке значення змінної виведення:

$$z_0 = \frac{\alpha_1 z_1^* + \alpha_2 z_2^*}{\alpha_1 + \alpha_2}.$$

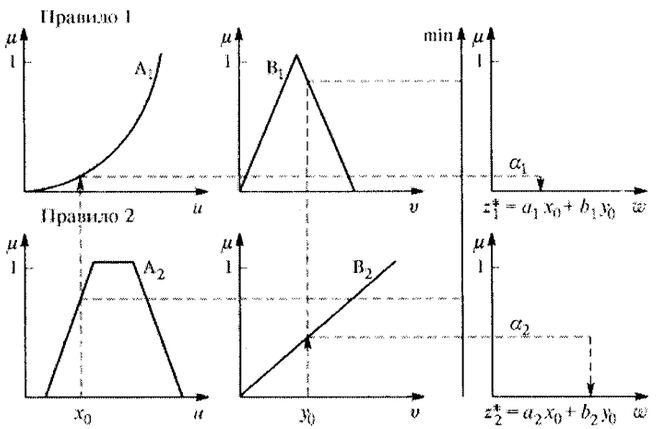


Рис. 4.8. Ілюстрації до алгоритму Sugeno.

Алгоритм Larsen. [82] В алгоритмі Larsen нечітка імплікація моделюється з використанням оператора множення.

Опис алгоритму

1. Перший етап – як в алгоритмі Mamdani.
2. На другому етапі, як в алгоритмі Mamdani спочатку знаходяться значення

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0),$$

$$\alpha_2 = A_2(x_0) \wedge B_2(y_0),$$

а потім – окремі нечіткі підмножини

$$\alpha_1 C_1(z), \alpha_2 C_2(z).$$

3. Знаходиться підсумкова нечітка підмножина з функцією залежності

$$\mu_\Sigma(z) = C(z) = (\alpha_1 C_1(z)) \vee (\alpha_2 C_2(z)),$$

(у загальному випадку n правил $\mu_\Sigma(z) = C(z) = \bigvee_{i=1}^{n} (\alpha_i C_i(z))$.

4. При необхідності виробляється приведення до чіткості.

Алгоритм Larsen проілюстровано на рис. 4.9.

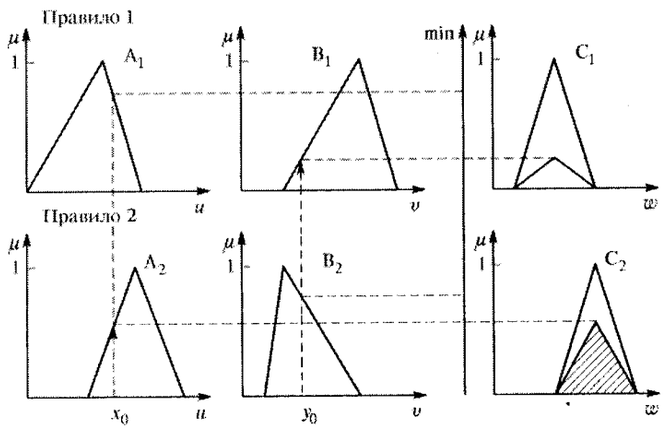


Рис. 4.9. Ілюстрація алгоритму Larsen.

Спрощений алгоритм нечіткого виведення

Вихідні правила в цьому випадку задаються у вигляді:

$П_1$: якщо $x \in A_1$ і $y \in B_1$, то $z_1 = c_1$,

$П_2$: якщо $x \in A_2$ і $y \in B_2$, то $z_2 = c_2$,

де c_1 і c_2 – деякі звичайні (чіткі) числа.

Опис алгоритму:

1. Перший етап – як в алгоритмі Mamdani.
2. На другому етапі знаходяться числа $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$, $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$.
3. На третьому етапі знаходиться чітке значення вихідних змінних за формулою:

$$z_0 = \frac{\alpha_1 c_1 + \alpha_2 c_2}{\alpha_1 + \alpha_2},$$

або – у загальному випадку наявності n правил – за формулою:

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i c_i}{\sum_{i=1}^{n} \alpha_i}.$$

Ілюстрація спрощеного алгоритму нечіткого виведення наведена на рис. 4.10.

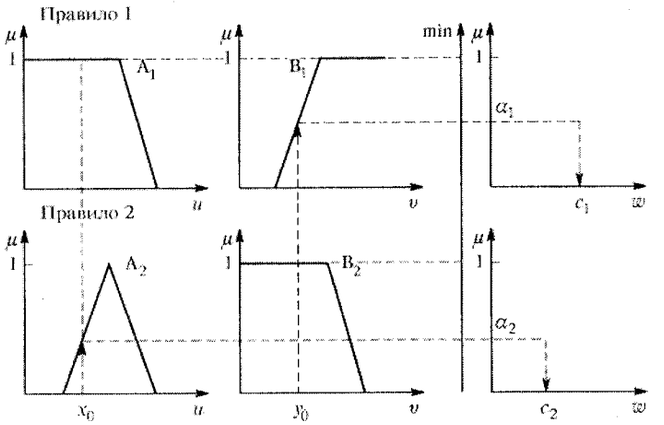


Рис. 4.10. Ілюстрація спрощеного алгоритму нечіткого виведення.

Приклад: нечіткий регулятор

Наведемо приклад використання апарату нечіткої логіки у задачі керування. Розглянемо замкнуту систему регулювання, де через О позначений об'єкт керування, через Р – регулятор, а через u, y, e, x – відповідно вхідний сигнал системи, її вихідний сигнал, сигнал помилки (неузгодженості), що надходить на вхід регулятора, і вихідний сигнал регулятора.

У розглянутій системі регулятор виробляє керівний сигнал x відповідно до обраного алгоритму регулювання, наприклад, пропорційно сигналу помилки, або її інтегралу. Покажемо, що в цьому випадку для вироблення такого сигналу застосовані розглянуті вище методи апарату нечіткої логіки.

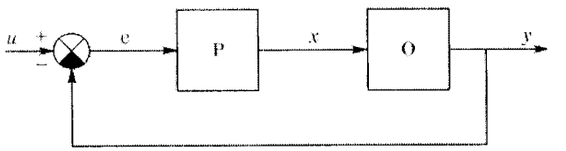


Рис. 4.11. Структура замкнутої системи керування.

Припустимо, що функції регулятора виконує мікроконтролер, при цьому аналоговий сигнал є обмежений діапазоном [–1, 1] і перетвориться в цифрову форму аналого-цифровим перетворювачем (АЦП) з дискретністю 0,25, а вихідний сигнал регулятора x формується за допомогою цифроаналогового перетворювача й має всього 5 рівнів: –1, –0,5, 0, 0,5, 1.

Беручи до уваги вказані рівні, введемо лінгвістичні змінні:

A_1: великий позитивний,

A_2: малий позитивний,

A_3: нульовий,

A_4: малий негативний,

A_5: великий негативний,

і на дискретній множині можливих значень сигналу неузгодженості е визначимо функції залежності.

Припустимо, що функціонування регулятора визначається наступними правилами:

$П_1$: якщо, $e = A_3$ і $\Delta e = A_3$, то $x = 0$,

$П_2$: якщо, $e = A_2$ і $\Delta e = A_2$, то $x = -0{,}5$,

$П_3$: якщо, $e = A_4$ і $\Delta e = A_4$, то $x = 1$,

$П_4$: якщо, $e = A_1$ і $\Delta e = A_1$, то $x = -1$,

де Δe – перша різниця сигналу помилки в поточний дискретний момент часу.

Табл. 4.1. Значення функцій належності

	-1	-0,75	-0,5	-0,25	0	0,25	0,5	0,75	1
$A_1(e)$	0	0	0	0	0	0	0,3	0,7	1
$A_2(e)$	0	0	0	0	0,3	0,7	1	0,7	0,3
$A_3(e)$	0	0	0,3	0,7	1	0,7	0,3	0	0
$A_4(e)$	0,3	0,7	1	0,7	0,3	0	0	0	0
$A_5(e)$	1	0,7	0,3	0	0	0	0	0	0

Зазначимо, що набір правил може бути загалом і іншим. Якщо, наприклад, використовується спрощений алгоритм нечіткого виведення, то при значеннях, скажімо $e = 0{,}25$ і $\Delta e = 0{,}5$ маємо:

$a_1 = \min(0{,}7;0{,}3) = 0{,}3$ і $x_1 = 0x_1$,

$a_2 = \min(0{,}7;1) = 0{,}7$ і $x_2 = -0{,}5$,

$a_3 = \min(0;0) = 0$ і $x_3 = 1$,

$a_4 = \min(0;0{,}3) = 0{,}3$ і $x_4 = -1$, і вихід регулятора

$$x = \frac{0{,}3 \cdot 0 + 0{,}7 \cdot (-0{,}5) + 0 \cdot 1 + 0{,}3 \cdot (-1)}{0{,}3 + 0{,}7 + 0 + 0{,}3} = \frac{-0{,}6}{1{,}3} = -0{,}5.$$

Аналогічним чином значення вихідного сигналу регулятора розраховуються при інших значеннях е і Δе.

При проектуванні подібних («нечітких») регуляторів основним етапом є задання набору нечітких правил. Інші аспекти: вибір форми функцій належності, алгоритму приведення до чіткості, виглядають завданнями простішими.

4.3. Інші методи моделювання нечіткостей в інтелектуальних системах

4.3.1. Використання коефіцієнтів впевненості (КВ)

У відомій експертній системі MYCIN для подання нечіткостей використовуються коефіцієнти впевненості, які введені для вимірювання ступеня довіри до будь-якого поданого висновку, який є результатом одержаних до цього моменту тверджень. Коефіцієнт впевненості (k) – це різниця між двома мірами:

$$k\,[h{:}e] = md\,[h{:}e] - mnd\,[h{:}e].$$

У цьому виразі $k\,[h{:}e]$ – впевненість у гіпотезі h з врахуванням тверджень e; $md\,[h{:}e]$ – міра довір'я h при заданому e, тоді як $mnd\,[h{:}e]$ – міра недовіри гіпотезі h при твердженні e.

Коефіцієнт впевненості може змінюватись від -1 (абсолютна хибність) до +1 (абсолютна істина), приймаючи також всі проміжкові значення, причому 0 означає повне незнання. Значення ж md і mnd, іншого боку, можуть змінюватись лиш від 0 до 1. Отже, k – це простий спосіб зважувати твердження «за» і «проти».

Ця формула не дозволяє відрізняти випадки протилежних тверджень (і md, і mnd – обидві великі) від випадку недостатньої інформації (і md, і mnd – обидві малі), що інколи досить корисно.

Зауважимо, що ні k, ні md, ні mnd не є імовірнісними мірами, md і mnd підпорядковуються деяким аксіомам теорії ймовірності, але не є вибірками з якої-небудь популяції, і тому їм не можна дати статистичну інтерпретацію. Вони просто дозволяють впорядкувати гіпотези відповідно до того ступеня правдоподібності, який у них є. У системі MYCIN операції min, max і віднімання від одиниці були використані аналогічно, як і в нечіткій логіці.

Те, що було додано Шортліффом [307], – це формула уточнення, за якою нову інформацію можна було співставляти зі старими результатами. Вона застосовується до мір довір'я і недовір'я, пов'язаних з кожним реченням. Формула для md виглядає наступним чином:

$$md\,[h{:}e1, e2] = md\,[\text{h:e1}] + md\,[h{:}e2](1 - md\,[h{:}e1]),$$

де кома між $e1$ і $e2$ означає, що $e2$ наступає за $e1$. Аналогічно уточнюються значення mnd.

Зміст формули полягає в тому, що ефект другого твердження ($e2$) на гіпотезу h при заданому $e1$ відчувається у зміщенні md у сторону певної визначеності на відстань, яка залежить від другого твердження. Ця формула має дві важливі властивості:

- вона симетрична у тому змісті, що порядок $e1$ і $e2$ не суттєвий.
- в міру накопичення тверджень, які підтверджують гіпотезу md (або mnd), рухається до визначеності.

4.3.2. Байесівський підхід

Одним із обчислень невизначеності, яке має міцні теоретичні результати, є теорія ймовірностей.

У багатьох системах, включаючи систему знаходження корисних копалин PROSPECTOR, теорема Байеса використовується як нитка, яка допомагає зв'язати докупи інформацію, що надходить з різних джерел. Це правило дозволяє обчислити відносну правомірність гіпотез, що конкурують, виходячи із сили тверджень. В основі правила лежить формула:

$$v\,(h{:}e) = p\,(e{:}h)/p\,(e{:}h),$$

де v відношення правомірності визначається як ймовірність подій або твердження е при умові заданої конкретної гіпотези h, яка поділена на ймовірність цього твердження при умові хибності заданої гіпотези (h). Так, якщо ми знаємо ймовірності твердження при заданій гіпотезі та її доповнення, то ми можемо визначити правомірність цієї гіпотези у світі наявних тверджень. Відношення правомірності може бути використане для уточнення шансів на користь розглядуваної гіпотези, якщо стає відомо, що відбулась подія e.

Дамо це правило на мові шансів. Шанси на користь чогось (o) і ймовірність (p) просто перетворяться один в одного:

$$o = p/(1 - p); \; p = o/(1 + o).$$

Перетворення оцінки «шанси проти» (a) в оцінку «шанси за» також досить просте:

$$o = 1/a.$$

Отже, 7 проти 4 може бути виражене як 1,75 проти 1, що відповідає 0,5714 (до 1) «за» (або 4 до 7 на користь розглянутої гіпотези).

Байесівська схема уточнення може бути зведена до виразу: $o'(K) = o\,(h) \bullet v\,(h : e)$, де $o\,(h)$ – апріорні шанси на користь h, а $o\,(h)$ – результатні апостеорні шанси за умови настання події e відповідно до відношень правомірності.

Тоді інформація від різних джерел знань може комбінуватись простим множенням. При відомих апріорних шансах для гіпотез, що конкурують на користь h, $o\,(h)$ і подій, про які відомо, що вони відбулися, легко обчислюються апостеорні шанси, а за ними – і ймовірності. Відношення правомірності отримуються з простої двомірної таблиці, яка показує, як часто відбувається кожна подія при кожній з гіпотез.

Відношення правомірності завжди додатні. Хоча нуль і нескінченність можуть зустрітися, завжди є можливість уникнути цих значень, при необхідності трохи змінивши дані. Тоді твердження $v > 1$ вказує на користь гіпотези, $v < 1$ – проти неї, а $v = 1$ говорить про те, що твердження не впливають на правомірність розглянутої гіпотези.

Множник v показує, наскільки ймовірнішою стає певна гіпотеза при існуванні тверджень, ніж при їх відсутності.

Запитання для повторення та контролю знань

1. Які Ви знаєте способи врахування невизначеностей в інтелектуальних системах?
2. Який математичний апарат використовується для врахування фактору невизначеності?
3. Що таке нечітка логіка?
4. Що таке нечіткі множини?
5. Які існують способи формалізації нечітких понять?
6. Яка область застосування нечітких множин?
7. Яка функція називається характеристичною функцією належності?
8. Дайте означення аналітичної моделі нечіткої множини.
9. Що таке носій нечіткої множини?
10. Що таке підмножина α-рівня?
11. Які основні характеристики нечітких множин?
12. Які Ви знаєте операції над нечіткими множинами?
13. Які властивості операцій над нечіткими множинами?
14. Яка норма називається трикутною?
15. Що таке трикутна конорма?
16. Які операції алгебри над нечіткими множинами?

17. Дайте означення нечіткої змінної.
18. Дайте означення лінгвістичної змінної.
19. Що таке нечіткі відношення?
20. Які існують методи подання нечітких відношень?
21. Яке нечітке бінарне відношення називається симетричним? Наведіть приклади.
22. Яке нечітке бінарне відношення називається рефлексивним? Наведіть приклади.
23. Яке нечітке бінарне відношення називається транзитивним? Наведіть приклади.
24. Наведіть кроки алгоритму Mamdani.
25. Наведіть кроки алгоритму Tsukamoto.
26. Наведіть кроки алгоритму Sugeno.
27. Наведіть кроки алгоритму Larsen.
28. Наведіть кроки спрощеного алгоритму нечіткого виведення.
29. Які інші методи моделювання нечіткостей Ви знаєте?
30. Що таке коефіцієнт впевненості?
31. Як використовується теорія ймовірності для подання нечіткостей?
32. Дайте означення апріорного й апостеорного шансів.

Завдання для самостійного розв'язування

1. За вихідною таблицею впливу кашлю на захворювання грип, обчислити апостеорну оцінку захворювання грипом, якщо відомо, що пацієнт кашляє.

	Захворювання		Всього
	Грип	Не грип	
Кашляє	20	33	53
Не кашляє	24	23	47
Всього	44	56	100

2. Знайти нечітку підмножину $B = A \circ F$, якщо відомо, що $\mu_A(X) = (1;\ 0.6;\ 0.3)$ і нечітке відношення F задане таблицею:

F	*1*	*2*	*3*
1	1	0.4	0.3
2	0.4	1	0.7
3	0.3	0.7	1

3. Використовуючи формулу Шортліффа знайдіть загальну міру довіря, якщо міра довіря до першого висновку рівна 0,4, а для другого – 0,6.
4. Розглянемо правило експертної системи MYCIN, представлене нижче:

   ЯКЩО 1) організм володіє грам додатнім кольором, І
   2) організм має форму колбочки, І
   3) організм в процесі росту утворює ланцюжки,

   ТО можна вважати (0,7), що цей мікроорганізм відноситься до класу streptococcus.

Вважатимемо, що сформульовані у правилі умови характеризуються такими коефіцієнтами впевненості:

Умова 1: 0,8

Умова 2: 0,2

Умова 3: 0,5

Який коефіцієнт впевненості характеризує висновок про те, що даний організм відноситься до класу streptococcus, яке видасть MYCIN у відповідності до сформованого правила?

5. Розглянемо таку пару правил експертної системи MYCIN:

ЯКЩО 1) культура взята із аналізу крові І

2) пацієнт страждає ушкодженням шкіри ecthyma gangrenosum,

ТО можна вважати (0,6), що цей мікроорганізм відноситься до класу pseudomonas.

ЯКЩО 1) тип інфекції бактеріальний І

2) пацієнт має серйозні опіки,

ТО можна вважати (0,4), що цей мікроорганізм відноситься до класу pseudomonas.

Вважатимемо, що сформовані у першому правилі умови характеризуються коефіцієнтами впевненості 0,8 і 0,9, а сформовані у 2-му правилі коефіцієнтами впевненості 0,2 і 0,3. Який коефіцієнт впевненості буде характеризувати виведення, що даний організм відноситься до класу pseudomonas?

6. Вважатимемо, що поняття «незвичайна оцінка з десяти» визначена як нечітка множина:

A={(0/1), (1/0,9), (2/0,7), (3/0,5), (4/0,3), (5/0,1), (6/0,1), (7/0,3), (8/0,5), (9/0,9), (10/0,9)},

а поняття „висока оцінка з десяти" визначена як нечітка множина

B={(0/0), (1/0), (2/0), (3/0,1), (4/0,2), (5/0,3), (6/0,4), (7/0,6), (8/0,7), (9/0,8), (10/1)}.

Побудуйте нечітку множину «незвичайно висока оцінка з десяти».

6. Нехай задано дві нечіткі множини A={(1|0,2),(2|0,5),(3|0,8),(4|1)} та B={(1|0,1), (2|0,3),(3|1),(4|0,75)}. Знайти A∪B, A∩B, A-B, B-A, A⊕B, AB.

РОЗДІЛ 5

ТЕОРЕТИЧНІ АСПЕКТИ ІНЖЕНЕРІЇ ЗНАНЬ

- Поле знань
- Стратегії одержання знань
- Теоретичні аспекти видобування знань
- Теоретичні аспекти структурування знань

Цей розділ цілком присвячений теоретичним проблемам інженерії знань, іншими словами – проектуванню баз знань – одержанню і структуризації знань фахівців для подальшого розроблення баз знань. Центральним поняттям на стадіях одержання і структуризації є так зване поле знань.

5.1. Поле знань

Інженерія знань – досить молодий напрям штучного інтелекту (ШІ), який з'явився тоді, коли практичні розробники зіткнулися з досить нетривіальними проблемами труднощів «видобування» і формалізації знань. У перших книгах зі ШІ ці факти зазвичай тільки постулювалися, надалі почалися серйозні дослідження з виявлення оптимальних стратегій видобування знань [211, 45].

Цей розділ цілком присвячений теоретичним проблемам інженерії знань, іншими словами – проектуванню баз знань – одержанню і структуризації знань фахівців для подальшого розроблення баз знань. Центральним поняттям на стадіях одержання і структуризації є так зване поле знань.

Поле знань – це умовний неформальний опис основних понять і взаємозв'язків між поняттями предметної області, виявлених із системи експерта, у вигляді графа, діаграми, таблиці або тексту.

5.1.1. Мова опису поля знань

Поля знань як перший крок до формалізації подає модель знань про предметну область у тому вигляді, в якому її зумів виразити аналітик на деякій «своїй» мові. Що це за мова? Відомо, що словник мови конкретної науки формується шляхом поповнення загальновживаної мови спеціальними термінами і знаками, які або запозичуються з повсякденної мови, або винаходяться [83]. Назвемо цю мову L і розглянемо її властивості, враховуючи, що стандарту цієї мови наразі не існує, а кожен інженер зі знань змушений сам її винаходити.

По-перше, як і в мові будь-якої науки, в ній має бути якомога менше неточностей, властивих повсякденним мовам. Частково точність досягається більш строгим визна-

ченням понять. Ідеалом точності, зазвичай, є мова математики. Мова L, мабуть, займає проміжне місце між природною мовою та мовою математики.

По-друге, бажано не використовувати в ній термінів інших наук в іншому, тобто новому, сенсі. Це викликає непорозуміння.

По-третє, мова L, мабуть, буде або символьною мовою, або графічною (схеми, малюнки, піктограми).

Під час вибору мови опису поля знань не слід забувати, що на стадії формалізації необхідно замінити її на машинно-реалізовану мову подання знань (МПЗ), вибір якої залежить від структури поля знань. Існує ряд мов, досить універсальних, щоб претендувати на роль мови інженерії знань, – це структурно-логічна мова SLL, що включає апарат лямбда-конверсії [35], мова K-систем [84], УСК [113] тощо. Однак вони не знайшли широкого застосування. У деякому розумінні створення мови L дуже близьке до ідей розроблення універсальних мов науки [83]. До XVII століття утворилося два підходи в розробленні універсальних мов: створення мов-класифікацій і логіко-конструктивних мов. До першої відносять проекти, висхідні до ідеї Ф. Бекона, – це мови Вілкінса і Далгарно. Другий підхід пов'язаний з дослідженнями в рамках пошуку універсального методу пізнання, найчіткіше висловленого Р. Декартом, а потім у проекті універсальної характеристики Г. Лейбніца. Саме Лейбніц позначив основні контури вчення про символи, які відповідно до його задумів у XVIII столітті розвивав Г. Ламберт, який дав ім'я науці «семіотика». Семіотика, в основному, знайшла своїх адептів у сфері гуманітарних наук. Останнім часом утворилася також нова галузь семіотики – прикладна семіотика.

Представники природничих наук ще не до кінця усвідомили позитивні якості семіотики тільки через те, що мають справу з досить простими і «жорсткими» предметними областями. Їм вистачає апарату традиційної математики. Проте, в інженерії знань ми маємо справу з «м'якими» предметними областями, де явно не вистачає виразної адекватності класичного математичного апарату і де велике значення має ефективність дотації (її компактність, простота модифікації, зрозумілість інтерпретації, наочність тощо).

Мови семіотичного моделювання [133, 150] як природний розвиток мов ситуаційного керування є, як нам здається, першим наближенням до мови інженерії знань. Саме мінливість і умовність знаків роблять семіотичну модель пристосованою до складних сфер реальної людської діяльності. Тому головне на стадії концептуалізації – збереження природної структури поля знаннями, а не виразні можливості мови. Традиційно семіотика включає (рис. 5.1):

- *синтаксис* (сукупність правил побудови мови або відношення між знаками);
- *семантику* (зв'язок між елементами мови та їх значень або відношення між знаками і реальністю);
- *прагматику* (відношення між знаками та їх користувачами).

Рис. 5.1. Структура семіотики

5.1.2. Семіотична модель поля знань

Поле знань Pz є деякою семіотичною моделлю, що може бути зображена як граф, малюнок, таблиця, діаграма, формула або текст залежно від смаку інженера зі знань і особливостей предметної області.

Особливості ПО можуть вплинути на форму і зміст компонентів структури Pz.

Розглянемо відповідні компоненти Pz (рис. 5.2).

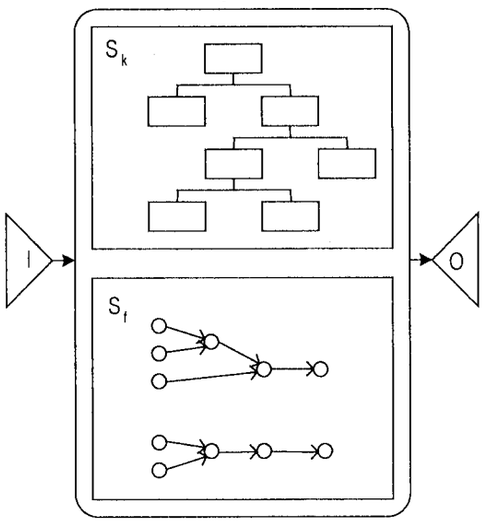


Рис. 5.2. Структура поля знань.

Синтаксис. Узагальнено синтаксичну структуру поля знань можна подати як

$$П = (I, O, M),$$

де I – структура початкових даних, які підлягають опрацюванню та інтерпретації в експертній системі; O – структура вихідних даних, тобто результату роботи системи; M – операційна модель предметної області, на підставі якої відбувається модифікація I в O.

Віднесення компонентів I та O в P обумовлене тим, що складові і структура цих інтерфейсних компонентів імпліцитно (тобто неявно) присутні в моделі репрезентації в пам'яті експерта. Операційна модель M може бути подана як сукупність *концептуальної* структури S_k, що відображає понятійну структуру предметної області, і *функціональної* структури S_f, що моделює схему міркувань експерта:

$$M=(S_k,S_f).$$

S_k, виступає як статична, незмінна складова P, у той час як S_f відображає динамічну, змінювану складову.

Формування S_f засноване на виявленні понятійної структури предметної області. Далі описується досить універсальний алгоритм здійснення концептуального аналізу на основі модифікації парадигми структурного аналізу і побудови ієрархії понять (так звана «піраміда знань»).

Структура S_f включає поняття предметної області A і моделює основні функціональні зв'язки R_A або відношення між поняттями, що утворять S_k. Ці зв'язки відображають модель або стратегію ухвалення рішення у вибраній ПО. У такий спосіб S_f утворить стратегічну складову M.

Семантика. Семантика, що надає певне значення пропозиціям будь-якої формальної мови, визначається на деякій області. Фактично це набір правил інтерпретації пропозицій і формул мови. Семантика L повинна бути композиційною, тобто значення пропозиції визначається як функція значень його складових.

Семантика мови L залежить від особливостей предметної області, вона має властивість поліморфізму, тобто ті самі оператори мови в різних завданнях можуть мати свої особливості.

Семантику поля знань Pz можна розглядати на двох рівнях. На першому рівні P^i_{Zg}, є семантична модель знань експерта 1 про деяку предметну область O_g. На другому рівні будь-яке поле знань Pz є моделлю деяких знань, і, отже, можна говорити про зміст його як деякого дзеркала дійсності. Розглядати перший рівень у відриві від конкретної області недоцільно, тому зупинимося докладніше на другому.

Поле P^{ij}_{Zg} – це результат, отриманий «після 4-ї трансляції» (якщо говорити мовою інформатики).

- 1-ша трансляція (I_j) – це сприйняття та інтерпретація дійсності *O* предметної області g і-м експертом. У результаті у пам'яті експерта утвориться модель M_{gi} як семантична репрезентація дійсності і його особистого досвіду по роботі з нею.
- 2-га трансляція (V_i) – це вербалізація досвіду і-го експерта, коли він намагається пояснити свої міркування S_f, і передати свої знання Z_i інженерові зі знань. У результаті V_i утвориться або текст T_i, або мовне повідомлення C.
- 3-тя трансляція (I_j) – це сприйняття й інтерпретація повідомлень T_i, або C_i j-м інженером зі знань. У результаті в пам'яті інженера по знаннях утвориться модель світу M_{gj}.

- 4-та трансляція (K_j) – це кодування й вербалізація моделі M_{gj} у формі поля знань P^{ij}_{Zg}.

Найбільше ця схема нагадує дитячу гру в «зіпсований телефон»; перед інженером по знаннях стоїть важке завдання – добитися максимальної відповідності M_{gj} і V_i. У читачів не повинно виникати ілюзій, що P_{Zg} відображає O_g. У жодному разі, адже знання – річ сугубо авторизована, варто було б на кожній ЕС ставити чіткий ярлик i-j, тобто «база знань експерта і в розумінні інженера зі знань.)». Варто замінити, наприклад, інженера зі знань j на h, і вийде зовсім інша картина.

Наведемо приклад впливу суб'єктивних поглядів експерта на M_{gj} і V_i. Реальність (O_g): два чоловіки прибігають на вокзал за 2 хвилини до відходу поїзда. Біля каси – черга. В автоматичних касах вільно, але ні одного, ні в другого немає дрібних грошей. Наступний поїзд через 40 хвилин. Обоє спізнюються на важливу зустріч.

Інтерпретація 1-го експерта (I_1,): не можна приходити на вокзал пізніше ніж за 10 хвилин. Інтерпретація 2-го експерта (I_2): треба завжди мати дрібні гроші у кишені.

Вербалізація 1-го експерта (V_1,): спізнився на потрібний поїзд, тому що не розрахував час.

Вербалізація 2-го експерта (V_2): спізнився, тому що на вокзалі плутанина, біля кас черги.

Наступні трансляції ще більше будуть спотворювати й видозмінювати модель, але тепер вже з урахуванням суб'єктивного сприйняття інженерів зі знань.

Отже, якщо вважати поле знань змістовою (семантичною) моделлю предметної області, то ця модель двічі суб'єктивна. І якщо модель M_{gj} – це усічене відображення O_g, то саме Pz – лише відблиск M_{gj} через призму V_i і M_{gj}.

Прагматика. Як прагматичну складову семіотичної моделі варто розглядати технології здійснення структурного аналізу ПО, користуючись яким інженер зі знань може сформувати Pz за результатами стадії здобуття знань.

Отже, під прагматикою будемо розуміти практичні аспекти розроблення й використання поля, тобто як від хаосу чернеток і стенограм сеансів здобуття знань перейти до стрункої або хоча б зрозумілої моделі.

Докладніше ці питання висвітлені нижче. Однак поле знань, незважаючи на всі старання інженера зі знань і експерта, завжди будуть лише блідим відбитком реально існуючої предметної області, адже світ, що нас оточує, такий мінливий, складний і різноманітний, а те, що зберігається в нашій свідомості, так погано піддається вербалізації. Проте, з погляду наукової методології, без продуманого, чіткого й гарного поля знань не може йти й мови про створення бази знань промислової ЕС.

5.1.3. «Піраміда» знань

Ієрархічність понятійної структури свідомості підкреслюється в роботах багатьох психологів [23]. Поле знань можна стратифікувати, тобто розглядати на різних рівнях абстракції понять. У «піраміді знань» кожний наступний рівень служить для сходження на новий ступінь узагальнення і поглиблення знань у предметній області. Отже, можлива наявність декількох рівнів понятійної структури S_k. Видається доцільним пов'язати це із глибиною професійного досвіду (наприклад, як у системі АВТАНТЕСТ [44] або з рівнем ієрархії в структурних щаблях організації. Зазвичай, стратегії прийняття рішень, тобто функціональні структури S, на різних рівнях будуть істотно відрізнятися. Якщо спробувати дати математичну інтерпретацію рівнів піраміди знань $U=(U_1,U_2,U_3,\ldots,U_N)$,

то найбільш найпрозорішим є поняття *гомоморфізму* – відображення деякої системи E, що зберігає основні операції і основні відношення цієї системи. Нехай

$$E = (E, (o_i : i \in I), (r_j : j \in J)$$

деяка система з основними поняттями $o_i, i \in I$ та основними відношеннями r ; $j \in J$.

Гомоморфізмом системи E в однотипну їй систему E' буде: $E' = (E', (o'_i : i \in I), (r'_j : j \in J)$, називається відображення,

$$\Phi : E \Rightarrow E',$$

яке задовольняє наступні дві умови:

$$\Phi(o_i(e_1,...,e_{j_i}) = o'_i(\Phi(e_1),...,\Phi(e_{j_i}));$$

$$(e_1,...,e_m) \in r_j \Rightarrow (\Phi(e_1),...,\Phi(e_m) \in r'_j.$$

для всіх елементів $e_1,...,e_m$ з E та всіх $i \in I, j \in J$

Відповідно до введених позначень рівні піраміди суть гомоморфізми моделей (тобто понять і відношень) предметної області

$$\Phi : M \Rightarrow M',$$

де $M = (A, R, S); M' = (A', R', S'), A'$ – мета-поняття, або поняття вищого рівня абстракції; R' – мета-відношення; S' – мета-стратегії. Сходячи по ступенях піраміди, ми отримуємо систему гомоморфізмів, що відповідає результатам, отриманим в когнітивній психології про зменшення розмірності семантичного простору пам'яті зі збільшенням досвіду експертів.

5.2. Стратегії одержання знань

Під час формування поля знань ключовим питанням є сам процес отримання знань, коли відбувається перенесення компетентності експертів на інженерів зі знань. Для назви цього процесу в літературі по ЕС набуло поширено кілька термінів: придбання, здобування, витягання, одержання, виявлення, формування знань. У англомовній спеціальній літературі в основному використовуються два: acquisition (придбання) і elicitation (виявлення, витягання, встановлення).

Термін «придбання» трактується або дуже широко – тоді він включає весь процес передавання знань від експерта до бази знань ЕС, або вже як спосіб автоматизованої побудови бази знань за допомогою діалогу експерта і спеціальної програми (при цьому структура поля знань заздалегідь закладається в програму). В обох випадках термін «придбання» не стосується самого таїнства екстрагування структури знань з потоку інформації про предметну область. Цей процес описується поняттям «видобування».

Автори схильні використовувати цей термін як більш ємний і такий, що точніше виражає зміст процедури перенесення компетентності експерта через інженера зі знань у базу знань ЕС.

Видобування знань (knowledge elicitation) – це процедура взаємодії експерта із джерелом знань, у результаті якої стають явними процес міркувань фахівців при ухваленні рішення і структура їх уявлень про предметну область.

На сьогодні більшість розробників ЕС відзначають, що процес видобування знань залишається «найвужчим» місцем при побудові промислових ЕС. При цьому їм доводиться практично самостійно розробляти методи витягання, зіштовхуючись із такими труднощами:

- організаційні непогодженості;
- невдалий метод витягання, що не збігається зі структурою знань у певній області;
- неадекватна модель (мова) для подання знань.

Можна додати до цього [44]:

- невміння налагодити контакт із експертом;
- термінологічний різнобій;
- відсутність цілісної системи знань в результаті витягання тільки «фрагментів»;
- спрощення «картини світу» експерта тощо.

Процес видобування – це тривала і трудомістка процедура, в якій інженерові зі знань, озброєному спеціальними знаннями з когнітивної психології, системного аналізу, математичної логіки та ін., необхідно відтворити модель предметної області, якою користуються експерти для прийняття рішення. Часто розроблювачі-початківці ЕС, бажаючи спростити цю процедуру, намагаються підмінити інженера зі знань самим експертом. З багатьох причин це небажано.

По-перше, велика частина знань експерта – це результат численних нашарувань ступенів досвіду. І часто, знаючи, що з А виводиться В, експерт не усвідомлює, що ланцюжок його міркувань був набагато довший, наприклад $A \to D \to C \to B$ або $A \to Q \to R \to B$

По-друге, як було відомо ще Платонові, мислення діалогічне. І тому діалог інженера зі знань і експерта – найприродніша форма вивчення лабіринтів пам'яті експерта, у яких зберігаються знання, частина яких має невербальний характер, тобто виражені не у формі слів, а, наприклад, у формі наочних образів. І саме в процесі пояснення інженерові зі знань експерт на ці розмиті асоціативні образи навішує чіткі словесні ярлики, тобто вербалізує знання.

По-третє, експертові важче створити модель предметної області внаслідок глибини й обсягу інформації, якою він володіє. Ще в ситуаційному керуванні [150] було виявлено: об'єкти реального світу зв'язані більш ніж 200 типами відношень (тимчасові, просторові, причинно-наслідкові, типу » частина-ціле» та ін.). Ці відношення і зв'язки предметної області утворять складну систему, з якої виділити «скелет» або головну структуру іноді доступніше аналітикові, що до того ж володіє системною методологією.

Термін «придбання» у цього залишений за автоматизованими системами прямого спілкування з експертом. Вони дійсно безпосередньо здобувають вже готові фрагменти знань відповідно до структур, закладених розроблювачами систем. Більшість цих інструментальних засобів спеціально орієнтовано на конкретні ЕС із жорстко позначеною предметною областю й моделлю подання знань, тобто не є універсальними.

Придбання знань (knowledge acquisition) – процес наповнення бази знань експертом з використанням спеціалізованих програмних засобів.

Наприклад, система TEIRESIAS [229], що стала прародичкою всіх інструментаріїв для придбання знань, призначена для поповнення бази знань системи MYCIN або її дочірніх галузей, побудованих на «оболонці» EMYCIN [307] в області медичної діагностики з використанням продукційної моделі подання знань. Три покоління та основні тенденції СПЗ будуть детально описані нижче.

Термін формування знань традиційно закріпився за надзвичайно перспективною областю інженерії знань, що активно розвивається і займається розробленням моделей, методів і алгоритмів навчання. Вона включає індуктивні моделі формування знань і автоматичного породження гіпотез, наприклад, ДСМ-метод на основі навчальних вибирань, навчання за аналогією та іншими методами. Ці моделі дозволяють виявити причинно-наслідкові емпіричні залежності в базах даних з неповною інформацією, що містять структуровані числові й символьні об'єкти (часто в умовах неповноти інформації).

Формування знань (machine learning) – процес аналізу даних і виявлення прихованих закономірностей з використанням спеціального математичного апарату програмних засобів.

Традиційно до завдань формування знань або машинного навчання відносяться завдання прогнозування, ідентифікації (синтезу) функцій, розшифровування мов, індуктивного виведення й синтезу з додатковою інформацією [64]. У широкому сенсі до навчання на прикладах можна віднести і методи навчання розпізнавання образів [10].

Індуктивне виведення правил із фактів застосоване також у системах AQ, AQUINAS, KSSI, INSTIL і деяких інших.

Найдослідженішими серед методів машинного навчання є, очевидно, методи розпізнавання образів, зокрема алгебраїчний підхід, в якому передбачається збагачення вихідних евристичних алгоритмів за допомогою алгебраїчних операцій і побудова сімейства алгоритмів, що гарантує одержання коректного алгоритму для рішення досліджуваного класу завдань, тобто алгоритму, що правильно класифікує кінцеву вибірку по всіх класах [15]. Однак застосування методів формування знань наразі не стало промисловою технологією розроблення баз знань.

Щоб ці методи стали елементами технології інтелектуальних систем, необхідно вирішити ряд завдань [134]:

- забезпечити механізм сполучення незалежно створених баз даних, що мають різні схеми, з базами знань інтелектуальних систем;
- встановити відповідність між набором полів бази даних і безліччю елементів декларативного компонента бази знань;
- виконати перетворення результату роботи алгоритму навчання у спосіб подання, підтримуваний програмними засобами інтелектуальної системи.

Крім перерахованих, існують також і інші стратегії одержання знань, наприклад, у випадку навчання на прикладах (case-based reasoning), коли джерело знань – це безліч прикладів предметної області [146]. Навчання на основі прикладів (прецедентів) включає настроювання алгоритму розпізнавання на завдання за допомогою надання прикладів, класифікація яких відома [68].

Навчання на прикладах тісно пов'язане з машинним навчанням. Відмінність полягає в тому, що результат навчання в розглянутому тут випадку повинен бути інтерпретований у деякій моделі, в якій, можливо, вже містяться факти та закономірності предметної області, і перетворений у спосіб подання, що допускає використання результату навчання в базі знань, для моделювання міркувань, для роботи механізму пояснення тощо, тобто робить результат навчання елементом відповідної технології. Наприклад, у системі INDUCE [80] породжується несуперечливий опис деякого класу об'єктів по великій кількості прикладів і контрприкладів заданого класу. Як мова подання використовується мова багатозначної логіки першого порядку).

Зазначимо також появу двох нових «прапорців» у стані прихильників методів машинного навчання – це data mining і knowledge discovery. Обидва підходи базуються на аналізі даних і пошуку закономірностей.

Отже, можна виділити три основні стратегії проведення стадії одержання знань під час розроблення інтелектуальних систем (див. рис. 5.3).

1. З використанням ЕОМ при наявності відповідного програмного інструментарію, інакше – придбання знань.
2. З використанням програм навчання при наявності репрезентативної (тобто досить представницької) вибірки прикладів прийняття рішень у предметній області та відповідних пакетів прикладних програм, інакше – формування знань.
3. Без використання обчислювальної техніки шляхом безпосереднього контакту інженера зі знань і джерела знань (експерт, спеціальна література або інші джерела), інакше – видобування знань.

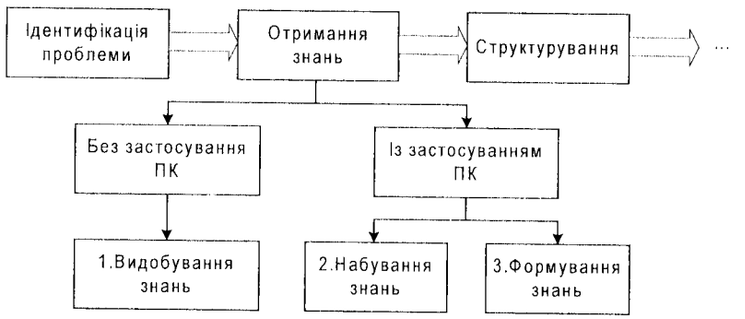


Рис. 5.3. Стратегії одержання знань.

У підручнику докладно будуть розглядатися процеси видобування і придбання знань, тому що на сучасному етапі розроблення інтелектуальних систем ці стратегії є найефективнішими й найперспективнішими. Формування знань, що тяжіє більшою мірою до області machine learning, тобто індуктивного навчання, ґрунтуючись на добре дослідженому апараті розпізнавання образів і виявлення подібності об'єктів, виходить за рамки цього підручника. Також поза цією книгою залишилися питання формування знань із даних (data mining, knowledge discovery) та ін.

5.3. Теоретичні аспекти видобування знань

Оскільки основною проблемою інженерії знань є процес видобування знань, інженерові по знаннях необхідно чітко розуміти природу й особливості цих процесів. Щоб розібратися в природі видобування знань, виділимо три основних аспекти цієї процедури (рис. 5.4):

A = {A1, A2, A3} = {психологічний, лінгвістичний, гносеологічний}.

Рис. 5.4. Теоретичні аспекти інженерії знань.

5.3.1. Психологічний аспект

Із трьох аспектів видобування знань психологічний – А1 – є провідним, оскільки він визначає успішність і ефективність взаємодії інженера зі знань (аналітика) з основним джерелом знань – експертом-професіоналом. Психологічний аспект виділяється ще й тому, що видобування знань відбувається найчастіше в процесі безпосереднього спілкування розроблювачів системи. А у спілкуванні психологія є домінантною.

Спілкування, або комунікація (від лат. communicatio – зв'язок) – це міждисциплінарне поняття, що позначає всі форми безпосередніх контактів між людьми – від дружніх до ділових. Воно широко досліджується у психології, філософії, соціології, етології, лінгвістиці, семіотиці й інших науках. Існує кілька десятків теорій спілкування, і єдине, у чому сходяться всі автори, – це складність, багатоплановість процедури спілкування. Підкреслюється, що спілкування – не просто односпрямований процес передавання повідомлень і не двотактний обмін порціями відомостей, а нерозчленований процес циркуляції інформації, тобто спільний пошук істини [74] (див. рис. 5.5).

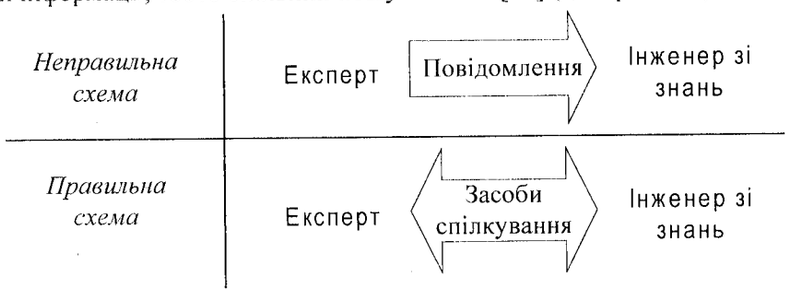


Рис. 5.5. Структура процесу спілкування

Отже, спілкування є процес вироблення нової інформації, загальної для людей, що спілкуються, і яка породжує їхню спільність. І хоча спілкування – перший вид діяльності, яким опановує людина в онтогенезі, по-справжньому володіють культурою й наукою спілкування одиниці.

Можна виділити чотири основних рівні спілкування [158].

1. Рівень маніпулювання, коли один суб'єкт розглядає іншого як засіб або перешкоду стосовно проекту своєї діяльності.
2. Рівень «рефлексивної гри», коли у процесі своєї діяльності людина враховує «контрпроект» іншого суб'єкта, але не визнає за ним самооцінки і прагне до «виграшу», до реалізації свого проекту.
3. Рівень правового спілкування, коли суб'єкти визнають право на існування проектів діяльності один одного й намагаються погодити їх хоча б зовні.
4. Рівень морального спілкування, коли суб'єкти внутрішньо приймають загальний проект взаємної діяльності.

Прагнення й уміння спілкуватися на вищому, четвертому, рівні може характеризувати ступінь професіоналізму інженера зі знань. Видобування знань – це особливий вид спілкування, який можна віднести до духовно-інформаційного типу. Відповідно до роботи [74] спілкування ділиться на матеріально-практичне; духовно-інформаційне; практично-духовне. При цьому інформаційний аспект спілкування для інженера зі знань із прагматичної точки зору найважливіший.

Відомо, що втрати інформації під час розмовного спілкування великі (рис. 5.6).

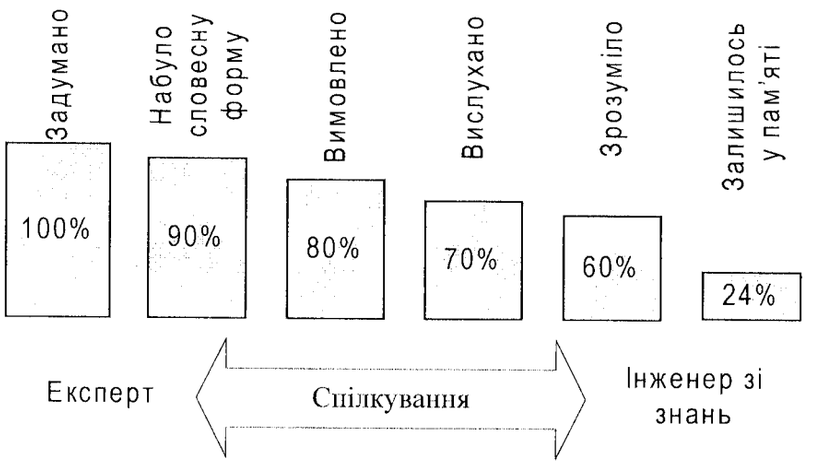


Рис. 5.6. Втрати інформації при розмовному спілкуванні.

У зв'язку із цим розглянемо проблему збільшення інформативності спілкування аналітика й експерта за рахунок використання психологічних знань.

Можна виділити такі структурні компоненти моделі спілкування під час видобування знань:

- учасники спілкування (партнери);
- засоби спілкування (процедура);
- предмет спілкування (знання).

Відповідно до цієї структури виділимо три «шари» психологічних проблем, які виникають під час видобування знань (рис. 5.7):

А1 – {Sll, S12, S13 } – {контактний, процедурний, когнітивний}.

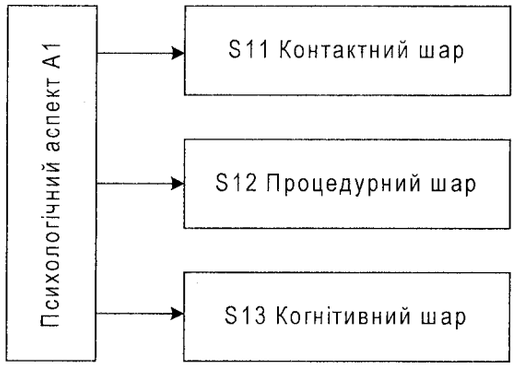


Рис. 5.7. Психологічний аспект видобування знань.

Контактний шар (S11)

Практично всі психологи відзначають, що на будь-який колективний процес впливає атмосфера, що виникає в групі учасників. Існують експерименти, результати яких незаперечно показують, що часто дружня атмосфера в колективі більше впливає на результат, ніж індивідуальні здібності окремих членів групи [125]. Особливо важливо, щоб у колективі розроблювачів складалися кооперативні, а не конкурентні відносини. Для кооперації характерна атмосфера співробітництва, взаємодопомоги, зацікавленості в успіхах один одного, тобто вищий рівень морального спілкування, а для відносин конкурентного типу – атмосфера індивідуалізму й міжособистісного суперництва (нижчий рівень спілкування).

Прогнозувати зараз сумісність у спілкуванні зі 100%-ю гарантією неможливо. Однак можна виділити ряд факторів і риси особистості, характеру й інших особливостей учасників спілкування, які, безсумнівно, впливають на ефективність процедури.

Розроблення проблематики контактного шару дозволило виявити наступні параметри партнерів, що впливають на результати процедури видобування знань:

S11 = {s11_i} = (стать, вік, особистість, темперамент, мотивація та ін.), частина з яких згодом увійшли у формування моделі користувача. Значення параметрів статі (s11_1) і віку (s11_2) хоча й впливають на ефективність контакту, але не є критичними. У літературі [69] відзначається, що гарні результати дають гетерогенні пари (чоловік/жінка) і співвідношення: 20 > (Ве - Ва) > 5, де Ве – вік експерта; Ва – вік аналітика.

Під особистістю (s11_3) зазвичай розуміється стійка система психологічних рис, що характеризує індивідуальність людини. Компоненти (s11_3), що рекомендуються, досліджені в роботі [39] і доповнені якостями з посібника для журналістів у роботі [303]. (s11_3) = (доброзичливість, аналітичність, гарна пам'ять, увага, спостережливість, уява, вразливість, більша зібраність, наполегливість, товариськість, спритність).

Із часів Галена і Гіпократа, що виділили чотири класичних типи темпераменту (s11_4), увійшли в наукову термінологію поняття (s11_4) = (холерик, сангвінік, меланхолік, флегматик). Відомо, що флегматики й меланхоліки повільніше засвоюють інформацію [109]. І для забезпечення психологічного контакту з ними не слід задавати бесіді занадто швидкий темп, квапити їх з відповіддю. Зате вони набагато краще засвоюють нове, на відміну від холериків, для яких властиве поверхневе засвоювання інформації. Останніх варто спеціально наводити на міркування й рефлексію. У меланхоліків часто

занижена самооцінка, вони сором'язливі й у бесіді їх треба підбадьорювати. Отже, найуспішнішими в рамках шару S11 є сангвініки й холерики. На ефективність колективного рішення завдань впливає також і мотивація (s11_5), тобто прагнення до успіху. Інженер зі знань залежно від умов розроблення повинен вишукувати різноманітні стимули для експертів (включаючи й матеріальні). Експерт передає аналітику один з найдорожчих у світі продуктів – знання. І якщо одні люди діляться досвідом добровільно та із задоволенням, то інші досить неохоче відкривають свої професійні таємниці. Іноді корисно розбудити в експерті дух суперництва, конкуренції (не порушуючи, звичайно, обстановки кооперативності в колективі).

Процедурний шар (S12)

Параметри процедурного шару S12 описують безпосередньо процес здійснення процедури видобування знань. Фактично це професійні параметри:

S12 = {s12_i} = {ситуація спілкування (місце, час, тривалість); устаткування (допоміжні засоби, освітленість, меблі); професійні прийоми (темп, стиль, методи тощо)}.

Інженер зі знань, що успішно опанував науку встановлення атмосфери довіри й взаєморозуміння з експертом (контактний шар – S11), повинен ще зуміти скористатися сприятливим впливом цієї атмосфери. Проблема процедурного шару стосується здійснення самої процедури видобування знань. Тут мало проникливості й чарівності, корисних для рішення проблеми контакту, тут необхідні професійні знання.

Зупинимося на загальних закономірностях здійснення процедури. s12_1 – ситуація спілкування визначається наступними компонентами:

- s12_1_1 – місце здійснення сеансів;
- s12_1_2 – тривалість здійснення сеансів;
- s12_1_3 – час здійснення сеансів.

Розмовляти з експертом найкраще в невеликому приміщенні наодинці (s12_1_1: місце), оскільки сторонні люди порушують довірчість бесіди й можуть породити ефект «фасаду». Робоче місце експерта є не найоптимальнішим, тому що його можуть відволікати телефонні дзвінки, співробітники та ін. Атмосфера замкнутого простору й самітності позитивно впливає на ефективність.

Американський психолог І.Атватер вважає, що для ділового спілкування найбільш сприятлива дистанція від 1,2 до 3 метрів [303]. Мінімально «комфортною» відстанню можна вважати 0,7-0,8 метра. Реконструкція власних міркувань – трудомісткий процес, і тому тривалість одного сеансу (s12_1_2: тривалість) зазвичай не перевищує 1,5-2 годин. Ці дві години краще вибрати в першій половині дня, наприклад, з 10 до 12 години, якщо експерт типу «жайворонок» (в12_1_3: час). Відомо, що взаємна стомленість партнерів під час бесіди настає через 20-25 хвилин [130], тому в сеансі потрібні паузи.

s12_2_2 (устаткування) включає:

- s12_2_1 – допоміжні засоби;
- s12_2_2 – освітленість;
- s12_2_3 – меблі.

Допоміжні засоби (s12_2_1):

- засоби для збільшення ефективності самого процесу видобування знань;
- засоби для протоколювання результатів.

До засобів для збільшення ефективності процесу видобування знань перш за все відноситься наочний матеріал. Незалежно від методу видобування, обраного в конкретній ситуації, його реалізація можлива різними способами. Наприклад, можна враховувати наступний фактор: широко відомо, що людей, які займаються інтелектуальною діяльністю, можна віднести до художнього або розумового типу. Терміни тут умовні й не мають відношення до тої діяльності, що традиційно називають художньою або розумовою. Важливо, що, визначивши тип експерта, інженер зі знань може плодотворніше використовувати будь-який з методів видобування, знаючи, що люди художнього типу легше сприймають зорову інформацію у формі малюнків, графіків, діаграм, тому що ця інформація сприймається через першу сигнальну систему. І навпаки, експерти розумового типу краще розуміють мову формул і текстову інформацію. При цьому враховується факт, що більшу частину інформації людина одержує від зору. Пораду користуватися активніше наочним матеріалом з роботи [56] можна вважати універсальною. Такі методи, як вільний діалог та ігри, надають широкі можливості використання слайдів, креслень, малюнків. Для протоколювання результатів у цей час використовуються такі способи:

- запис на папір безпосередньо в ході бесіди (недоліки – це часто заважає бесіді, крім того, важко встигнути записати все, навіть при наявності навиків стенографії);
- магнітофонний запис (диктофон), що допомагає аналітикові проаналізувати весь хід сеансу й свої помилки (недолік – може сковувати експерта);
- запам'ятовування з наступним записом після бесіди (недолік – прийнятий тільки для аналітиків із блискучою пам'яттю).

Найпоширенішим способом на сьогодні є перший. При цьому найбільша небезпека тут – втрата знань, оскільки будь-який запис відповідей – це вже інтерпретація, тобто привнесення суб'єктивного розуміння предмета. Значення параметрів освітленості (s12_2_2) і меблів (s12_2_3) очевидні й пов'язані із впливом зовнішніх факторів на експерта.

s12_3 – професійні прийоми аналітика включають, зокрема:

- s12_3_1 – темп;
- s12_3_2 – стиль;
- s12_3_3 – методи.

Урахування індивідуального темпу (s12_3_1) і стилю (s12_3_2) експерта дозволяє аналітикові знизити напруженість процедури видобування знань. Типовою помилкою є нав'язування власного темпу й стилю. На успішність також впливає довжина фраз, які вимовляє інженер зі знань. Цей факт був установлений американськими вченими – лінгвістом Інгве та психологом Міллером під час проведення дослідження про причини низької засвоюваності команд на Військово-морському флоті США [245]. Причина була в довжині команд. Виявилося, що людина найкраще сприймає речення глибиною (або довжиною) 7 ± 2 слова. Це число (7 ± 2) одержало назву число Інгве-Міллера. Можна вважати його мірою «розмовності» мовлення. Досвідчені лектори використовують у лекції в основному короткі фрази, зменшуючи втрату інформації з 20-30% (у поганих лекторів) до 3-4% [54]. Більша частина інформації надходить до інженера зі знань у формі пропозицій природною мовою. Однак зовнішня мова експерта є відтворенням його внутрішньої мови (мислення), яка значно багатша і образніша.

При цьому для передаванні цієї внутрішньої мови експерт використовує й невербальні засоби, такі як інтонація, міміка, жести. Досвідчений інженер зі знань намагається по можливості записувати у протоколи (у формі ремарок) цю додаткову інформацію.

У цілому, невербальний компонент стилю спілкування важливий і для проблем контактного шару під час встановлення контакту, коли по окремих жестах і виразу обличчя експерта інженер зі знань може встановити границю можливої «дружності» спілкування.

Значення параметра методів (s12_3_3) докладно розглянуто в наступному розділі, виходячи з позиції, що метод повинен підходити експерту як «ключ до замка».

Когнітивний шар (S13)

Когнітивні (від англ. cognition – пізнання) науки досліджують пізнавальні процеси людини з позицій можливості їхнього моделювання (психологія, нейрофізіологія, ергономіка, інженерія знань). Найменш досліджені на сьогодні проблеми когнітивного шару S13, пов'язані з вивченням семантичного простору пам'яті експерта й реконструкцією його понятійної структури і моделі міркувань.

Основними факторами, що впливають на когнітивну адекватність, будуть:

S13 = {s13_i} = {когнітивний стиль, семантична репрезентативність поля знань і концептуальної моделі}.

Під когнітивним стилем (s13_1) людини розуміється сукупність критеріїв переваги при рішенні завдань і пізнанні світу, специфічна для кожної людини. Когнітивний стиль визначає не стільки ефективність діяльності, скільки спосіб досягнення результату [4]. Це спосіб пізнання, що дозволяє людям з різними здібностями досягати однакових результатів у діяльності. Це система засобів та індивідуальних прийомів, до яких звертається людина для організації своєї діяльності. Інженерові зі знань корисно вивчити і прогнозувати свій когнітивний стиль, а також стиль експерта. Особливо важливі такі характеристики когнітивного стилю, як:

- s13_1_1 – (полезалежність – поленезалежність);
- s13_1_2 – (імпульсивність – рефлективність (рефлексивність));
- s13_1_3 – (ригідність – гнучкість);
- s13_1_4 – (когнітивна еквівалентність).

s13_1_1. Поленезалежність дозволяє людині акцентувати увагу лише на тих аспектах проблеми, які необхідні для рішення конкретного завдання, і вміти відкидати все зайве, тобто не залежати від фону або оточення завдання, впливу шумового поля. Ця характеристика корелює з такими рисами особи, як невербальний інтелект, аналітичність мислення, здатність до розуміння суті. Очевидно, що крім того, що самому аналітикові необхідно мати високе значення параметра s13_1_1, поленезалежний експерт – це теж бажаний чинник. Однак доводиться враховувати, що більше мають потребу в спілкуванні полезалежні люди, а тому вони й більш контактні [132].

Особливо корисні для спілкування гетерогенні (змішані) пари, наприклад, «полезалежний – поленезалежний» [69]. У літературі описані різні експерименти, що моделюють спілкування, що вимагає розуміння і спільної діяльності. Найуспішнішим у розумінні при випробуванні виявилися поле-незалежні (92 % успіху), для порівняння полезалежні давали 56 % успіху [86].

Для спільної професійної діяльності важлива також гнучкість когнітивної організації, що пов'язана з поленезалежністю. Отже, більшу здатність до адекватного розуміння

партнера демонструють суб'єкти з високою психологічною диференціацією, тобто поле незалежністю. Поленезалежність є однією з характерних професійних рис когнітивного стилю найбільш кваліфікованих інженерів зі знань. За деякими результатами [4] чоловіки поленезалежніші, ніж жінки.

s13_1_2. Під імпульсивністю розуміється швидке прийняття рішення (часто без його достатнього обґрунтування), а під рефлексивністю – схильність до міркувань. Рефлексивність за експериментальними даними корелює зі здатністю до формування понять і продуктивністю стратегій рішення логічних завдань [86]. Отже, і інженерові зі знань, і експертові бажано бути рефлексивним, хоча власний стиль змінюється лише частково і з більшим напруженням.

s13_1_3. Ригідність – стан, при якому знижена здатність до переключення психічних процесів і пристосування до умов середовища. Очевидно, що якщо експерт ще може собі дозволити ригідність (що характерно для фахівців, особливо старшого віку, які довго працюють над однією проблемою), то для інженера зі знань ця характеристика когнітивного стилю явно протипоказана. Збільшення ригідності з віком відзначається багатьма психологами [86].

s13_1_4. Когнітивна еквівалентність характеризує здатність людини до розрізнення понять і розбивання їх на класи та підкласи. Чим вужчий діапазон когнітивної еквівалентності, тим тоншу класифікацію здатний здійснити індивід, тим більшу кількість ознак понять він може виділити. Зазвичай у жінок діапазон когнітивної еквівалентності вужче, ніж у чоловіків. Семантична репрезентативність (s13_2) має на увазі підхід, що виключає традиційне нав'язування експертові якоїсь моделі подання знань(наприклад, продукційної або фреймової), і змушує інженера зі знань послідовно відтворювати модель світу експерта, використовуючи як неформальні методи, так і математичний апарат, наприклад, багатомірне шкалювання (див. наступний розділ). Проблема семантичної репрезентативності орієнтована на досягнення когнітивної адекватності поля знань і концептуальної моделі. На певний момент вона може бути сформульована як проблема «зіпсованого телефону» [43] – можливі трансформації та втрати в ланцюзі передавання інформації:

(Q_k : предметна область або реальний світ) →

[I_i : інтерпретація і-го експерта] →

(Mg_i : модель світу експерта) →

[V_i : вербалізація моделі світу експерта] →

($T_i \varepsilon_\tau C_j$: вербальні і невербальні повідомлення і-го експерта j-му аналітику) →

[I_i : їх інтерпретація j-им аналітиком] →

(Mg_j: модель світу j-го аналітика) →

[K_j : кодування при формуванні поля знань з послідовною структуризацією в концептуальній моделі].

Круглі дужки визначають поняття, квадратні – процеси.

5.3.2. Лінгвістичний аспект

Лінгвістичний (А2) аспект стосується досліджень мовних проблем, тому що мова – це основний засіб спілкування у процесі видобування знань. Область розроблення природно-мовних інтерфейсів і весь спектр проблем, пов'язаних з нею – лексичних, синтаксичних, семантичних, прагматичних тощо [31; 111; 144], у цій книзі не розглядається.

В інженерії знань можна виділити три шари лінгвістичних проблем (рис. 5.8):
A2 = {S21, S22, S23} – {«загальний код», понятійна структура, слова}.

Рис. 5.8. Лінгвістичний аспект видобування знань.

«Загальний код» (S21)

«Загальний код» вирішує проблему мовних ножиць між професійною термінологією експерта і повсякденною літературною мовою інженера зі знань і включає наступні компоненти:

S21 = {s21_i} = {загальнонаукова термінологія; спеціальні поняття із професійної літератури; елементи побутової мови; неологізми, сформовані за час спільної роботи; професійний жаргон і ін.}.

Деталізація схеми спілкування (див. рис. 5.5) дозволяє зобразити засоби спілкування як два потоки [54], у яких нас цікавлять компоненти V1 і V2 – мови, якими говорять аналітик і експерт ($V1^1$, $V2^1$ – невербальні компоненти). Розходження мов V1 і V2 обумовлює «мовний бар'єр» або «мовні ножиці» у спілкуванні інженера зі знань і експерта.

Ці дві мови є відображенням «внутрішньої мови» експерта й аналітика, оскільки більшість психологів і лінгвістів вважають, що мова – це основний засіб мислення поряд з іншими знаковими системами «внутрішнього користування» (універсальний семантичний код – УСК [113], мови «змісту» [116], концептуальні мови [181] і ін.). *Мова аналітика* V1 складається із трьох компонентів:

- s21_1 – загальнонаукової термінології з її «теоретичним багажем;
- s21_2 – термінів предметної області, які аналітик почерпнув зі спеціальної літератури в період підготовки;
- s21_3 – побутової розмовної мови, якою користується аналітик.

Мова експерта V2 включає:

- s21_1 – загальнонаукову термінологію;
- s21_2 – спеціальну термінологію, прийняту в предметній області;
- s21_3 – побутову мову;
- s21_4 – неологізми, створені експертом за час роботи, тобто його професіональний жаргон.

Якщо вважати, що побутова й загальнонаукова мови у двох учасників спілкування приблизно збігаються (хоча реально обсяг другого компонента в експерта істотно

більший), то деяка загальна мова або код, який необхідно виробити партнерам для успішної взаємодії, буде складатися з потоків, зображених на рис. 5.9.

Надалі цей загальний код перетвориться в деяку понятійну (семантичну) мережу, що є прообразом поля знань предметної області. Вироблення загального коду починається з виписуванням аналітиком всіх термінів, уживаних експертом, і уточнення їхнього змісту. Фактично це є складання словника предметної області. Потім випливає групування термінів і вибір синонімів (слів, що означають те саме). Розробка загального коду закінчується складанням словника термінів предметної області з попереднім угрупованням їх за змістом, тобто за понятійною близькістю (це вже перший крок структурування знань).

На цьому етапі аналітик повинен уважно віднестись до всіх спеціальних термінів, намагаючись максимально вникнути в суть розв'язуваних проблем і термінологію. Освоєння аналітиком мови предметної області – перший рубіж на підступах до створення адекватної бази знань.

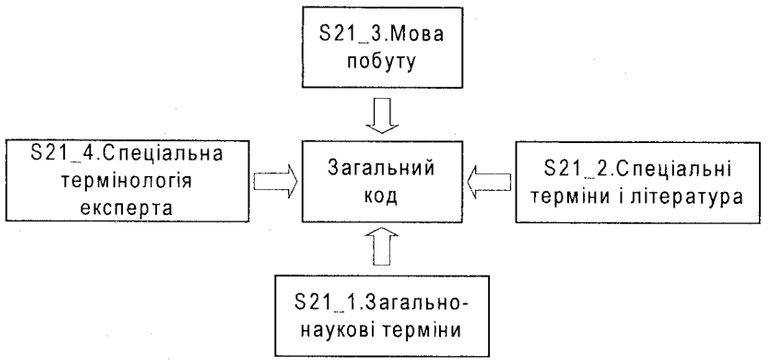


Рис. 5.9. Структура загального коду.

Рис. 5.9 дає уявлення про процес неоднозначності інтерпретації термінів двома фахівцями. У семіотиці, науці про знакові системи, проблема інтерпретації є однією із центральних. Інтерпретація зв'язує «знак» і «означуваний предмет». Тільки в інтерпретації знак одержує зміст.

Увага до лінгвістичного аспекту проблеми видобування знань сприяє зближенню між собою двох образів.

Отже, шар S21 включає вивчення й керування процесом розроблення спеціальної проміжної мови, необхідної для взаємодії інженера зі знань і експерта.

Понятійна структура (S22)

Проблеми формування понятійної структури надають наступний шар S22 лінгвістичного аспекту проблеми видобування знань. Особливості формування понятійної структури обумовлені встановленим постулатом когнітивної психології про взаємозв'язок понять у пам'яті людини й наявності семантичної мережі, що поєднує окремі терміни у фрагменти, фрагменти у сценарії й тощо. Побудова ієрархічної мережі понять, так званої «піраміди знань», – найважливіша ланка в проектуванні інтелектуальних систем.

Більшість фахівців зі штучного інтелекту й когнітивної психології вважають, що основна особливість природного інтелекту і пам'яті зокрема – це зв'язаність всіх понять у деяку мережу. Тому для розроблення бази знань потрібний не словник, а «енциклопедія» [182], в якій всі терміни пояснені у словникових статтях з посиланнями на інші терміни.

Отже, лінгвістична робота інженера зі знань на цьому шарі проблем полягає в побудові таких зв'язаних фрагментів за допомогою «зшивання» термінів. Фактично ця робота є підготовкою до етапу концептуалізації, де це «шиття» (по Шенку – КІП, концептуальна організація пам'яті [183]) набуває деякого закінченого вигляду. При ретельній роботі аналітика й експерта в понятійних структурах починає проглядатися ієрархія понять, докладно про яку мова йде нижче. Такі структури мають найважливіші гносеологічне і дидактичне значення й останнім часом для них використовується спеціальний термін – онтології. Відзначимо, що ця ієрархічна організація добре узгоджується з теорією універсального предметного коду (УПК) [66], відповідно до якої при мисленні використовуються не мовні конструкції, а їхні коди у формі деяких абстракцій, що в загальному узгоджуються з результатами когнітивної психології [26]. Ієрархія абстракцій – це глобальна схема, що може бути покладена в основу концептуального аналізу структури знань будь-якої предметної області. Лінгвістичний еквівалент ієрархії – ієрархія понять, яку необхідно побудувати в понятійній структурі, сформованій інженером зі знань (рис. 5.10).

Підкреслимо, що робота зі складання словника та понятійної структури вимагає лінгвістичного «чуття», легкості маніпулювання термінами та багатого словникового запасу інженера зі знань, тому що найчастіше аналітик змушений самостійно розробляти словник ознак. Чим багатший та виразніший виходить загальний код, тим більш повніша база знань.

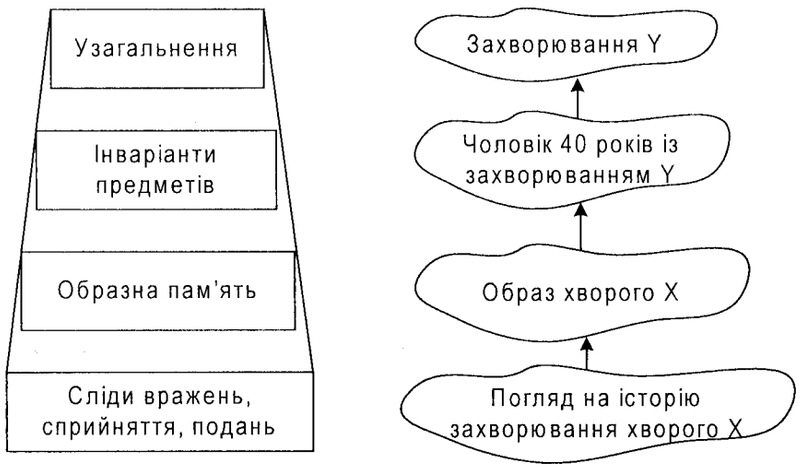


Рис. 5.10. Приклад ієрархії.

Аналітик змушений весь час пам'ятати про труднощі передавання образів і подань у вербальній формі. Корисними тут виявляються властивості багатозначності слів природної мови. Часто інженерові зі знань доводиться підказувати слова і вирази експертові, і такі нові лексичні конструкції виявляються корисними.

Здатність до словесної інтерпретації залежить і від статі аналітика (параметр s11_1). Встановлено, що традиційно жінки надають більшу перевагу невербальним компонентам спілкування, а вербальні мають більший алфавіт ознак. І взагалі, існують статеві

розходження сприйняття не тільки в побутовій сфері, але й у професійній. Отже, в експерта-чоловіка й в експерта-жінки можуть істотно відрізнятися алфавіти для вербалізації ознак сприйнятих об'єктів.

Словник користувача (S23)

Лінгвістичні результати, співвіднесені до шарів загального коду й понятійної структури, спрямовані на створення адекватної бази знань. Однак часто професійний рівень кінцевого користувача не дозволяє йому застосувати спеціальну мову предметної області в повному обсязі.

Для розроблювачів-початківців несподіваними є проблеми формування окремого словника для створення дружнього інтерфейсу з користувачем ЕС, досліджувані в шарі S23. Необхідні спеціальні прийоми, що збільшують «прозорість» і доступність системи. Для розроблення користувальницького інтерфейсу потрібне додаткове доопрацювання словника загального коду з виправленням на доступність і «прозорість» системи.

Так, під час розроблення експертної системи з психодіагностики АВТАНТЕСТ [44] довелося розробити два словники термінів – один для психологів-професіоналів, другий – для неспеціалістів (клієнтів). Оскільки результат психодіагностичного тестування завжди цікавий клієнту, йому видається лістинг із психологічним висновком загальнолітературною мовою без уживання спеціальних термінів. Цікаво, що при впровадженні системи використовувався, в основному, цей другий словник; навіть професійні психологи віддавали перевагу текстам з повсякденною мовою.

5.3.3. Гносеологічний аспект видобування знань

Гносеологія – це розділ філософії, пов'язаний з теорією пізнання, або теорією відображення дійсності у свідомості людини. Гносеологічний аспект (А3) видобування знань поєднує методологічні проблеми одержання нового наукового знання, оскільки при створенні БЗ експерт часто вперше формулює деякі закономірності, які до цього моменту складали його особистий досвід. Інженерія знань як наука, якщо можна так висловитися, двічі гносеологічна – спочатку дійсність відбивається у свідомості експерта (М1), а потім діяльність і досвід експерта інтерпретуються свідомістю інженера зі знань (М2), що служить вже основою для побудови третьої інтерпретації (Р) – поля знань інтелектуальної системи. Процес пізнання, за суттю, скерований на створення внутрішньої репрезентації навколишнього світу у свідомості людини.

Якщо описати процеси І2 і І3 в термінології, введеній на початку розділу, то ми маємо справу з перетворенням експертного знання і теоретичного (книжного) досвіду Z1 в полі знань Z2, яке є матеріалізацією моделі світу М2 інженера зі знань.

У процесі видобування знань аналітика, в основному, цікавить компонент Z1, пов'язаний з неканонічними індивідуальними знаннями експертів, оскільки предметні області, що вимагають саме такого типу знань, вважаються найбільш сприйнятливими до впровадження інтелектуальних систем. Ці області зазвичай називають емпіричними, оскільки в них накопичений великий обсяг окремих емпіричних фактів і спостережень, тоді як їх теоретичне узагальнення – питання майбутнього.

Якщо вважати, що інженер зі знань витягає тільки фрагмент Z1', тобто частину із системи знань експерта Z1, то його завдання, по-перше, намагатися, щоб структура Z1' відповідала Z1, і, по-друге, щоб Z1' якомога повніше відображала Z1.

Пізнання часто супроводжується створенням нових понять і теорій. Іноді експерт породжує нові знання прямо під час бесіди з аналітиком. Така генерація знань корисна

і самому експертові, який до того моменту міг не усвідомлювати ряд співвідношень і закономірностей предметної області. Аналітикові може допомогти тут і інструментарій системної методології, що дозволяє використовувати відомі принципи логіки наукових досліджень, понятійної ієрархії науки. Ця методологія змушує його завжди прагнути за часткою побачити загальне, тобто будувати ланцюжки.

Гносеологічний ланцюжок: факт – узагальнений факт – емпіричний закон – теоретичний закон.

Не завжди вдається дійти до останньої ланки цього ланцюжка, але вже саме прагнення до руху буває надзвичай плідним. Такий підхід повністю узгоджується зі структурою самого знання, яке має два рівні:

1) емпіричний (спостереження, явища);
2) теоретичний (закони, абстракції, узагальнення).

Але теорія – це не лише стрункa система узагальнення наукового знання, це також деякий спосіб виробництва нових знань. Основними методологічними критеріями науковості, що дозволяють вважати науковим і саме нове знання й спосіб його одержання, є [187]:

А3 = {S31, S32, S33} = (внутрішня узгодженість, системність, об'єктивність, історизм}.

Внутрішня узгодженість (S31)

Основні характеристики емпіричного знання:

S31 = {s31_i} = {модальність, суперечливість, неповнота}.

На перший погляд критерій внутрішньої узгодженості знання не відповідає реальним характеристикам, що описують знання з точки зору шару {s31_i}. Ці характеристики емпіричних знань підкреслюють його «багатоукладність» – так, часто факти не узгоджуються один з одним, визначення суперечать, критерії дифузні тощо. Аналітикові, що розуміє особливості емпіричного знання, доводиться згладжувати ці «шорсткості» емпірики.

Модальність (s31_1) знання означає можливість його існування в різних категоріях, тобто в конструкціях існування і повинності. Отже, частина закономірностей можлива, інша обов'язкова тощо. Крім того, доводиться розрізняти такі відтінки модальності, як:

- експерт знає, що...;
- експерт думає, що...;
- експерт хоче, щоб...;
- експерт вважає, що...

Можлива суперечність (s31_2) емпіричного знання – природний наслідок з основних законів діалектики, і суперечності ці не завжди повинні вирішуватися в полі знань, а навпаки, саме суперечності служать найчастіше відправною крапкою в міркуваннях експертів.

Неповнота (s31_3) знання пов'язана з неможливістю повного опису предметної області. Завдання аналітика цю неповноту обмежити певними рамками «повноти», тобто звузити границі предметної області або ввести ряд обмежень і допущень, що спрощують проблему.

Системність (S32)

Системно-структурний підхід до пізнання (введений Гегелем) орієнтує аналітика на розгляд будь-якої предметної області з позицій закономірностей системного цілого і взаємодії складових його частин. Сучасний структуралізм виходить із багаторівневої ієрархічної організації будь-якого об'єкту, тобто всі процеси і явища можна розглядати як дрібніші підмножини (ознаки, деталі) і, навпаки, будь-які об'єкти можна (і потрібно) розглядати як елементи вищих класів узагальнень. Наприклад, системний погляд на проблематику структурування знань дозволяє побачити його ієрархічну організацію. Докладніше про це нижче.

Об'єктивність (S33)

Процес пізнання глибоко суб'єктивний, тобто він істотно залежить від особливостей самого суб'єкта, що пізнає. «Факти існують для одного ока й відсутні для іншого» (Віпшер). Отже, суб'єктивність починається вже з опису фактів і збільшується в міру поглиблення ідеалізації об'єктів.

Коректніше говорити про глибину розуміння, ніж про об'єктивність знання. Розуміння – це співтворчість, процес тлумачення об'єкту з погляду суб'єкта. Це складний і неоднозначний процес, що здійснюється в глибинах людської свідомості і вимагає мобілізації всіх інтелектуальних і емоційних здібностей людини. Всі свої зусилля аналітик повинен зосередити на розумінні проблеми.

У психології відомий результат [27], який підтверджує факт, що люди, які швидко і успішно вирішують інтелектуальні завдання, більшу частину часу витрачають на розуміння його, тоді як погані вирішувачі швидко розпочинають до пошук рішення і найчастіше не можуть його знайти.

Історизм (S34)

Цей критерій пов'язаний з розвитком. Пізнання сьогодення – є пізнання породжувача його минулого. І хоча більшість експертних систем дають «горизонтальний» зріз знань – без урахування часу (у статиці), інженер зі знань повинен завжди розглядати процеси з урахуванням тимчасових змін – як зв'язок з минулим, так і зв'язок з майбутнім. Наприклад, структура поля знань і база знань повинні допускати відлагодження і корекцію як у період розроблення, так і під час експлуатації ЕС.

Розглянувши основні критерії науковості пізнання, спробуємо тепер описати його структуру. Методологічна структура пізнання може бути подана як деяка послідовність етапів. Параметри {83і} органічно вписуються в цю структуру пізнання, яка може подаватися як послідовність етапів, описаних далі з позицій інженера зі знань:

- E_1: опис і узагальнення фактів;
- E_2: встановлення логічних і математичних зв'язків, дедукція й індукція законів;
- E_3: побудова моделі, що ідеалізується;
- E_4: пояснення і прогноз явищ.

E_1. Опис і узагальнення фактів

Ретельність і повнота ведення протоколів під час процесу видобування і пунктуальна «домашня робота» над ними – ось гарантія продуктивного першого етапу пізнання та матеріал для опису й узагальнення фактів.

На практиці виявляється важко дотримуватися принципів об'єктивності й системності, описаних вище. Найчастіше на цьому етапі факти просто збирають і як би кидають у «загальний мішок»; досвідчений інженер зі знань часто відразу намагається знайти «поличку» або «скриньку» для кожного факту, тим самим приховано готуючись до етапу концептуалізації.

E_2. Встановлення зв'язків і закономірностей

У пам'яті експерта всі поняття пов'язані й закономірності встановлені, хоча часто і неявно завдання інженера – виявити каркас висновків експерта. Реконструюючи міркування експерта, інженер зі знань може спиратися на дві найпопулярніші теорії мислення – логічну й асоціативну. При цьому, якщо логічна теорія завдяки гарячим шанувальникам в особі математиків широко цитується і всіляко експлуатується в роботах зі штучного інтелекту, то друга, асоціативна, набагато менш відома і популярна, хоча має також стародавнє коріння. Так, Р. Фейнман у своїх «Лекціях з фізики» відзначає, що у фізиці, як і раніше, переважальним є вавілонський, а не грецький метод побудови знань. Відомо, що давньосхідні математики вміли робити складні обчислення, але формули їх не були логічно зв'язані. Навпаки, грецька математика дедуктивна (наприклад, «Початок» Евкліда).

Традиційна логіка формує критерії, які гарантують точність, валідність, несуперечність загальних понять, міркувань і висновків. Її основи закладені ще в «Органоне» Арістотеля в IV ст. до н. е. Великий внесок у розвиток логіки зробив Джон Стюарт Мілль (1806-1873).

Інженер зі знань і сам використовує операції традиційної логіки, і виділяє їх у схемі міркувань експерта. Це такі операції:

- визначення;
- порівняння і розрізнення;
- аналіз;
- абстрагування;
- узагальнення;
- класифікація;
- категоризація;
- виникнення думок;
- висновок;
- складання силогізмів тощо.

Проте краса і стрункість логічної теорії не повинні затуляти того, що людина рідко мислить у категоріях математичної логіки [151]. Теорія асоціацій подає мислення як ланцюжок ідей, зв'язаних загальними поняттями. Основними операціями такого мислення є:

- асоціації, придбані на основі різних зв'язків;
- пригадування минулого досвіду; проби і помилки з випадковими успіхами;
- звичні («автоматичні») реакції та ін.

Проте ці дві теорії не вичерпують усього різноманіття психологічних шкіл. Великий інтерес для інженерії знань може становити гештальт-психологія. Одним з її засно-

ників є видатний німецький психолог М. Вертгеймер (1880-1943). Під ґештальтом (нім. Gestalt) розуміється принцип цілісності сприйняття як основа мислення. Ґештальт-психологи прагнуть у всьому виділяти якийсь цілісний образ або структуру як базис для розуміння процесів і явищ навколишнього світу. Ця теорія близька до теорії фреймів і об'єктного підходу й спрямована на збагнення глибинного знання, що характеризується стабільністю і симетрією. При цьому важливий так званий «центр ситуації», щодо якого розвивається знання про предметну область.

Для інженера зі знань це означає, що, виявляючи різні фрагменти знань, він не повинен забувати про головний, про ґештальте фрагмента, який впливає на решту компонентів і пов'язує їх у деяку структурну одиницю. Ґештальтом може бути якийсь головний принцип, або ідея, або гіпотеза експерта, або його віра через якісь окремі концепції. Цей принцип рідко формулюється експертом явно, він завжди так би мовити, за «кадром», і мистецтво інженера зі знань – виявити цей основний ґештальт експерта.

У ґештальт-теорії існує закон «прагнення до хорошого ґештальту», згідно з яким структури свідомості прагнуть до гармонії, зв'язності, простоти. Це близьке до стародавнього класичного принципу «бритви Оккама» – «сутності не повинні множитися без необхідності» – і формулюється як принцип прегнантності Вертгеймера [29]: «Організація поля має тенденцію бути настільки простою і зрозумілою, наскільки дозволяють задані умови». Міркування про ґештальт підводять впритул до третього етапу в структурі пізнання.

E_3. Побудова ідеалізованої моделі

Для побудови моделі, що відображає уявлення суб'єкта про предметну область, необхідна спеціалізована мова, за допомогою якої можна описувати і конструювати ті ідеалізовані моделі світу, які виникають у процесі мислення. Мова ця створюється поступово за допомогою категоріального апарата, прийнятого у відповідній предметній області, а також формально-знакових засобів математики і логіки. Для емпіричних предметних областей така мова наразі не розроблена, і поле знань, яке напівформалізованим способом опише аналітик, може бути першим кроком до створення такої мови.

Будь-яке пізнавальне відбиття містить у собі умовність, тобто спрощення та ідеалізацію. Інженерові зі знань необхідне оволодіння такими специфічними гносеологічними прийомами, як ідеалізація, огрублення, абстрагування, які дозволяють адекватно відображати в моделі реальну картину світу. Ці прийоми доводять властивості й ознаки об'єктів до меж, що дозволяють відтворювати закони дійсності в лаконічнішому вигляді (без впливу несуттєвих деталей).

На тернистому шляху пізнання перевірений діалектичний підхід виявляється кращим «поводирем». Інженер зі знань, що прагне пізнати проблемну область, має бути готовим постійно змінювати свої вже затверділі способи сприйняття й оцінювання світу і навіть відмовлятися від них. При цьому найретельніше треба перевіряти правильність думок, які здаються найочевиднішими.

E_4. Пояснення і пророкування моделей

Цей завершальний етап у структурі пізнання є одночасно і частковим критерієм істинності отриманого знання. Якщо виявлена система знань експерта повна й об'єктивна, то на її підставі можна робити прогнози й пояснювати будь-які явища з певної предметної області. Зазвичай бази знань ЕС страждають фрагментарністю і модульною незв'язаністю компонентів. Все це не дозволяє створювати дійсно інтелектуальні системи, які, рівняючись на людину, могли б передбачати нові закономірності й пояснювати випадки, не вказані в явному вигляді в базі. Виключенням тут є навчальні системи, які орієнтовані на генерацію нових знань і «прогноз».

Пропонована методологія озброює аналітика апаратом, що дозволяє уникнути традиційних помилок, що приводять до неповноти, суперечливості, фрагментарності БЗ, і вказує напрямок, в якому необхідно рухатися розробникам. І хоча на сьогодні більшість БЗ опрацьовуються лише до етапу E_3, знання повної схеми збагачує і поглиблює процес проектування.

5.4. Теоретичні аспекти структурування знань

Поділ стадій видобування і структурування знань є досить умовним, оскільки хороший інженер зі знань, вже видобуваючи знання, починає роботу зі структурування і формування поля знань, описаному в підрозділі 5.1.

Проте зараз простежується тенденція випередження технологічних засобів розроблення інтелектуальних систем стосовно їхнього теоретичного обґрунтування. Практично зараз існує прірва між блискучими, але трохи «пристарілими» математичними основами кібернетики (праці Вінера, Ешбі, Шеннона, Джорджа, Кліра, Йордона, Ляпунова, Глушкова й ін.), і сучасним поколінням інтелектуальних систем, які засновані на парадигмі опрацювання знань (експертні системи, лінгвістичні процесори, що навчають системи, тощо).

З одного боку, це пояснюється тим, що з перших кроків наука про штучний інтелект (ШІ) була спрямована на моделювання слабоформалізованих сенсових завдань, у яких не застосовується традиційний математичний апарат; з іншого боку, ШІ – це галузь інформатики і активно розвивається як промислова індустрія програмних засобів в умовах жорсткої конкуренції, де часом важливіше швидке впровадження нових ідей і підходів, ніж їхній аналіз і теоретичне пророблення.

Необхідність розроблення теоретичних основ науки про методи розроблення систем, заснованих на знаннях – інженерії знань, – обґрунтовується в роботах Поспєлова Д. А., Попова Е. В., Стефанюка В. Л., Шенка Р., Мінського М. – провідних спеціалістів в області ШІ. Перші кроки в створенні методології (роботи Осипова Г. С., Хорошевського В. Ф., Яшина А. М., Wielinga, Slagle, та ін.) фактично є піонерськими і найчастіше орієнтовані на певний клас завдань, що моделюються в рамках конкретного програмного інструментарію.

У цьому підрозділі розглянута нова методологія [38], що дозволяє здійснити стадію структуризації незалежно від подальшої програмної реалізації, спираючись на досягнення в області розроблення складних систем.

5.4.1. Історична довідка

Стадія концептуального аналізу або структуризації знань традиційно є (разом зі стадією видобування) «вузьким місцем» у життєвому циклі розроблення інтелектуальних систем [201]. Методологія структуризації близька до сучасної теорії великих систем [49] або складних систем [224], де традиційно акцент робиться на процесі проектування таких систем. Великий внесок до цієї теорії внесли класики об'єктно-орієнтованого аналізу [25].

Розроблення інтелектуальних систем з упевненістю можна віднести до цього класу завдань, оскільки вони мають основні ознаки складності (ієрархія понять, внутрішньоелементні й міжелементні зв'язки тощо). Складність проектування ІС визначається, в основному, складністю предметних областей і керування процесом розроблення, а також складністю забезпечення гнучкості кінцевого програмного продукту й опису поведінки окремих підсистем.

Серед перших прихильників досліджень з теорії систем найпомітнішими були Берталанфі [208], Раппопорт і Боулдінг [216].

Аналогічні концепції, але пов'язані не із загальносистемними дослідженнями, а інформаційними процесами, які розглядають у системах, таких як зв'язок і керування, поклали початок кібернетиці як самостійній науці [30]. Цей підхід був істотно підтриманий роботами Шеннона з математичного моделювання поняття інформації [184].

Пізніше, в 1960-х рр., було зроблено кілька спроб сформулювати і розвинути математичні теорії систем високого рівня спільності [117]. Істотний внесок до математичної теорії систем і основи структуризації внесли вітчизняні дослідники Моїсєєв Н.Я. [122], Глушков В.М. [52], Івахненко А.Г. [70], Поспєлов Д.А. [149] та інші. Системний аналіз тісно переплітається з теорією систем і включає сукупність методів, орієнтованих на дослідження і моделювання складних систем – технічних, економічних, екологічних і т. ін.

5.4.2. Ієрархічний підхід

Проектування складних систем і методи структуризації інформації традиційно використовували ієрархічний підхід [117] як методологічний прийом розчленовування формально описаної системи на рівні (або блоки, або модулі). На вищих рівнях ієрархії використовуються найменш деталізовані уявлення, що відображають тільки загальні риси і особливості проектованої системи. На наступних рівнях ступінь подрібленості зростає, при цьому система розглядається не в цілому, а окремими блоками.

У теорії САПР такий підхід називається блочно-ієрархічним (БІП) [129]. Одне з переваг БІП полягає в тому, що складне завдання великої розмірності розбивається на послідовно вирішувані групи завдань малої розмірності.

На кожному рівні вводяться свої уявлення про систему та елементи. Елемент k-го рівня є системою для рівня k–1. Просування від рівня до рівня має строгу спрямованість, обумовлену стратегією проектування – зверху вниз або знизу до верху.

Спадна концепція (top-down) декларує рух від $n = n+1$, де n – n-й рівень ієрархії понять ПО (предметної області) з подальшою деталізацією понять, що належать відповідним рівням

$$STR_{td} : P_i^n \Rightarrow P_1^{n+1},...,P_{k_i}^{n+1},$$

де n – номер рівня концепту, що породжує; i – номер концепту, що породжує;

k_i – число породжуваних концептів, сума всіх k_i по i становить загальне число концептів на рівні n+1.

Вихідна концепція (bottom-up) пропонує рух $n \Rightarrow n-1$ з послідовним узагальненням понять.

$$STR_{bd} : P_i^n,...,P_{k_i}^n \Rightarrow P_i^{n-1},$$

де n – номер рівня концептів, що породжують; i – номер породжуваного концепту;

k_i – число концептів, що породжують, сума всіх k_i по i становить загальне число концептів на рівні n).

Підставою для припинення агрегації та дезагрегування є повне використання словника термінів, яким користується експерт, при цьому число рівнів є значущим чинником успішності структуризації (див. «вербальні звіти» у наступному розділі).

5.4.3. Традиційні методології структуризації

Наявні підходи до проектування складних систем можна розділити на два великі класи:

- структурний (системний) підхід або аналіз, заснований на ідеї алгоритмічної декомпозиції, де кожний модуль системи виконує один з найважливіших етапів загального процесу;
- об'єктний підхід, пов'язаний з декомпозицією і виділенням не процесів, а об'єктів, при цьому кожен об'єкт розглядається як екземпляр певного класу.

У структурному аналізі [220] розроблена велика кількість виразних засобів для проектування, зокрема графічних [25]: діаграми потоків даних (DFD – data-flow diagrams), структуровані словники (тезауруси), мови специфікації систем, таблиці рішень, стрілочні діаграми «об'єкт-зв'язок» (ERD – entity-relationship diagrams), діаграми переходів (станів), дерева цілей, блок-схеми алгоритмів (у нотації Нассі-Шнейдермана, Гамільтона-Зельдіна, Фестля та ін.), засоби керування проектом (PERT-Діаграми, діаграми Ганта та ін.), моделі оточення.

Множинність засобів і їх деяка надмірність пояснюються тим, що кожна предметна область, використовуючи структурний підхід як універсальний засіб моделювання, вводила свою термінологію, найбільш відповідну для відображення специфіки конкретної проблеми. Оскільки інженерія знань має справу із широким класом ПО (це «м'які» ПО), постає завдання розроблення достатньо універсальної мови структуризації.

Об'єктний (об'єктно-орієнтований) підхід (ООП), що виник як технологія програмування більших програмних продуктів, заснований на таких основних елементарних поняттях [25]: об'єкти, класи як об'єкти, зв'язані спільністю структури і властивостей, і класифікації як засоби впорядкування знань; ієрархії зі спадкуванням властивостей; інкапсуляції як засоби обмеження доступу; методи і поліморфізм для визначення функцій і відносин.

ООП має свою систему умовних позначень і пропонує багатий набір логічних і фізичних моделей для проектування систем високого ступеня складності, при цьому ці системи добре структуровані, що породжує легкість їхньої модифікації. Вперше принцип ООП розроблений у 1979 р. [265], а потім розвинений в роботах [174].

Широке розповсюдження об'єктно-орієнтованих мов програмування C++, CLOS, Smalltalk і ін. успішно демонструє життєздатність і перспективність цього підходу.

5.4.4. Об'єктно-структурний підхід (ОСП)

Можна запропонувати як базисну парадигму методології структурного аналізу знань і формування поля знань Pz узагальнений об'єктно-структурний підхід (ОСП), послідовно розроблений від математичного обґрунтування до технології і програмної реалізації. Основні постулати цієї парадигми запозичені з ООП і розширені.

1. Системність (взаємозв'язок між поняттями).

2. Абстрагування (виявлення істотних характеристик поняття, які відрізняють його від інших).
3. Ієрархія (ранжирування на впорядковані системи абстракцій).
4. Типізація (виділення класів понять із частковим спадкуванням властивостей у підкласах).
5. Модульність (розбиття завдання на підзадачі або «можливі світи»).
6. Наочність і простота нотації.

Використання п'ятого постулату ОСП в інженерії знань дозволяє будувати глобальні БЗ із можливістю виділити локальні завдання за допомогою горизонтальних і вертикальних перетинів на окремі модулі простору-опису предметної області.

Шостий постулат внесений у список останнім, але не за значущістю. В інженерії знань формування Pz традиційно є критичною точкою, оскільки створювана неформальна модель предметної області повинна бути гранично зрозумілою і лаконічною. Традиційно мовою інженерії знань були діаграми, таблиці та інші графічні елементи, що сприяють наочності уявлень. Саме тому пропонований у цій роботі підхід до мови пов'язаний з можливою візуалізацією процесу проектування.

ОСП дозволяє наочно і компактно відобразити об'єкти і відносини предметної області на основі використання шести постулатів.

Об'єктно-структурний підхід має на увазі інтегроване використання сформульованих вище постулатів від першої до останньої стадій розроблення БЗ інтелектуальних і навчальних систем. На основі ОСП пропонується алгоритм об'єктно-структурного аналізу (ОСА) предметної області, що дозволяє оптимізувати і упорядкувати досить розмиті процедури структуризації знань.

Стратифікація знань

Основи ОСА були запропоновані ще в роботах [37] і успішно застосовувалися при розробленні ЕС (таких як МІКРОЛЮШЕР [41] і АВЕКС [40]).

ОСА має на увазі дезагрегацію ПО, як правило, на вісім страт або шарів (табл. 5.1).

Таблиця 5.1. Стратифікація знань предметної області

s_1	Навіщо-Знання	*Стратегічний аналіз*: призначення і функції системи
s_2	Хто-Знання	*Організаційний аналіз*: колектив розробників системи
s_3	Що-Знання	*Концептуальний аналіз*: основні концепти, понятійна структура
s_4	Як-знання	*Функціональний аналіз*: гіпотези і моделі прийняття рішення
s_5	Де-Знання	*Просторовий аналіз*: оточення, устаткування, комунікації
s_6	Коли-Знання	*Часовий аналіз*: часові параметри й обмеження
s_7	Чому-Знання	*Каузальний або причинно-наслідковий аналіз*: формування підсистеми пояснень
s_8	Скільки-Знання	*Економічний аналіз*: ресурси, витрати, прибуток, окупність

Об'єктно-структурний аналіз має на увазі розроблення і використання матриці ОСА (див. табл. 5.2), що дозволяє всю зібрану інформацію дезагрегувати послідовно по шарах-стратах (вертикальний аналіз), а потім по рівнях – від рівня проблеми до рівня підзадачі (горизонтальний аналіз). Або навпаки – спочатку по рівнях, а потім по стратах.

Табл. 5.2. Матриця об'єктно-структурного аналізу

Рівні Страти	Рівень області u_1	Рівень проблеми u_2	Рівень задачі u_3	Рівень підзадачі u_4	...	u_n
Стратегічний аналіз s_1	E_{11}	E_{12}	E_{13}	E_{14}		E_{1n}
Організаційний аналіз s_2	E_{21}					
Концептуальний аналіз s_3	E_{31}					
Функціональний аналіз s_4	E_{41}					
Просторовий аналіз s_5	E_{51}					
Часовий аналіз s_6	E_{61}					
Каузальний аналіз s_7	E_{71}					
Економічний аналіз s_8	E_{81}					
					E_{ij}	
s_m	E_{m1}					E_{mn}

За потребою число страт може бути збільшено. У свою чергу, знання кожної страти піддаються подальшому ОСА й декомпозуються на складові

$$\|e_{mn}\|,$$

де m – номер рівня, n – номер страти, а e_{mn} належить множині K всіх концептів (понять) предметної області.

$$E_{mn} = \begin{Vmatrix} e_{11} & \dots & e_{1n} \\ \dots & \dots & \dots \\ e_{m1} & \dots & e_{mn} \end{Vmatrix}. \qquad (5.1)$$

Матриця (5.1) є матрицею над К. Нехай $M(K)$ – сукупність всіх mxn матриць над K. Тоді можна визначити клітинну матрицю E, у якій $m=m_1+\dots+m_k$, $n=n_1+\dots+n_l$, де m і n – цілі додатні числа $E \in M_{mn}(K)$, і її можна зобразити у вигляді:

$$E = \begin{Vmatrix} E_{11} & \dots & E_{1l} \\ \dots & \dots & \dots \\ E_{k1} & \dots & E_{kl} \end{Vmatrix}, \qquad (5.2)$$

де $E_{st} \in M_{st}(K), s = 1,\dots,k; t = 1,\dots,l.$

Матриця E є несиметричною, оскільки частина клітинних елементів E_{st} можуть піддаватися декомпозиції, а частина відображає деякі базисні атомарні концепти з K, що не підлягають деталізації.

Пропонований підхід припускає реалізацію концепції послідовного генезису ОСП через ОСА до *об'єктно-структурного розроблення* (ОСР).

Алгоритм ОСА

Алгоритм ОСА (об'єктно-структурного аналізу) призначений для детальної практичної структуризації знань ПО. В основу ОСА закладений алгоритм заповнення ОСА-матриці E_{mn}. Алгоритм містить послідовність аналітичних процедур, що дозволяють спростити і оптимізувати процес структуризації. Алгоритм розділяється на дві складові:

- A_I. Глобальний (вертикальний) аналіз, що включає розбиття ПО на методологічні страти (Які-Знання, Як-Знання і т. д.) на рівні всієї ПО. У результаті заповнюється перший стовпець матриці (5.2).
- A_II. Аналіз страт (горизонтальний), що включає побудову багаторівневих структур за окремими стратами. Число рівнів n визначається особливостями стратифікованих знань ПО і може істотно відрізнятися для різних страт. Зі сторони методології $n < 3$ свідчить про слабке опрацювання ПО.

Перший рівень відповідає рівню всієї ПО (рівень області). Другий – рівню проблеми, виділеної для вирішення. Третій – рівню конкретного вирішуваного завдання. Подальші рівні відповідають підзадачам, якщо має сенс їх виділяти.

При цьому можливе як послідовне застосування висхідної (bottom-up) і спадної концепцій (top-down), так і їхнє одночасне застосування.

Глобальний аналіз

Технологія глобального аналізу зводиться до розбиття простору основного завдання структуризації ПО на підзадачі, що відповідають особливостям ПО. Для розроблення інтелектуальних систем існує мінімальний набір s-страт, що забезпечує формування БЗ. Мінімальний набір включає три страти:

- s_3 – формування концептуальної структури S_k;
- s_4 – формування функціональної структури S_f;
- s_7 – формування підсистеми пояснень S_o.

Формування інших страт дозволяє істотно оптимізувати процес розроблення і уникнути багатьох традиційних помилок проектування. Страти s_4 і s_5 є додатковими і формуються у випадках, коли знання предметної області істотно залежать від тимчасових і просторових параметрів (системи реального часу, планування дій роботів і т.ін.). Алгоритм A_1 глобального аналізу може бути коротко сформульований отже:

- A_1_1. Зібрати всі матеріали з ідентифікації завдання і за результатами з витягу знань.
- A_1_2. Вибрати набір страт N, що підлягають формуванню (Nmin=3).
- A_1_3. Відібрати всю інформацію з першої обраної страти (i–1, де i – номер з вибраного набору страт N).
- A_1_4. Повторити крок A_1_3 для i+1 для всіх вибраних страт до N.
- A_1_5. Якщо частина інформації залишиться невикористаною, збільшити число страт і повторити для нових страт крок A_1_3; інакше перейти до послідовної реалізації алгоритмів горизонтального аналізу страт A_2.

Аналіз страт

Послідовність кроків горизонтального аналізу залежить від номера страти, але фактично зводиться до реалізації дуальної концепції структуризації для вирішення конкретної підзадачі.

Нижче пропонується алгоритм ОСА для однієї з обов'язкових страт s_3 (Що-Аналіз), результатом якого є формування концептуальної структури предметної області S_k.

- A_2_3_1. Із групи інформації, що відповідає Що-Страті, вибрати всі значущі поняття і сформулювати відповідні концепти.
- A_2_3_2. Виявити наявні ієрархії і зафіксувати їх графічно у вигляді структури.
- A_2_3_3. Деталізувати концепти, користуючись спадною концепцією (top-down).
- A_2_3_4. Утворити метапоняття за концепцією (bottom-up).
- A_2_3_5. Виключити повтори, надмірність і синонімію.
- A_2_3_6. Обговорити поняття, що не увійшли в структуру S_f, з експертом і перенести їх в інші страти або виключити.
- A_2_3_7. Отриманий граф або набір графів розділити на рівні і позначити – відповідно до матриці ОСА.

Запитання для повторення та контролю знань

1. Що таке інженерія знань?
2. Що таке поле знань?
3. Як виглядає мова опису поля знань?
4. Як задаються мови семіотичного моделювання?
5. Що в себе включає семіотика?
6. Як виглядає семіотична модель поля знань?
7. Чи можуть особливості предметної області вплинути на форму і зміст компонентів структури поля знань?
8. Як виглядає узагальнена синтаксична структура поля знань?
9. Як задається понятійна структура предметної області?
10. Як задається семантика поля знань?
11. Як визначити: чи рівні семантики поля знань?
12. Як відбувається інтерпретація інформації різними експертами? Наведіть приклади.
13. Як задається прагматика поля знань?
14. Що таке «піраміда» знань?
15. Які існують стратегії одержання знань?
16. Яка процедура називається видобуванням знань?
17. У чому полягає складність процесу видобування знань?
18. Який процес називається придбанням знань?
19. Дайте означення терміну формування знань.
20. Як навчання на прикладах пов'язане з машинним навчанням?
21. Які існують стратегії одержання знань під час розроблення інтелектуальних систем?
22. Які аспекти процедури видобування знань?

23. Який аспект процедури видобування знань є головним? Поясніть чому.
24. Наведіть чотири основних рівні спілкування.
25. Чому відбуваються втрати інформації при розмовному спілкуванні?
26. Які існують структурні компоненти моделі спілкування під час видобування знань?
27. Які існують психологічні аспекти видобування знань?
28. У чому полягає суть контактного шару?
29. У чому полягає суть процедурного шару?
30. У чому полягає суть когнітивного шару?
31. У чому полягає суть лінгвістичного аспекту видобування знань?
32. Які шари лінгвістичних проблем можна виділити в інженерії знань?
33. Що є відображенням «внутрішньої мови» експерта й аналітика?
34. Чому існує неоднозначна інтерпретація термінів двома фахівцями?
35. Чому існують проблеми формування понятійної структури?
36. Як задається лінгвістичний еквівалент ієрархії?
37. Для чого потрібен словник користувача?
38. У чому полягає гносеологічний аспект видобування знань?
39. Поясніть гносеологічний ланцюжок.
40. Які існують рівні структури знань?
41. Які існують основні характеристики емпіричного знання?
42. У чому полягає системно-структурний підхід до пізнання?
43. Чому коректніше говорити про глибину розуміння, ніж про об'єктивність знання?
44. Як виглядає структура пізнання?
45. Як здійснювати опис і узагальнення фактів?
46. Які операції традиційної логіки використовує інженер зі знань?
47. Як відбувається встановлення логічних і математичних зв'язків, дедукція і індукція законів?
48. Як відбувається побудова моделі, що ідеалізується?
49. Як відбувається пояснення і прогноз явищ?
50. У чому полягають теоретичні аспекти структурування знань?
51. Чому методи структуризації інформації традиційно використовують ієрархічний підхід?
52. Який підхід називається блочно-ієрархічним?
53. Які існують традиційні методології структуризації?
54. У чому полягає об'єктно-структурний підхід формування поля знань?
55. Як відбувається стратифікація знань предметної області?
56. Наведіть приклад матриці об'єктно-структурного аналізу.
57. У чому полягає суть алгоритму об'єктно-структурного аналізу?

РОЗДІЛ 6

ТЕХНОЛОГІЇ ІНЖЕНЕРІЇ ЗНАНЬ

- Класифікація методів практичного видобування знань
- Комунікативні методи
- Текстологічні методи

Розглядаються практичні методи одержання знань, їх класифікація та принципи використання. Спочатку детально описуються комунікативні методи, а потім текстологічні методи видобування знань.

6.1. Класифікація методів практичного видобування знань

Детально розглянувши у попередньому розділі теоретичні аспекти інженерії знань, ми, однак, в явному вигляді не визначили, яким практичним методом ми одержимо ці знання. У неявному вигляді передбачалось, що це деяка взаємодія інженера зі знань та експерта, що відбувається шляхом безпосереднього живого спілкування. Проте, це не єдина форма одержання знань, хоча вона є достатньо поширеною. У працях [34, 134] згадується близько 15 ручних (неавтоматизованих) методів видобування і понад 20 автоматизованих методів одержання і формування знань.

Рис. 6.1 ілюструє запропоновану класифікацію методів видобування, в якій використовуються найбільш вживані терміни, що дозволить інженерам зі знань залежно від конкретної задачі та ситуації обрати метод, який найбільш влаштовує.

Із запропонованої схеми класифікації видно, що основний принцип поділу пов'язаний із джерелом знань.

Комунікативні методи видобування знань охоплюють методи та процедури контактів інженера зі знань з безпосереднім джерелом знань – експертом, а текстологічні включають методи видобування знань з документів (методик, посібників, інструкцій) та спеціальної літератури (статей, монографій, підручників).

Поділ цих груп методів на верхньому рівні класифікації не означає їх антагоністичності, зазвичай інженер зі знань комбінує різні методи. Наприклад, спочатку вивчає літературу, після цього спілкується з експертами, або навпаки.

Водночас комунікативні методи також можна поділити на дві групи: активні та пасивні. Пасивні методи передбачають, що головна роль у процедурі видобування, передається експерту, а інженер зі знань лише складає протокол міркувань експерта під час його реальної роботи з прийняття рішень або записує те, що, на думку експерта, варто самостійно розповісти у формі лекції. В активних методах, навпаки, ініціатива повністю в руках інженера зі знань, який активно контактує з експертом різноманітними способами – в іграх, діалогах, бесідах за круглим столом тощо.

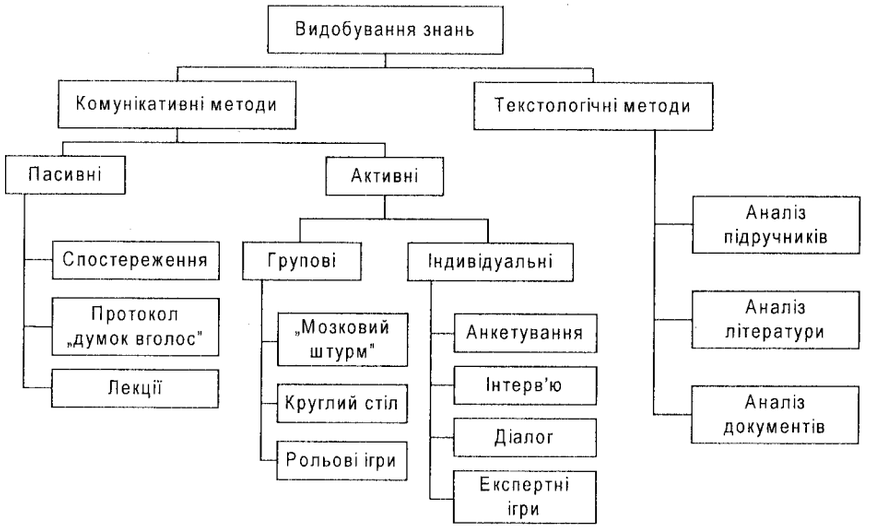


Рис. 6.1. Класифікація методів видобування знань.

Ще раз наголосимо на тому, що активні та пасивні методи можуть чергуватися навіть у рамках одного сеансу видобування знань. Наприклад, якщо інженер зі знань сором'язливий і не має великого досвіду, то спочатку він може використовувати пасивні методи, а поступово, краще знайомлячись з експертом, захоплювати ініціативу і переходити в «наступ».

Пасивні методи на перший погляд достатньо прості, але насправді вони вимагають від інженера зі знань вміння чітко аналізувати потік відомостей від експерта і виявляти в них значущі фрагменти знань. Відсутність зворотнього зв'язку (пасивність інженера зі знань) значно зменшує ефективність цих методів. Саме цим пояснюється їх допоміжна роль при активних методах.

Активні методи можна поділити на дві групи залежно від кількості експертів, які віддають свої знання. Якщо їх більше одного, то доцільно, окрім індивідуальних контактів з кожним, застосовувати ще й метод групових обговорень предметної області. Такі групові методи зазвичай активізують міркування учасників дискусії та дозволяють виявляти достатньо нетривіальні аспекти їх знань. Водночас, індивідуальні методи на сьогодні залишаються головними, оскільки така делікатна процедура, як «видобування знань», не може відбуватись при зайвих свідках.

Окремо варто сказати про ігри. Ігрові методи зараз широко використовують в соціології, економіці, менеджменті, в педагогіці для підготовки керівників, вчителів, лікарів та інших спеціалістів. Гра – це особлива форма діяльності та творчості, де людина стає розкутою і відчуває себе дещо вільніше, ніж у звичайній трудовій діяльності.

На вибір методу впливають три фактори: особисті особливості інженера зі знань, особисті особливості експерта і характеристика предметної області.

Одна з можливих класифікацій людей за психологічними характеристиками [131] ділить всіх на три типи:

- мислитель (пізнавальний тип);
- співбесідник (емоційно-комунікативний тип);

- практик (практичний тип).

Мислителі зорієнтовані на інтелектуальну роботу, навчання, теоретичне узагальнення і мають такі характеристики когнітивного стилю, як полі-незалежність і рефлективність (див. попередній розділ).

Співбесідники – це комунікабельні, відкриті люди, готові до співпраці.

Практики надають перевагу діям, а не розмовам, добре реалізують задуми інших, спрямовані на результативність роботи.

Для характеристики предметних областей можна запропонувати наступну класифікацію:

- добре документовані;
- середньо документовані;
- слабко документовані.

Ця класифікація пов'язана зі співвідношенням двох видів знань Z_1 та Z_2, введених в п. 5.3, де Z_1 – це експертне „особисте" знання, а Z_2 – матеріалізоване у книгах „загальне" знання в певній конкретній області. Якщо подати знання $Z_{ПО}$ предметної області як об'єднання Z_1 і Z_2, тобто $Z_{ПО} = Z_1 \cup Z_2$, то рис. 6.2 наглядно ілюструє запропоновану класифікацію.

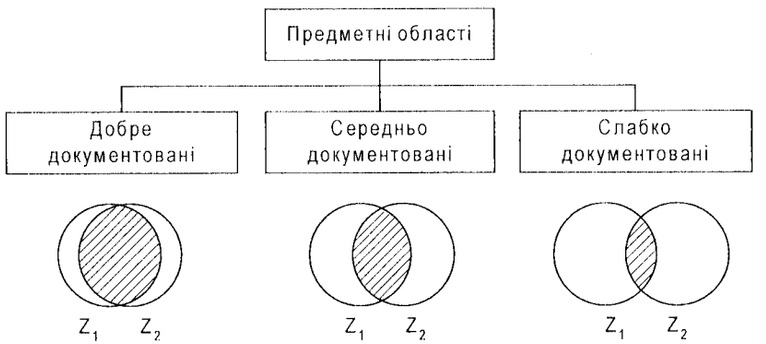


Рис. 6.2. Класифікація предметних областей.

Крім того, предметні області можна поділити за критерієм структурованості знань. Під структурованістю будемо розуміти ступінь теоретичного осмислення і виявлення основних закономірностей та принципів, що діють у певній предметній області. І хоча ЕС традиційно застосовуються у слабко структурованих предметних областях, зараз спостерігаємо тенденцію розширення сфери проникнення експертних систем. За ступенем структурованості знань предметні області можуть бути:

- добре структурованими – з чіткою аксіоматизацією, широким застосуванням математичного апарату, сталою термінологією;
- середньо структурованими – з визначеною термінологією, теорією, яка розвивається, та явним взаємозв'язком між явищами;
- слабко структурованими – з розмитими визначеннями, багатою емпірикою, прихованими взаємозв'язками, з великою кількістю «білих плям».

Введені в цьому параграфі класифікації методів та предметних областей допоможуть інженеру зі знань, чітко визначивши свою предметну область, співвіднести її з запропонованими типами і обрати такий метод чи групу методів видобування знань,

які найбільш підходять. Проте, скоріш за все, реальна робота повністю заперечить його вибір, і виявиться, що його добре документована область насправді є слабко документованою, а метод спостережень треба негайно замінити іграми. Такою є реальна важкість процедури видобування знань.

6.2. Комунікативні методи

За класифікацією, яка подана на рис. 6.1, розглянемо детальніше обидва різновиди комунікативних методів: пасивні та активні.

6.2.1. Пасивні методи

Термін «пасивні» не повинен викликати ілюзій, оскільки він введений на противагу «активним» методам. Насправді ж пасивні методи вимагають від інженера зі знань не менше віддачі, ніж такі активні методи, як ігри та діалог.

Пасивні методи видобування знань включають такі методи, де головна роль у процедурі видобування фактично передається експерту, а інженер зі знань тільки фокусує міркування експерта під час роботи з прийняття рішень.

Згідно із класифікацією (див. рис. 6.1) до цієї групи належать:

- спостереження;
- аналіз потоків «думок вголос»;
- лекції.

Спостереження

У процесі спостережень інженер зі знань перебуває безпосередньо поруч з експертом під час його професійної діяльності чи імітації цієї діяльності. При підготовці до сеансу видобування експерту необхідно пояснити ціль спостережень і попросити максимально коментувати свої дії. Під час сеансу аналітик записує всі дії експерта, його репліки та пояснення. Може бути здійснений і відеозапис в реальному масштабі часу. Обов'язковою умовою цього методу є невтручання аналітика в роботу експерта хоча б на перших етапах роботи. Саме цей метод є єдино «чистим» методом, який виключає втручання інженера зі знань та нав'язування ним якихось своїх структур уявлень:

- спостереження за реальним процесом;
- спостереження за імітацією процесу.

Зазвичай використовують обидва різновиди. Спочатку інженеру зі знань корисно спостерігати за реальним процесом, щоб глибше зрозуміти предметну область і відзначити всі зовнішні особливості процесу прийняття рішень. Це необхідно для проектування ефективного інтерфейсу користувача, оскільки майбутня ЕС повинна працювати саме у контексті такого реального виробничого процесу. Крім того, тільки спостереження дозволить аналітикові побачити предметну область, а, як відомо, «краще один раз побачити, ніж сто разів почути».

Спостереження за імітацією процесу здійснюють зазвичай також за робочим місцем експерта, але сам процес діяльності запускається спеціально для аналітика. Перевага цього різновиду полягає в тому, що експерт менш напружений, ніж у першому варіанті, коли він працює «на два фронти» – і веде професійну діяльність, і демонструє її. Недолік

збігається з перевагою – саме менше напруження експерта може вплинути на результат – оскільки робота не є справжньою, то і результат може відрізнятись від справжнього.

Спостереження за імітацією здійснюють також у тих випадках, коли спостереження за реальним процесом з будь-яких причин неможливе (наприклад, професійна етика лікаря-психіатра може не дозволяти присутності стороннього на прийомі).

Сеанси спостереження можуть вимагати від інженера зі знань:

- оволодіння технікою стенографії для фіксації дій експерта в реальному масштабі часу;
- ознайомлення з методиками хронометражу для чіткого структурування виробничого процесу за часом;
- розвитку навичок «читання по очах», тобто уважності до жестів, міміки та інших невербальних елементів спілкування;
- серйозного попереднього ознайомлення з предметною областю, оскільки через відсутність «зворотнього зв'язку» деколи у діях експертів є багато незрозумілого.

Протоколи спостережень після сеансів під час виконання домашньої роботи уважно розшифровують, а потім обговорюють з експертом.

Отже, спостереження – це один з найбільш поширених методів видобування знань на початкових етапах розроблення. Зазвичай він застосовується не самостійно, а в поєднанні з іншими методами.

Аналіз протоколів «думок вголос»

Складання протоколу «думок вголос» відрізняється від спостережень тим, що експерта просять не тільки прокоментувати свої дії та рішення, але й пояснити, як це рішення було прийняте, тобто продемонструвати весь ланцюжок своїх міркувань. Під час міркувань експерта всі його слова, весь «потік свідомості» заносить у протокол інженер зі знань, при цьому варто зазначити навіть паузи та вигуки. Деколи цей спосіб називають «вербальними звітами» [124].

Питання про використання із цією метою магнітофонів та диктофонів є дискусійним, оскільки магнітофон деколи діє на експерта, як паралітичний засіб, знищуючи атмосферу довіри, яка може і повинна виникати при безпосередньому спілкуванні.

Основними труднощами, які виникають під час складання протоколу «думок вголос» є труднощі, які пов'язані з тим, що для будь-якої людини важко пояснити як вона думає. При цьому існують експериментальні докази того факту, що люди не завжди в змозі достовірно описувати процеси мислення. Крім того, частина знань, яка зберігається у невербальній формі (наприклад, різноманітні процедурні знання типу «як зав'язати шнурівки»), взагалі важко корелюють з їхнім словесним описом. Автор теорії фреймів М. Мінський вважає, що «лише як виняток, а не як правило, людина може пояснити те, що вона знає» [286]. Проте існують люди, здатні до рефлексії, для яких ця робота є доступною. Отже, описана в попередньому розділі така характеристика когнітивного стилю, як рефлективність, є для експерта більш ніж бажаною.

Розшифрування отриманих протоколів здійснюється інженером зі знань самостійно з корекцією на наступних сеансах видобування знань. Успішне складання протоколів «думок вголос» є одним з найефективніших методів видобування знань, оскільки в ньому експерт може проявити себе найяскравіше, він нічим не скутий, ніхто йому не заважає, він наче вільно парує в потоці власних висновків та міркувань. Він може тут показати

свою ерудицію, продемонструвати глибину своїх знань. Для більшої кількості експертів це найприємніший і привабливий спосіб видобування знань.

Від інженера зі знань метод «думки вголос» вимагає тих самих вмінь, що і метод спостережень. Зазвичай «думки вголос» доповнюються потім одним з активних методів для реалізації зворотнього зв'язку між інтерпретацією інженера зі знань та уявленнями експерта.

Лекції

Лекція є найстарішим способом передавання знань. Лекційне мистецтво здавна високо цінувалось у всіх областях науки та культури. Але нас зараз цікавить не стільки здатність до підготовки та читання лекції, скільки здатність цю лекцію слухати, конспектувати та засвоювати. Вже згадувалося, що найчастіше експертів не обирають, а тому інженер зі знань не зможе вчити експерта читати лекції. Але якщо експерт має досвід викладача (наприклад, професор клініки чи досвідчений керівник виробництва), то можна скористатись таким концентрованим фрагментом знань, як лекція. У лекції експерту також дано багато ступенів свободи для самовираження, при цьому треба сформулювати для експерта тему та завдання лекції. Наприклад, тема циклу лекцій «Встановлення діагнозу – запалення легенів», тема конкретної лекції – «Міркування з аналізу рентгенограм», завдання – навчити слухачів за переліченими експертом ознаками встановити діагноз запалення легенів, робити прогноз. При такій постановці досвідчений лікар може заздалегідь структурувати свої знання і хід думок. Від інженера зі знань вимагається лише грамотно законспектувати лекцію і в кінці задати потрібні запитання.

Студенти добре знають, що конспекти лекцій одного й того ж лектора у різних студентів суттєво відрізняються. Списати конспект лекцій просять, як правило, у одного-двох студентів з групи. Люди, які вміло ведуть конспект, зазвичай є сильними студентами. Зворотнє невірно. У чому ж полягає мистецтво ведення конспекту? У «перешкодостійкості». Записувати головне, пропускати другорядне, виділяти фрагменти знань (параграфи, підпараграфи), записувати лише обдумані речення, вміти узагальнювати.

Гарне запитання під час лекції допомагає і лектору, і слухачу. Серйозні та глибокі запитання можуть суттєво підняти авторитет інженера зі знань в очах експерта.

Досвідчений лектор знає, що всі запитання можна розбити на три групи:

- розумні запитання, що поглиблюють лекцію;
- нерозумні запитання або запитання не по суті;
- запитання «на засипку», або провокаційні.

Якщо інженер зі знань задає запитання другого типу, то можливі дві реакції. Ввічливий експерт буде розмовляти з таким аналітиком як з дитиною, яка зараз не розуміє і так чи інакше нічого не зрозуміє. Зарозумілий експерт просто вийде з контакту, не бажаючи втрачати час. Якщо ж інженер зі знань захоче продемонструвати свою ерудицію запитаннями третього типу, то нічого крім дратування та відчуження, він не отримає у відповідь.

Тривалість лекції рекомендується стандартна – від 40 до 50 хвилин і через 5-10 хвилин – ще стільки ж. Курс зазвичай від двох до п'яти лекцій.

Метод видобування знань у вигляді лекцій, як і всі пасивні методи, використовують на початку розроблення, як ефективний спосіб швидкого занурення інженера зі знань у предметну область.

Наостанку декілька порад, як слухати лекції [156].

1. Підготуйтесь до лекції, тобто ознайомтесь з предметною областю.
2. Слухайте з максимальною увагою, для цього усуньте фактори, які заважають (скрип дверей, шарудіння тощо); зручно вмостіться, менше рухайтесь.
3. Вчіться відпочивати під час слухання (наприклад, коли лектор наводить цифри, які можна взяти з довідника).
4. Слухайте одночасно і лектора, і себе (паралельно з думками лектора в асоціації виникають власні думки).
5. Слухайте і одночасно записуйте, але записуйте текст скорочено, використовуючи умовні знаки (для цього зовсім не обов'язково бути стенографістом, достатньо тільки встановити для себе ряд умовних значків і незмінно ними користуватися).
6. Розшифруйте записи лекції того ж дня.
7. Не сперечайтеся з лектором під час лекції.
8. Раціонально використовуйте перерви у лекції для підведення підсумків прослуханого.

Порівняльні характеристики пасивних методів видобування знань подані в табл. 6.1.

Табл. 6.1. Порівняльні характеристики пасивних методів видобування знань

Пасивний метод видобування знань	Спостереження	"Думки вголос"	Лекції
Переваги	Відсутність впливу аналітика та його суб'єктивної позиції. Максимальне наближення аналітика до предметної області.	Свобода самовираження для експерта. Оголеність структур міркувань.	Свобода самовираження для експерта. Структурований виклад. Висока концентрація.
Недоліки	Відсутність зворотнього зв'язку. Фрагментарність отриманих коментарів.	Відсутність впливу аналітика та його суб'єктивної позиції. Відсутність зворотнього зв'язку. Можливість відходу від теми в міркуваннях експерта.	Відсутність впливу аналітика та його суб'єктивної позиції. Занадто велика деталізованість. Слабкий зворотній зв'язок Недостатність хороших лекторів серед експертів-практиків.
Вимоги до експерта (типи та основні якості)	Співбесідник чи мислитель (здатність до вербалізації + думок + аналітичність + відкритість + рефлексивність)	Співбесідник чи мислитель (здатність до вербалізації + думок + аналітичність + відкритість + рефлексивність)	Мислитель (лекторські здібності)

продовження табл. 6.1

Вимоги до експерта (типи та основні якості)	Мислитель (Спостережливість + полінезалежність)	Мислитель чи співрозмовник (контрастність +полінезалежність)	Мислитель (полінезалежність + здатність до узагальнення)
Характеристика предметної області	Слабо та середньо структуровані; слабо і середньо документовані	Слабо та середньо структуровані; слабо і середньо документовані	Слабо документовані та слабо структуровані

6.2.2. Активні індивідуальні методи

Активні індивідуальні методи видобування знань сьогодні – найбільш поширені. У тій чи іншій мірі до них звертаються при розробленні майже будь-якої інтелектуальної системи. До основних активних методів можна віднести:

- анкетування;
- інтерв'ю;
- вільний діалог;
- ігри з експертом.

У всіх цих методах активну функцію виконує інженер зі знань, який пише сценарій і є режисером сеансів видобування знань. Ігри з експертом суттєво відрізняються від трьох інших методів. Методи анкетування, інтерв'ю, вільного діалогу є дуже схожими між собою і відрізняються лише ступенем свободи, який може собі дозволити інженер зі знань при здійсненні сеансів видобування знань. Їх можна назвати запитальними методами пошуку знань.

Анкетування

Анкетування – це найжорсткіший метод, тобто найбільш стандартизований. У цьому випадку інженер зі знань заздалегідь складає список запитань чи анкету, розмножує її та використовує для опитування декількох експертів. Це основна перевага анкетування. Сам процес може здійснюватись двома способами.

1. Аналітик вголос задає запитання і сам заповнює анкету за відповідями експерта.
2. Експерт самостійно заповнює анкету після попереднього інструктування.

Вибір способу залежить від конкретних умов (наприклад, від оформлення анкети, її зрозумілості, готовності експерта). На нашу думку, перевагу варто надавати другому методу, оскільки в експерта з'являється необмеженість у часі при обдумуванні відповідей.

Якщо згадати схему спілкування, подану на рис. 3.8, то основними факторами, якими можна суттєво вплинути при анкетуванні, є засоби спілкування (у цьому випадку це список запитань) і ситуація спілкування.

Список запитань (анкета) заслуговує особливої розмови. Існує декілька загальних рекомендацій при складанні анкет. Ці рекомендації є універсальними, тобто не залежать від предметної області. Найбільший досвід роботи з анкетами нагромаджений в соціології та психології, тому частина рекомендацій взята з [141].

- Анкета не повинна бути монотонною і викликати нудьгу чи втому. Це досягається варіаціями форми запитань, зміною тематики, включенням запитань-жартів чи ігрових запитань.

- Анкета має бути пристосована до мови експертів (дивитись вище).
- Треба враховувати, що запитання впливають одне на одне і тому послідовність запитань має бути строго продумана.
- Бажано прагнути до оптимальної надлишковості. Відомо, що в анкеті завжди є багато зайвих запитань, частина з них необхідна – це так звані контрольні запитання (див. про них нижче), а інша частина має бути мінімізована.

Приклад

Зайві запитання з'являються, наприклад, у таких ситуаціях. Фрагмент анкети: «В12. Чи вважаєте Ви, що для лікування ангіни ефективний еритроміцин?».

«В13. Які дози еритроміцину Ви зазвичай рекомендуєте?».

При запереченні на 12-те запитання 13-те є зайвим. Його можна уникнути, ускладнивши попереднє запитання.

«В12. Чи використовуєте Ви еритроміцин для лікування ангіни і якщо так, то в яких дозах?».

- Анкета повинна мати «хороші манери», тобто її мова повинна бути зрозумілою і достатньо ввічливою. Методичною майстерністю складання анкети можна оволодіти тільки на практиці.

Інтерв'ю

Під інтерв'ю будемо розуміти специфічну форму спілкування інженера зі знань з експертом, в якій інженер зі знань задає експерту серію заздалегідь підготовлених запитань з метою видобувань знань про предметну область. Найбільший досвід у проведенні інтерв'ю накопичений, напевно, в журналістиці та соціології. Більшість спеціалістів цих областей відзначають тим не менше крайню недостатність теоретичих та методичних досліджень по темі проведення інтерв'ю [130].

Інтерв'ю дуже близьке до того способу анкетування, коли аналітик сам заповнює анкету, вносячи туди відповіді експерта. Основна відмінність інтерв'ю полягає в тому, що воно дозволяє аналітику опускати ряд запитань і, залежно від ситуації, вставляти нові запитання в інтерв'ю, змінювати темп, урізноманітнювати ситуацію спілкування. Крім того, у аналітика з'являється можливість «взяти в полон» експерта своїм шармом, зацікавити його самою процедурою і тим самим збільшити ефективність сеансу видобування.

Запитання для інтерв'ю

Тепер дещо детальніше про центральну ланку активних індивідуальних методів – про запитання. Інженери зі знань рідко сумніваються у своїй здатності ставити запитання, тоді як і у філософії, і у математиці ця проблема обговорюється з давніх давен. Існує навіть спеціальна гілка математичної логіки – еротетична логіка (логіка запитань). Існує цікава робота Белнапа «Логіка запитань та відповідей» [14], але, на жаль, використовувати результати, отримані логіками, безпосередньо при розробленні інтелектуальних систем не вдається.

Усі запитальні речення можна розбити на два типи [160]:

- запитання з невизначеністю, що стосуються усього речення («Правда, що введення великих доз антибіотиків може викликати анафілактичний шок?»);
- запитання з неповною інформацією («При яких умовах потрібно натискати кнопку?»), які часто починаються зі слів «хто», «що», «де», «коли» тощо.

Цей поділ можна доповнити класифікацією, частково описаною в роботі [187] і поданою на рис. 6.3.

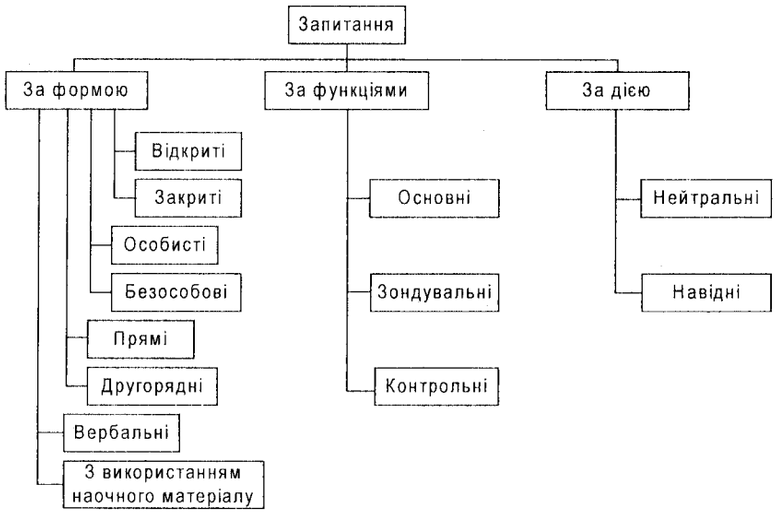


Рис. 6.3. Класифікація запитань.

Відкрите запитання називає тему чи предмет, залишаючи повну свободу експерту щодо форми та змісту відповіді («Чи не могли б Ви розповісти, як найкраще збити високу температуру у хворого із запаленням легенів?»). У закритому запитанні експерт вибирає відповідь з набору запропонованих («Вкажіть, будь ласка, що Ви рекомендуєте при ангіні: а) антибіотики, б) полоскання, в) компреси, г) інгаляції. Закриті запитання легше опрацьовувати при наступному аналізі, проте вони небезпечніші, оскільки «закривають» хід міркувань експерта і «програмують» його відповідь у певному напрямку. При складанні сценарію інтерв'ю корисно чергувати відкриті й закриті запитання, особливо ретельно продумувати закриті, оскільки для їх складання вимагається ерудиція в предметній області.

Особисте запитання стосується безпосередньо особистого індивідуального досвіду експерта («Скажіть, будь ласка, Іване Даниловичу, у вашій практиці Ви застосовуєте вулнузан при фурункульозах?»). Особисті запитання зазвичай активізують мислення експерта, «грають» на його самолюбстві, вони завжди прикрашають інтерв'ю. Безособове запитання спрямоване на виявлення найбільш поширених і загальноприйнятих закономірностей предметної області («Що впливає на швидкість процесу ферментизації лізину?»).

Під час складання запитань потрібно враховувати, що мовні здібності експерта, як правило, обмежені і внаслідок скованості, замкнутості, сором'язливості він не може одразу висловлювати свою думку і подавати знання, які від нього вимагаються (навіть якщо припустити, що він чітко для себе їх формулює). Тому часто при «затиснутості» експерта використовують не прямі запитання, які безпосередньо вказують на предмет чи тему («Як Ви ставитесь до методики лікаря Сухарєва?»), а непрямі, які лише непрямо вказують на предмет зацікавлення («Чи використовуєте Ви методику доктора Сухарєва? Опишіть, будь ласка, результати лікування»). Деколи доводиться задавати декілька десятків непрямих запитань замість одного прямого.

Вербальні запитання – це традиційні усні запитання. Запитання з використанням наглядного матеріалу урізноманітнюють інтерв'ю і знижують втомлюваність експерта. У цих запитаннях використовують фотографії, малюнки і картки. Наприклад, експерту пропонують кольорові картонні картки, на яких виписані ознаки захворювання. Потім аналітик просить розкласти ці картки в порядку зменшення значущості для встановлення діагнозу. Поділ запитань за функцією на основні, зондуючі, контрольні пов'язаний з тим, що часто основні запитання інтерв'ю, націлені на виявлення знань, не спрацьовують – експерт з деяких причин, ухиляється від відповіді, відповідає нечітко.

Тоді аналітик використовує зондувальні запитання, які скеровують міркування експерта в потрібне русло. Наприклад, якщо не спрацювало основне запитання: «Які параметри визначають момент закінчення процесу ферментації лізингу?» – аналітик починає задавати зондувальні запитання: «Чи завжди процес ферментації триває 72 години? А якщо він закінчується раніше, то як про це дізнатися? Якщо він триватиме довше, то що заставить мікробіолога не закінчити процес на 72-й годині?» тощо.

Контрольні запитання застосовують для перевірки достовірності та об'єктивності інформації, отриманої в інтерв'ю раніше («Скажіть, будь ласка, а московська школа психологів так само, як Ви трактує шкалу К списку запитань MMPI?» або «Чи рекомендуєте Ви ін'єкції АТФ?» (АТФ – препарат, який знято з виробництва). Контрольні запитання повинні бути «хитро» складені, щоб не образити експерта недовір'ям (для цього використовують повторення запитань у іншій формі, уточнення, посилання на інші джерела). «Краще два рази запитати, ніж один раз налякати» (Шолом-Алейхем).

І нарешті, про нейтральні та навідні запитання. У принципі, інтерв'юеру (в нашому випадку інженерові зі знань) рекомендують бути неупередженим, тому його запитання повинні мати нейтральний характер, тобто не мають вказувати на ставлення інтерв'юера до заданої теми. Навпаки, навідні запитання примушують респондента (в нашому випадку експерта) прислухатись чи навіть взяти до уваги позицію інтерв'юера. Нейтральне запитання: «Чи співпадають симптоми крововиливу в мозок та струсу мозку?». Навідне запитання: «Чи це правда, що дуже важко диференціювати симптоми крововиливу в мозок?».

Крім перелічених вище, корисно розрізняти та включати в інтерв'ю наступні запитання:

- контактні (ті, що «ламають кригу» між аналітиком та експертом);
- буферні (для розділення окремих тем інтерв'ю);
- ті, що оживляють пам'ять експертів (для реконструкції окремих випадків з практики);
- «провокаційні» (для отримання спонтанних, непідготовлених відповідей).

У кінці опису інтерв'ю вкажемо три основні характеристики запитань [187], які впливають на якість інтерв'ю:

- мова запитання (зрозумілість, лаконічність, термінологія);
- порядок запитань (логічна послідовність і немонотонність);
- доцільність запитань (етика, ввічливість).

Запитання в інтерв'ю – це не просто спосіб спілкування, але й спосіб передавання думок та позиції аналітика.

«Запитання являє собою форму руху думки, у ньому яскраво виражений момент переходу від незнання до знання, від неповного, неточного знання до більш повного і до більш точного» [89]. Звідси необхідність у протоколах фіксувати не тільки відповіді, але й запитання, попередньо ретельно відпрацьовуючи їх форму і зміст.

Очевидно, що будь-яке запитання має зміст лише у контексті. Тому запитання може готувати інженер зі знань, який вже оволодів ключовим набором знань.

Запитання мають для експерта діагностичне значення – декілька відвертих «дурних» запитань можуть повністю розчарувати експерта і відбити в нього бажання до подальшої співпраці. Відома відповідь Маркса на запитання Прудона: «Запитання було до такої міри неправильно поставлене, що на нього неможливо було дати правильну відповідь».

Вільний діалог

Вільний діалог – це метод видобування знань у формі бесіди інженера зі знань та експерта, в якій немає чітко регламентованого плану і списку запитань. Це не означає, що до вільного діалогу не треба готуватися. Навпаки, зовні вільна і легка форма цього методу потребує найвищої професійної та психологічної підготовки. Підготовка до вільного діалогу практично може співпадати із запропонованою в роботі [187] підготовкою до журналістського інтерв'ю. Рис. 6.4 графічно ілюструє схему такої підготовки, доповненої у зв'язку зі специфікою інженерії знань. Підготовка займає різний час залежно від ступеня професіоналізму аналітика, але в будь-якому випадку вона необхідна, оскільки дещо зменшує вірогідність найбільш нераціонального методу – методу спроб та помилок.

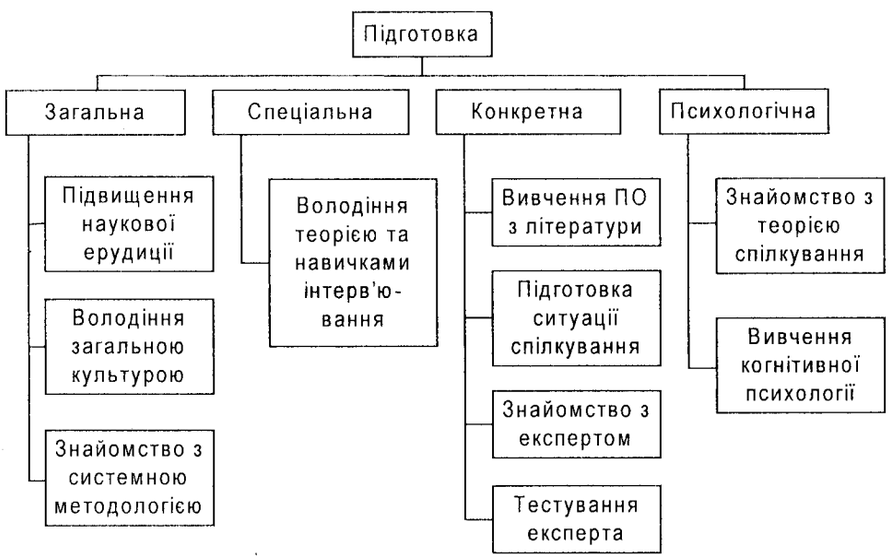


Рис. 6.4. Підготовка до видобування знань.

Кваліфікована підготовка до діалогу допомагає аналітику стати істинним драматургом або сценаристом майбутніх сеансів, тобто запланувати гладкий хід процедури видобування: від приємного враження на початку бесіди перехід до професійного контакту через пробудження зацікавлення і завоювання довіри експерта. При цьому для забезпечення бажання експерта продовжити бесіду необхідно здійснювати «погладжування» (термінологія Е. Берна [16]), тобто підбадьорювати експерта і всіляко підтверджувати його впевненість у власній компетентності (фрази-вставки: «Я вас розумію...», «...це дуже цікаво» тощо).

Так, в одному дослідженні з техніки ведення професійних журналістських діалогів було експериментально доведено, що схвалювальне «хмикання» інтерв'юера збільшує довжину відповідей респондента. При цьому схвалення повинно бути чесним, як показало опитування інтерв'юерів Інституту демоскопії Німеччини: «Найкраща хитрість – це уникнення будь-якої хитрості: ставитись до опитуваного з істинним людинолюбством, не з вигаданою, а зі справжньою цікавістю» [130]. Щоб розговорити співрозмовника, можна спочатку аналітику розповісти про себе, про роботу, тобто поговорити самому.

Ми вже писали про професійну придатність інженерів зі знань і необхідність попереднього психологічного тестування при підготовці інженерів зі знань. Тут лише наведемо каталог якостей ідеального інтерв'юера [130]. На наш погляд, це цілком придатний зразок портрету інженера зі знань перед серією вільних діалогів: «Він повинен виглядати здоровим, спокійним, впевненим, викликати довір'я, бути чесним, веселим, проявляти зацікавлення бесідою, бути гарно вдягненим, доглянутим». Хороший аналітик особистим шармом і вмінням може приховати недостатність підготовки. Блискуча коротка характеристика інтерв'юера наведена в тій же роботі – «комунікабельний педант».

У вільному діалозі важливо також обрати правильний темп чи ритм бесіди: без великих пауз, оскільки експерт може відволіктися, але і без поспіху, інакше швидко втомлюються обидва учасника і збільшується напруга, крім того деякі люди говорять і думають дуже повільно. Вміння чергувати різні темпи, напругу і розрядку в розмові суттєво впливає на результат. Підготовка до діалогу так, як і до інших активних методів видобування знань, включає складання плану проведення сеансу видобування, в якому необхідно передбачити ряд стадій.

1. Початок бесіди (знайомство, створення у експерта «образу» аналітика, пояснення цілей та завдань роботи).
2. Діалог з видобування знань.
3. Завершальна стадія (подяка експерту, підведення підсумків, домовленість про наступні зустрічі).

Девізом для інженера зі знань можуть послужити погляди одного із класиків вітчизняного літературознавства М. М. Бахтіна [12]:

«Діалог – зіткнення різних розумів, різних істин, несумісних культурних позицій, що складають єдиний розум, єдину істину, загальну культуру». «Діалог передбачає:

- унікальність кожного партнера та їх принципову рівність один одному;
- відмінність і оригінальність їх точок зору;
- орієнтацію кожного на розуміння і на активну інтерпретацію його точки зору партнером;
- очікування відповіді та його передзахоплення у власному висловлюванні;
- взаємну доповнюваність позицій учасників спілкування, співвідношення яких і є метою діалогу».

Порівняльні характеристики активних індивідуальних методів видобування знань подані в табл. 6.2.

Табл. 6.2. Порівняльні характеристики активних індивідуальних методів видобування знань

Активний індивідуальний метод видобування знань	Анкетування	Інтерв'ю	Вільний діалог
Переваги	Можливість стандартизованого опиту-вання декількох експертів	Наявність зворотнього зв'язку для експерта	Гнучкість Значний зворотній зв'язок змін
	Не вимагає особливого напруження від аналітика під час процедури анкетування		Можливіст змін сценарію і форми сеансу
Недоліки	Вимагає вміння і досвіду складання анкет	Вимагає значного часу на підготовку запитань інтерв'ю	Вимагає від аналітика найбільшого напруження
	Відсутність контексту між експертом, немає зворот-нього зв'язку. Запитання анкети можуть бути неправильно зрозумілі експертом		Відсутність формальних методик проведення Складність протоколювання результатів
Вимоги до експерта (типи, якості)	Практик і мислитель	Співрозмовник або мислитель	Співрозмовник або мислитель
Вимоги до аналітика (типи, якості)	Мислитель (педантизм у складанні анкет, уважність)	Співрозмовник (журналістські навички, вміння слухати,шарм)	Співрозмовник (спостережливість, вміння слухати,шарм)
Характеристика предметної області	Слабо структуровані, слабо і середньо документовані	Слабо структуровані, слабо і середньо документовані	Слабо структуровані, слабо і середньо документовані

6.2.3. Активні групові методи

До групових методів видобування знань належать рольові ігри, дискусії за «круглим столом» за участю декількох експертів і «штурми мозку».

Основні переваги групових методів – це можливість одночасного «поглинання» знань від декількох експертів, взаємодія яких вносить у цей процес елемент принципової новизни від накладання різних поглядів та позицій. Оскільки ці методи менш популярні, ніж індивідуальні (що пов'язане зі складністю організації), спробуємо описати їх детально.

Дискусії за «круглим столом»

Учасники висловлюються в певному порядку, а потім переходять до живої вільної дискусії. Кількість учасників змінюється від трьох до п'яти-семи. Більшість загальних рекомендацій з видобування знань, запропонованих раніше, можна застосувати і до цього методу. Однак існує і специфіка, пов'язана з поведінкою людини в групі.

По-перше, від інженера зі знань підготовка «круглого столу» вимагає додаткових зусиль: як організаційних (місце, час, обстановка, мінеральна вода, чай, кворум тощо), так і психологічних (вміння вставляти доцільні репліки, почуття гумору, пам'ять на імена та по батькові, здатність гасити конфліктні ситуації тощо).

По-друге, більшість учасників буде говорити під дією «ефекту фасаду» зовсім не те, що вони сказали б в іншій обстановці, тобто бажання справити враження на інших експертів буде суттєво «підсвічувати» їхні висловлювання. Цей ефект часто спостерігається на захистах дисертацій. Члени вченої ради запитують зазвичай не те, що їх насправді цікавить, а те, що демонструє їх особисту компетентність.

Хід бесіди за «круглим столом» зручно записувати на магнітофон, а при розшифруванні й аналізі результатів враховувати цей ефект, а також взаємні відносини учасників. Завдання дискусії – колективно, з різних точок зору, під різними кутами, дослідити спірні гіпотези предметної області. Зазвичай, емпіричні області багаті на такий дискусійний матеріал. Для гостроти на «круглий стіл» запрошують представників різних наукових напрямів і різних поколінь, це також зменшує небезпеку одержання однобічних знань. Обмін думками з наукових питань має давню традицію в історії людства (антична Греція, Індія). До наших днів дійшли літературні пам'ятки обговорення спірних питань (наприклад, Протагор «Мистецтво сперечатись», роботи софістів), які послужили першоосновою діалектики – науки вести бесіду, сперечатись, розвивати теорію. У самому слові «дискусія» (від лат. discussio – дослідження) містяться вказівки на те, що це метод наукового пізнання, а не просто суперечки (для порівняння, полеміка – від грец. polemikos – войовничий, ворожий). Декілька практичних порад з процедурних питань «круглого столу» з роботи [165]. Перед початком дискусії ведучому корисно:

- впевнитись, що всі правильно розуміють завдання (тобто відбувається сеанс видобування знань);
- встановити регламент;
- чітко сформулювати тему.

Під час дискусії важливо відслідковувати, щоб надміру емоційні та балакучі експерти не підмінили тему і щоб критика позицій один одного була обґрунтованою. Наукова плідність дискусії робить цей метод привабливим і для самих експертів, особливо для тих, хто знає менше. Це помітив ще Епікур: «При філософській дискусії більше виграє переможений – в тому відношенні, що він примножує знання» [115]. Хорошою настановою для проведення «круглого столу» є слова П. Л. Капіци: «Коли в якійсь науці немає протилежних поглядів, немає боротьби, то ця наука йде по шляху до цвинтаря, вона йде ховати себе» [75].

«Штурм мозку»

Активні групові методи зазвичай використовуються як гостра приправа при видобуванні знань, самі по собі вони не можуть слугувати джерелом більш чи менш повного знання. Їх застосовують як допоміжні до традиційних індивідуальних методів (спостереження, інтерв'ю тощо); для активізації мислення та поведінки експертів.

«Штурм мозку», або «мозкова атака», – один з найпоширеніших методів для активізації творчого мислення. Інші методи (метод фокальних об'єднань, синектика, метод контрольних запитань [185]) застосовуються рідше через меншу ефективність.

Вперше цей метод був використаний у 1939 р. у США А. Осборном як спосіб отримання нових ідей в умовах забороненої критики. Помічено, що страх перед критикою заважає творчому мисленню, тому основна ідея штурму – це відокремлення процедури генерування ідей у замкненій групі спеціалістів від процесу аналізу й оцінки висловлених ідей.

Як правило, штурм триває недовго (близько 40 хв). Учасникам (до 10 осіб) пропонується говорити будь-які ідеї (жартівливі, фантастичні, хибні) на задану тему (критика заборонена). Зазвичай висловлюється понад 50 ідей. Регламент – до 2 хвилин на виступ. Найцікавіший момент штурму – це настання піку (ажіотажу), коли ідеї починають «фонтанувати», тобто відбувається невимушене генерування гіпотез учасниками. Цей пік має теоретичне обґрунтування в роботах видатного швейцарського психолога і психіатра З. Фрейда про несвідоме. При наступному аналізі лише 10-15% ідей є розумними, але серед них є доволі оригінальні. Оцінює результати, зазвичай, група експертів, що не брала участі в генеруванні.

Ведучий «штурму мозку» – інженер зі знань – повинен вільно володіти аудиторією, підібрати активну групу експертів – «генераторів», не затискати погані ідеї – вони можуть слугувати каталізаторами хороших. Мистецтво ведучого – це мистецтво задавати питання аудиторії, «підігріваючи» генерування. Запитання слугують «гачком» [185], яким видобуваються ідеї. Запитання також можуть зупиняти багатослівних експертів і слугувати способом розвитку ідей інших.

Основний девіз штурму – «чим більше ідей, тим краще». Фіксація ходу сеансу – традиційна (протокол чи магнітофон).

Переваги та недоліки активних групових методів видобування знань подані в табл. 6.2.

Експертні ігри

Грою називають такий вид людської діяльності, який відображає (відтворює) інші її види [79]. При цьому для гри характерні одночасно умовність і серйозність.

Поняття експертної гри чи гри із експертами з метою видобування знань пов'язане із трьома джерелами – це ділові ігри, широко використовувані при підготовці спеціалістів та моделюванні [20]; діагностичні ігри, описані у працях [7], і комп'ютерні ігри, що все частіше використовуються у навчанні.

Зараз у психолого-педагогічних науках немає розвиненої теоретичної концепції ділових ігор та інших ігрових методів навчання. Тим не менше, на практиці ці ігри широко використовуються. Під діловою грою найчастіше розуміють експеримент, де учасникам пропонується виробнича ситуація, а вони на основі свого життєвого досвіду, своїх загальних і спеціальних знань розв'язують її.

Ігри з експертом

У цьому випадку з експертом грає інженер зі знань, який бере на себе будь-яку роль у змодельованій ситуації. Наприклад, одному з авторів часто доводилося розігрувати з експертом гру «Вчитель і учень», в якій інженер зі знань бере на себе роль учня і на очах у експерта виконує його роботу (наприклад, пише психодіагностичний висновок), а експерт виправляє помилки «учня». Ця гра – зручний спосіб розговорити сором'язливого експерта.

Приклад

Інша гра (взята з роботи [47]) змушує інженера зі знань взяти на себе роль лікаря, який знає добре хворого, а експерт грає роль консультанта. Консультант ставить запитання і робить прогноз про доцільність використання того чи іншого методу лікування (в описаній грі це був прогноз доцільності електростимульної терапії при серцевій аритмії). Гра «двох лікарів» дозволила виявити, що експерту знадобилось лише 30 запитань для успішного прогнозу, в той час як початковий варіант списку запитань, складений медиками з тією ж метою, містив 170 запитань. Приклад

Спочатку експерта просять написати обґрунтування для власного прогнозу. Наприклад, чому він вважає, що виразка у хворого Х загоюється. Накопичується декілька таких обґрунтувань [47], а через деякий час експерту зачитують лише його обґрунтування і просять зробити прогноз. Як правило, цього він зробити не може, тобто обґрунтування (його знання) було неповним. Експерт доповнює обґрунтування, таким чином виявляються приховані (для самого експерта) шари знань. Так, в іграх «Обґрунтування прогнозу рецидиву виразкової кровотечі» вдалося виявити, що суттєвими для прогнозу є усі три правила. При чому два правила входили в традиційно-діагностичний список запитань, а третє було сформульоване під час гри.

Приклад

Гра «фокусування на контексті»: експерт грає роль ЕС, а інженер по знанням – роль користувача. Розігрується ситуація консультації. Перші запитання експерта виявляють найбільш значимі поняття, найважливіші аспекти проблеми. Роль користувача може взяти на себе й інший експерт.

Основні поради інженеру зі знань з проведення індивідуальних ігор:

- грайте сміливіше, вигадуйте ігри самі;
- не нав'язуйте гру експерту, якщо він не налаштований;
- в грі «не тисніть» на експерта, не забувайте мети гри;
- грайте весело, нешаблонно;
- не забувайте про час і про те, що гра втомлива для експерта.

Рольові ігри в групі

Групові ігри передбачають участь у грі декількох експертів. Для такої гри зазвичай заздалегідь складається сценарій, розподіляються ролі, до кожної ролі готують портрет-опис (краще з девізом) і розробляють систему оцінювання гравців [20].

Існує декілька способів проведення рольових ігор. В одних іграх гравці можуть вигадувати собі нові імена і грати під ними, в інших – всі гравці переходять на «ти». У третіх – ролі обирають гравці, в четвертих – ролі витягують за жеребом. Роль – це комплекс зразків поведінки. Роль пов'язана з іншими ролями. «Короля грає челядь». Оскільки у нашому випадку режисером і сценаристом гри є інженер зі знань, то йому і надається повна свобода вибору форми проведення гри.

Приклад

Так, в роботі [87] описана гра «План», призначена для видобувань знань із спеціалістів підприємства, що розробляють виробничі плани випуску для цехів і приймають різноманітні рішення з керування виробництвом. У грі експертів розбили на три ігрові

групи: ЛПР1 – група планування; ЛПР2 – група менеджерів; Е-група експертизи з оцінювання дій ЛПР1 та ЛПР2. Групам ЛПР1 та ЛПР2 пропонувалися різні промислові ситуації і ретельно протоколювали їх суперечки, міркування, аргументи з прийняття рішень. У результаті гри був створений прототип бази знань експертної системи планування.

Зазвичай у грі приймає участь від трьох до шести експертів, якщо їх більше, то можна розділити всіх на декілька конкурентних ігрових бригад. Наприклад, чий діагноз виявиться ближче до істинного, чий план раціональніше використовує ресурси, хто швидше виявить причину несправності в технічному блоці.

Створення ігрової обстановки потребує чимало фантазії та творчої вигадки від інженера зі знань. Рольова гра, як правило, потребує декількох найпростіших заготовок (наприклад, табличок «Директор», «Бухгалтерія», «Плановий відділ», спеціально надрукованих інструкцій з правилами гри). Але головне, звичайно, щоб експерти в грі дійсно «заграли», стали розкутими і «розкрили свої карти».

Ігри з тренажерами

Ігри з тренажерами у більшій мірі ближчі не до ігор, а до імітаційних вправ у ситуації, наближеній до реальної. Наявність тренажера допоможе відтворити майже промислову ситуацію і поспостерігати за експертом. Тренажери широко використовують для навчання (наприклад, пілотів або операторів атомних станцій). Очевидно, що застосування тренажерів для видобування знань дозволить зафіксувати фрагменти знань із літання, які виникають під час і на місці реальних ситуацій, і випадають із пам'яті при виході за межі ситуації.

Комп'ютерні експертні ігри

Ідея використовувати комп'ютери у ділових іграх відома давно. Але лише коли комп'ютерні ігри взяли в полон майже всіх користувачів персональних ЕОМ від малого до великого, стала очевидною привабливість такого роду ігор.

Традиційна сучасна класифікація комп'ютерних ігор із журналу GAME.EXE:

- **Action/Arcade games (екшн/аркади).** Ігри-дії. Вимагають хорошого окоміру та швидкої реакції.
- **Simulation games (симулятори).** Базуються на модельованій реальній дійсності та відпрацюванні практичних навичок, наприклад, у керуванні автомобілем, пасажирським літаком, поїздом, авіадиспетчера. Існують навіть симулятори рибної ловлі. Також популярні спортивні симулятори – теніс, бокс тощо.
- **3D Action games („стрілялки").** Те ж, що й екшн, але з активним використанням трьохмірної графіки.
- **Strategy games (стратегічні ігри).** Вимагають стратегічного планування та відповідальності під час прийняття рішень, наприклад, розвиток цивілізацій, суперництво світів, економічна боротьба. Особливий клас стратегічних ігор – wargames (військові ігри). Останнім часом спрямованість в 3D Action робиться на багатокористувальницький режим (гру в мережі).
- **Puzzles (настільні ігри-головоломки).** Комп'ютерні реалізації різноманітних логічних ігор.
- **Adventure/Quest (пригодницькі ігри).** Зазвичай володіють розгалуженим сценарієм, хорошою графікою та звуком. Керуючи одним чи декількома персонажами, гравець повинен правильно вести діалоги, розгадувати багато загадок та головоломок, помічати та правильно використовувати предмети, заховані у грі.
- **Role-plaing games RPG (рольові ігри).** Поширений жанр, що бере свій початок у старих англійських настільних іграх. Існує один або декілька героїв, що мають індивідуальні властивості та характеристики. Їм доводиться боротися з ворогами, вирішувати загадки. У міру виконання цих завдань, у героїв накопичується досвід і при досягненні певного значення їхні характеристики покращуються.

Варто відзначити, що багато ігор можна одночасно віднести до декількох класів, і в цілому, цю класифікацію не можна вважати строгою. Ігри деколи бувають корисними для розважання експертів перед початком сеансу видобування знань. Крім того, очевидно, що експертні ігри, поєднуючи елементи перелічених вище класів, можуть успішно застосовуватися для безпосереднього видобування знань. Проте розроблення та програмна реалізація такої гри вимагає суттєвих вкладень часових та грошових ресурсів.

Приклад

Одна з перших вітчизняних експертних комп'ютерних ігор описана в роботі [50]. Основний принцип гри «Зоосад» полягає у створенні ігрової ситуації при організації діалогу з експертом. При цьому завдання видобування знань маскується спрямованістю на розв'язання чисто ігрової задачі: необхідно визначити вміст «чорної скриньки», в якій міститься якась тварина, при цьому треба набрати якнайбільшу кількість очок, не витративши виділеного ресурсу грошей. У ході гри експерт робить ставки на різні гіпотези, вказуючи при цьому, які ознаки має та чи інша тварина.

Після кожної відповіді він отримує необхідну інформацію про результати. У ході гри непомітно для експерта формуються правила, що відображають знання експерта на основі зроблених ним ходів. У цій грі – це знання про те, які ознаки мають ті чи інші тварини. Таким чином виявляється алфавіт значущих ознак для діагностики та класифікації тварин.

6.3. Текстологічні методи

Група текстологічних методів об'єднує методи видобування знань, засновані на вивченні спеціальних текстів з підручників, монографій, статей, методик та інших носіїв професійних знань.

У буквальному розумінні текстологічні методи не відносять до текстології – науки, яка народилася в руслі філології з метою критичного читання літературних текстів, вивчення та інтерпретації джерел з вузькоприкладної задачі – підготування текстів до видання.

Зараз текстологія розширила свої межі включенням аспектів суміжних наук – герменевтики (науки правильного тлумачення древніх текстів – біблії, античних рукописів та ін.), семіотики, психолінгвістики тощо.

Текстологічні методи видобування знань, безумовно, використовуючи основні положення текстології, відрізняються принципово від її методології, по-перше, характером і природою своїх джерел (професійна спеціальна література, а не художня, що живе за своїми власними законами), а по-друге, жорсткою граматичною спрямованістю видобування конкретних професійних знань.

Серед методів видобування знань ця група є найменш розробленою, з неї майже немає бібліографії, тому подальший виклад є наче введенням в методи вивчення текстів в тому вигляді, як це уявляють автори.

Задачу видобування знань з текстів можна сформулювати як задачу розуміння і виокремлення суті тексту. Сам текст на звичайній мові є лише провідником суті, а задум і знання автора лежать у вторинній структурі (змістовній структурі чи макроструктурі тексту), надбудованій над звичайним текстом [28], або, як сформульовано в роботі [173], «текст не містить і не передає суті змісту, а є тільки інструментом для автора тексту».

При цьому можна виділити дві такі змістові структури:

M_1 – зміст, який намагався вкласти автор, це його модель світу, і M_2 – зміст, який розуміє читач, у нашому випадку інженер зі знань (рис. 6.5), в процесі інтерпретації I. При цьому T – це словесна оболонка M, тобто результат вербалізації V.

Складність процесу полягає в принциповій неможливості співпадіння знань, що утворюють M_1 і M_2 через те, що M_1 утворюється за рахунок всієї сукупності уявлень, потреб, інтересів та досвіду автора, лише невелика частина яких знаходить відображення у тексті T. Відповідно, і M_2 утворюється в процесі інтерпретації тексту T за рахунок залучення всієї сукупності наукового та людського багажу читача. Отже, два інженери зі знань видобудуть з одного T дві різні моделі M_1 і M_2.

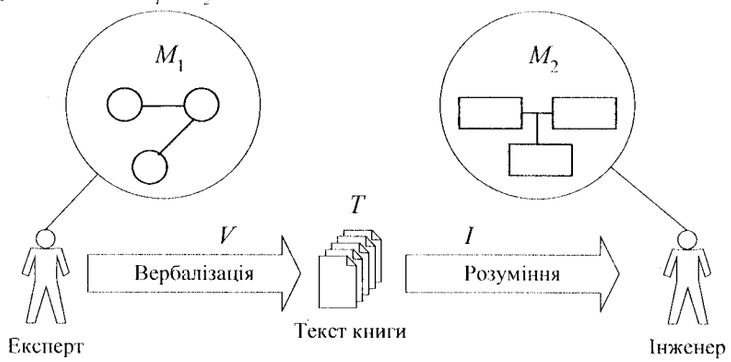


Рис. 6.5. Схема видобування знань зі спеціальних текстів.

Постає завдання: з'ясувати, за рахунок чого можна досягти максимальної адекватності M_1 і M_2, пам'ятаючи при цьому, що розуміння завжди відносне, оскільки це синтез двох змістів „своє – чуже" [13].

Розглянемо детальніше, які джерела живлять модель M_1 і створюють текст T. У роботі [13] вказані два компоненти будь-якого наукового тексту. Це перший матеріал спостережень α та система наукових понять β в момент створення тексту. На додаток до цього, на наш погляд, крім об'єктивних даних експериментів та спостережень, в тексті обов'язково є суб'єктивні погляди автора γ, як результат його особистого досвіду, а також деякі „загальні місця" або „вода" δ. Крім цього, будь-який науковий текст містить запозичення з інших джерел (статей, монографій) і т.ін., при цьому всі компоненти занурені в мовне середовище L. Можна записати:

$$T = (\alpha,\beta,\gamma,\delta,\theta)_L.$$

Отже, компоненти наукового тексту можна подати у вигляді наступної схеми (рис. 6.6). При цьому компоненти β, γ частина α входять і в модель M_1.

Рис. 6.6. Компоненти наукового тексту.

При видобуванні знань аналітику, який інтерпретує текст, доводиться розв'язувати задачу декомпозиції цього тексту на перераховані вище компоненти для виділення справді важливих для реалізації баз знань фрагментів. Складність інтерпретації наукових і спеціальних текстів полягає і в тому, що будь-який текст набирає значення тільки в контексті, де під контекстом розуміється оточення, в яке «занурений» текст.

Розрізняють мікро- та макроконтекст. Мікроконтекст – це найближче оточення тексту. Так, речення отримує зміст в контексті абзацу, абзац в контексті розділу і т. д. Макроконтекст – це вся система знань, пов'язана з предметною областю (тобто знання про особливості та властивості, явно не вказані в тексті). Іншими словами, будь-яке знання набуває сенсу в контексті деякого метазнання.

Тепер детальніше про центральну ланку процедури вилучення знання – про розуміння тексту. Класичним в текстології є визначення німецького філософа і мовознавця В. фон Гумбольдта [176]:

«...Люди розуміють один одного не тому, що передають співбесіднику ознаки предметів, і навіть не тому, що взаємно налаштовують один одного на точне і повне відтворення ідентичного поняття, а тому, що взаємно зачіпають один в одному одну й ту саму ланку чуттєвих уявлень і початків внутрішніх понять, доторкаються до одних і тих самих клавіш інструмента свого духа, завдяки чому у кожного спалахують у свідомості відповідні, а не тотожні розуміння».

Спілкуючись мовою сучасного мовознавства, розуміння – це формування «другого тексту», тобто семантичної структури (поняттєвої структури) [161]. У нашій термінології – це спроба відтворення семантичної структури M_1 в процесі формування моделі M_2, тобто це перший крок структурування знань.

Як відбувається процес розуміння? Одна з можливих схем викладена в роботі [164]. Ми внесли деякі зміни в цю схему у зв'язку з тим, що в ній трактується розуміння тексту на іноземній мові, а нас цікавить розуміння тексту в новій для людини, що пізнає, предметні області. Крім цього, доповнимо її деякими положеннями герменевтики. У цілому одержана схема узгоджується зі стратегією вивчення всього нового.

Основними моментами розуміння тексту є:

- висування попередньої гіпотези про суть всього тексту (передбачення);
- визначення значень незрозумілих слів (тобто спеціальної термінології);
- виникнення загальної гіпотези про зміст тексту (про знання);
- уточнення значення термінів та інтерпретація окремих фрагментів тексту під впливом загальної гіпотези (від цілого до частин);

- формування деякої змістової структури за рахунок встановлення внутрішніх зв'язків між окремими важливими (ключовими) словами, фрагментами, а також за рахунок виникнення абстрактних понять, що узагальнюють окремі фрагменти знань;
- корекція загальної гіпотези відносно фрагментів знань, що містяться в тексті (від частин до цілого);
- прийняття основної гіпотези, тобто формування M_2.

Варто зауважити наявність як дедуктивної (від цілого до частин), так і індуктивної (від частин до цілого) складових процесу розуміння. Такий двоєдиний підхід дозволяє охоплювати текст як змістову єдність особливого роду, з її основними ознаками, такими як зв'язність, цілісність, завершеність та ін. [161].

Центральними поняттями процесу I є кроки 5 та 7, тобто формування змістової структури чи виділення «опірних», ключових слів чи «змістових віх» [161], а також заключне зв'язування «змістових віх» в єдину семантичну структуру.

При аналізі тексту важливе виявлення внутрішніх зв'язків між окремими елементами тексту та поняттями. Традиційно виділяють два види зв'язків у тексті – експліцитні (явні зв'язки), які виражаються у зовнішньому подрібненні тексту, та імпліцитні (приховані зв'язки). Експліцитні зв'язки ділять текст на параграфи за допомогою перелічення компонентів, вставних слів (або конекторів) типу «по-перше...», «по-друге...», «проте» тощо. Імпліцитні (внутрішні) зв'язки між окремими «змістовими віхами» викликають основні складнощі при розумінні.

Отже, семантична структура тексту виникає у свідомості суб'єкта, який пізнає за допомогою знань про мову, знань про світ, а також загальних (фонових) знань у тій предметній області, якій присвячений текст: «тексти пишуть для посвячених». Іншими словами, якщо текст не є науково-популярним, то для його адекватного читання потрібна деяка підготовка.

Отже, шлях до знань збільшується ще на одну ланку. Якщо раніше ми говорили, що самі текстологічні методи рідко вживаються як самостійний метод видобування, а зазвичай використовується як деяка підготовка до комунікативної взаємодії, то тепер стверджуємо, що і для читання текстів потрібна підготовка. Яка ж?

Підготовкою для читання спеціальних текстів є вибір разом з експертами деякого «базового» списку літератури, який поступово введе аналітика в предметну область. У цьому списку можуть бути підручники для початківців, розділи і фрагменти з монографій, популярні видання. Лише після ознайомлення з «базовим» списком доцільно переходити до читання спеціальних текстів.

Отже, на процес розуміння (чи інтерпретації) I та модель M_2 впливають наступні компоненти (рис. 6.7):

- екстракт компонентів $(\alpha,\beta,\gamma,\theta)'$, почерпнутий з тексту T;
- попередні знання аналітика про предметну область ПО;
- загальнонаукова ерудиція аналітика ε;
- його особистий досвід D.

$$M_2=[(\alpha,\beta,\gamma,\theta)', \text{ПО}, \varepsilon, D].$$

Процес I – це складний процес, що не піддається формалізації, на який суттєвим чином впливають такі чисто індивідуальні компоненти, як когнітивний стиль пізнання, інтелектуальні характеристики та ін.

Рис. 6.7. Компоненти формування змісту тексту.

Але процедури розбиття тексту на частини («змістові групи»), а потім згущення, стиснення вмісту кожного змістового шматка у «змістову віху» є, очевидно, основою для будь-якого індивідуального процесу розуміння. Така компресія (стиснення) тексту у вигляді набору ключових слів, що передають основний зміст тексту, може слугувати зручною методологічною основою для здійснення текстологічних процедур видобування знань.

Як ключове слово може слугувати будь-яка частина мовлення (іменник, прикметник, дієслово і т. ін.) або їх поєднання. Набір ключових слів (НКС) – це набір опірних точок, за якими розгортається текст при кодуванні в пам'ять і усвідомлюється при декодуванні; це семантичне ядро цілості [161].

Приклад

Як приклад подано результати експерименту з формування НКС. Знання видобувались з наступного тексту [172].

«Теорія фреймів належить до психологічних понять, які стосуються розуміння того, що ми бачимо та чуємо. Ці способи сприйняття трактуються з послідовної точки зору, на їх основі здійснюється концептуальне моделювання, доцільність одержаних моделей досліджується разом з різними проблемами, що виникають в цих двох областях».

Для усвідомлення того факту, що задана інформація в цих областях має єдиний зміст, людська пам'ять передусім повинна бути здатною вказувати цю інформацію зі спеціальними концептуальними об'єктами. В іншому випадку не вдається систематизувати інформацію, яка виглядає різною. В основі теорії фреймів лежить сприйняття фактів через співставлення одержаної ззовні інформації з конкретними елементами та значеннями.

6.3.1. Методи структурування

Усі методи виявлення понять ми поділили на:

- традиційні, основані на математичному апараті розпізнавання образів та класифікацій;
- нетрадиційні, основані на методології інженерії знань.

Якщо перші достатньо добре висвітлені в літературі, то другі ще маловідомі.

Приклад

Цікавий експеримент з виявлення понять описаний в роботі [85].

Тридцятьом студентам, що мають права водія, запропонували створити словник термінів предметної області за допомогою чотирьох методів:

- формування переліку понять (17%);
- опитування спеціалістів (інтерв'ювання) (35%);
- складання списку елементарних дій (18%);
- створення заголовку підручника (30%).

Цифри у дужках характеризують продуктивність відповідного методу, тобто показують, який відсоток понять із загального виявленого списку (702 терміни) був одержаний відповідним методом. Для класифікації понять були залучені ще два учасники експерименту, які розділили 702 виявлених поняття на сім категорій (методом сортування карток). Таблиця 6.3 відображає числові дані концептуалізації.

Таблиця 6.3 Дані концептуалізації

Категорії	Відсоток від загального числа термінів	Відсоток від загального числа термінів, одержаних відповідним методом			
		Список запитань	Інтерв'ювання	Список операцій	Створення заголовку
Пояснення	6	5,5	7,2	7,0	4,9
Загальні правила	22,0	43,6	18,9	36,8	4,9
Режимні правила	9,0	9,8	8,4	11,6	6,6
Поняття	42,0	18,4	38,9	8,5	77,7
Процедури	9,0	5,1	9,5	25,6	1,2
Факти	9,0	15,0	12,5	8,9	1,2
Інші поняття	3,0	2,6	4,6	1,6	3,5

Загалом результати показали, що для виявлення безпосередньо концептів найрезультативнішими виявились методи інтерв'ювання і створення заголовку підручника. Проте найбільша кількість загальних правил була породжена за методом списку дій. Отже, ще раз справдилося твердження про те, що немає «найкращого» методу, є методи, які підходять для тих чи інших ситуацій і типів знань.

Цікаво, що число правил – продукцій «якщо..., то...» – склало найбільший відсоток у всіх чотирьох методах. Це говорить про те, що популярна продукційна модель навряд чи є звичною для людських моделей репрезентації знань.

Методи виявлення зв'язків між поняттями

Концепти не існують незалежно, вони включені в загальну понятєву структуру за допомогою відношень. Виявлення зв'язків між поняттями при розробленні баз знань створює для інженера зі знань чимало проблем. Те, що знання в пам'яті – це деякі пов'язані структури, а не окремі фрагменти, загальновідомо і очевидно. Тим не менш основний наголос в наявних моделях подання знань робиться на поняття, а зв'язки вводять доволі примітивні (в основному, причин-но-наслідкові).

В останніх роботах з теорії ШІ щоразу більшу увагу приділяють взаємопов'язаності структур знань. Так, в роботі [182] введено поняття сценарію (script) як деякої структури

подання знань. Основу сценарію складає КОП (концептуальна організація пам'яті) та мета-КОПи – деякі узагальнюючі структури. Сценарії, у свою чергу, поділяються на фрагменти, чи сцени (chunks). Зв'язки між фрагментами – часові чи просторові, всередині фрагменту – найрізноманітніші: ситуаційні, асоціативні, функціональні тощо.

Усі методи виявлення таких зв'язків можна поділити на дві групи:

- формальні;
- неформальні (основані на додатковій роботі з експертом).

Неформальні методи виявлення зв'язків вигадує інженер зі знань для того, щоб змусити експерта вказати явні та неявні зв'язки між поняттями. Найбільш поширеним є метод «сортування карток» у групи [297], що широко застосовується і для формування понять. Другим неформальним методом є побудова замкнутих кривих. У цьому випадку експерта просять обвести замкнутою кривою пов'язані між собою поняття [293]. Цей метод може бути реалізований як на папері, так і на екрані дисплею. У цьому випадку можна говорити про залучення елементів когнітивної графіки [67].

Після того, як визначені зв'язки між поняттями, всі поняття немов би розпадаються на групи. Такі групи являють собою метапоняття, присвоєння імен яким відбувається на наступній стадії процесу структурування.

Методи виділення метапонять та деталізація понять «піраміда знань»

Процес утворення метапонять, тобто інтерпретації груп понять, одержаних на попередній стадії, як і зворотня процедура – деталізація понять, очевидно, є такими операціями, що принципово не піддаються формалізації. Вони вимагають високої спеціалізації експертів, а також наявності здатності до «наклеювання» лінгвістичних ярликів. Якщо на рис. 6.8 показані схеми узагальнення та деталізації на тривіальних прикладах, то в реальних предметних областях ця задача виявляється доволі трудомісткою. При цьому незалежно від того, формальними чи неформальними методами були виявлені поняття чи деталі понять, присвоєння імен чи їхня інтерпретація – завжди неформальний процес, в якому інженер зі знань просить експерта дати назву деякій групі понять чи окремих ознак.

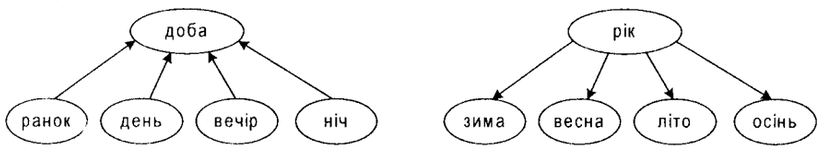


Рис. 6.8. Узагальнення та деталізація понять.

6.3.2. Еволюція систем одержання знань

Перше покоління таких систем появилося в середині 80-х років XX ст. – це так звані системи одержання знань (СОЗ) (TEIRESIAS [0], АРІАДНА [0]). Це засіб наповнення так званих „порожніх" ЕС, тобто систем, з БЗ яких вилучені знання (наприклад, EMYCIN – EMPTY MYCIN, спустошена медична ЕС MYCIN зі спеціальною діалоговою системою заповнення бази знань TEIRESIAS). Їхні автори вважали, що прямий діалог експерта з комп'ютером через СОЗ допоможе скоротити життєвий цикл розроблення. Однак досвід створення та впровадження СОЗ продемонстрував недосконалість такого підходу. Основні недоліки СОЗ першого покоління:

- слабке опрацювання методів вилучення та структурування знань;

- жорсткість моделі подання знань, вбудованої в СОЗ і прив'язаної до програмної реалізації;
- обмеження на предметну область.

Отже, традиційна схема розроблення СОЗ І-го покоління: створення конкретної ЕС спустошення БЗ розроблення СОЗ для нових наповнень БЗ формування нової БЗ для іншої ЕС виявилась непридатною для промислового застосування.

Друге покоління СОЗ з'явилося наприкінці 80-х років минулого століття і було орієнтоване на ширший модельний підхід [19] з наголосом на попередньому детальному аналізі предметної області. Так, в Європі широке застосування одержала методологія KADS (Knowledge Acquisition and Documentation Structuring), в основі якої лежить поняття інтерпретаційної моделі, яка дозволяє процеси вилучення, структурування та формалізації знань розглядати як «інтерпретацію» лінгвістичних знань в інші подання та структури.

KADS-методологія

Рис. 6.9 демонструє перетворення знань за методологією KADS [218] через специфікацію п'ятьох кроків аналізу «ідентифікація – концептуалізація – гносеологічний рівень – рівень аналізу виконання» та стадії чи простору проектування. Результатом аналізу є концептуальна модель експертизи, що складається з чотирьох рівнів (рівня області – рівня виведення – рівня задачі – стратегічного рівня), яка згодом вводиться в простір проектування. При розв'язуванні реальних задач KADS використовує бібліотеку інтерпретаційних моделей, що описують загальні експертні задачі, такі як діагностика, моніторинг та ін., без конкретного наповнення об'єктами предметної області. Інтерпретаційна модель являє собою концептуальну модель без рівня області. На основі вилучених лінгвістичних даних відбувається відбір, комбінація та вкладення верхніх рівнів моделі, тобто рівнів виведення та задачі, які наповнюються конкретними об'єктами й атрибутами з рівня області та відображають у результаті концептуальну модель задачі, яку розглядають. На рис. 6.9 подана модель життєвого циклу KADS.

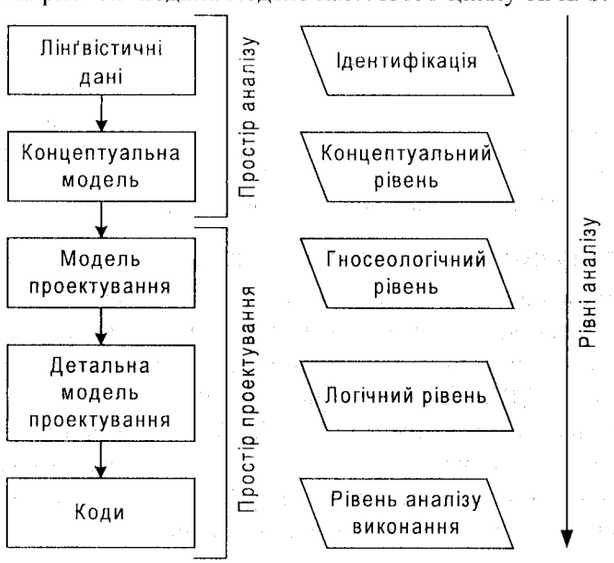


Рис. 6.9. Модель життєвого циклу KADS.

Перші системи програмної підтримки KADS-методології зображені набором інструментальних засобів KADS Power Tools. У цей набір входять наступні системи: редактор протоколів PED (Protocol Editor); редактор систем понять (Concept Editor); редактор концептуальних моделей CME (Conceptual Model Editor) та IM-бібліотекар IML (Interpretation Model Librarian).

Редактор протоколів – програмний засіб, що допомагає інженеру зі знань у здійсненні аналізу знань про предметну область на лінгвістичному рівні. При роботі зі знаннями на цьому рівні вихідним матеріалом є тексти (протоколи) – записи інтерв'ю з експертом, протоколи «думок вголос» та будь-які інші тексти, корисні з точки зору інженера зі знань. Редактор протоколів реалізований як гіпертекстова система, що забезпечує виділення фрагментів у тексті, який аналізують, встановлення зв'язків між фрагментами, групування фрагментів, анотування фрагментів. Фрагменти можуть мати будь-яку довжину – від окремого слова до протоколу в цілому.

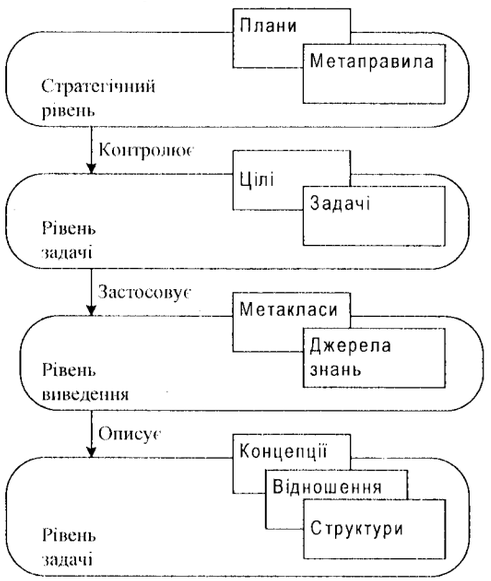


Рис. 6.10. Основні моделі KADS

Фрагменти можуть не перекривати один одного. Можливі наступні типи зв'язків між фрагментами:

- анотація (зв'язок між фрагментом протоколу та деяким текстом, введеним інженером зі знань для специфікації цього фрагменту);
- член групи (зв'язок між фрагментом та назвою – ім'ям групи фрагментів; об'єднання фрагментів у групу дозволяє інженеру зі знань структурувати протоколи. При цьому група фрагментів одержує унікальне ім'я);
- поіменований зв'язок (зв'язок між двома фрагментами, ім'я зв'язку обирає інженер зі знань);
- понятійний зв'язок (поіменований зв'язок між фрагментом та поняттям, зазвичай використовується, якщо фрагмент містить визначення понять).

Редактор понять допомагає інженеру зі знань організувати предметні знання у вигляді набору понять і відношень, що їх поєднують. Кожне поняття має ім'я і може мати

атрибути; кожен атрибут може мати значення. Які атрибути використовуються – це визначає інженер зі знань із врахуванням специфіки предметної області. За допомогою редактора понять інженер зі знань може вводити довільні відношення між поняттями і створювати ієрархічні структури з того чи іншого поняття. Існує єдине відношення (ISA), семантика якого «вбудована» в редактор. Якщо інженер зі знань встановлює це відношення між двома поняттями, то має місце успадкування атрибутів. Обравши атрибут «визначення», користувач зможе побачити на екрані графічне зображення структури виведення, елементами якої є джерела знань та метакласи. Як джерела знань, так і метакласи мають свої набори атрибутів; інженер зі знань може переглянути їх, вказуючи на відповідний елемент.

Психосемантика

Окрім ідеології KADS, на розроблення СОЗ II-го покоління значний вплив мали методи суміжних наук, зокрема психосемантики, одного з молодих напрямів прикладної психології [140], перспективного інструменту, що дозволяє реконструктурувати семантичний простір у пам'яті і тим самим моделювати глибинні структури знань експерта (див. наступний розділ). Уже перші вклади психосемантики в ШІ в середині 80-х років XX ст. дозволили одержати достатньо наглядні результати [85]. У подальшому розвиток цих методів відбувався по лінії розроблення зручних пакетів прикладних програм, основаних на методах багатомірного шкалювання, факторного аналізу, а також спеціалізованих методів опрацювання репертуарних решіток [177] (див. наступний розділ). Прикладами СОЗ такого типу є системи KELLY [153], MEDIS [6]. Специфіка конкретних додатків вимагала розвитку також «нечисельних» методів, які використовують парадигму логічного виведення. Прикладами систем цього напряму є системи ETS [210] і AQUINAS [212]. Успіхи СОЗ II-го покоління дозволили значно розширити ринок ЕС, який до кінця 80-х років XX ст. оцінювався в 300 млн. доларів на рік [145]. Тим не менше і ці системи не були вільними від недоліків, до найважливіших з яких можна віднести:

- недосконаліть інтерфейсу, в результаті чого непідготовлені експерти не можуть оволодіти системою і відмовляються від неї;
- важкість налаштування на конкретне професійне мовне середовище;
- необхідність розроблення дорогих лінгвістичних процесорів для аналізу звичайних-мовних повідомлень та текстів.

Третє покоління СОЗ – KEATS [223], MACAO [195], NEXPERT-OBJECT [45] – перенесло акцент у проектуванні з експерта на інженера зі знань. Нові СОЗ – це програмні засоби для аналітика, глибокі, а головне такі, що використовують *графічні* можливості сучасних робочих станцій та досягнення CASE-технологій (Computer-Aided Software Engineering). Ці системи дозволяють не задавати заздалегідь інтерпретаційну модель, а формувати структуру БЗ динамічно. Існують різні класифікації СОЗ – за виразністю та потужністю інструментальних засобів; за узагальненими характеристиками; в межах структурно-функціонального підходу [45].

Методологічні проблеми

Основна проблема, що постає перед розробниками, – відсутність теоретичного базису процесу вилучення та структурування знань – породжує дочірні вужчі питання та казуси на всіх етапах створення інтелектуальних систем. Навіть ретельно опрацьована

методологія KADS, описана в попередньому розділі, страждає громіздкістю та явною надмірністю. Нижче перелічені найбільш загальні проблеми, що виникають в послідовності, яка відповідає стадіям життєвого циклу:

- розмитість критеріїв вибору задачі;
- слабке опрацювання теоретичних аспектів процесів вилучення знань (філософські, лінгвістичні, психологічні, педагогічні, дидактичні та інші аспекти), а також відсутність обґрунтованої класифікації методів вилучення знань та розкиданість термінології;
- відсутність єдиного теоретичного базису процедури структурування знань;
- жорсткість моделей подання знань, що змушує розробників зменшувати та урізати реальні знання експертів;
- недосконаліть математичного базису моделей подання знань (дескриптивний, а не конструктивний характер більшості наявних математичних моделей);
- емпіричність процедури вибору програмного інструментарію та процесу тестування (відсутність критеріїв, розрізнені класифікації), тестування (відсутність критеріїв, різноманітність класифікації тощо).

Технологічні проблеми

Більша частина технологічних проблем є звичайним наслідком методологічних і спричинена ними. Найсерйознішими з технологічних проблем є:

- відсутність концептуальної цілісності та узгодженості між окремими прийомами та методами інженерії знань;
- недостатність чи відсутність кваліфікованих спеціалістів у галузі інженерії знань;
- відсутність техніко-економічних показників оцінки ефективності ЕС;
- незважаючи на велику кількість методів вилучення знань (фактично понад 200 в огляді [213]), практична недоступність методичних матеріалів з практики здійснення сеансів вилучення знань;
- явна неповнота та недостатність наявних методів структурування знань, відсутність класифікацій та рекомендацій з вибору належного методу;
- незважаючи на багатство ринку програмних засобів, недостатність промислових систем підтримки розроблення та їх вузька спрямованість (залежність від платформи, мови реалізації, обмежень предметної області), розрив між мовами ПЗ та мовами, вбудованими в «оболонки» ЕС;
- жорсткість програмних методів, їх низька адаптивність, відсутність індивідуальної спрямованості на користувача та предметну область;
- слабкі графічні можливості програмних засобів, недостатнє врахування когнітивних та ергономічних факторів.

Запитання для повторення та контролю знань

1. Як класифікуються методи практичного видобування знань?
2. Який основний принцип поділу методів практичного видобування знань?
3. Які методи видобування знань називаються комунікативними?

4. На які дві групи можна поділити комунікативні методи?
5. Які методи видобування знань називаються ігровими?
6. Як класифікуються предметні області з точки зору їх документованості?
7. Як класифікуються предметні області за ступенем структурованості знань?
8. Які методи видобування знань належать до пасивних?
9. Які методи видобування знань належать до індивідуальних активних?
10. Які існують різновиди методу спостережень?
11. Які виникають труднощі під час складання протоколу «думок вголос»?
12. Як діляться питання під час читання лекцій?
13. Обґрунтуйте порівняльні характеристики пасивних методів видобування знань.
14. Які існують способи здійснення анкетування?
15. Які рекомендації під час складання анкет?
16. Що розуміють під інтерв'ю?
17. Як відбувається класифікація запитань під час інтерв'ю?
18. Що таке вільний діалог?
19. Які стадії видобування знань під час вільного діалогу?
20. Як і порівняльні характеристики активних індивідуальних методів видобування знань?
21. Які методи видобування знань належать до групових активних?
22. Які переваги групових методів?
23. У чому полягає метод «мозкової атаки»?
24. Обґрунтуйте порівняння активних групових методів видобування знань.
25. У чому полягає метод експертної гри? Наведіть приклади.
26. У чому полягає метод рольових ігор? Наведіть приклади.
27. У чому полягає метод ігор з тренажерами? Наведіть приклади.
28. У чому полягає метод комп'ютерних експертних ігор? Наведіть приклади.
29. Які методи видобування знань належать до текстологічних?
30. Сформуйте задачу видобування знань з текстів.
31. Яка схема видобування знань зі спеціальних текстів?
32. Які компоненти наукового тексту?
33. Що таке мікроконтекст та макроконтекст?
34. Що таке основні моменти розуміння тексту?
35. Що таке семантична структура тексту?
36. Які компоненти впливають на інтерпретацію тексту?
37. Які Ви знаєте методи виявлення понять?
38. Як діляться методи виявлення зв'язків між поняттями?
39. Які існують методи виділення метапонять та деталізації понять?

40. Наведіть еволюцію систем одержання знань.
41. Що таке KADS-методологія?
42. Які основні моделі KADS?
43. Які існують типи зв'язків між фрагментами?
44. Що таке психосемантика?
45. Які існують методологічні проблеми KADS?
46. Які існують технологічні проблеми KADS?

Завдання для самостійного розв'язування

Чому задача придбання знань є вузьким місцем в проектуванні інтелектуальних систем? Які рішення пропонуються для запобігання такої ситуації?

РОЗДІЛ 7

НОВІ ТЕНДЕНЦІЇ ТА ПРИКЛАДНІ АСПЕКТИ ІНЖЕНЕРІЇ ЗНАНЬ

- Латентні структури знань і психосемантика
- Метод репертуарних решіток
- Керування знаннями
- Візуальне проектування баз знань як інструмент пізнання
- Проектування гіпермедіа БД і адаптивних навчальних систем

У розділі розглядаються прикладні аспекти інженерії знань, використання латентних структур знань та психосемантики для видобування глибинних знань. Розглянуто метод репертуарних решіток, керування знаннями та проектування бази знань. В кінці розділу описано використання адаптивних навчальних систем та їх проектування.

7.1. Латентні структури знань і психосемантика

Більшість систем, що видобувають знання, полегшують складний і трудомісткий процес формування баз знань і реалізують прямий діалог з експертом. Однак виявлені в такий спосіб структури знань часто відбивають лише поверхневу складову знань експерта, не зачіпаючи їхньої глибинної структури. Такий самий недолік має більшість методів безпосереднього видобування знань.

Щоб видобути глибинні шари експертного знання, можна скористатися методами психосемантики – науки, що виникла на стику когнітивної психології, психолінгвістики, психології сприйняття і досліджень індивідуальної свідомості. Психосемантика досліджує структури свідомості через моделювання індивідуальної системи знань [139] і виявлення тих категоріальних структур свідомості, які можуть не усвідомлюватися (латентні, імпліцитні або приховані).

7.1.1. Семантичні простори і психологічне градуювання

Основним методом експериментальної психосемантики є метод реконструкції суб'єктивних семантичних просторів. Тут психосемантика тісно пов'язана із психолінгвістикою й лінгвістичною семантикою – з методологією виявлення значень слів, лексикографією, «відмінковою граматикою» і структурними дослідженнями [9]. Однак лінгвістичні методи, в основному, спрямовані на аналіз текстів, відчужених від суб'єкта, від його мотивів і задумів.

Психолінгвістичні методи звертаються безпосередньо до випробуваного. Більшість із них пов'язані з різними формами суб'єктивного градуювання. Перед випробуваним ста-

виться завдання оцінити «подібність знань» за допомогою деякої градуйованої шкали, наприклад, від 0 до 5 або від 0 до 9. У цьому випадку дослідник отримує чисельно подані стандартизовані дані, що легко піддаються статистичному опрацюванню.

Психосемантика – як один з нових напрямів сучасної психології [140] – відразу була оцінена фахівцями в галузі штучного інтелекту як перспективний інструмент, що дозволяє реконструювати семантичний простір пам'яті як психологічну модель глибинної структури знань експерта. Уже перші досягнення психосемантики в середині 80-х XX ст. років дозволили одержати досить наочні результати. У психології семантичні простори виступають як модель категоріальної структури індивідуальної свідомості. При концептуальному аналізі знань структуру семантичного простору експерта можна вважати основою для формування поля знань. При цьому окремі параметри семантичного простору відповідають різним компонентам поля (розмірність простору співвідноситься зі складністю поля, виділені понятійні структури – з метапоняттями, змістовні зв'язки між поняттями-стимулами – це суть відношення тощо).

Найлегше ознайомитися з експериментальною психосемантикою на прикладі, пов'язаному з виявленням структури і розмірності семантичного простору знань із деякої предметної області [140]. В основі побудови семантичних просторів, як правило, лежить статистична процедура (наприклад, факторний аналіз [1], багатомірне градуювання [169] або кластерний аналіз [63]), що дозволяє групувати ряд окремих ознак опису в місткіші категорії-фактори. Говорячи мовою поля знань, це – побудова концептів вищого рівня абстракції. При геометричній інтерпретації семантичного простору значення окремої ознаки відображається як точка або вектор із заданими координатами усередині n-мірного простору, координатами якого виступають виділені фактори. Побудова семантичного простору, таким чином, включає перехід до опису предметної області на вищому рівні абстракції, тобто перехід від мови, що містить великий алфавіт ознак описування, до місткішої мови концептуалізації, що містить менше число концептів і виступає своєрідною метамовою стосовно першої. Залежно від досвіду та професійної компетентності тих, кого випробовують, розмірність простору і розташування в ній первинних понять може істотно варіюватися. Ця особливість семантичних просторів може бути використана на стадії контролю в процесі навчання, при тестуванні експериментів і користувачів.

Так, для перевірки знань і розуміння англійської мови в роботі [168] було взято десять найпоширеніших прийменників, які досить важко перекладати. На екрані дисплея випробовуваному пропонували кілька прийменників і запитували, чи часто в нього викликає труднощі вибір одного із цих прийменників. Ступінь утруднення оцінювалася в балах від 1 до 9. На цих даних методами багатомірного градуювання було побудовано структуру складності вживання англійських прийменників з погляду носія російської мови. Ця модель істотно залежить від рівня знань. Так, модель новачка не є організованою структурою. У людей, що мають певні навички, виявилася деяка структурованість семантичного простору, у ньому чітко позначилися пари й трійки подібних прийменників.

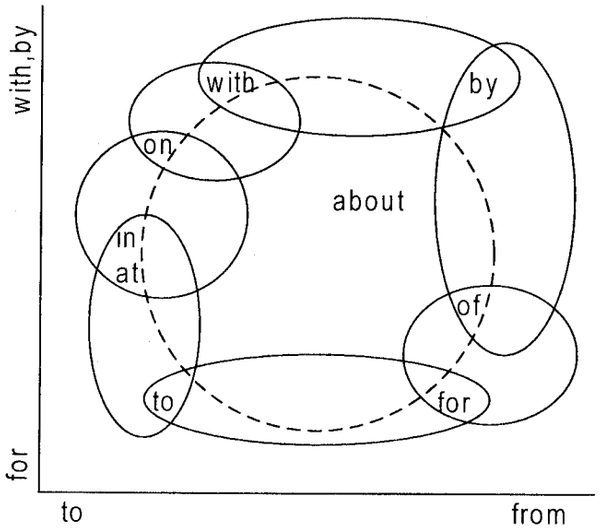


Рис. 7.1. Семантичний простір близькості англійських прийменників.

Найбільш чітка й зв'язна структура наведена на рис. 7.1. За її допомогою можна пояснити особливості диференціації прийменників в англійській мові. Основа структури подається у вигляді кола, близькі крапки на якому відповідають прийменникам, які важко диференціювати. Структура спирається на дві ортогональні осі. Вісь абсцис відповідає прийменникам напрямку руху, а вісь ординат – прийменникам мети, або засобу.

На підставі одержаних методами психосемантики моделей можна здійснювати контроль знань. При аналізі індивідуальних семантичних просторів виявляються питання, які не засвоєні й не уклалися в систему. Контроль структури знань здійснюється на основі зіставлення семантичних просторів хороших фахівців і новачків (студентів, слухачів, молодих фахівців). Ступінь погодженості семантичних просторів (їхньої розмірності, ознаки й конфігурації понять) буде визначати рівень знань новачка.

Однак тут необхідно врахувати, що семантичні простори двох кваліфікованих фахівців можуть бути різними, тому що містять індивідуальні розходження сприйняття, що віддзеркалюють досвід і характер діяльності людини. Тому не завжди можна формально порівняти семантичні простори експерта й новачка, варто попередньо вивчити семантичні простори декількох фахівців, а потім уже робити порівняння.

У роботі [85] описаний подібний експеримент. Були отримані когнітивні структури знань досвідченого льотчика-винищувача й пілота-новачка з використанням двох методів: багатовимірного градуювання (алгоритм MDS – Alscal) і мережного градуювання із врахованими зв'язками (алгоритм Pathfinder). Обидва алгоритми засновані на використанні оцінок психологічної близькості. Досвідчений пілот і новачок оцінювали всі можливі парні сполучення 30-ти пов'язаних з польотом понять, приписуючи числа від 0 до 9 кожній парі, де 0 позначав найслабший ступінь зв'язку між поняттями, а 9 – найсильніший. Ці оцінки потім опрацьовувалися із застосуванням обох алгоритмів градуювання.

Відповідно до алгоритмів MDS кожний концепт, що виражає деяке поняття, вміщують в k-вимірний простір таким чином, що відстань між точками відбиває психологічну

близькість відповідних концептів. Алгоритм Pathfinder будує семантичну мережу. Дуги можуть бути або орієнтованими (несиметричне відношення), або неорієнтованими (симетричне відношення). Обидва методи забезпечують стискання великих обсягів даних (у формі попарних оцінок) до меншого набору параметрів; але націлені вони на виявлення різних властивостей досліджуваних структур. Якщо в алгоритмі Pathfinder центром уваги є локальні відношення між концептами, то алгоритм MDS забезпечує ширше розуміння властивостей простору концептів, що підлягає метризації.

Результуючі когнітивні структури виявилися близькими для пілотів-винищувачів з однаковим рівнем досвіду, але були різними для різних груп випробуваних. Автори експериментів виявили, що за когнітивною структурою, характерною для пілота-винищувача, можна встановити, новачок він чи досвідчений пілот. Нарешті, здійснений ними ж аналіз когнітивних структур виявив наявність концептів і відношень, спільних для подань досвідченого фахівця й новачка, і, крім того, ряд концептів і відносин, які з'явилися тільки в одному з подань. Прямим розвитком розглянутої роботи стала експертна система керування повітряним боєм ACES [248].

Аналогічне дослідження механізмів нагромадження досвіду було здійснене в області програмування на ЕОМ [36]. За допомогою методів градуювання було показано, що один з аспектів досвіду програміста – організація знань відповідно до задуму програми, або семантики, а не відповідно до синтаксису.

Мережне подання абстрактних понять програмування на основі оцінок зв'язності концептів у програмістів показане на рис. 7.2. Експеримент показав, що всіх програмістів на основі аналізу структури семантичного простору можна розбити на три групи: новачки, недосвідчені фахівці середнього рівня, досвідчені фахівці. Цей висновок збігається з результатами, отриманими в роботі [36]. Крім того, досліджувалася еволюція когнітивної структури програміста в міру його просування від новачка до досвідченого фахівця.

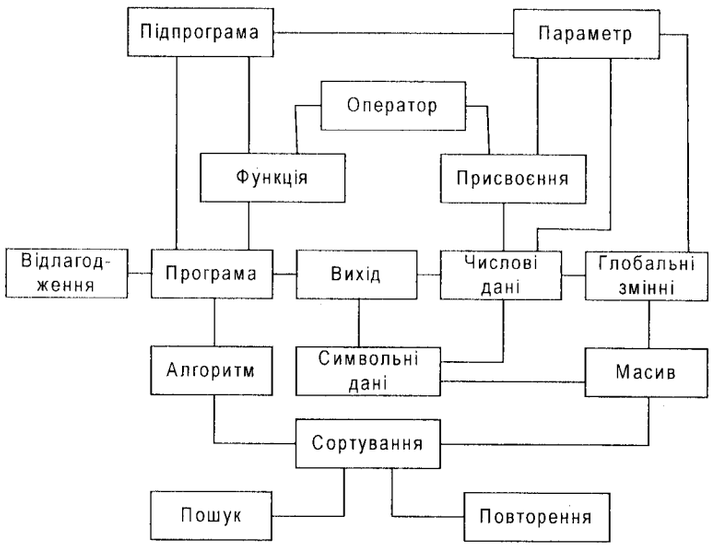


Рис. 7.2. Асоціативна мережа структури знань експерта-програміста.

Інтерпретація виявлених відношень (зв'язків) між поняттями вимагає додаткових зусиль від колективу розроблювачів інтелектуальних систем. Так, наприклад, для озна-

чення дуг на рис. 7.2 необхідним став додатковий експеримент, учасникам якого була запропонована пара понять і поставлено завдання дати словесний опис зв'язку між поняттями пари. Результати наведено в табл. 7.1. Отже, асоціативна мережа на рис. 7.2 може бути перетворена в семантичну.

Таблиця 7.1. Опис зв'язку між поняттями

Пов'язані пари понять	Відношення між пов'язаною парою понять
Підпрограма – програма	Є частина
Символьні дані – вихід	Є тип
Параметр – програма	Використовується
Програма – вихід	Робить
Сортування – пошук	Містить у собі
Чисельні дані – параметр	Може бути
Функція – оператор	Є
Функція – програма	Є частина
Налагодження – програма	Піддається
Вихід – чисельні дані	Складається з...
Масив – символьні дані	Може складатися з...
Функція – підпрограма	Є
Присвоєння значень – параметр	Використовується для
Масив – глобальна зміна	Може бути
Масив – чисельні дані	Може складатися з...
Повторення – сортування	Є частина
Програма – алгоритм	Є виконання
Сортування – алгоритм	Є
Сортування – чисельні дані	Робиться на
Присвоєння – оператор	Є

Всі вищезгадані методи (включаючи кластерний аналіз) можна віднести до методів психологічного градуювання. Їхню основу становлять алгоритми перетворення складних структур даних у зрозумілішу форму, що передбачається психологічно змістовною. Результуюче подання залежить від методу:

- кластерний аналіз породжує деревоподібну структуру;
- багатомірне градуювання й факторний аналіз – просторову;
- алгоритм MDS – мережну;
- репертуарні решітки породжують конструкти або мета-виміри [177].

Формальна методологія психосемантичного градуювання дозволяє частково автоматизувати процес структурування знань і одержувати «когнітивний розріз» його подань про предметну область. Методологія градуювання дозволяє виявляти структури знання непрямим шляхом при одержанні відповідей від експертів на досить прості запитання (наприклад, «наскільки близькі поняття X1 і X2» замість «скажіть, який зв'язок між X1 і X2, як вони впливають один на одного»).

Майже всі експерименти дозволили виявити одну закономірність. Розмірність семантичного простору з підвищенням рівня професіоналізму зменшується. Цей висновок збігається з відомими твердженнями когнітивної психології про те, що процес пізнання супроводжується узагальненням.

Побудова семантичного простору зазвичай включає три послідовних кроки.

1. Вибір і застосування відповідного методу оцінки семантичної подібності. Цей крок містить у собі експеримент із випробовуваними особами, яким пропонується оцінити спільність поданих стимульних ознак на деякій шкалі.
2. Побудова структури семантичного простору на основі математичного аналізу отриманої матриці подібності. При цьому відбувається зменшення числа досліджуваних понять за рахунок узагальнення й одержання генералізованих осей.
3. Ідентифікація, інтерпретація виділених факторних структур, кластерів або груп об'єктів, осей тощо. На цьому кроці необхідно знайти значеннєві еквіваленти, мовні «ярлики» для виділених структур. Тут великого значення набуває лінгвістичне чуття й професіоналізм фахівця, що здійснює дослідження, і випробуваних експертів. Часто до інтерпретації залучають групу експертів.

7.1.2. Методи багатовимірного градуювання

Надалі розвиток методів психосемантики пішов шляхом розроблення зручних пакетів прикладних програм, заснованих на методах багатовимірного градуювання (БГ), факторного аналізу, а також спеціалізованих методів (статистичного) опрацювання репертуарних решіток [177]. Прикладами пакетів такого типу є системи KELLY [153], MADONNA [168], MEDIS [181]. З іншого боку, специфіка ряду конкретних додатків, насамперед – в інженерії знань, вимагала також розвитку інших (не чисельних) методів опрацювання психосемантичних даних, що використовують – у тій або іншій формі – парадигму логічного виведення на знаннях. Яскравим прикладом цього напряму є система AQUINAS [211]. Однак аналіз практичного застосування систем обох типів до завдань інженерії знань приводить до висновку про недосконалість наявних методик і необхідності їхнього розвитку відповідно до сучасних вимог інженерії знань. Найбільші перспективи в цій галузі, очевидно, мають методи багатовимірного градуювання.

Багатовимірне градуювання (БГ) сьогодні – це математичний інструментарій, призначений для опрацювання обробки даних про попарні подібності, зв'язки або відношення між аналізованими об'єктами з метою подання цих об'єктів у вигляді точок деякого координатного простору. БГ є одним із розділів прикладної статистики, наукової дисципліни, що розробляє й систематизує поняття, прийоми, математичні методи й моделі, призначені для збирання, стандартного запису, систематизації й опрацювання статистичних даних з метою їхнього лаконічного подання, інтерпретації й одержання наукових і практичних висновків. Традиційно БГ використовується для рішення трьох типів завдань:

- пошук і інтерпретація латентних (тобто схованих, безпосередньо неспостережуваних) змінних, що пояснюють задану структуру попарних відстаней (зв'язків, наближеностей);
- верифікація геометричної конфігурації системи аналізованих об'єктів у координатному просторі латентних змінних;
- стиснення вихідного масиву даних з мінімальними втратами в їхній інформативності.

Незалежно від завдання БГ завжди використовується як інструмент наочного подання (візуалізації) вихідних даних. БГ широко застосовується в дослідженнях з антропології, педагогіці, психології, економіці, соціології [60].

В основі такого підходу лежить інтерактивна процедура суб'єктивного градуювання, коли випробовуваній особі (тобто експертові) пропонується оцінити подібність між різними елементами за допомогою деякої градуйованої шкали (наприклад, від 0 до 9, або від –2 до +2). Після такої процедури аналітик має у своєму розпорядженні чисельно подані стандартизовані дані, що піддаються опрацюванню наявними пакетами прикладних програм, що реалізують різні алгоритми формування концептів вищого рівня абстракції та будують геометричну інтерпретацію семантичного простору в евклідовій системі координат.

Основний тип даних у БГ – міри близькості між двома об'єктами (i, j) – dij. Якщо міра близькості така, що найбільші значення dij відповідають парам найбільш схожих об'єктів, то dij – міра подібності, якщо, навпаки, найменш схожим, то dij – міра розходження.

БГ використовує дистанційну модель розходження, використовуючи поняття відстані в геометрії як аналогію подібності й розходження понять. Щоб функція d, визначена на парах об'єктів (a, b), була евклідовою відстанню, вона повинна задовольняти наступні чотири аксіоми:

d(a, b)≥0,

d(a, a)=0,

d(a, b)=d(b, a),

d(a, b)+d(b, c)≥d(a, c).

Тоді, відповідно до звичайної формули евклідової відстані, міра розходження двох об'єктів i та j зі значеннями ознаки k в об'єктів i та j відповідно x_{ik} і x_{jk}:

$$\delta_{ij} = d_{ij} = \left[\sum_{k=1}^{K}(x_{ik} - x_{jk})^2\right]^{1/2}.$$

Дистанційна модель була багаторазово перевірена в соціології та психології [287], що дає можливість оцінити її придатність для використання.

У більшості робіт з БГ використовується матрична алгебра. Геометрична інтерпретація дозволяє зобразити абстрактні поняття матричної алгебри в конкретній графічній формі. Для полегшення інтерпретації рішення завдання БГ до попередньо оціненої матриці координат стимулів X застосовується обертання.

Серед безлічі алгоритмів БГ широко використовуються різні модифікації метричних методів Торгерсона [311], а також неметричні моделі, наприклад, Крускала [271].

При порівнянні методів БГ із іншими методами аналізу, теоретично застосовними в інженерії знань (ієрархічний кластерний аналіз або факторний аналіз), БГ виграє за рахунок можливості дати наочне кількісне координатне зображення, найчастіше найпростішої й тому легше інтерпретованої експертами.

7.1.3. Використання метафор для виявлення «прихованих» структур знань

Незважаючи на згадану близькість завдань, інженерія знань і психосемантика істотно відрізняються як у теоретичних підвалинах, на яких вони базуються, так і в практичних методиках. Але головна відмінність полягає в тім, що інженерія знань спрямована на виявлення – як остаточного результату – моделі міркувань, динамічної або операційної складової ментального простору (або функціональної структури поля знань S_f), у той час як психосемантика, намагаючись подати ментальний простір у вигляді евклідового

простору, дозволяє робити видимою статичну структуру взаємного «розташування» об'єктів у пам'яті у вигляді проекцій скупчень об'єктів (концептуальна структура S_k).

Крім цього, зазначимо ряд недоліків методів психосемантики з погляду практичної інженерії знань.

1. Оскільки в основі психосемантичного експерименту лежить процедура вимірювання суб'єктивних відстаней між пропонованими стимулами, то й результати опрацювання такого експерименту, як правило, використовують геометричну інтерпретацію – евклідів простір невеликого числа вимірів (найчастіше – двовимірний). Таке сильне спрощення моделі пам'яті може призвести до неадекватних баз знань.
2. Природність ієрархії як глобальної моделі понятійних структур свідомості служить методологічною базою ОСП. Крім того, і в природній мові поняття явно тяжіють до різних рівнів узагальнення. Однак у більшості прикладних пакетів не передбачено розбивки семантичного простору на рівні, що відбивають різні ступені спільності понять, включених в експериментальний план. У результаті одержувані кластери понять, просторово ізольовані в геометричній моделі градуювання, мають таксономічно неоднорідний характер і важко піддаються інтерпретації.
3. Єдині відносини, які виявляють процедури психосемантики, – це «далеко – близько» за деякою шкалою. Для проектування й побудови баз знань виявлення відношень є на порядок складнішим завданням, ніж виявлення понять. Тому семантичні простори, отримані в результаті градуювання й кластеризації, треба піддавати подальшому опрацюванню на предмет визначення відношень, особливо функціональних і каузальних.

Не варто очікувати, що ці протиріччя можуть бути вирішені швидко й безболісно, у силу того, що математичний апарат, покладений в основу всіх пакетів прикладних програм зі психосемантики, має певні межі застосування. Однак одним з можливих шляхів зближення без порушення чистоти процедури бачиться розширення простору конкретних об'єктів-стимулів предметної області за рахунок додавання деяких абстрактних об'єктів зі світу метафор, які змусять випробуваного експерта вийти за рамки об'єктивності у світ суб'єктивних уявлень, які найчастіше більшою мірою впливають на його міркування й модель прийняття рішень, ніж традиційні правильні погляди.

Нижче описано підхід, що дозволяє вивести експерта за межі традиційної установки і тим самим виявити суб'єктивні, часто приховувані або приховані структури його професійного досвіду.

Більшість результатів, отриманих у когнітивній психології, підтверджують, що в людини (у тому числі й в експерта) формальні знання про світ (зокрема, про ту предметну область, де він є експертом) і його особистий поведінковий досвід не можуть існувати ізольовано, вони утворять цілісну, стабільну структуру. У західній літературі за цією структурою закріпилася назва «модель (або картина) світу». Принциповою властивістю моделі світу є те, що її структура не дана людині – її носію – в інтроспекції, вона працює на істотно латентному, неусвідомлюваному рівні, найчастіше взагалі нічим не відзначеному (у символічній формі) на поверхні вербальної свідомості.

Вивчення моделі світу людини є завданням когнітивної психології, психосемантики й інших споріднених дисциплін. Що ж стосується експерта як об'єкта спостережень інженерії знань, то тут одна із проблем, у рішенні яких може допомогти психосемантика, пов'язана з необхідністю (і неминучістю) відділення досвіду експерта від його

об'єктивних знань у процесі формалізації та структурування останніх для експертних систем.

Строго кажучи, неможливо вказати ту грань, за якою знання (які можна формалізувати й видобувати) переходять у досвід (тобто в те, що залишається унікальною, невідчужуваною власністю експерта). Більш коректно, очевидно, говорити про деякий континуум, детальність градації якого може залежати від конкретного завдання.

Наприклад, можна виділити такі три рівні:

1) знання, призначені для викладу або доказу (аргументації), наприклад, на міждисциплінарному рівні або для популярної лекції (вербальні);
2) знання, які застосовуються в реальній практиці, – знання ще вербалізовані, але вже такі, що не рефлектуються;
3) власне досвід, тобто знання, що лежать на найбільш глибокому, неусвідомлюваному рівні, відповідальні за ті рішення експерта, які зовні (у тому числі й для нього самого) виглядають як миттєве осяяння або «інсайт», інтуїтивний творчий акт (інтуїтивні).

Класична методика психосемантичного експерименту також не дозволяє виділити з його результатів інтуїтивний рівень. Це видно із самої тестової процедури. Чи просять випробувану особу оцінити подібність-розходження стимулів прямо або ж пропонують оцінити їхню відповідність деяким «конструктам» – у будь-якому разі випробуваний вільно або мимоволі настроюється на необхідність доказовості своєї відповіді в термінах об'єктивних властивостей стимулів.

Більшість методів видобування знань орієнтовані на верхні – вербальні або вербалізовані – рівні знання. Необхідний непрямий метод, орієнтований на виявлення прихованих переваг практичного досвіду або операційних складових досвіду. Таким методом може слугувати метафоричний підхід. Метафоричний підхід, уперше описаний із чисто лінгвістичних позицій [301], а також із позицій практичної психології [53], був видозмінений для потреб інженерії знань. Наприклад, в експериментах з об'єктивного порівняння мов програмування між собою були також використані два метафоричних «світи» – світ тварин і світ транспорту.

У рамках цього підходу вдалося експериментально довести наступні тези:

- метафора працює як фільтр, що виділяє за допомогою підбору адекватного об'єкту порівняння певні властивості основного об'єкту (тобто того, про яке власне і йде мова); ці виділені властивості мають суто операційний характер, що проявляється на рівні поліморфізму методів, тому що метафора за самою своєю суттю виключає можливість порівняння об'єктів за їх внутрішніми об'єктивними властивостями;
- метафора має на меті скоріше не повідомити що-небудь про певний об'єкт (тобто відповісти на запитання «що це?»), а закликати до певного відношення до нього, вказати на якусь парадигму, що говорить про те, як варто поводитися стосовно заданого об'єкту;
- суб'єктивному зрушенню у відношенні до основного об'єкту (наприклад, до мови програмування) сприяє також і зрушення у сприйнятті об'єкту порівняння (наприклад, до конкретної тварини) у силу вищевказаної специфіки властивостей, які фільтрують метафорою; тому об'єкт порівняння виступає в метафорі не за своїм прямим призначенням, тобто це не просто «лев» як представник фауни, а втілення сили, спритності й могутності;

- у тому випадку, коли метафора зіставляє не одиничні об'єкти, а деякі їхні множини, у яких об'єкти зв'язані осмисленими відношеннями, простір об'єктів порівняння повинен бути ізоморфним щодо простору основних об'єктів за системою зазначених відношень.

На ці тези спирається запропонована модифікація класичної методики зіставлення об'єктів, застосована, наприклад, в оцінних решітках Келлі [267]. Під час здійснення експерименту була використана система MEDIS [181]. Ця система дозволяє планувати, здійснювати експеримент і опрацьовувати дані довільного психосемантичного експерименту. Крім класичної парадигми багатовимірного градуювання, система MEDIS містить у собі деякі можливості тесту репертуарних решіток. Зокрема вона дозволяє працювати зі стимулами двох видів – так званими елементами й конструктами (з єдиним винятком: конструкти в системі MEDIS – на відміну від класичного тесту репертуарних решіток – монополярні). Природно, вибір базового інструментарію істотно вплинув на описувану експериментальну реалізацію методики. Як предметна область був обраний світ мов програмування. У простір базових понять (що виступали в методиці як елементи) було включено декілька більш-менш популярних мов програмування, що належать до таких класів:

- мови штучного інтелекту;
- традиційні процедурні мови;
- так звані «макромови», зазвичай реалізовані в оболонках операційних систем, текстових редакторах тощо.

Як метафоричні простори обрані світ тварин і світ транспорту. Об'єкти цих світів виступали в методиці як монополярні конструкти. На першому етапі експерименту кожний з респондентів виконував класичне попарне суб'єктивне градуювання елементів. На запитання «Чи є що-небудь спільне між даними мовами програмування?», респондентові пропонувалося відповісти однією з наступних альтернатив:

ТАК!	1	Об'єкти дуже близькі
Так	2	Між об'єктами є щось спільне
???	3	Невизначена відповідь
Ні	4	Об'єкти різні
НІ!	5	Об'єкти зовсім несумісні

Дані цього етапу (окремо для кожного з респондентів) піддавалися опрацюванню методами багатовимірного градуювання (див. вище).

Результатом такого опрацювання є деякий евклідів простір невеликого числа вимірів, у якому вихідні оцінки розходжень відображені геометричними відстанями між точками. Чим краще ці відстані відповідають вихідним розходженням, тим адекватнішими вважається результат опрацювання в цілому. При цьому буквальний збіг відстаней і числових кодів відповідей, природно, не є обов'язковим (хоча таке й можливе в деяких модельних експериментах). Важливішою виявляється рангова відповідність відстаней вихідним оцінкам. А саме, в ідеальному випадку всі відстані між точками, що відповідають, наприклад, відповідям «ТАК!» у вихідних даних, повинні бути менші (хоча б і на частки відсотка масштабу шкали) від всіх відстаней, що відповідають відповідям «Так», і т. д.

У реальному експерименті ідеальна відповідність неможлива в принципі, оскільки метою опрацювання є стискання, скорочення розмірності даних, що обмежує число координатних осей результуючого простору. Проте алгоритм градуювання намагається, наскільки це можливо, мінімізувати рангову невідповідність моделі вихідним даним.

Геометричну модель градуювання можна інтерпретувати по-різному:

- по-перше, можна з'ясувати зміст координатних осей результуючого простору. Ці осі за суттю аналогічні факторам у факторному аналізі, що дозволяє використовувати відповідну парадигму інтерпретації, детально розроблену в експериментальній психології. У цьому випадку можна вважати, що виявлені фактори відіграють роль базових категорій, або базових (латентних) конструктів, за допомогою яких респондент (як правило, неусвідомлено) упорядковує свою картину світу (точніше, її проекцію на задану предметну область);
- по-друге, можна проаналізувати компактні угруповання стимулів у цьому просторі, ототожнивши їх з деякими істотними (хоча й схованими від інтроспекції) таксономічними одиницями, реально присутніми в моделі світу експерта. Істотно, що робиться спроба інтерпретувати кластери, отримані по моделі градуювання, а не по вихідних числових кодах розходжень. Це випливає із припущення, що інформація, не відтворена головними (найбільш навантаженими) факторами, є «шумом», що супроводжує кожен (а особливо психологічний) експеримент.

Основна експериментальна процедура – попарне порівняння деяких об'єктів і вираження ступеня їхньої подібності (відмінності) на числовій осі або виділення пар близьких об'єктів із поданої тріади – сама по собі накладає велику кількість обмежень на структуру, яку виявляють, а саме:

1) через вибір стимульного матеріалу (вибір об'єктів залишається за інженером зі знань);
2) через недосконалість шкали вимірів;
3) у зв'язку з рядом припущень математичного апарату; але головне, що отримана структура знань найчастіше має академічний характер, тобто відбиває об'єктивно наявні, але легко з'ясовні закономірності, що ніби лежать на поверхні.

Це пов'язано із психологічною установкою самого експерименту, під час якого експерта ніби перевіряють, екзаменують і він, природно, прагне давати правильні відповіді.

Щодо реального процесу видобування знань ця обставина стає принциповою, тому що дозволяє насправді відокремити ті знання, завдяки яким експерт є таким (рівень В), від загальнозначущих, банальних (для експертів у заданій предметній області) знань (рівень А), які можливо й не варті того, щоб заради них створювати власне інтелектуальну систему.

Однак очікується, що в багатьох (якщо не в більшості) випадків виявлені латентні структури можуть повністю перевернути уявлення інженера зі знань про предметну область і дозволити йому суттєво поглибити базу знань. Введення світу метафор – це деяка гра, а гра розкріпачує свідомість експерта і, як всі ігрові методики видобування знань (див. попередні розділи), є гарним каталізатором трудомістких серій інтерв'ю з експертом, без яких сьогодні неможливе розроблення промислових інтелектуальних систем.

7.2. Метод репертуарних решіток

7.2.1. Основні поняття

Серед методів когнітивної психології – науки, що вивчає те, як людина пізнає й сприймає світ, інших людей і самого себе, як формується цілісна система уявлень і відношень конкретної людини, особливе місце займає такий метод особистісної психодіагностики, як метод репертуарних решіток («repertory grid»).

Уперше метод був сформульований автором теорії особистісних конструктів Джорджем Келлі в 1955 р. Чим ширший набір особистісних конструктів у суб'єкта, тим більш багатовимірним, диференційованим є образ світу, людини, інших явищ і предметів, тобто тим вища його когнітивна складність.

Репертуарні решітки являють собою матрицю, що заповнюється або самим респондентом, або експериментатором у процесі обстеження чи бесіди. Стовпцю матриці відповідає певна група об'єктів, або, інакше, елементів. Як об'єкти можуть виступати люди, предмети, поняття, відносини, звуки, кольори – усе, що цікавить психодіагностика. Рядки матриці являють собою конструкти – біполярні ознаки, параметри, шкали, альтернативні протилежні відносини або способи поводження. Конструкти або задаються дослідником, або виявляються у випробовуваної особи за допомогою спеціальних прийомів і процедур виявлення. Уводячи поняття конструкта, Келлі поєднує дві функції: функцію узагальнення (установлення подібності) і функцію протиставлення. Він пропонує кілька визначень поняття «конструкт». Одне з них: конструкт – це деяка ознака або властивість, за якою два або кілька об'єктів подібні між собою й, отже, відмінні від третього об'єкта або декількох інших об'єктів.

Наприклад, виділення із трьох предметів «диван, крісло, стілець» двох «диван і крісло» виявляє конструкт «м'якість меблів». Келлі у своїх роботах підкреслює біполярність конструктів. Він вважає, що, затверджуючи що-небудь, ми завжди одночасно щось заперечуємо. Саме біполярність конструктів уможливлює побудову репертуарних решіток. Наприклад, «північ-південь» – це референтна вісь: елементи, які в одному контексті є «північчю», в іншому стають «півднем».

Можливості конструкта обмежені. Вони можуть бути застосовані тільки до деяких об'єктів. Це знайшло своє відбиття в понятті діапазону придатності конструкта. Англійські психологи Франселла й Банністер [177] вважають правило діапазону придатності характерною рисою техніки репертуарних решіток. Під діапазоном придатності можна розуміти область уявлень людини про світ, поняття якої можна порівняти з конкретною референтною віссю відокремлюваного конструкта. Психологічно осмислений результат вийде тільки в тому випадку, якщо елементи, використовувані в репертуарних решітках, будуть потрапляти в «діапазон придатності» конструктів респондента.

Конструкти – не ізольовані утворення. Вони взаємодіють один з одним, причому характер цієї взаємодії не випадковий, а має цілісний системний характер.

У процесі заповнення репертуарних решіток випробовувана особа повинна оцінити кожний об'єкт за кожним конструктом або іншим чином поставити у відповідність елементи конструктам.

Визначення репертуару означає, що елементи вибираються за певними правилами так, щоб вони відповідали якійсь одній області й всі разом були осмислено пов'язані

(контекстом) аналогічно до репертуару ролей у п'єсі. Передбачається, що, змінюючи репертуар елементів, можна «набудовувати» методики для виявлення конструктів різних рівнів спільності й стосовно різних систем.

При перекладі з англійської мови термін «матриця» не використовується, оскільки репертуарні решітки не завжди є матрицею в точному значенні цього терміну: в ній на перетині рядків і стовпців не обов'язково розташовані числа, не завжди дотримується прямокутний формат, рядки можуть бути різної довжини.

Другий зміст цього визначення полягає в тому, що в техніці репертуарних решіток часто елементи задаються у вигляді узагальнених інструкцій, репертуару ролей, на місце яких кожна конкретна людина подумки підставляє своїх знайомих людей або конкретні предмети, якщо як елементи задані назви предметів.

Як видно, репертуарні решітки краще вважати специфічним різновидом структурованого інтерв'ю. Зазвичай ми досліджуємо систему конструктів іншої людини в ході розмови з нею. У процесі бесіди ми поступово починаємо розуміти, як вона бачить світ, що із чим пов'язане, що для чого важливо, а що ні, як вона оцінює інших людей, події й ситуації.

Решітки формалізують цей процес і дають математичне обґрунтування зв'язків між конструктами певної людини, дозволяють детальніше вивчити окремі підсистеми конструктів, помітити індивідуальне, специфічне у структурі й змісті світогляду людини.

Важливе положення техніки репертуарних решіток: орієнтація на виявлення власних конструктів випробовуваного, а не нав'язування їх йому ззовні.

Гнучкість і ефективність репертуарних решіток, якість і кількість одержуваної інформації роблять їх придатними для вирішення широкого кола завдань. Методики цього типу використовуються в різних галузях практичної діяльності: у педагогіці й соціології, у медицині, рекламі й дизайні. Метод репертуарних решіток виявився ідеально пристосованим для реалізації у вигляді діалогових програм на комп'ютері, що також сприяло його широкому розповсюдженню. Достоїнства й переваги цього методу повністю розкриваються тоді, коли є можливість, здійснивши дослідження, швидко опрацювати результати й проаналізувати їх для того, щоб уже при наступній зустрічі з випробуваним можна було уточнити й перевірити виниклі припущення, скласти й уточнити репертуарні решітки іншого типу, а якщо це необхідно, і доповнити колишню, змінивши репертуар елементів або вибірку конструктів.

7.2.2. Методи виявлення конструктів. Метод мінімального контексту

Метод мінімального контексту або метод тріад найчастіше використовується для виявлення конструктів. Елементи подаються групами по три. Це мінімальне число, що дозволяє визначити подібність і розходження.

Випробовуваній особі надаються три елементи з усього списку й пропонується назвати яку-небудь важливу якість, за якою два з них подібні між собою й, отже, відмінні від третього. Після того як експериментатор запише відповідь, респондента просять назвати, у чому конкретно полягає відмінність третього елемента від двох інших (якщо випробовуваний не вказав, які саме два елементи були оцінені як подібні між собою, то його просять зробити це). Відповідь на це запитання і являє собою протилежний полюс конструкта. Випробовуваному надається стільки тріад елементів, скільки вважає за

потрібне експериментатор. Специфічних правил не існує. Все залежить лише від величини вибірки, тобто від кількості конструктів, що підлягають дослідженню.

Приклад

Є список із назв фруктів. Береться тріада «яблуко-груша-апельсин». Респондент виділяє два подібних об'єкти – «яблуко й груша»; якість, що визначає подібність, – «відсутність алергійної реакції в респондента», відмінність третього об'єкта – «алергічність». Так виявлено особистісний конструкт «алергічність/ її відсутність».

Інші методи виявлення конструктів

Франселла й Банністер описують методи, які також використовують тріади:

- послідовний метод;
- метод самоідентифікації;
- метод рольової персоніфікації.

У двох останніх методах у тріаду включається елемент «я сам». Келлі запропонував використовувати тріади для виявлення конструктів, оскільки цей метод відбивав його теоретичні уявлення про те, як конструкти вперше виникають. Однак у зв'язку з тим, що у випробовуваного виявляються вже сформовані конструкти, не обов'язково використовувати неодмінно три елементи. Тріада не є єдиним способом виявлення протилежного полюса.

Для виявлення конструктів можна використовувати два елементи (виявлення конструктів за допомогою діод елементів) або більш, ніж три, як це робиться в методі повного контексту. Часто використовуваний метод – техніка сходового спускання Хінкла:

- конструкти видобувають стандартним методом;
- із приводу окремого конструкта задається запитання «До якого полюса цього конструкта ви б хотіли бути віднесені?»;
- потім: «Чому Ви віддаєте перевагу цьому полюсу?», «Що протистоїть цьому?».

Таким чином утворюється новий конструкт, більше узагальнений, ніж вихідний. Процес повторюється, і виділяється ієрархія конструктів.

7.2.3. Аналіз репертуарних решіток

Аналіз репертуарних решіток дозволяє визначити силу й спрямованість зв'язків між конструктами респондента, виявити найбільш важливі й значимі параметри (глибинні конструкти), що лежать в основі конкретних оцінок і відношень, побудувати цілісну підсистему конструктів, що дозволяє описувати й пророкувати оцінки й відношення людини.

Аналіз одиничних репертуарних решіток

Можна використовувати форму кластерного аналізу для групування конструктів. Цей алгоритм структурує конструкти в лінійний порядок так, що конструкти, які перебувають близько в просторі, також виявляються близькими в порядку. Цей алгоритм має переваги під час демонстрації, тому що подання просто реорганізує решітки, показуючи сусідство конструктів і елементів. Отже, формуються дві матриці – одна для елементів, інша для конструктів. Кластери визначаються вибором найбільших значень у цих матрицях – тобто найбільш зв'язаних складових матриці – доти, поки всі елементи й конструкти не будуть включені в кластерне дерево. Програма робить ієрархічну кластеризацію системи конструктів експерта й відображає видобуті знання.

Крім того, для кожного конструкта є чисельні значення в решітках як вектор величин, пов'язаних з розташуванням елементів щодо полюсів цього конструкта. Із цього погляду кожний конструкт може бути зображений як точка в багатомірному просторі, а його площина визначається кількістю пов'язаних з ним елементів. Природною мірою відношення між конструктами є відстанню між ними в цьому багатовимірному просторі. Два конструкти з нульовою відстанню між ними – це конструкти, стосовно яких елементи структуруються однаково. Отже, можна вважати, що вони використовуються однаково. У деякому сенсі – це еквівалентні конструкти.

Для нееквівалентних конструктів можна аналізувати їхні просторові відношення, визначаючи ряд осей як проекцію кожного конструкта на вісь, найбільш віддалену від них, проекцію на другу вісь, пов'язану з відстанями, що залишилися, і т. д. Це метод аналізу головних компонент простору конструктів. Він пов'язаний з факторним аналізом семантичного простору, використаного у вивченні семантичного диференціала. Метод аналізу головних компонент дозволяє подати елементи й конструкти так, що між ними можуть бути виявлені взаємозв'язки. Можна побудувати логічний аналіз репертуарних решіток, використовуючи конструкти як предикати щодо елементів.

Аналіз декількох репертуарних решіток

Досить часто виникає ситуація, коли потрібно зрівняти кілька репертуарних решіток. Аналіз серії репертуарних решіток, заповнюваних тією самою людиною в різні моменти часу, дозволяє стежити за динамікою конструктів і оцінок, будувати траєкторії зміни стану людини в системі його власних суб'єктивних шкал.

Проаналізуємо кілька репертуарних решіток, заповнюваних різними людьми.

Аналіз пар системних конструктів використовується для вимірювання згоди й порозуміння між людьми. Для цього два експерти, що мають різні точки зору, створюють і заповнюють решітки зі спільної області знань. При цьому кожний незалежно від іншого вибирає елементи, виявляє конструкти й оцінює їх. Потім кожний робить дві порожні копії своїх решіток, залишаючи елементи й конструкти без значень їхньої оцінки. Обидві ці решітки заповнюються партнерами. При цьому одна заповнюється так, як він сам собі це уявляє, а друга так, як він уявляє собі заповнення оригінальної решітки її автором. Порівняння пар решіток допомагають досягти згоди й порозуміння між двома людьми. Існують три способи порівняння двох решіток.

1. Зчеплення решіток, що мають спільні елементи, і їхня подальше опрацювання одним з описаних алгоритмів так, ніби вони становлять одну більшу решітку. Таким чином можна досліджувати взаємодію ідей через перевірку змішаних кластерів конструктів з різних решіток.
2. Цей шлях вимагає наявності двох решіток з однаковими назвами елементів і конструктів і показує розбіжності між ними через вимірювання відстані між тими самими назвами. Результати показують згоду в розумінні й виявляють розходження між двома решітками, заснованими на однакових назвах і конструктах.
3. Цей спосіб також використовує дві решітки з однаковими назвами елементів і конструктів, знаходить елементи й конструкти, що змінюються найбільше, й видаляє їх із решіток. Таким чином визначаються базові елементи й конструкти, які показують згоду й порозуміння.

Аналіз груп системних конструктів. Аналізується серія репертуарних решіток, отриманих від групи людей, що використовували однакові елементи. Порівняється кожна пара й показується «групова мережа», що відображає зв'язки подібних конструктів усередині групи. Створюються решітки, що відображають конструкти, які розуміються більшістю груп, і це служить підставою для подальшого аналізу. Кожний конструкт, невикористаний у рамках групи, оцінюється за силою зв'язаності з іншими конструктами.

7.2.4. Автоматизовані методи

Цей параграф присвячений огляду деяких найвідоміших методів і систем видобування знань на основі методу репертуарних решіток, частково з робіт [134], [123].

Уперше автоматизоване створення репертуарних решіток і видобування з експертів конструктів було реалізовано в системі PLANET [244]. Подальшим розвитком системи PLANET є інтегроване середовище KITTEN, що підтримує ряд методів видобування знань. Буза Д. у системі ETS [211] використав метод репертуарних решіток для виявлення понятійної системи предметної області. Нащадками ETS є система NewETS та інтегроване середовище для видобування експертних знань AQUINAS [202].

Відомо багато прототипів ЕС, для створення яких використовувалася ETS. Серед них:

1) порадник вибору інструментарію для розробників ЕС;
2) консультант з мов програмування;
3) аналізатор геологічних даних;
4) порадник із налагодження Фортран-програм;
5) консультант із СКБД та ін.

Проте сфера застосування ETS обмежена видобуванням експертних знань для нескладних завдань аналізу, що не вимагають для свого рішення процедурних, каузальних і стратегічних знань.

ETS взаємодіє з експертом у діалоговому режимі: проводить з ним інтерв'ю і допомагає аналізувати створювану БЗ. В архітектурі ETS можуть бути виділені підсистеми: видобування елементів; виявлення конструктів; побудови репертуарних решіток; побудови графа імплікативних зв'язків; генерації продукційних правил; тестування БЗ; корекції БЗ; генерації БЗ для різних інструментальних засобів створення ЕС.

У діагностичній системі MORE [266] використано принципи, подібні до тих, які лежать в основі обох описаних вище систем. Тут уперше використано кілька різних стратегій інтерв'ю. Техніка інтерв'ю, використана в MORE, спрямована на виявлення таких сутностей:

- гіпотези – підтвердження яких має своїм результатом діагноз;
- симптоми – спостереження яких наближає наступне прийняття гіпотези;
- умови – деяка множина подій, що не є безпосередньо симптоматичною для якої-небудь гіпотези, але яка може мати діагностичне значення для деяких інших подій;
- зв'язки – з'єднання сутностей;
- шляхи – виділений тип зв'язку, що з'єднує гіпотези із симптомами.

Відповідно до цього в системі використовуються такі стратегії інтерв'ю: диференціація гіпотез, розрізнення симптомів, симптомна обумовленість, розподіл шляху й деякі інші.

Стратегія диференціації гіпотез спрямована на пошук симптомів, які забезпечують точніше розрізнення гіпотез. Найпотужнішими є ті симптоми, які спостерігаються під час однієї події, яку піддають діагностиці.

Стратегія розрізнення симптомів виявляє специфічні характеристики симптому, які, з одного боку, ідентифікують його як наслідок деякої гіпотези, з іншого боку – протиставляють іншим.

Стратегія симптомної обумовленості спрямована на виявлення негативних симптомів, тобто симптомів, відсутність яких має більшу діагностичну вагу, ніж їхня присутність.

Стратегія розподілу шляхів забезпечує знаходження симптоматичних подій, які лежать на шляху до вже знайденого симптому. Якщо такий симптом існує, то він має більше діагностичне значення, ніж знайдений раніше. У системі KRITON [230] для видобування знань використовуються два джерела: експерт із його знаннями, отриманими на практиці; книжкові знання, документи, описи, інструкції (ці знання добре структуровані й фіксовані традиційними засобами). Для видобування знань із першого джерела в KRITON застосовано техніку інтерв'ю, що використовує стратегію репертуарних решіток – розбивання на щаблі. Стратегія розбивання на щаблі спрямована на виявлення спадкової структури предметної області. Акцент робиться на виявленні структури родових і видових понять (супертипів). При цьому типи, виявлені на черговому кроці застосування стратегії, стають базисом для подальшого її застосування.

У системі застосований прийом перемикання стратегій: якщо при роботі стратегії репертуарних решіток, надаючи трійки семантично зв'язаних понять, експерт не в змозі назвати ознаку, що відрізняє два з них від третього, система запускає стратегію розбивання на щаблі й, задаючи експертові запитання про поняття, пов'язані з попередніми відношеннями «рід – вид», робить спробу з'ясувати таксономічну структуру цих понять з метою виявлення ознак, що їх розрізняють.

7.3. Керування знаннями

7.3.1. Що таке «керування знаннями»?

Поняття «керування знаннями» (КМ – Knowledge Management) виникло в середині 90-х років XX ст. у великих корпораціях, де проблеми опрацювання інформації набули особливої гостроти й стали критичними. При цьому стало очевидним, що основним вузьким місцем є опрацювання знань, накопичених фахівцями компанії, тому що саме знання забезпечують переваги над конкурентами. Часто інформації в компаніях накопичено навіть більше, ніж вони здатні опрацювати. Різні компанії намагаються вирішувати це питання по-різному, але при цьому кожна компанія прагне збільшити ефективність опрацювання знань [281].

Ресурси знань розрізняються залежно від галузей індустрії й додатків, але, як правило, включають керівництва, листи, новини, інформацію про замовників, відомості про конкурентів і дані, що нагромадилися в процесі розроблення. Для застосування КМ-систем використовуються різноманітні технології:

- електронна пошта;
- бази й сховища даних (Data Wharehouse);

- системи групової підтримки;
- броузери й системи пошуку;
- корпоративні мережі й Інтернет;
- експертні системи й бази знань; інтелектуальні системи.

Традиційно проектувальники КМ-систем орієнтувалися лише на окремі групи споживачів – головним чином менеджерів. Сучасніші КМ-системи спроектовані вже з розрахунку на цілу організацію.

Сховища даних, які працюють за принципом центрального складу, були одним з перших інструментаріїв КМ. Як правило, сховища містять багаторічні версії звичайної БД, фізично розташовані в тій самій базі. Коли всі дані утримуються в єдиному сховищі, вивчення зв'язків між окремими елементами може бути плідніше.

При цьому активи знань можуть перебувати в різних місцях: у базах даних, базах знань, у картотечних блоках, у фахівців і можуть бути розосереджені по всьому підприємству. Занадто часто одна частина підприємства повторює роботу іншої частини просто тому, що неможливо знаходити й використовувати знання, що перебувають в інших частинах підприємства.

Керування знаннями – це сукупність процесів, які керують створенням, поширенням, опрацюванням й використанням знань усередині підприємства.

Необхідність розроблення систем КМ обумовлена такими причинами:

- працівники підприємства витрачають занадто багато часу на пошук необхідної інформації;
- досвід провідних і найбільш кваліфікованих співробітників використовується тільки ними самими;
- цінна інформація міститься у величезній кількості документів і даних, доступ до яких ускладнений;
- помилки, що потім так дорого коштують, повторюються через недостатню інформованість та ігнорування попереднього досвіду.

Важливість систем КМ обумовлена також тим, що знання, що не використовуються й не примножуються, в остаточному підсумку стають застарілими й марними, так само, як гроші, які зберігаються не для того, щоб стати оборотним капіталом, в остаточному підсумку втрачають свою вартість, поки не знеціняться. Тоді як знання, що поширюються, здобуваються й обмінюються, генерують нові знання.

7.3.2. Керування знаннями і корпоративна пам'ять

Більшість оглядів концепції керування знання (КМ) приділяють увагу тільки первинному опрацюванню корпоративної інформації типу електронної пошти, програмного забезпечення колективної роботи або гіпертекстових баз даних (наприклад, Wiig, 1996). Вони формують істотну частину з необхідної, але виразно недостатньої, технічної інфраструктури для керування знаннями.

Одним з нових рішень щодо керування знаннями є поняття корпоративної пам'яті (corporate memory), що за аналогією з людською пам'яттю дозволяє користуватися попереднім досвідом і уникати повторення помилок.

Корпоративна пам'ять фіксує інформацію з різних джерел підприємства й робить цю інформацію доступною для фахівців з метою вирішення виробничих завдань.

Корпоративна пам'ять не дозволяє зникнути знанням фахівців, що вибувають (вихід на пенсію, звільнення та ін.). Вона зберігає великі обсяги даних, інформації й знань із різних джерел підприємства. Вони відображені в різних формах, таких як бази даних, документи й бази знань (рис. 7.3).

Введемо два рівні корпоративної пам'яті (так звані явні й неявні знання [292]).

Рівень 1. Рівень матеріальної або явної інформації – це дані й знання, які можуть бути знайдені в документах організації у формі повідомлень, листів, статей, довідників, патентів, креслень, відео- і аудіозаписів, програмного забезпечення й т. ін.

Рівень 2. Рівень персональної або схованої інформації – це персональні знання, невідривно пов'язані з індивідуальним досвідом. Вони можуть бути передані через прямий контакт – «віч-на-віч», через процедури видобування знань. Саме приховане знання – те практичне знання, що є ключовим при ухваленні рішення й керуванні технологічними процесами.

У дійсності ці два типи інформації, подібні до двох сторін однієї й тої самої медалі, однаково важливі у структурі корпоративної пам'яті (рис. 7.3).

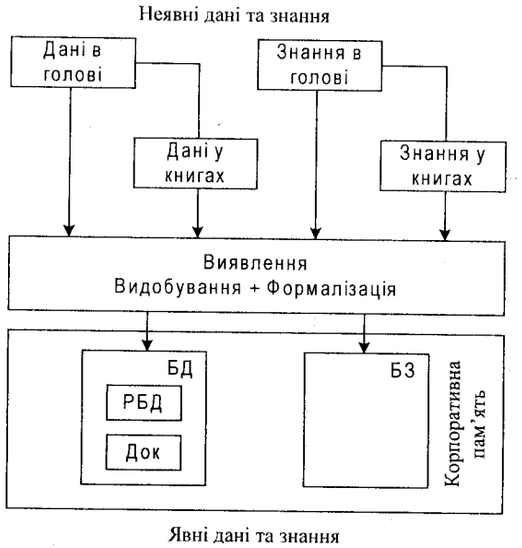


Рис. 7.3. Дані й знання в системах корпоративної пам'яті.

При розроблені систем КМ можна виділити ряд етапів.

1. Нагромадження. Стихійне й безсистемне нагромадження інформації в організації.
2. Видобування. Процес, ідентичний традиційному видобуванню знань для ЕС (див. розділи 2 і 3). Це один з найбільш складних і трудомістких етапів. Від його успішності залежить подальша життєздатність системи.
3. Структурування. На цьому етапі повинні бути виділені основні поняття, вироблена структура подання інформації, що має максимальну наочність, простоту зміни й доповнення.
4. Формалізація. Подання структурованої інформації у форматах машинного опрацювання, тобто на мовах опису даних і знань.

5. Обслуговування. Під процесом обслуговування розуміється коректування формалізованих даних і знань (додавання, відновлення): «чищення», тобто видалення застарілої інформації; фільтрація даних і знань для пошуку інформації, необхідної користувачам.

Якщо перші чотири етапи звичайні для інженерії знань, то останній є специфічним для систем керування знаннями. Як уже було сказано, він розпадається на три дрібніших процеси:

- коректування формалізованих знань (додавання, відновлення);
- видалення застарілої інформації;
- фільтрація знань для пошуку інформації, необхідної користувачеві, виділяє компоненти даних і знань, що відповідають вимогам конкретного користувача. За допомогою тої самої процедури користувач може довідатися про місцезнаходження інформації, що його цікавить.

Розглянута вище класифікація не є єдиною, але вона дозволяє зрозуміти, що відбувається в реальних системах керування знаннями.

7.3.3. Системи OMIS

Автоматизовані системи КМ, або Organizational Memory Information Systems (OMIS), призначені для нагромадження й керування знаннями підприємства [272]. OMIS включають роботу як на рівні 1 – з явним знанням компанії у формі баз даних і електронних архівів, так і на рівні 2 – зі схованим знанням, фіксуючи його в певному (більш-менш формальному) поданні у формі експертних систем або БД.

OMIS часто використовують допоміжні довідкові системи, так звані helpdesk-додатки.

Вкажемо основні функції OMIS.

- Збирання і систематична організація інформації з різних джерел у централізоване й структурне інформаційне сховище.
- Інтеграція з наявними автоматизованими системами. На технічному рівні це означає, що корпоративна пам'ять повинна бути безпосередньо пов'язана за допомогою інтерфейсу з інструментальними засобами, які в цей час використовуються в організації (наприклад, текстові процесори, електронні таблиці, системи).
- Забезпечення потрібної інформації на запит (пасивна форма) і при необхідності – активна форма. Занадто часті помилки – це наслідок недостатньої інформованості. Цього неможливо уникнути за допомогою пасивної інформаційної системи, тому що службовці часто надто зайняті, щоб шукати інформацію, або просто не знають, що потрібна інформація існує. Корпоративна пам'ять може нагадувати службовцям про корисну інформацію й бути компетентним партнером для спільного вирішення завдань.

Кінцева мета OMIS полягає в тому, щоб забезпечити доступ до знань щоразу, коли це необхідно. Щоб забезпечити це, OMIS реалізує активний підхід поширення знань, що не надається на запити користувачів, а автоматично забезпечує корисні для рішення завдання знання. Щоб запобігати інформаційному перевантаженню, цей підхід має бути поєднаний з високою вибірковою оцінкою доречності. Закінчена система повинна діяти як інтелектуальний помічник користувача.

Використання корпоративної пам'яті часто переслідує більш помірковані цілі, ніж використання інтелектуальних систем. Це пов'язано з тим, що технології опрацювання даних (баз даних і гіпертекстових систем) застосовуються набагато ширше, ніж технології систем, заснованих на знаннях. OMIS зберігають і забезпечують видачу на запит потрібної інформації, але залишають її інтерпретацію й оцінку в специфічному контексті завдання головне користувачеві.

З іншого боку, корпоративна пам'ять розширює ці технології роботою зі знаннями, щоб поліпшити якість рішення завдань. Так, OMIS включає підсистеми пояснень, які дозволяють безпосередньо відповідати на запитання: «Чому?» і «Чому немає?». У простій базі даних або гіпертекстовій системі користувачі повинні були б шукати потрібну інформацію для відповіді на такі запитання безпосередньо, а для цього необхідно відфільтрувати велику кількість потенційно потрібної інформації, що, однак, не буде застосовуватися у специфічному випадку.

Нарешті, OMIS не тільки видає інформацію, але повинна також бути завжди готовою сприймати нову інформацію від її користувачів.

Рис. 7.4 відображає архітектуру для OMIS і корпоративної пам'яті (частково з роботи [214]). Ядром системи є Інформаційне сховище (Information Depository).

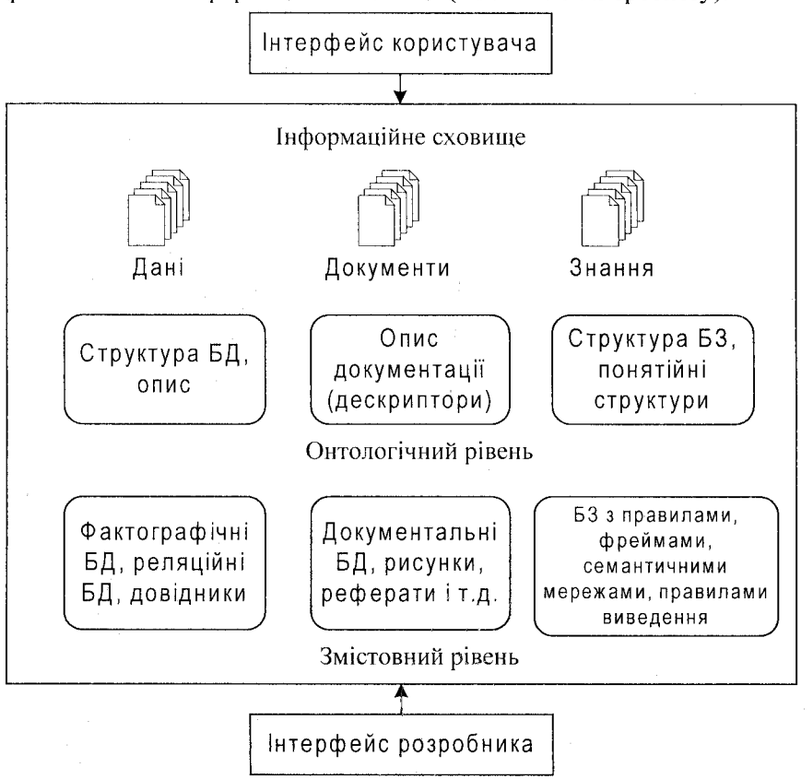


Рис. 7.4. Архітектура OMIS.

Рисунок також дає уявлення про деякі види інформації, що включається до корпоративної пам'яті. Якщо сховища даних містять, в основному, кількісну інформацію, то сховища знань більше орієнтовані на якісний матеріал. КМ-системи генерують системи із широкого діапазону даних, сховищ даних, статей новин, зовнішніх баз, WWW-сторінок.

Програмний інструментарій для OMIS включає як оригінальні розроблення, наприклад, KARAT [313], так і стандартні засоби, наприклад, LOTUS NOTES, яка забезпечила один з перших інструментаріїв зберігання якісною й документальною інформацією. Але сьогодні у зв'язку з бурхливим розвитком Інтернету, КМ-системи все частіше використовують Web-технологію.

7.3.4. Особливості розроблення ОМШ

Оскільки розроблення систем корпоративної пам'яті – це, насамперед, програмний проект, то для нього застосовують традиційні технології розроблення великих програмних систем. У кожному програмному проекті першим кроком у розробленні є аналіз вимог, у якому мають бути знайдені відповіді на такі запитання:

- Які завдання повинні підтримуватися?
- Яка інформація необхідна, щоб вирішити ці завдання?
- Який тип підтримки хочуть користувачі?
- Який рівень витрат на розроблення?
- Які зміни очікуються в майбутньому?

При пошуку відповідей на ці запитання варто враховувати наступне.

1. **Людський фактор.** Основна причина невдач ранніх відомих проектів OMIS полягала в тім, що розроблювачі ігнорували реальні потреби, здатності й цілі користувачів системи [272].
2. **Аналіз вартості**. По-перше, ядро проекту має орієнтуватися на критичні процеси, що «страждають» від недостачі інформаційної підтримки. По-друге, не слід перевантажувати початкову систему занадто великою кількістю послуг, які можуть бути бажаними, але не обіцяють швидке повернення інвестицій.
3. **Еволюція знань**. Електронна підтримка особливо цінна в галузях, що піддаються швидким змінам, тому що на таких підприємствах важко забезпечити доступ до оперативної сучасної інформації. У системах OMIS часто використовують різні нові технології опрацювання знань, що не мають наразі загальноприйнятих слов'яномовних термінів і пов'язані з одержанням нового знання шляхом аналізу даних, наприклад «відкриття або розвідка знань» (Knowledge Discovery) і «видобування даних» (Data Mining). Розвідка знань являє собою новий напрям, що швидко розвивається і який займається «нетривіальним видобуванням точної, раніше невідомої й потенційно корисної інформації з даних» [294]. У методах розвідки даних використовують різні підходи до аналізу тексту й числовий даних, плюс спеціальний інструментарій статистичного аналізу.
4. **Чутливість до контексту для природно-мовних запитів.** Система повинна «розуміти» контекст запитувачів. Наприклад, вона повинна розрізняти терміни «розмноження тварин» і «розмноження документів».
5. **Гнучкість.** Система повинна мати можливість опрацьовувати знання в різній формі й із різних тем у контексті роботи певного підприємства.

6. **Інтелектуальність**. Система повинна накопичувати інформацію про своїх користувачів і про знання, які вона одержує під час роботи. Отже, з часом її можливість «продумано» надавати користувачам знання повинна вдосконалюватися.

До останнього часу при розробленні OMIS залишається цілий ряд дослідницьких питань [272].

- Проблема узагальнення моделей даних, словників понять або тезаурусів, онтології. Підстава для об'єднаної експлуатації даних, документів і формального знання – побудова об'єднаних метамоделей даних і знань. Корисними були б процедури автоматичного породження тезауруса з наявних масивів документів. Об'єднана онтологія/тезаурус може використовуватися, щоб поліпшити пошук, фільтрацію й маршрутизацію документів.
- Проблема об'єднання логічного висновку й інформаційного пошуку. Об'єднана експлуатація формальних і неформальних подань знань і даних – це послідовне зближення логічних методів і методів інформаційного пошуку й індексації даних.
- З'єднання ділових процесів і керування знаннями. Остаточна мета полягає в тому, щоб виявляти інформаційну потребу протягом виконання виробничого процесу й визначати доречне знання у специфічному контексті завдання. Перший прагматичний крок у цьому напрямі описаний у роботі [259], де автори пропонують використовувати інформацію контексту завдання для інформаційної фільтрації.

Корпоративна пам'ять інтегрує знання, щоб у рішенні нових завдань обпертися на попередньо накопичений досвід. Отже, можна уникати повторення помилок, досвід може розширюватися систематично й інформаційноємні процеси роботи можуть бути виконані ефективнішими способами. На відміну від експертних систем, первинна мета систем OMIS – не підтримка одного специфічного завдання, а краща експлуатація необхідного загального ресурсу – знань.

Зараз існує значний інтерес до КМ із боку промислових компаній, які усвідомлюють високий прикладний потенціал корпоративної пам'яті для рішення цілого ряду практичних завдань опрацювання інформації. З іншого боку, небагато проектів ідуть далі, ніж стадія прототипу, що очевидно показує, що компанії намагаються уникати витрат і ризику вкладення капіталу в нові технології, які ще не знайшли значного поширення.

7.4. Візуальне проектування баз знань як інструмент пізнання

Візуальні методи специфікації та проектування баз знань і розроблення концептуальних структур є досить ефективним гносеологічним інструментом або інструментом пізнання [264]. Використання методів інженерії знань як дидактичних інструментів, як формалізмів подання знань сприяє швидшому і повнішому розумінню структури знань цієї предметної області, що особливо корисно для новачків на стадії вивчення особливостей професійної діяльності.

Методи візуальної інженерії знань можна широко використовувати в різних навчальних закладах – від шкіл до університетів – як для поглиблення процесу розуміння, так і для контролю знань. Більшість учнів і студентів опановують навички візуального структурування протягом декількох годин.

7.4.1. Від понятійних карт до семантичних мереж

У розділі 5 було запропоноване визначення поля знань, що дозволяє інженерові зі знань трактувати форму зображення поля досить широко, зокрема семантичні мережі або понятійні карти (concept maps) є можливою формою зображення. Це означає, що сам процес побудови семантичних мереж допомагає усвідомлювати пізнавальні структури.

Програми візуалізації є інструментом, що дозволяє зробити видимими семантичні мережі пам'яті людини. Мережі складаються з вузлів і впорядкованих співвідношень, або зв'язків, що з'єднують ці вузли. Вузли виражають поняття або припущення, а зв'язки описують відношення між цими вузлами. Тому розроблення семантичних мереж має на увазі аналіз структурних взаємодій між окремими поняттями предметної області.

У процесі створення семантичних мереж експерт і аналітик змушені аналізувати структури своїх власних знань, що допомагає їм включати нові знання у структури вже наявних знань. Результатом цього є більш осмислене використання придбаних знань.

Візуальні специфікації у формі мереж можуть використовуватися новачками й експертами як інструменти для оцінки змін, що відбулися в їхньому мисленні. Якщо погодитися, що семантична мережа є досить повним відображенням пам'яті людини, то процес навчання із цього погляду можна розглядати як реорганізацію семантичної пам'яті.

Козма [270], один із розроблювачів програми організації семантичної мережі Learning Tool, вважає, що ці засоби є інструментами пізнання, що підсилюють і розширюють пізнання людини. Розроблення семантичних мереж чекає від учнів:

- реорганізації знань;
- вичерпного опису понять і зв'язків між ними;
- глибокого опрацювання знань, що сприяє кращому запам'ятовуванню й видобуванню з пам'яті знань, а також підвищує здатності застосовувати знання в нових ситуаціях;
- зв'язування нових понять із наявними поняттями й уявленнями, що поліпшує розуміння;
- просторового вивчення за допомогою просторового подання понять у досліджуваній галузі [239].

Корисність семантичних мереж і карт понять, мабуть, найкраще демонструється їхніми зв'язками з іншими формами мислення вищого порядку. Вони тісно пов'язані з формальним обґрунтуванням у хімії і здатністю аргументувати свої висловлювання в біології. Також було показано, що семантичні мережі мають зв'язок з виконанням досліджень [247].

7.4.2. База знань як пізнавальний інструмент

Коли семантична мережа створюється як прообраз бази знань, розроблювач повинен фактично моделювати знання експерта. Особливо глибокого розуміння вимагає розроблення функціональної структури.

Визначення структури «Якщо..., то ...» області знань змушує чітко формулювати принципи ухвалення рішення. Не можна вважати, що просто розроблення поля знань системи обов'язково приведе до одержання повних функціональних знань у певній області.

Розроблення інтелектуальних систем почало використовуватися як інструмент пізнання порівняно недавно. Ліпперт [279], що є одним з піонерів застосування експерт-

них систем як інструменту пізнання, стверджує, що завдання зі створення невеликих базисів правил є дуже корисними для рішення педагогічних проблем і структурування знань для учнів від шостого класу до дорослих. Вивчення при цьому стає більш осмисленим, тому що учні оцінюють не тільки сам процес мислення, але також і результати цього процесу, тобто отриману базу знань. Створення бази знань жадає від учнів уміння відокремлювати одне від одного факти, зміни й правила, що застосовуються до зв'язків між складовими галузі знань.

Наприклад, Лей [215] установив, що після того, як студенти-медики створять медичну експертну систему, вони підвищують своє вміння в плані аргументації й одержують глибші знання із досліджуваного предмету. Шість студентів-першокурсників фізичного факультету, які використовували експертні системи для складання запитань, прийняття рішень, формулювання правил і пояснень щодо руху частки відповідно до законів класичної фізики, одержали глибші знання в цій галузі завдяки ретельній роботі, пов'язаній з кодуванням інформації й опрацюванням великої кількості матеріалу для одержання зрозумілого зв'язного змісту, а отже, і більшої семантичної глибини [279].

Отже, створення бази знань експертної системи сприяє глибшому засвоєнню знань, а візуальна специфікація підсилює прозорість і наочність уявлень.

Коли комп'ютери використовуються в навчанні як інструмент пізнання, а не як контрольно-навчальні системи (навчальні комп'ютери), вони розширюють можливості автоматизованих навчальних систем (АНС), одночасно розвиваючи розумові здібності та знання учнів. Результатом такої співпраці учня й комп'ютера є значне підвищення ефективності навчання. Комп'ютери не можуть і не повинні керувати процесом навчання. Скоріше, комп'ютери повинні використовуватися для того, щоб допомогти учням набути знань.

7.5. Проектування гіпермедіа БД і адаптивних навчальних систем

7.5.1. Гіпертекстові системи

Перші інформаційні системи на основі гіпертекстових (ГТ) моделей з'явилися ще в середині 60-х років XX ст., але справжній розквіт настав у 80-ті роки минулого століття після появи перших комерційних ГТ-систем для GUIDE (1986 р.) і HyperCard (1987 р.). У цей час ГТ-технологія є стандартом de facto в області автоматизованих навчальних систем (АНС) і надвеликих документальних БД [290].

Під гіпертекстом розуміють технологію формування інформаційних масивів у вигляді асоціативних мереж, елементами або вузлами якої виступають фрагменти тексту, малюнки, діаграми тощо.

Навігація з такими мережами здійснюється по зв'язках між вузлами. Основні функції зв'язків [221]:

- перейти до нової теми;
- приєднати коментар до документу;
- з'єднати посилання на документ із документом, показати на екрані графічну інформацію (малюнок, креслення, графік, фотографію тощо);
- запустити іншу програму тощо.

З огляду на широке використання графічних структур у моделюванні, ГТ може набувати рис складніших моделей, наприклад, мереж Петрі, діаграм станів тощо.

На сьогодні не існує стандартизованої технології розроблення «добре структурованого» ГТ, хоча важливість когнітивно-наочної й «прозорої» структури ГТ очевидна.

Як робочий спрощений алгоритм складання ГТ можна запропонувати наступний.

Інструкція з розроблення гіпертекстового додатка:

1. Дайте назву уявлюваному документу, наприклад, «Підручник з кулінарії», «Довідник абітурієнта» тощо.
2. Уявіть, що вся інформація повинна бути згрупована, як у гарному підручнику, тобто у вигляді розділів і параграфів. Основна умова для розділів, параграфів і підпараграфів – збалансованість, тобто приблизно однаковий обсяг.
3. Проставте перехресні посилання між поняттями усередині параграфів, дотримуючись принципу «чим менше, тим краще».
4. Досягніть балансу аудіо-, відео- й графічної інформації.
5. Надайте можливість повернення на крок назад або на більш високий рівень ієрархії.
6. Відобразіть ієрархічне положення поточної сторінки. Наприклад, якщо ми перебуваємо в підпараграфі, то необхідно вивести назву поточного розділу й параграфа.
7. Ще раз перевірте гіпертекстовий зміст, у якому відбиті всі розділи, параграфи й підпараграфи. З будь-якої сторінки повинен бути доступ до цього змісту.
8. Всі необхідні гіпертекстові посилання краще розташовувати не в самому тексті, а розмістити їх в одному місці на сторінці, наприклад, наприкінці сторінки.
9. Посилання, якими користувач уже користувався, повинні виділятися іншим кольором.

7.5.2. Від мультимедіа до гіпермедіа

Мультимедіа (ММ) сьогодні розуміється як інтегроване комп'ютерне середовище, що дозволяє використовувати поряд із традиційними засобами взаємодії людини й ЕОМ (алфавітно-цифровий і/або графічний дисплей, принтер, клавіатуру) нові можливості: звук (живий людський голос, музику й ін.), відеокліпи (кольорові художні й документальні кліпи), озвучену мультиплікацію тощо.

Коли елементи ММ об'єднані на основі мережі гіпертексту, можна говорити про гіпермедіа (ГМ). Основною сферою застосування ГМ сьогодні є автоматизовані навчальні системи (АНС) або електронні підручники. Розміщення у вузлах мережі не тільки текстової й цифрової інформації, але й графіків, відеокліпів, фото й музики збагачує гіпертекст і збільшує цікавість і наочність електронних підручників. В останні роки ММ активно впроваджується і в інтелектуальні системи, особливо в системи-консультанти, як ефективний засіб збільшення наочності та зрозумілості рекомендацій, виданих системою. Використання анімації, звуку й відео істотно полегшує засвоєння навчального матеріалу зі структурування знань і знижує рівень когнітивних зусиль учнів при одночасному зменшенні часу, потрібного для вивчення певної проблематики.

Основною передумовою інтересу до гіпермедіа-систем є фактично безпрецедентний успіх глобальної мережі Інтернет і WWW-технології.

Іншим фактором появи ГМ можна вважати потужний прорив на ринку машинних носіїв інформації з боку фірм, що роблять компакт-аудіодиски (PHILIPS, SONY, NIMBUS)

у вигляді CD-ROM (Compact Disc Read Only Memory), що дозволяють зберігати до 600 Мб інформації, тим самим відкриваючи двері для опрацювання BLOB (Binary Large Objects) – більших двійникових об'єктів, наприклад, цифрових відеозображень.

Проте, існує цілий ряд проблем, з якими зіштовхуються розроблювачі ММ-систем, незалежно від того, орієнтовані вони на Інтернет-додатки чи на автономні підручники на CD-ROM:

- відсутність методології та технології структурування різнорідної інформації;
- відсутність єдиних стандартів на системи кодування й опрацювання;
- складні апаратні й програмні рішення проблем аналого-цифрових перетворень і синхронізації повноекранного відео.

Зараз не існує єдиної методології, а тим більше технології розроблення ГМ систем, такий стан характеризується як «no science, rather art», тобто «не наука, а мистецтво». Однак перші кроки до створення технології вже зроблені [299], і розроблення в цій області продовжують активно розвиватися.

Традиційно процес розроблення ГМ передбачає кілька фаз.

Фаза 1 – проектування і розроблення структури й окремих фрагментів гіпермедіа-додатка, включаючи звукові відео-ролики, програмне оточення. Фаза 1 – найбільш трудомісткий за часом і людськими ресурсами період. Цю фазу здійснює команда, що складається з:

- менеджера;
- фахівця із систем навчання;
- експерта предметної області;
- дизайнера-графіка;
- системного аналітика;
- сценариста;
- програміста.

Фаза 2 залежить від того, на який варіант додатка розрахована система.

Для Інтернет-додатків розробляються Web-сторінки мовою HTM із включенням цифрових у спеціальних форматах звуків, зображень, анімації й відео. При цьому можуть використовуватися спеціальні пакети, наприклад, Dreamweaver або FrontPage.

Для виготовлення CD-дисків використовується як HTM-формат, так і спеціальні програмні засоби (authoring languages and systems), як, наприклад, ToolBook або Macromedia Director.

Фаза 3 включає прототип CD-ROM на спеціальному пристрої WORM (Write Once Read Many) і його тестування або розміщення додатка на Інтернет-сайті.

Фаза 4 – це тиражування й виробництво готових CD-ROM.

7.5.3. На шляху до адаптивних навчальних систем

Інструментом пізнання вважатимемо процес проектування структури гіпертексту. Нами був запропонований підхід [57], що трактує структуру (граф) гіпертексту (ГТ) як «кістяк» поля знань, тобто найбільш інформаційно-навантажений елемент поля знань. Саме це віддзеркалює структуру знань, яку можна назвати гіпернаннями. Засвоєння цієї структури є найважливішим компонентом процесу навчання. Очевидно, чим більше буде конкретна структура ГТ відповідати індивідуальній когнітивній структурі, тим

ефективніше буде відбуватися процес навчання. Фактично це означає, що навчання й тренінг будуть організовані не через нав'язування конкретних когнітивних структур тому, кого навчають, і ламання старих уявлень, а через проекцію нового знання на каркас індивідуального досвіду й нарощування вже наявних когнітивних структур. Така адаптація до індивідуальних пізнавальних структур може істотно підвищити ефективність навчальних систем.

Пропонований підхід дозволяє зобразити поле знань предметної області у вигляді релевантної гіперструктури HS, вузлами якої є концепти A, виділені експертами як «опорні», тобто принципово значимі зв'язки або відношення, що використовуються для переходів між вузлами. Таке трактування є природним розвитком моделей подання знань типу семантичних мереж, які на сучасному етапі вважаються найбільш адекватними структурі людської пам'яті. Природно, що такі концептуально-когнітивні структури вирізняються різкою індивідуальністю, пов'язаною з особистими, інтелектуальними й професійними розходженнями носіїв знань. У деякому сенсі така структура є когнітивною моделлю індивіда.

Пропонований підхід узгоджується з концепцією мультидерева [240], що реалізує множинне відображення ПО. Модель користувача, сформована на цьому принципі, буде відбивати «модель світу» певного конкретного користувача у вигляді гіперграфа, вузли якого, у свою чергу, можуть бути розкриті у вигляді гіперструктур нижчого рівня. Фактично це відповідає суб'єктивним концептуальним структурам у пам'яті. Часто в проблематиці АНС використовують поняття сценарію. Традиційно під сценарієм розуміється опис змістовної, логічної й тимчасової взаємодії структурних одиниць програми, за допомогою яких реалізується авторська мета [17].

Очевидно, що в гіпермедіа навчальних додатках природним поданням сценарію є граф гіпертексту. При цьому важливо домогтися, щоб цей сценарій відповідав (був релевантним) уявленням як учителів, так і учнів.

Релевантна ГТ-структура відображає стратифіковану мережу HS = <A,R>, що відображає ієрархію понять предметної області у формі, яка відповідає професійним уявленням експертів і не викликає когнітивного дисонансу в користувача.

Для формування стратифікованої множини вершин A можна використовувати модифікацію алгоритмів об'єктно-структурного аналізу ОСА.

Алгоритм «ОСА-гіпер»

1. Сформулювати мету й зібрати від експертів всю доступну інформацію.
2. Визначити кількість страт N, що підлягають формуванню.
3. З інформації, отриманої від експертів і з літератури, вибрати всі значимі основні об'єкти й поняття {A} і розмістити їх за витратами, сформувавши опорні метавузли ГТ-структури.
4. Деталізувати концепти, користуючись спадною концепцією (top-down).
5. Утворити метапоняття за концепцією (bottom-up).
6. Виключити повтори, надмірність і синонімію.
7. Обговорити поняття, що не ввійшли в структуру ГТ із експертом і включити їх або вилучити.
8. Виявити основні відносини {R} і спроектувати ескіз ГТ структури.

9. Здійснити обстеження потенційних користувачів з метою виявлення їхніх когнітивних уявлень про задану ПО, а також формування моделі користувача.
10. Привести у відповідність ескіз ГТ і подання користувача.
11. Сформувати ряд сценаріїв обходу ГТ із метою спрощення навігації й обліку необхідних сценарних зв'язків і включити їх у структуру.

Цей алгоритм виявляє канонічний або основний сценарій, альтернативні сценарії формуються на підставі деякого іншого досвіду викладання, або з урахуванням категорії користувачів.

При розробленні гіпермедіа-додатків необхідно також урахувати фактор збалансованості звукових і відеовузлів опорної ГТ-структури, тобто аудіо-і відеофрагменти повинні рівномірно розподілятися по мережі. Для цього введене поняття «розфарбування» вузлів, що дозволяє наочно оцінити збалансованість мережі за аудіо- і відеонавантаженням. Алгоритм «ОСА-гіпер» враховує необхідність аналізу не тільки вихідної інформації, але й навігаційних можливостей:

- переміщення за екраном назад;
- повернення до початку;
- повернення до початку секції;
- перегляд структури;
- озвучування екрана;
- включення відео;
- допомога;
- переміщення на екран уперед.

Особливо корисним в реалізації графічної навігації є те, що в довільний момент користувач може подивитися структуру сценарію й зрозуміти, в якому місці він зараз перебуває.

Такий підхід використаний при створенні ГТ АНС в області інженерії знань для систем дистанційного навчання. Системи навчання в області ІШ мають наразі невелику історію, хоча при відсутності підручників із цих дисциплін ці електронні підручники особливо необхідні. Відомі лише поодинокі роботи в нашій країні й за її межами – «База знань для розроблювачів ЕС» і KARTT (Knowledge Aqcuisition Research and Teaching Tool).

При проектуванні будь-якої ГТ-структури нетривіальним завданням є виділення «опорних» концептів – питань, практично не висвітлених у літературі. У роботі [43] наведений огляд різних методів видобування знань, що частково включають і дослідження з виявлення значущих концептів. Можна запропонувати використовувати для цієї мети методи багатовимірного градуювання, що дозволяють виявляти структуру індивідуальних ментальних просторів, аналізуючи попарні зв'язки понять предметної області.

Ці методи використовувалися й раніше для вивчення семантичних просторів пам'яті, однак можна використовувати новий підхід, орієнтований на аналіз не тільки осей ментальних просторів з виявленням відповідних конструктів, але й точок згущення понять, названих «атракторами» для виявлення метапонять або концептів.

Надалі ці концепти утворять вузли {A} релевантної ГТ-структури, яку можна фактично трактувати як модель користувача UM (user model):

$$UM = \{A, R\}.$$

Глибші дослідження [224] показали, що психосемантичні методи збагачені новими елементами (використання метафоричних планів), можуть сприяти виявленню імпліцитних структур знань, що не піддаються виявленню іншими методами.

Використання UM як гіперструктури в гіпертекстових АНС дозволяє створювати «гнучкі» релевантні сценарії, орієнтовані на когнітивні особливості певних груп користувачів. Останнє зауваження можна також віднести до розроблення систем гіпермедіа. Очевидно, що глибоке й конструктивне вивчення людського фактора в області комп'ютерних наук може істотно розсунути межі сучасних інтелектуальних і навчальних систем.

Запитання для повторення та контролю знань

1. Для чого слід використовувати методи психосемантики?
2. Які форми використовують психолінгвістичні методи?
3. Що таке семантичний простір знань? Наведіть приклад.
4. Чому семантичні простори двох кваліфікованих фахівців можуть бути різними?
5. Наведіть приклад асоціативної мережі структури знань.
6. Які методи відносять до методів психологічного градуювання?
7. Як здійснюється побудова семантичного простору?
8. Що таке багатовимірне градуювання?
9. Для чого використовуються методи багатовимірного градуювання?
10. Яку модель використовують методи багатовимірного градуювання?
11. Чим відрізняється психосемантика від інженерії знань?
12. Поясніть використання метафор для виявлення «прихованих» структур знань.
13. Які Ви знаєте рівні градуювання знань?
14. Які орієнтовані методи видобування знань?
15. У чому полягає підхід використання метафор?
16. Як використовуються решітки Келлі для психосемантичних тестів?
17. Як можна інтерпретувати геометричну модель градуювання?
18. У чому полягає метод репертуарних решіток?
19. Що являють собою репертуарні решітки?
20. Які Ви знаєте методи виявлення конструктів?
21. Опишіть метод мінімального контексту.
22. Що дозволяє визначити аналіз репертуарних решіток?
23. Що дозволяє визначити аналіз декількох репертуарних решіток?
24. Які існують способи порівняння двох решіток?
25. У чому полягає аналіз груп системних конструктів?
26. Наведіть приклади систем, в яких реалізовані автоматизоване створення репертуарних решіток і видобування з експертів конструктів.

27. Що означає термін «керування знаннями»?
28. Які технології використовуються для застосування КМ-систем?
29. Які причини розроблення КМ-систем?
30. Що означає поняття «корпоративна пам'ять»?
31. Які існують рівні корпоративної пам'яті?
32. Які існують етапи під час розроблення систем КМ?
33. Які основні функції OMIS?
34. Як виглядає архітектура OMIS?
35. Які особливості розроблення OMIS?
36. Які Ви знаєте візуальні методи специфікації й проектування баз знань і розроблення концептуальних структур?
37. Чим є релевантна ГТ-структура?

РОЗДІЛ 8

МАШИННЕ НАВЧАННЯ ТА НЕЙРОННІ МЕРЕЖІ

- Поняття машинного навчання
- Генетично-адаптивні алгоритми
- Інтелектуальний аналіз даних. Побудова дерева рішень
- Штучні нейронні мережі

Очевидно, що саме знання є центральною ланкою будь-якої інтелектуальної системи. Однак завдання отримання знань від людей-спеціалістів є надто складним, тому в таких системах використовуються методи та алгоритми машинного навчання. Даний розділ й присвячено таким методам та алгоритмам.

8.1. Поняття машинного навчання

Під машинним навчанням розуміємо будь-яке покращення роботи інтелектуальної системи, яка є результатом накопичення досвіду. Навчальні алгоритми націлені на досягнення одного або декількох наступних результатів:

1) розширити коло розв'язувальних задач;
2) видавати точніші розв'язки;
3) отримувати відповіді з меншими затратами;
4) спростити вже наявні знання.

Будь-яка система, створена з таким розрахунком, щоб вона могла модифікувати і покращувати свою роботу, повинна містити наступні 4 головні компоненти:

1) множина інформаційних структур, в яких кодується поточний рівень інтелектуальності системи (база знань);
2) алгоритм задачі («виконавець»), який використовує цю базу знань для керування активністю системи;
3) модуль зворотнього зв'язку («критик»), який співставляє досягнуті результати з бажаними;
4) навчальний механізм («учень»), який використовує зворотній зв'язок, що надходить від критика для покращення бази знань.

8.1.1. Базові визначення

Поняття «навчання» та «самонавчання» є одними з ключових у теорії штучного інтелекту. Вони явно або неявно зустрічаються в усіх розділах підручника.

У психології під навчанням розуміється набуття умінь та навичок, які раніше були невідомими. Кажуть, що система навчилася чомусь, якщо вона стала здатною до вирішення якихось завдань, які до цього вона вирішувати не могла. Звичайно, якщо нова процедура просто кимсь програмується, навряд чи є сенс говорити про справжнє навчання (згадайте поняття декларативного і процедурного підходів, а також дискусію про визначення інтелекту взагалі). Тому часто спеціально підкреслюється, що при навчанні інтелектуальної системи відбувається самостійне виведення нової інформації з наявної або щойно отриманої [71].

Уїнстон [170] виділяє такі рівні навчання: навчання шляхом програмування; навчання шляхом пояснення; навчання на прикладах; навчання шляхом відкриття.

Можна сказати, що при переході від нижчих рівнів до вищих «учень» виконує все більшу частину тієї роботи, яку раніше виконував «учитель».

Навчання без «учителя» називається самонавчанням.

8.1.2. Автомати з лінійною тактикою

Історично першими були моделі навчання на основі стимульно-реактивної теорії. За суттю, це було моделювання процесу утворення умовних рефлексів на основі «покарань» і «заохочень», які отримує система внаслідок своїх дій. Створенню цих моделей передував ряд фізіологічних експериментів над тваринами. Досить показовим є, наприклад, такий.

Піддослідний пацюк має можливість обрати одну із двох дій: побігти направо або наліво. Якщо він побіжить наліво, він отримує заохочення – його годують. Якщо ж пацюк побіжить направо, він отримує покарання – удар електричним струмом. Пацюк заздалегідь не знає, куди йому бігти, і він навчається цьому шляхом проб і помилок.

Для моделювання такого типу навчання виявилося зручним застосовувати автомати з лінійною тактикою. Розглянемо основні принципи функціонування цих автоматів.

Нехай проводиться серія експериментів, і система в кожному експерименті має можливість вибрати одну з m можливих дій. Тоді автомат з лінійною тактикою, який моделює поведінку такої системи, має m груп станів, кожна група станів відповідає певній дії. Точніше, якщо автомат перебуває в будь-якому стані i-ї групи, він вибере дію, яка відповідає цій групі.

Кожна група має q станів. Параметр q називається ***глибиною пам'яті автомата***. Стани кожної групи пронумеровані від 1 до q – від найменш глибокого стану до найбільш глибокого. Позначимо через $k^{(j)}$ j-й стан k-ї групи.

Автомат з лінійною тактикою, який має три групи станів і глибину пам'яті 4, зображено на рис. 8.1. Чорною крапкою відзначено поточний стан (у нашому випадку – 1-й стан 1-ї групи). Це означає, що при черговому експерименті автомат вибере першу дію.

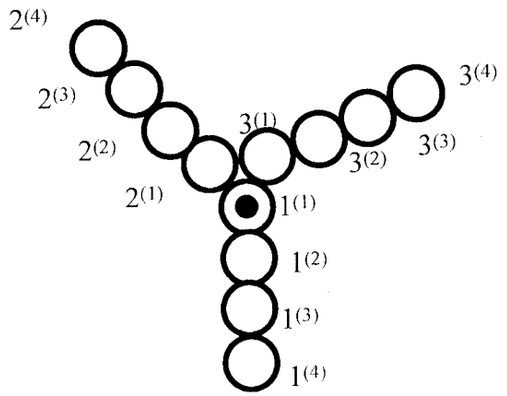


Рис. 8.1. Автомат з лінійною тактикою.

Навчання автомату з лінійною тактикою здійснюється шляхом переходу від одного стану до іншого. Ці переходи відбуваються відповідно до наступних правил.

Якщо автомат, перебуваючи в стані $k^{(j)}$ (j-й стан k-ї групи), в результаті вибору k-ї дії отримує заохочення, він переходить до стану $k^{(j-1)}$ (більш глибокого стану цієї ж групи), якщо $j < q$. Інтуїтивно це можна розуміти як "**підкріплення впевненості у правильності свого вибору**". Якщо ж $j = q$ (тобто якщо автомат перебуває у найглибшому стані деякої групи), він залишається у цьому ж стані.

Так, автомат, зображений на рис. 8.1, перейде до другого стану цієї ж групи (рис. 8.2).

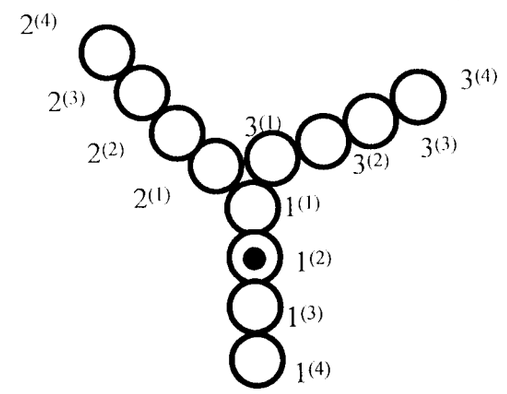


Рис. 8.2. Зміна стану в результаті заохочення.

Якщо автомат, перебуваючи в стані $k^{(j)}$, в результаті вибору k-ї дії отримує покарання, він переходить до стану $k^{(j+1)}$ (менш глибокого стану цієї ж групи), якщо $j > 1$. Інтуїтивно це можна розуміти як "**послаблення впевненості у правильності дії**". Якщо ж $j = 1$ (тобто, якщо автомат перебуває у першому стані деякої групи), він переходить до пер-

шого стану деякої іншої групи. Це означає зміну поведінки: в наступному експерименті автомат вибере іншу дію.

Так, автомат, зображений на рис. 8.1, перейде до першого стану другої групи (рис. 8.3).

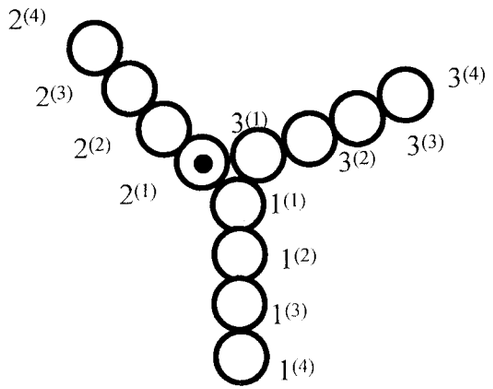


Рис. 8.3. Зміна стану в результаті покарання.

Ми бачимо, що глибина пам'яті має дуже велике значення. Вона характеризує, скільки покарань в середньому повинен отримати автомат для того, щоб змінити свою поведінку.

У найпростішому випадку, коли кожна дія завжди пов'язана або з покаранням, або із заохоченням, автомат врешті-решт опиняється в найглибшому стані найоптимальнішої дії та залишиться там назавжди. Глибина пам'яті в цьому випадку повинна бути мінімальною.

Нехай тепер при виборі кожної дії покарання й заохочення відбуваються з певними ймовірностями. У цьому випадку збільшення глибини пам'яті автомата збільшує його «інертність» та зменшує вірогідність випадкових переходів від більш вигідної поведінки до менш вигідної.

Ситуація ще більш ускладнюється, якщо характеристики середовища, тобто ймовірності заохочень і покарань, не залишаються сталими, а змінюються з часом. У цьому випадку автомат повинен перенавчатися відповідно до змін середовища, і глибина пам'яті, що характеризує швидкість такого перенавчання, не повинна бути ні надто великою, ні надто малою.

Нарешті, важливий напрямок досліджень був пов'язаний з колективною поведінкою автоматів, коли розглядається функціонування багатьох автоматів з лінійною тактикою, кожний з яких переслідує свої цілі.

8.1.3. Формування та засвоєння понять

Одне з найважливіших місць у процесі навчання займає узагальнення, яке полягає у виділенні спільних рис, характерних для тих чи інших явищ і понять. Формування понять є одним з найважливіших видів узагальнення. Проблему формування понять можна неформально сформулювати таким чином: на основі наявних спостережень виробити структурований опис поняття, або ж набір правил, які дозволяють визначити, чи

охоплюється той чи інший об'єкт певним поняттям. Якщо ж поняття уже сформовані і потрібно лише передати знання про ці поняття інтелектуальній системі, йдеться про засвоєння понять.

Однією з перших формалізацій процесу формування і засвоєння понять стала класична схема Бонгарда [51]. Правила, які відділяють одне поняття від іншого, шукаються у вигляді диз'юнкції деяких логічних функцій. Пошук інформативних ознак, від яких повинні залежати ці функції, та комбінацій ознак, які мають значення для відділення одного поняття від іншого, здійснюється шляхом перебору.

Уїнстон у [170] описує метод засвоєння понять на прикладах шляхом зміни модельного опису. При демонстрації першого прикладу система шляхом його безпосереднього аналізу формує у своїй пам'яті початкову модель поняття у вигляді деякої семантичної мережі. Далі описується методика корекції цієї семантичної мережі при демонстрації підкріплювальних прикладів (представників цього поняття) і контрприкладів типу «майже того». «Майже те» – це приклад, який не є представником цього класу через деякі невідповідності.

Згадаємо програми ЕВРІСКО та АМ, створені Д. Ленатом [277]. Ці програми мали можливість на основі аналізу певного набору базових математичних правил і понять встановлювати нові співвідношення і формувати нові поняття.

Параметричне навчання є, очевидно, найбільш дослідженим підходом до навчання систем штучного інтелекту. Воно знаходить широке застосування, зокрема в системах розпізнавання образів.

Нехай, наприклад, система повинна навчитися відрізняти одне поняття від іншого. При параметричному навчанні відразу задається загальний вигляд функції, яка може бути використана для розділення понять. Невідомими залишаються лише деякі параметри, від яких залежить ця функція, і саме ці параметри повинні бути визначені в процесі навчання.

8.1.4. Базові поняття теорії індуктивних виведень

Слово «індукція» означає здійснення виведення «від часткового до загального». Індуктивне виведення полягає у тому, що інтелектуальна система спостерігає певний набір фактів і на основі цього формує загальне правило, яке описує всі ці факти і дозволяє отримувати нові подібні факти. В іншій інтерпретації, спостерігається набір об'єктів, для яких справджується певне правило, і на основі цих спостережень робиться висновок, що це правило справджується для всіх предметів. На відміну від дедуктивного, індуктивне виведення може виявитися хибним.

Слід розрізняти індуктивну перевірку гіпотез та індуктивне формування гіпотез на основі прикладів. У першому випадку перевіряється деяка вже сформована гіпотеза. Якщо при цьому не вдається здійснити повне перебирання, ми ніколи не зможемо бути повністю впевнені у справедливості гіпотези; ми можемо лише збільшити або зменшити ступінь нашої впевненості.

У другому ж випадку на початку дослідження гіпотези ще не існує. Вона формується на основі аналізу прикладів, і після цього її часто можна перевірити іншим шляхом.

8.1.5. Правила формування гіпотез Мілля

Дж. Мілль запропонував чотири правила індуктивних виведень [58]. Суть їх зводиться до такого.

Правило 1 (*метод виявлення подібностей*). Нехай ми маємо такі спостереження: **(a, d, q_1); (a, d, q_2); …, (a, d, q_n).** Тут **a, d** – деякі факти; $q_1, \ldots, q_n$ – набори деяких інших фактів. Іншими словами, ми маємо ***n*** спостережень, в кожному з яких мало місце і **a**, і **d**. На підставі таких спостережень можна висунути гіпотезу: *"a та d завжди зустрічаються разом"*. Якщо у нашому розпорядженні є більше інформації, на основі цієї додаткової інформації інколи можна зробити складніші виведення: *"a є причиною d"* або *"d є причиною a"*.

Правило 2 (*метод виявлення відмінностей*). Це правило дозволяє зробити ті самі виведення, що й правило 1, але з більшою мірою впевненості. Точніше, відповідне індуктивне виведення *"a та d завжди зустрічаються разом"* робиться, якщо ми маємо ***n*** спостережень, причому в ***q*** випадках спостерігалися і **a**, і **d,** а в ***n-q*** випадках не мали місця ні **a**, ні **d**.

Правило 3 (*метод виявлення залишку*). Якщо у нас є певний набір спостережень, на основі яких можна зробити виведення, що поява елементів *(a, b)* тягне за собою появу елементів *(c, d)*, а на основі інших спостережень встановлено, що *"b є причиною c"*, то можна зробити індуктивне виведення, що *"a є причиною d"*

Правило 4 (*метод зв'язних змін*). Нехай $\mathbf{a_i} \in \mathbf{A}$, $\mathbf{d_i} \in \mathbf{D}$, і ми маємо групу спостережень: **(a_i, …, d_i);** крапками позначені несуттєві елементи. Можна зробити висновок про те, що поява елемента з множини **A** завжди тягне за собою появу елемента з множини **D**.

На основі правил Мілля був запропонований відомий ДСМ-метод автоматичного формування гіпотез [152], названий так на честь Джона Стюарта Мілля. Широко розповсюджений також GUHA-метод [51].

Як один із класичних прикладів застосування теорії індуктивного виведення можна навести таку задачу: заданий числовий ряд, наприклад, 1, 4, 9, 16, 25; знайти наступний член цього ряду. Розвивається формалізована математична теорія індуктивного виведення [18].

8.1.6. Індуктивна перевірка гіпотез і парадокс Хемпеля

З індуктивною перевіркою гіпотез пов'язаний відомий парадокс Хемпеля, суть якого проілюструємо на класичному прикладі про чорну ворону.

Нехай учений бажає перевірити гіпотезу «Всі ворони чорні». Природний шлях для індуктивної перевірки цієї гіпотези – дослідити якомога більшу кількість ворон і перевірити, чорні вони чи ні. Кожна виявлена не чорна ворона спростовує гіпотезу, але кожна виявлена чорна ворона збільшує впевненість у справедливості цієї гіпотези, тобто є підкріплювальним прикладом для неї.

Але твердження «Всі ворони чорні» еквівалентне твердженню «Всі не чорні предмети не є воронами». Спостереження, наприклад, зеленої (жовтої, синьої) книги збільшує міру впевненості у справедливості останньої гіпотези, але твердження про те, що воно ж є підкріпленням для гіпотези «Всі ворони чорні», викликає сумнів. Якщо це так, то Зелена книга з тим же успіхом підкріплює і гіпотезу «Всі ворони білі».

До цього можна додати ще й таке міркування. Нехай деякий інопланетянин перевіряє гіпотезу «Всі люди на Землі нижчі за 5 метрів». Зрозуміло, що виявлення людини зі зростом 4 метри 99 сантиметрів, хоч і підтверджує гіпотезу, не підкріплює її, а навпаки, послаблює.

8.1.7. Поняття про генетичні алгоритми

Загальновідомо, що жива природа розвивається шляхом природного еволюційного відбору. Останнім часом інтенсивно розвивається напрям досліджень, пов'язаний з набуттям системами штучного інтелекту здатності до подібного еволюційного розвитку. Для моделювання процесу еволюції були розроблені спеціальні методики, які дістали назву генетичних алгоритмів. Перші роботи з цього напряму відносяться до середини 70-х років XX століття.

Генетичний алгоритм починає свою роботу з деякого набору даних і процедур і намагається комбінувати їх різним чином (відбувається «схрещування»). Далі з цих комбінацій відбираються найкращі за певним критерієм («селекція»), і саме вони беруть участь у наступних схрещуваннях. Інколи до даних і процедур, які залучені до еволюційного відбору, вносяться певні випадкові зміни (відбуваються «мутації»). Генетичні алгоритми непогано зарекомендували себе для розв'язання ряду погано формалізованих та слабко структурованих задач.

8.2. Генетично-адаптивні алгоритми

Природа вражає своєю складністю і багатством всіх своїх проявів. Серед прикладів можна назвати складні соціальні системи, імунні й нейронні системи, складні взаємозв'язки між видами. Вони – всього лише деякі з див, що виявилися очевиднішими, коли ми почали глибше досліджувати себе самих і світ навколо нас. Наука – це одна із систем віри, що змінює інші, якими ми намагаємось пояснити те, що спостерігаємо, цим самим змінюючи себе, щоб пристосуватися до нової інформації, одержуваної з навколишнього середовища. Багато чого з того, що ми бачимо і спостерігаємо, можна пояснити єдиною теорією: теорією еволюції через спадковість, змінність і відбір.

Теорія еволюції вплинула на зміну світогляду людей із самої своєї появи. Теорія, що Чарльз Дарвін навів у роботі, відомій як «Походження видів», 1859 року стала початком цієї зміни. В багатьох областях наукового знання є свобода думки, що багато чим завдячує революції, викликаній теорією еволюції та розвитку. Але Дарвін, як і багато його сучасників, які припускали, що в основі розвитку лежить природний відбір, не міг не помилятися. Наприклад, він не зміг показати механізм успадкування, при якому підтримується змінність. Його гіпотеза про пангенезис виявилася хибною. Це було за п'ятдесят років до того, як теорія спадковості почала поширюватися світом, і за тридцять років до того, як «еволюційний синтез» зміцнив зв'язок між теорією еволюції і відносно молодою наукою генетикою. Однак Дарвін виявив головний механізм розвитку: відбір у поєднанні зі змінністю або, як він його називав, «спуск із модифікацією». У багатьох випадках, специфічні особливості розвитку через змінність і відбір все ще не безперечні, однак, основні механізми пояснюють неймовірно широкий спектр явищ, які спостерігаються в природі.

Тому не дивно, що вчені, які займаються комп'ютерними дослідженнями, звернулися до теорії еволюції в пошуках натхнення. Можливість того, що обчислювальна система, наділена простими механізмами змінності і відбору, могла б функціонувати за аналогією з законами еволюції в природних системах, була дуже привабливою. Ця надія виявилася причиною появи ряду обчислювальних систем, побудованих на принципах природного відбору. Історія еволюційних обчислень почалася з розроблення ряду різних незалежних

моделей. Основними з них були генетичні алгоритми і класифікаційні системи Голанда (Holland), опубліковані на початку 60-х років минулого століття і які отримали загальне визнання після виходу у світ книги, що стала класикою в цій області, – «Адаптація в природних і штучних системах» («Adaptation in Natural and Artifical Systems», 1975).

Основні труднощі з можливістю побудови обчислювальних систем, заснованих на принципах природного відбору і застосуванням цих систем у прикладних задачах, полягають в тому, що природні системи досить хаотичні, а всі наші дії, фактично, носять чітку спрямованість. Ми використовуємо комп'ютер як інструмент для розв'язування визначених задач, які ми самі і формулюємо та акцентуємо увагу на максимально швидкому виконанні при мінімальних витратах. Природні системи не мають жодних таких цілей чи обмежень, у всякому разі, нам вони не відомі. Виживання в природі не спрямоване до деякої фіксованої мети, замість цього еволюція робить крок вперед у будь-якому доступному напрямі. Можливо це велике узагальнення, але є думки, що зусилля, спрямовані на моделювання еволюції за аналогією з природними системами, до певного часу можна розбити на дві великі категорії:

1) системи, що змодельовані на біологічних принципах; вони успішно використовуються для задач функціональної оптимізації і можуть легко бути описані небіологічною мовою;
2) системи, що є біологічно реалістичнішими, але які не виявилися особливо корисними в прикладному сенсі; вони більше схожі на біологічні системи і менш спрямовані (чи не спрямовані зовсім); вони мають складну і цікаву поведінку, і, мабуть, незабаром набудуть практичного застосування.

Зазвичай на практиці ми не можемо розділяти ці системи так строго. Ці категорії – просто два полюси, між якими лежать різні обчислювальні системи. Ближче до першого полюса – еволюційні алгоритми, такі як Еволюційне Програмування (Evolutionary Programming), Генетичні Алгоритми (Genetic Algorithms) і Еволюційні Стратегії (Evolution Strategies).

Ближче до іншого полюса – системи, що можуть бути класифіковані як Штучне Життя (Artificial Life).

Звичайно, еволюція біологічних систем не єдине «джерело натхнення» творців нових методів, що моделюють природні процеси.

8.2.1. Еволюційна теорія

Як відомо, еволюційна теорія стверджує, що життя на нашій планеті виникло спочатку лише в найпростіших формах – у вигляді одноклітинних організмів. Ці форми поступово ускладнювалися, пристосовуючись до навколишнього середовища, породжуючи нові види, і тільки через багато мільйонів років з'явилися перші тварини і люди. Можна сказати, що кожен біологічний вид з часом вдосконалює свої якості так, щоб найбільш найефективніше справлятися з найважливішими завданнями виживання, самозахисту, розмноження та ін. Таким шляхом виникло захисне забарвлення в багатьох риб і комах, панцир у черепахи, отрута в скорпіона і багато інших корисних застосувань.

За допомогою еволюції природа постійно оптимізує все живе, знаходячи часом неординарні рішення. Незрозуміло, за рахунок чого відбувається цей прогрес, однак йому можна дати наукове пояснення, ґрунтуючись лише на двох біологічних механізмах – природного відбору і генетичної спадковості.

8.2.2. Природний відбір і генетична спадковість

Головну роль в еволюційній теорії грає природний відбір. Його суть полягає в тому, що найбільш пристосовані особини краще виживають і мають більше нащадків, ніж менш пристосовані. Відзначимо, що сам собою природний відбір ще не забезпечує розвиток біологічного виду. Тому дуже важливо вивчити, яким чином відбувається успадкування, тобто як властивості нащадка залежать від властивостей батьків.

Основний закон спадковості інтуїтивно зрозумілий кожному – він полягає в тому, що нащадки схожі на батьків. Зокрема нащадки більш пристосованих батьків будуть, швидше за все, одними з найбільш пристосованих у своєму поколінні. Щоб зрозуміти, на чому заснована ця подібність, нам треба частково заглибитися в будову природної клітини – у світ генів і хромосом.

Майже в кожній клітині будь-якої особини є набір хромосом, що містять інформацію про цю особину. Основна частина хромосоми – нитка ДНК, що визначає, які хімічні реакції будуть відбуватися в цій клітині, як вона буде розвиватися і які функції виконуватиме.

Ген – це відрізок ланцюга ДНК, відповідальний за визначену властивість особини, наприклад, за колір очей, тип волосся, колір шкіри і т. ін. Вся сукупність генетичних ознак людини кодується за допомогою приблизно 60 тис. генів, довжина яких складає понад 90 млн. нуклеотидів.

Розрізняють два види клітин: статеві (такі, як сперматозоїд і яйцеклітина) і соматичні. У кожній соматичній клітині людини міститься 46 хромосом. Ці 46 хромосом – насправді 23 пари, причому в кожній парі одна з хромосом отримана від батька, а друга – від матері. Парні хромосоми відповідають за однакові ознаки – наприклад, батьківська хромосома може містити ген чорного кольору ока, а парна їй материнська – ген голубого кольору. Існують визначені закони, що керують участю тих чи інших генів у розвитку особини. Зокрема, у нашому прикладі нащадок буде, переважно, чорнооким, оскільки ген блакитних очей є «слабким» (рецесивним) і утискається геном будь-якого іншого кольору.

У статевих клітинах хромосом тільки 23, і вони непарні. При заплідненні відбувається злиття чоловічої і жіночої статевих клітин і утворюється клітина зародка, що містить саме 46 хромосом. Які властивості нащадок одержить від батька, а які – від матері? Це залежить від того, які саме статеві клітини брали участь у заплідненні. Справа в тім, що процес вироблення статевих клітин (так званий мейоз) в організмі піддається випадкам, завдяки яким нащадки все-таки багато в чому відрізняються від своїх батьків. При мейозі, зокрема, відбувається наступне: парні хромосоми соматичної клітини зближаються впритул, потім їхні нитки ДНК розриваються в декількох випадкових місцях і хромосоми обмінюються своїми частинами (рис. 8.4).

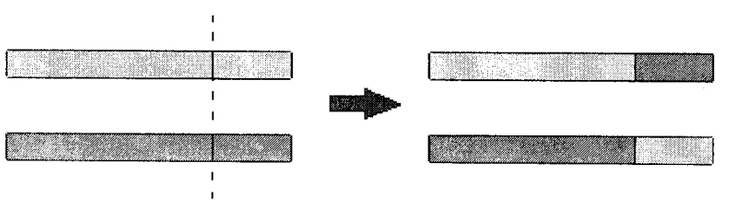


Рис. 8.4. Умовна схема кросинговеру

Цей процес забезпечує появу нових варіантів хромосом і зветься «кросинговер». Кожна з нових хромосом з'явиться потім всередині однієї зі статевих клітин, і її генетична інформація може реалізуватись в нащадках цієї особини.

Другий важливий фактор, що впливає на спадковість, – це мутації, що полягають у зміні деяких ділянок ДНК. Мутації також випадкові і можуть бути викликані різними зовнішніми факторами, такими, як радіоактивне випромінювання. Якщо мутація відбулася в статевій клітині, то змінений ген може передатися нащадку і проявитися у вигляді успадкованої хвороби або в інших нових властивостях нащадка. Вважається, що саме мутації є причиною появи нових біологічних видів, а кросинговер визначає вже змінність всередині виду (наприклад, генетичні розходження між людьми).

8.2.3. Задачі оптимізації

Як уже було відзначено вище, еволюція – це процес постійної оптимізації біологічних видів. Тепер ми в стані зрозуміти, як відбувається цей процес. Природний відбір гарантує, що найбільш пристосовані особини дадуть досить велику кількість нащадків, а завдяки генетичному успадкуванню ми можемо бути упевнені, що частина нащадків не тільки збереже високу пристосованість батьків, але буде мати і деякі нові властивості. Якщо ці нові властивості виявляються корисними, то з великою імовірністю вони перейдуть і в наступне покоління. Отже, відбувається нагромадження корисних якостей і поступове підвищення пристосованості біологічного виду в цілому. Знаючи, як розв'язується задача оптимізації видів у природі, ми тепер застосуємо схожий метод для розв'язування різних реальних задач.

Задачі оптимізації – найбільш розповсюджений і важливий для практики клас задач. Їх доводиться розв'язувати кожному з нас або в побуті, розподіляючи свій час між різними справами, або на роботі, домагаючись максимальної швидкості роботи програми чи максимальної прибутковості компанії – у залежності від посади. Серед цих задач є розв'язувані простим шляхом, але є і такі, точний розв'язок яких знайти практично неможливо.

Уведемо позначення і подамо кілька класичних прикладів. Як правило, у задачі оптимізації ми можемо керувати декількома параметрами (позначимо їх значення через x1, x2, …, xn, а нашою метою є максимізація (чи мінімізація) деякої функції, f (x1, x2, …, xn), що залежить від цих параметрів. Функція f називається цільовою функцією. Наприклад, якщо потрібно максимізувати цільову функцію «дохід компанії», тоді керованими параметрами будуть число співробітників компанії, обсяг виробництва, витрати на рекламу, ціни на кінцеві продукти тощо. Важливо відзначити, що ці параметри пов'язані між собою – зокрема, при зменшенні числа співробітників швидше за все впаде й обсяг виробництва.

Генетичний алгоритм – новітній, але не єдино можливий спосіб розв'язування задач оптимізації.

Відомо два основні шляхи розв'язування таких задач – переборний та градієнтний. Розглянемо класичну задачу комівояжера. Суть задачі полягає у знаходженні короткого шляху проходження всіх міст.

- Перебірний метод є найпростішим. Для пошуку оптимального розв'язку (максимум цільової функції) потрібно послідовно обчислити значення функції у всіх точках. Недоліком є велика кількість обчислень.

- Іншим способом є градієнтний спуск. Обираємо випадкові значення параметрів, а потім значення поступово змінюють, досягаючи найбільшої швидкості зростання цільової функції. Алгоритм може зупинитись, досягнувши локального максимуму. Градієнтні методи швидкі, але не гарантують оптимального рішення (оскільки цільова функція має декілька максимумів).

Генетичний алгоритм являє собою комбінацію перебірного та градієнтного методів. Механізми кросинговеру (схрещування) та мутації реалізують перебірну частину, а відбір кращих рішень – градієнтний спуск.

Тобто, якщо на деякій множині задана складна функція від декількох змінних, тоді генетичний алгоритм є програмою, яка за певний час знаходить точку, де значення функції перебуває достатньо близько до максимально можливого значення. Обираючи прийнятний час розрахунку, отримуємо одне з кращих рішень, які можна отримати за цей час.

8.2.4. Робота генетичного алгоритму

Уявимо собі штучний світ, населений множиною істот (особин), причому кожна особина – це деяке розв'язування нашої задачі. Будемо вважати особину тим більше пристосованою, чим кращий відповідний розв'язок (чим більше значення цільової функції вона дає). Тоді задача максимізації цільової функції зводиться до пошуку найбільш пристосованої істоти. Звичайно, ми не можемо поселити в наш віртуальний світ всі істоти відразу, тому що їх дуже багато. Замість цього ми будемо розглядати багато поколінь, що змінюють одне одного. Тепер, якщо ми зуміємо ввести в дію природний відбір і генетичне спадкування, тоді отримане середовище буде підкорятися законам еволюції. Відзначимо, що, відповідно до нашого визначення пристосованості, метою цієї штучної еволюції буде саме створення найкращих рішень. Очевидно, еволюція – нескінченний процес, у ході якого пристосованість особин поступово підвищується. Примусово зупинивши цей процес через досить тривалий час після його початку і вибравши найбільш пристосовану особину у поточному поколінні, ми одержимо не абсолютно точну, але близьку до оптимальної відповідь. Така, коротенько, ідея генетичного алгоритму. Перейдемо тепер до точних визначень і опишемо роботу генетичного алгоритму детальніше.

Щоб говорити про генетичне спадкування, потрібно наділити наші особини хромосомами. У генетичному алгоритмі хромосома – це деякий числовий вектор, що відповідає параметру, який підбирається, а набір хромосом певної особини визначає розв'язок задачі. Які саме вектори варто розглядати в конкретній задачі, вирішує сам користувач. Кожна з позицій вектора хромосоми називається ген.

Простий генетичний алгоритм випадковим чином генерує початкову популяцію структур. Робота генетичного алгоритму являє собою ітераційний процес, що продовжується доти, поки не буде досягнено заданої кількості поколінь або будь-якого іншого критерію зупинки. У кожному поколінні генетичного алгоритму реалізується відбір пропорційної пристосованості, одноточковий кросинговер і мутація. Спочатку пропорційний відбір призначає кожній структурі імовірність $p_s(i)$, рівну відношенню її пристосованості до сумарної пристосованості популяції:

$$p_s(i) = \frac{f(i)}{\sum_{i=1}^{n} f(i)}$$

Потім відбувається відбір (із заміщенням) усіх n особин для подальшого генетичного опрацювання, відповідно до величини $p_s(i)$.

При такому відборі члени популяції з більш високою пристосованістю з більшою імовірністю будуть частіше вибиратися, ніж особини з низькою пристосованістю. Після відбору, n обраних особин випадковим чином розбиваються на n/2 пари. Для кожної пари з імовірністю p_s може застосовуватися кросинговер. Відповідно до імовірності $1-p_s$ кросинговер не відбувається і незмінені особини переходять на стадію мутації. Якщо кросинговер відбувається, отримані нащадки заміняють собою батьків і переходять до мутації.

Визначимо тепер поняття, що відповідають мутації і кросинговеру в генетичному алгоритмі.

Мутація – це перетворення хромосоми, що випадково змінює одну чи декілька її позицій (генів). Найбільш розповсюджений вид мутацій – випадкова зміна тільки одного з генів хромосоми.

Кросинговер (у літературі з генетичних алгоритмів також вживається назва кросинговер або схрещування) – це операція, при якій із двох хромосом породжується одна чи декілька нових хромосом. Одноточковий кросинговер працює в такий спосіб. Спочатку, випадковим чином вибирається одна з точок розриву. (Точка розриву – ділянка між сусідніми бітами в рядку.) Обидві батьківські структури розриваються на два сегменти по цій точці. Потім, відповідні сегменти різних батьків склеюються і утворюються два генотипи нащадків.

Наприклад, припустимо, сегмент одного з батьків складається з 10 нулів, а іншого – з 10 одиниць. Нехай з 9 можливих точок розриву обрана точка 3. Батьки і їхні нащадки показані нижче.

Кросинговер

Батько 1 0000000000 000~0000000 → 111~0000000 1110000000 Нащадок 1

Батько 2 1111111111 111~1111111 → 000~1111111 0001111111 Нащадок 2

Після того як закінчується стадія кросинговеру, виконуються оператори мутації. У кожному рядку, що піддається мутації, кожен біт з імовірністю p змінюється на протилежний.

Популяція, отримана після мутації, записується поверх старої і цим цикл одного покоління завершується. Наступні покоління опрацьовуються подібним чином: відбір, кросинговер і мутація.

Зараз дослідники генів пропонують багато інших операторів відбору, кросинговеру і мутації. От лише найбільш розповсюджені з них.

Елітні методи відбору гарантують, що при відбиранні обов'язково будуть виживати кращий чи кращі члени популяції сукупності. Найбільш поширена процедура обов'язкового збереження тільки одної кращої особини, якщо вона не пройшла, як інші, через процес відбору, кросинговеру і мутації. Елітизм може бути впроваджений практично в будь-який стандартний метод відбору.

Двоточковий кросинговер і рівномірний кросинговер – цілком гідні альтернативи одноточковому оператору. У двоточковому кросинговері вибираються дві точки розриву, і батьківські хромосоми обмінюються сегментом, що міститься між двома цими точками. У рівномірному кросинговері кожен біт першого батька успадковується першим нащадком із заданою імовірністю; у протилежному випадку цей біт передається другому нащадку. І навпаки.

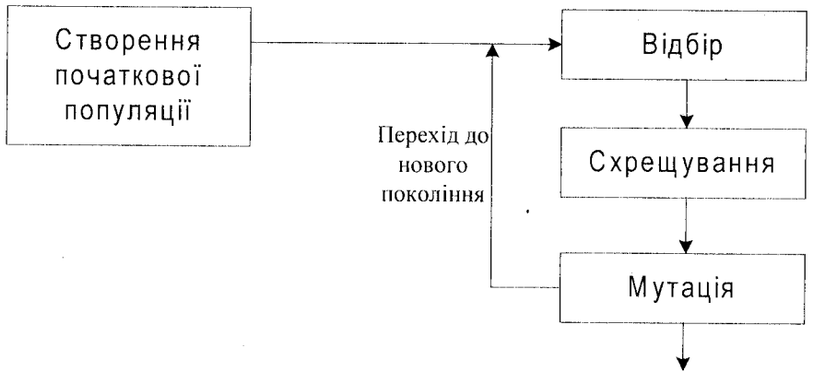


Рис. 8.5. Блок-схема генетичного алгоритму

Блок-схема генетичного алгоритму зображена на рис. 8.5. Спочатку генерується початкова популяція особин (індивідуумів), тобто деякий набір розв'язків задачі. Як правило, це робиться випадковим чином. Потім ми повинні змоделювати розмноження всередині цієї популяції. Для цього випадково відбираються декілька пар індивідуумів, відбувається схрещування між хромосомами в кожній парі, а отримані нові хромосоми втілюються в популяцію нового покоління. У генетичному алгоритмі зберігається основний принцип природного відбору – чим пристосованіший індивідуум (чим більше відповідне йому значення цільової функції), тим з більшою імовірністю він буде брати участь у схрещуванні. Тепер моделюються мутації – у декількох випадково обраних особин нового покоління змінюються деякі гени. Потім стара популяція частково або цілком знищується і ми переходимо до розгляду наступного покоління. Популяція наступного покоління в більшості реалізацій генетичних алгоритмів містить стільки ж особин, скільки початкова, але через відбір пристосованість у ній у середньому вища. Тепер описані процеси відбору, схрещування й мутації повторюються вже для цієї популяції і т. д.

У кожному наступному поколінні ми будемо спостерігати виникнення зовсім нових розв'язків нашої задачі. Серед них будуть як погані, так і хороші, але завдяки відбору кількість прийнятних розв'язків буде зростати. Відзначимо, що в природі не буває абсолютних гарантій, і найпристосованіший тигр може загинути від рушничного пострілу, не залишивши нащадків. Імітуючи еволюцію на комп'ютері, ми можемо уникати подібних небажаних подій і завжди зберігати життя кращому з індивідуумів поточного покоління – така методика називається «стратегією елітизму».

8.2.5. Застосовування генетичних алгоритмів

Генетичні алгоритми в різних формах застосовуються до вирішення багатьох наукових і технічних проблем. Генетичні алгоритми використовуються при створенні інших обчислювальних структур, наприклад, автоматів або мереж сортування. У машинному навчанні вони використовуються при проектуванні нейронних мереж або керуванні роботами. Вони також застосовуються при моделюванні розвитку в різних предметних областях, включаючи біологічні (екологія, імунологія і популяційна генетика) та соціальні системи (економіка і політичні системи).

Проте, можливо найпопулярніше застосування генетичних алгоритмів – оптимізація багатопараметричних функцій. Багато реальних задач можуть бути сформульовані як

пошук оптимального значення, де значення – складна функція, що залежить від певних вхідних параметрів. У деяких випадках потрібно знайти ті значення параметрів, при яких досягається найкраще точне значення функції. В інших випадках, точний оптимум не потрібний – розв'язком може вважатися будь-яке значення, краще за певну задану величину. У цьому випадку, генетичні алгоритми – часто найприйнятніший метод для пошуку «прийнятних» значень. Сила генетичного алгоритму полягає в його здатності маніпулювати одночасно багатьма параметрами, що використовується в сотнях прикладних програм, включаючи проектування літаків, налаштування параметрів алгоритмів і пошуку стійких станів систем нелінійних диференціальних рівнянь.

Отже, генетичні алгоритми є ефективною процедурою пошуку, що конкурує з іншими процедурами. Ефективність генетичних алгоритмів дуже залежить від таких деталей, як метод кодування рішень, операторів, налаштування параметрів, окремих критеріїв успіху. Теоретична робота, відбита в літературі, присвяченій генетичним алгоритмам, не дає підстав говорити про вироблення певних строгих механізмів для чітких передбачень.

8.3. Інтелектуальний аналіз даних. Побудова дерева рішень

8.3.1. Постановка задачі класифікації даних

Метод класифікації даних складається з двох основних етапів [256]. На першому етапі здійснюється аналіз даних, що зберігаються в базі даних, і будується модель, яка кожному об'єкту ставить у відповідність мітку класу, до якого цей об'єкт належить. Сукупність об'єктів, за якими створюється класифікаційна модель, називається **навчальним набором**.

Побудована модель відображається деяким способом подання знань, переважно у формі дерев рішень або класифікаційних правил. Класифікатор може мати як детермінований характер, коли кожному об'єкту ставиться у відповідність точно один клас, так і недетермінований, коли об'єкт відноситься до кількох класів з певним розподілом деяких мір невизначеності (наприклад, підтримки, ступеня довіри тощо).

На другому етапі створена класифікаційна модель використовується для класифікації нових об'єктів. Сукупність цих об'єктів називається **тестовим набором даних**.

Розглянемо формальну постановку задачі класифікації даних. Нехай маємо навчальний набір даних L, який складається з кортежів $t \in L$, а також множину класів C, яка складається з міток класів $c_i \in C$, $i = 1,...,m$. Для кожного кортежа з навчального набору відомо, до якого класу він належить, тобто, кортежі мають структуру $<A, c>$, де A – атрибути, що описують об'єкт, c – мітка класу. Крім того, задано тестовий набір даних T, для кортежів якого невідомо, до якого класу вони належать. Потрібно на основі навчального набору побудувати відображення $K:L \to C$ у формі знань, яке кожному кортежу t навчального набору L ставить у відповідність мітку c з множини класів C; кожному кортежу t тестового набору T, використовуючи відображення K, ставить у відповідність мітку c з множини класів C.

8.3.2. Метод класифікації на основі індукції дерев рішень

Основна вимога до математичного апарату виявлення закономірностей в даних (окрім, звичайно, вимоги ефективності) полягає в інтерпретації результатів. Правила,

що відображають знайдені закономірності, повинні формулюватися на простій і зрозумілій людині мові логічних висловлювань. Наприклад, **ЯКЩО {(подія 1) і (подія 2) і... і (подія N)} ТО ...** Іншими словами, це повинні бути логічні правила.

Дерева рішень (decision trees) є найпоширенішим зараз підходом до виявлення і зображення логічних закономірностей в даних. Знані представники цього підходу – системи CHAID (chisquare automatic interaction detection), CART (classification and regression trees) і ID3 (Interactive Dichotomizer – інтерактивний дихотомайзер). Розглянемо детальніше процедуру побудови дерев рішень на прикладі системи ID3.

В основі системи ID3 лежить алгоритм CLS. Цей алгоритм циклічно розбиває навчальні приклади на класи відповідно до змінної, що має найбільшу класифікаційну силу. Кожна підмножина прикладів (об'єктів), що виділяється такою змінною, знову розбивається на класи з використанням наступної змінної з найбільшою класифікаційною здатністю і т.д. Розбиття закінчується, коли в підмножині опиняються об'єкти лише одного класу. У ході процесу утворюється дерево рішень. Шляхи руху по цьому дереву з верхнього рівня на найнижчі визначають логічні правила у вигляді послідовностей кон'юнкцій.

Побудова класифікаційної моделі здійснюється у такій послідовності [202]:

Вибірка даних

На першому кроці здійснюється вибірка даних із бази даних, які становитимуть навчальний набір. При цьому особа, яка приймає рішення чи експерт аналізує предметну область та формує множину параметрів, які описують об'єкт дослідження і можуть впливати на вихідне рішення. У результаті вибірки створюється відношення, яке називається *таблицею рішень*, зі структурою типу $R(A,D)$, де A – набір атрибутів-факторів, значення яких впливають на формування рішення, D – набір атрибутів рішення.

Підготування даних

На цьому кроці здійснюється поповнення даних з невизначеностями, опрацювання аномальних даних, дискретизація числових величин та виділення із загального набору параметрів підмножини суттєвих факторів.

Після вибірки даних частина кортежів таблиці рішень може містити невизначені або пропущені значення. Крім того, деякі значення певного атрибута можуть різко відрізнятися від решти значень. Для коректного виконання алгоритму побудови дерева рішень усі невизначеності та аномалії потрібно або усунути з таблиці рішень, або довизначити і згладити їх на основі статистичного аналізу значень атрибута [2].

Для зменшення кількості можливих значень числових атрибутів та побудови дерева рішень на загальнішому рівні агрегації даних здійснюється дискретизація числових величин [68]. При цьому числовий домен атрибута розбивається на сукупність інтервалів, і кожне значення атрибута у таблиці рішень заміняється на відповідний інтервал.

Одним із критеріїв якості отриманих знань є їх цілковите розуміння особою, що приймає рішення. Тому важливо, щоб дерево рішень мало достатньо просту структуру. Одним із способів досягнення цього є виділення серед атрибутів-характеристик тих, що найбільше впливають на остаточне рішення. Цей крок здійснюється за допомогою методів факторного аналізу.

8.3.3. Побудова дерева рішень та набору класифікаційних правил

На цьому кроці виконується алгоритм побудови дерева рішень. Алгоритм виконується у наступній послідовності.

1. Створюється початковий вузол дерева.
2. Якщо всі кортежі навчального набору належать до одного класу, то вузол визначається як листковий, і йому присвоюється мітка класу.
3. Інакше, алгоритм використовує інформаційний приріст для визначення атрибута, на основі якого добудовується дерево.
4. Для кожного значення вибраного атрибута формується гілка дерева, і кортежі навчального набору, що залишилися, розділяються на відповідні підмножини.
5. Алгоритм виконується рекурсивно для побудови піддерева на основі набору атрибутів, що залишилися; при цьому атрибути, які вже задіювалися при побудові дерева, не розглядаються.
6. Алгоритм зупиняється при настанні однієї з таких умов:
   - всі кортежі навчального набору, що залишилися, належать до одного класу;
   - для побудови дерева рішень використані всі атрибути;
   - для побудови дерева використані усі кортежі навчального набору.

На основі побудованого дерева рішень формується набір класифікаційних правил типу "ЯКЩО <атрибут>=<значення> ТО <прогнозована подія>.

Алгоритм CLS здатний приводити до якісних рішень задачі пошуку логічних закономірностей тільки у разі незалежних ознак. У протилежному випадку він нерідко скеровує хід логічного виведення помилковим шляхом і створює лише ілюзію правильного міркування.

8.3.4. Виявлення логічних закономірностей в даних

Розглянемо рис. 8.6. На ньому схематично зображено обличчя людей. Ці обличчя з якихось причин, можливо важливих, розділені на два класи. Ставиться задача знайти закономірності здійсненого поділу.

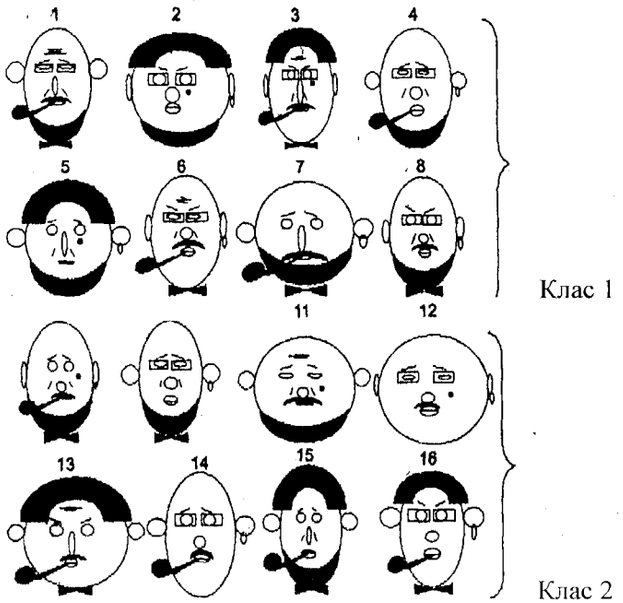


Рис. 8.6. Зображення облич людей.

Спробуйте візуально визначити, чим лиця людей різних класів відрізняються одне від одного і що об'єднує лиця одного класу. Відразу помітимо – рішення існує. Але ваш візуальний аналіз, швидше за все, не дасть відповіді на поставлене запитання. Звичайний людський розум не в змозі розв'язати навіть таку, на перший погляд просту, задачу виявлення прихованих закономірностей. Тут необхідне застосування комп'ютерних методів аналізу даних.

Перш за все виділимо ознаки, що характеризують зображені лиця. Це такі характеристики:

x_1 (голова) – кругла – 1, овальна – 0;
x_2 (вуха) – відстовбурені – 1, притиснуті – 0;
x_3 (ніс) – круглий – 1, довгий – 0;
x_4 (очі) – круглі – 1, вузькі – 0;
x_5 (чоло) – із зморшками – 1, без зморшок – 0;
x_6 (складка) – носогубна складка є – 1, носогубної складки немає – 0;
x_7 (губи) – товсті – 1, тонкі – 0;
x_8 (волосся) – є – 1, немає – 0;
x_9 (вуса) – є – 1, немає – 0;
x_{10} (борода) – є – 1, немає – 0;
x_{11} (окуляри) – є – 1, немає – 0;
x_{12} (родимка) – родимка на щоці є – 1, родимки на щоці немає – 0;
x_{13} (метелик) – є – 1, немає – 0;
x_{14} (брови) – підняті догори – 1, опущені донизу – 0;
x_{15} (сережка) – є – 1, немає – 0;
x_{16} (люлька) – палильна люлька є – 1, немає – 0.

Початкова матриця даних, що відповідає зображеним обличчям, подана в табл. 8.1. Рядки відповідають об'єктам (N=16), стовпці – виділеним бінарним ознакам (p=16). Об'єкти з номерами 1-8 відносяться до класу 1, а з номерами 9-16 – до класу 2.

Табл. 8.1.Матриця початкових даних

№ п/п	Голова	Вуха	Ніс	Очі	Чоло	Складка	Губи	Волосся	Вуса	Борода	Окуляри	Родимка	Метелик	Брови	Сережка	Люлька	Class
0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	
1	0	1	0	0	1	1	0	0	1	1	1	0	1	1	0	1	1
2	1	0	1	1	0	0	1	1	0	1	1	1	0	0	1	0	1
3	0	0	0	1	1	1	0	1	1	0	1	1	1	0	0	1	1
4	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1	1
5	1	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1

продовження табл. 8.1.

№ п/п	Голова	Вуха	Ніс	Очі	Чоло	Складка	Губи	Волосся	Вуса	Борода	Окуляри	Родимка	Метелик	Брови	Сережка	Люлька	Class
6	0	0	1	0	1	1	1	0	1	0	1	0	1	0	1	1	1
7	1	1	0	1	0	0	0	0	1	1	0	0	1	1	1	1	1
8	0	0	1	1	0	1	1	0	1	1	1	0	1	0	1	0	1
9	0	0	1	1	0	1	0	0	1	1	0	1	1	1	0	1	2
10	0	1	1	0	0	1	1	0	0	1	1	0	1	1	1	0	2
11	1	1	1	0	1	1	0	0	1	1	0	1	0	1	0	0	2
12	1	0	1	0	1	0	1	0	1	0	1	1	0	1	1	0	2
13	1	1	0	1	1	0	1	1	1	0	0	0	1	0	0	1	2
14	0	1	1	1	0	0	1	0	1	0	1	0	0	1	1	1	2
15	0	1	0	1	0	1	1	1	0	1	0	0	1	1	0	1	2
16	0	1	1	1	0	0	1	1	0	0	1	0	1	0	1	1	2

Класифікація облич у розглянутому прикладі може бути здійснена за допомогою чотирьох логічних правил:

1) ЯКЩО {(голова овальна) і (є носогубна складка) і (є окуляри) і (є люлька)} **ТО** (Клас 1);

2) ЯКЩО {(очі круглі) і (чоло без зморшок) і (є борода) і (є сережка)} **ТО** (Клас 1);

3) ЯКЩО {(ніс круглий) і (волосся немає) і (є вуса) і (брови підняті догори)} **ТО** (Клас 2);

4) ЯКЩО {(відстовбурнені вуха) і (товсті губи) і (немає родимки на щоці) і (є метелик)} **ТО** (Клас 2).

Математичний запис цих правил виглядає таким чином:

$[(x1=0)\wedge(x6=1)\wedge(x11=1)\wedge(x16=1)] \vee [(x4=1)\wedge(x5=0)\wedge(x10=1)\wedge(x15=1)] \Rightarrow \omega_1$,

$[(x3=1)\wedge(x8=1)\wedge(x9=1)\wedge(x14=1)] \vee [(x2=1)\wedge(x7=1)\wedge(x12=0)\wedge(x13=1)] \Rightarrow \omega_2$.

Тут значки $\wedge$ – кон'юнкція (і), $\vee$ – диз'юнкція (або), $\Rightarrow$ – імплікація (якщо, то).

Під правило 1 підпадають перша, третя, четверта і шоста особи з першого класу; під правило 2 – друга, п'ята, сьома і восьма особи з першого класу; під правило 3 – дев'ята, одинадцята, дванадцята і чотирнадцята особи з другого класу; під правило 4 – десята, тринадцята, п'ятнадцята і шістнадцята особи з другого класу.

На першому кроці алгоритму визначається ознака з найбільшою дискримінаційною силою. У нашому випадку однакову і максимальну силу мають відразу 7 ознак – *x2, x3, x6, x10, x11, x14* і *x15* (табл. 8.2). Тому тут ми ухвалюємо вольове рішення і призначаємо першою ознакою, наприклад, *x6*.

Табл. 8.2. Відношення одиниць (1) в різних класах об'єктів для різних ознак.

Ознаки	x_1	$\mathbf{x_2}$	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	$\mathbf{x_{14}}$	$\mathbf{x_{15}}$	x_{16}
Клас1/Клас2	3/3	**4/6**	4/6	5/5	3/3	6/4	4/6	3/3	5/5	6/4	6/4	3/3	5/5	**4/6**	**4/6**	5/5

Від першої ознаки відходять дві гілки. Перша для значення x_6=0, а друга x_6=1. У табл. 8.3 і табл. 8.4 містяться дані, відповідні цим гілкам.

Табл. 8.3. Таблиця даних, що відповідає гілці x_6=0.

Ознаки																
Об'єкт	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	$\mathbf{x_{10}}$	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}
2	1	0	1	1	0	0	1	1	0	**1**	1	1	0	0	1	0
7	1	1	0	1	0	0	0	0	1	**1**	0	0	1	1	1	1
12	1	0	1	0	1	0	1	0	1	**0**	1	1	0	1	1	0
13	1	1	0	1	1	0	1	1	1	**0**	0	0	1	0	0	1
14	0	1	1	1	0	0	1	0	1	**0**	1	0	0	1	1	1
16	0	1	1	1	0	0	1	1	0	**0**	1	0	1	0	1	1
Клас1/Клас2	2/2	1/3	1/3	2/3	2/2	2/4	1/4	1/2	1/3	**2/0**	1/3	1/1	1/2	1/2	2/3	1/3

Для гілки $\boldsymbol{x}_6$ = 0 остаточне рішення дає ознаку $\boldsymbol{x}_{10}$. Вона приймає значення 1 на об'єктах 2 і 7 з першого класу і значення 0 на об'єктах 12,13,14 і 16 з другого класу.

Табл. 8.4. Таблиця даних, відповідна гілці x_6=1.

Ознаки																
Об'єкт	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	$\mathbf{x_{11}}$	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}
1	0	1	0	0	1	1	0	0	1	1	**1**	0	1	1	0	1
3	0	0	0	1	1	1	0	1	1	0	**1**	1	1	0	0	1
4	0	1	1	0	0	1	1	0	0	1	**1**	0	0	1	1	1
5	1	1	0	1	0	1	0	1	0	1	**0**	1	0	1	1	0
6	0	0	1	0	1	1	1	0	1	0	**1**	0	1	0	1	1
8	0	0	1	1	0	1	1	0	1	1	**1**	0	1	0	1	0
9	0	0	1	1	0	1	0	0	1	1	**0**	1	1	1	0	1
10	0	1	1	0	0	1	1	0	0	1	**1**	0	1	1	1	0
11	1	1	1	0	1	1	0	0	1	1	**0**	1	0	1	0	0
15	0	1	0	1	0	1	1	1	0	1	**0**	0	1	1	0	1
Клас1/Клас2	2/2	1/3	1/3	2/3	2/2	2/4	1/4	1/2	1/3	**2/0**	**1/3**	1/1	1/2	1/2	2/3	1/3

Гілка $\boldsymbol{x}_6$=1 влаштована складніше. На цій гілці найбільше дискримінаційну силу має ознака $\boldsymbol{x}_{11}$ (табл. 8.4). Вона має значення 0 в об'єкту 5 з першого класу і об'єктів 9, 11, 15 з другого класу; і значення 1 в об'єктів 1, 3, 4, 6 з першого класу і об'єкту 10 з другого класу. Отже, потрібне додаткове розгалуження, яке здійснюється за допомогою ознак x_{15}, x_{16} і x_2.

Дерево логічного виведення, що виросло з ознаки x_6 (носогубна складка), має 6 результатів. Тільки два з цих результатів включають по чотири об'єкти (повнота 4/8). Один результат групує три об'єкти свого класу (повнота 3/8), один результат – два об'єкти (повнота 2/8) і три результати включають по одному об'єкту (повнота 1/8). Природно, не

можна вважати логічною закономірністю шлях по дереву, для якого результат охоплює таке мале відносне число об'єктів одного класу (має малу повноту). Як бачимо, рішення, що дається алгоритмом CLS, далеке від того, що вимагається.

Якщо спробувати побудувати дерево рішень з будь-якої іншої ознаки, наприклад, x_2, x_3, x_{10}, x_{11}, x_{14} або x_{15}, то результат також буде далекий від оптимального.

Отримане дерево рішень наведено на рис. 8.7.

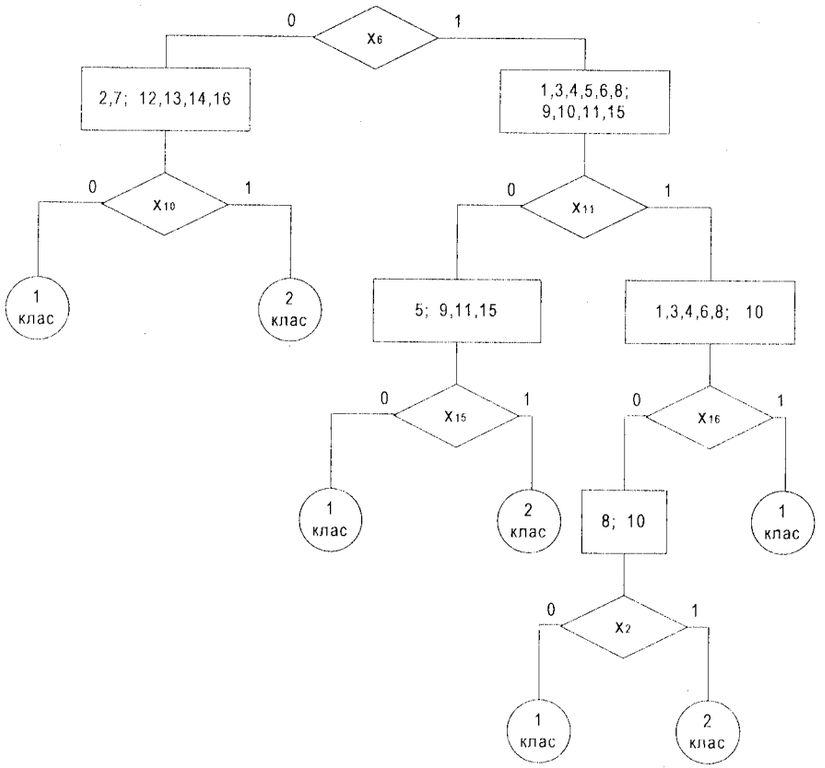


Рис. 8.7. Дерево рішень.

Отримані правила на основі побудованого дерева рішень:

Правило1. ЯКЩО (x6=0) І (x10=0) ТО клас 1.
Правило2. ЯКЩО (x6=0) І (x10=1) ТО клас 2.
Правило3. ЯКЩО (x6=1) І (x11=0) І (x15=0) ТО клас 1.
Правило4. ЯКЩО (x6=1) І (x11=0) І (x15=1) ТО клас 2.
Правило5. ЯКЩО (x6=1) І (x11=1) І (x16=1) ТО клас 1.
Правило6. ЯКЩО (x6=1) І (x11=1) І (x16=0) І (x2=0) ТО клас 1.
Правило7. ЯКЩО (x6=1) І (x11=1) І (x16=0) І (x2=1) ТО клас 2.

8.4. Штучні нейронні мережі

Що таке штучні нейронні мережі? Що вони можуть робити? Як вони працюють? Як можна використовувати їх? Ці та багато інших подібних запитань задають спеціалісти широкого кола галузей. Дати вичерпні відповіді є досить складною справою.

Після двадцяти років, коли дослідження в цій галузі практично не здійснювалися, останнім часом інтерес до штучних нейронних мереж різко зріс. Спеціалісти таких різних напрямків як інженерія, філософія, фізіологія і психологія заінтриговані потенційними можливостями цієї технології і шукають її практичне застосування в своїх галузях.

Цей відроджений інтерес підсилив розвиток теорії та практичні успіхи. Раптом з'явилася можливість застосовувати обчислення у сферах, які раніше вважалися доступними тільки для людського розуму; створювати машини, які здатні навчатися і запам'ятовувати шляхом, що надзвичайно нагадує розумовий процес людини; дати нове і важливе означення такому широковживаному терміну як штучний інтелект.

Штучні нейронні мережі виконують більшість поширених обчислень.

Вони обіцяють створення пристроїв, що виконують функції, які раніше був здатен виконати лише мозок людини. Нудна, багато раз повторювана чи небезпечна робота може бути перекладена на плечі машин.

Теоретичне підґрунтя штучних нейронних мереж розвивається дуже стрімко, але воно ще далеке від здійснення багатьох оптимістичних прогнозів. Типова ситуація, яка склалася історично, показує, що теорія розвивається швидше, аніж песимістичні прогнози скептиків, і повільніше, ніж сподіваються оптимісти. На сьогодні хвиля зацікавленості породила тисячі досліджень у цій галузі. Штучні нейронні мережі були розроблені у великому числі конфігурацій. Але, незважаючи на здавалося б велику різноманітність, всі нейронні парадигми ґрунтуються на спільній ідеї, що бере за основу принципи роботи нейронів головного мозку людини.

8.4.1. Біологічний прототип

Штучні нейронні мережі є біологічно запозиченими. Це означає, що дослідники, як правило, думають про організацію і діяльність мозку людини, коли розробляють конфігурації нейронних мереж та алгоритми їх роботи. Але знання про роботу головного мозку є дуже і дуже обмеженими. Отже, розробники мереж змушені самі розробляти структури, які мають під собою певні біологічні знання і задовольняють виконання найважливіших функцій. У багатьох випадках біологічну подібність треба відкинути, оскільки людський мозок повний метафор, а результати роботи нейронних мереж подекуди є органічно несумісними або дуже неправдоподібними щодо анатомії та функціонування мозку.

Але, незважаючи на розбіжності в біології, штучні нейронні мережі й надалі нагадують про людський мозок. Їх функції часто нагадують людське пізнання. Отож, тяжко позбутися подібної аналогії. На жаль, такі порівняння є не досить доцільними, оскільки створюють ілюзорне очікування нереальних результатів. Між тим, не погано було б ознайомитися з будовою та основними принципами роботи нервової системи, яку наслідують штучні нейронні мережі.

Людська нервова система складається із частинок, що носять назву нейронів. Вони мають надзвичайну комплексність. Приблизно 10^{11} нейронів беруть участь у 10^{15} міжнейронних з'єднань. Кожен нейрон має багато характеристик, подібних до інших частинок тіла, але тільки він має унікальні можливості отримувати, опрацьовувати і передавати електрохімічні сигнали через нейронні шляхи, що утворюють комунікаційну систему мозку.

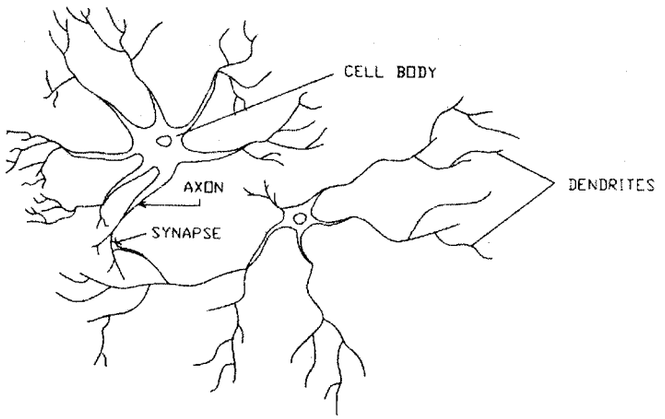


Рис. 8.8. Біологічний нейрон.

На рис. 8.8 показана структура пари типових біологічних нейронів. Нейрони з'єднуються один з одним за допомогою дендритів у точках, через які передаються сигнали і які називаються синапсами. На приймальній стороні синапсів є входи, що керують клітиною. Тут вхідні сигнали сумуються, причому деякі з входів служать для збудження клітини, а інші перешкоджають цьому. Всі збудження акумулюються і при досягненні їх певного порогового значення клітина посилає сигнали вздовж аксонів до інших нейронів. Звичайно ж цей базовий функціональний вихід є набагато складнішим і різностороннішим, але, незважаючи на це, більшість штучних нейрон-них мереж моделюють лише ці найпростіші характеристики.

8.4.2. Штучний нейрон

Штучний нейрон був розроблений, наслідуючи основні характеристики біологічного нейрона. За суттю це набір входів, кожен з яких є відповідним виходом іншого нейрона. Кожен вхід перемножується на відповідний ваговий коефіцієнт, аналогічний до синаптичної сили, і далі всі вагові входи підсумовуються до досягнення активізаційного рівня нейрона.

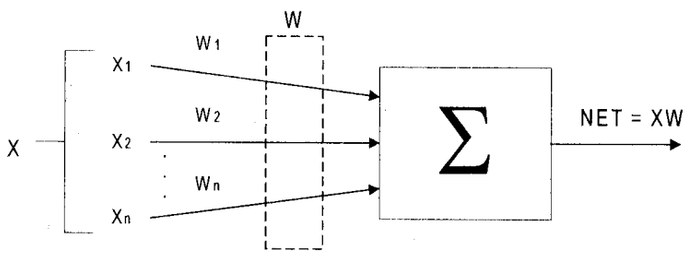


Рис. 8.9. Модель сумуючого блоку штучного нейрона.

На рис. 8.9 показана модель, що реалізує цю ідею. Незважаючи на різноманітність мережних парадигм, буквально всі вони в більшій чи меншій мірі базуються на цій конфігурації. Входи нейрона позначаються через x1, x2, x3, ... , xn. Всі ці входи складають вектор X, відповідно до сигналів в синапсах біологічних нейронів. Кожен сигнал

перемножується на відповідний коефіцієнт w1, w2, w3, ..., wn, після чого надходить в сумуючий блок. Кожен ваговий коефіцієнт відповідає синаптичній силі з'єднання в біологічній моделі. Набір вагових коефіцієнтів позначається як вектор W. Блок сумування, аналогічно біологічній клітині, виконує алгебраїчне сумування, продукуючи вихід, який ми назвемо NET. Це все може бути описано компактніше у векторній формі:

$$NET = X\,W.$$

Сигнал NET далі опрацьовується активізаційною функцією F, виробляючи вихідний сигнал OUT.

Це може бути проста лінійна функція:

$$OUT = K\,(NET),$$

де K є стала порогова функція,

$$OUT = 1, \text{ якщо } NET > T;$$

$$OUT = 0 \text{ в іншому випадку,}$$

де T – це константа порогового рівня чи функція, що більш точно відображає нелінійні характеристики передавання в біологічному нейроні і дозволяє існування більш загальних мережних функцій.

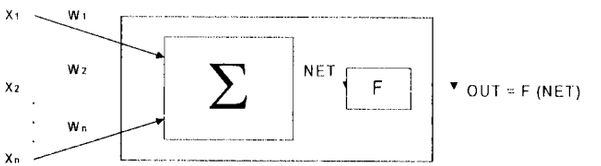


Рис. 8.10. Модель штучного нейрона.

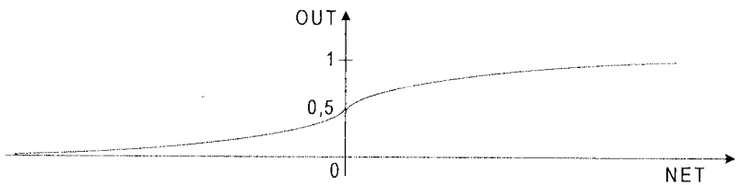


Рис. 8.11. Сигмоїдна логічна функція.

На рис. 8.10 відображено блок, позначений як F, який приймає вихід NET і продукує з нього сигнал OUT. Якщо блок F стискає межі значень NET таким чином, що OUT ніколи не перевищує деяких мінімальних допустимих значень, незважаючи при цьому на значення NET, то F називають функцією обмеження. Як обмежувальну функцію досить часто вибирають логічну функцію чи «sigmoid» (мається на увазі S-подібний вигляд), як показано на рис. 8.11. Ця функція подається математично у вигляді

$$F(x) = \frac{1}{1+e^{-x}}.$$

Отже, тоді будемо мати наступний вираз:

$$OUT = \frac{1}{1+e^{-NET}} = F(NET).$$

Як і в аналогових електронних системах, ми можемо подати активізаційну функцію для штучного нейрона як нелінійний приріст. Цей приріст визначається відношенням зміни величини OUT до малої зміни в NET. Отже, приріст – це нахил кривої певного активізаційного рівня. При великому негативному значенні збудження приросту майже нема (крива практично горизонтальна). При досягненні нульового значення приріст різко збільшується і практично затухає при набутті функцією збудження великих позитивних значень. Кроссберг (1973) встановив, що така характеристика нелінійного приросту вирішує поставлену ним дилему шумового насичення; тобто, чи може одна і та ж мережа виявляти одночасно малі і великі сигнали. Іншою загальновживаною активізаційною функцією є гіперболічний тангенс (рис. 8.12.). Вона має вигляд, подібний до логічної функції та часто використовується біологами як математична модель нервової клітини. Стосовно штучної нейронної мережі активізаційна функція має наступний вигляд:

$$OUT = \tanh(x) = \begin{cases} 1 - \dfrac{1}{1-x}, x \geq 0 \\ -1 + \dfrac{1}{1+x}, x < 0 \end{cases}.$$

Подібно до логічної функції, функція гіперболічного танґенса має S – подібну форму, але симетрична відносно осі абсцис.

Ця найпростіша модель штучного нейрона не бере до уваги багато з характеристик свого біологічного двійника. Наприклад, вона не здійснює часових затримок, що придає динаміку системі; отже, входи моментально продукують результатний вихід. Але суттєвішим недоліком є те, що ця модель не включає в себе механізми синхронізації або частотозадавану функцію біологічного нейрона, характеристики яких дослідники вважають найважливішими.

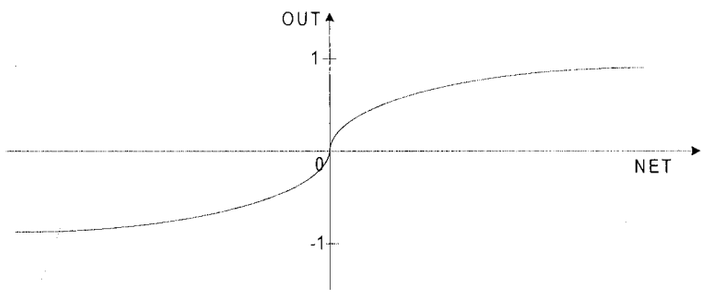


Рис. 8.12. Функція гіперболічного танґенса.

Але, всупереч всім обмеженням, нейронна мережа, що має наведені вище властивості, дуже нагадує біологічну систему.

8.4.3. Одноріневі штучні нейронні мережі

Хоча ізольований нейрон і може здійснювати найпростіші обчислювальні функції, але сила нейронних обчислень лежить у їх об'єднані в єдину мережу. Найпростіша мережа – це група нейронів, об'єднаних в одному шарі, як показано на рис. 8.13.

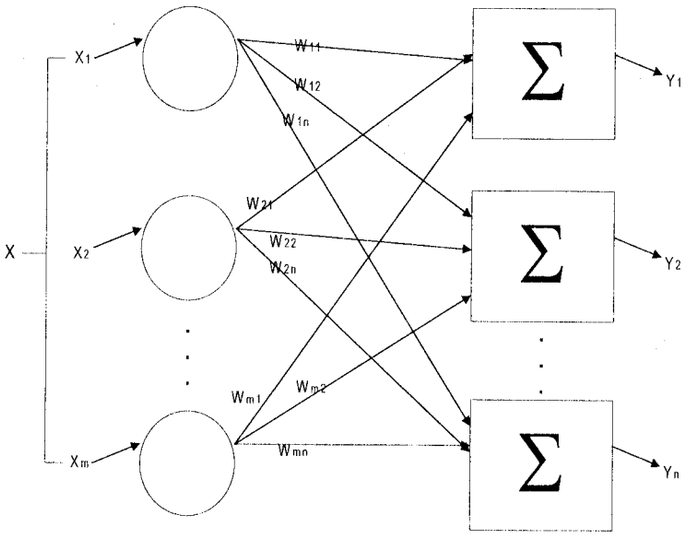


Рис. 8.13. Модель найпростішої нейронної мережі.

Як видно з рис. 8.13 набір входів X подається на вхід кожного нейрона з відповідним ваговим коефіцієнтом. Ранні штучні нейронні мережі були не набагато повнішими, ніж ця. У цьому випадку кожен нейрон просто виводить вагову суму входів мережі. Для зручності об'єднаємо вагові коефіцієнти у матрицю W. Розмірність цієї матриці буде m x n , де m – це число входів і n – кількість нейронів. Для прикладу, ваговий коефіцієнт від третього входу до другого нейрона буде позначений як $W_{3,2}$. Таким чином можна показати обчислення вихідних даних для такої мережі, як просте матричне перемноження:

$$N = XW$$

$$N = XW \ .$$

Така форма нейронної мережі є найпростішою і не володіє достатньою обчислювальною здатністю. Але разом з тим вона є досить наочною і зрозумілою для знайомства з принципом роботи штучної нейронної мережі загалом. Далі розглянемо модель нейронної мережі у вигляді перцептрона як такого, що став основою для побудови перших прототипів штучних нейронів.

8.4.4. Перцептрони

Вчення про штучні нейронні мережі вперше виникло у 40-х роках XX століття. Дослідники намагалися продублювати функції людського мозку, розробивши простіші апаратні (а пізніше і програмні) моделі біологічних нейронів і систему зв'язку між ними. Поступово вдосконалюючи знання про людську нервову систему, нейрофізіологи показали, що ці методи були лише примітивною спробою імітації.

McCulloch і Pitts (1943) опублікували перше систематичне вчення про штучні нейронні мережі. Пізніше (1947) вони створили мережеві парадигми для розпізнавання образів, незважаючи на їх переміщення і обертання в просторі. У більшості їх робіт використовується проста нейронна мережа, яку називають перцептроном. Її показано на рис. 8.14.

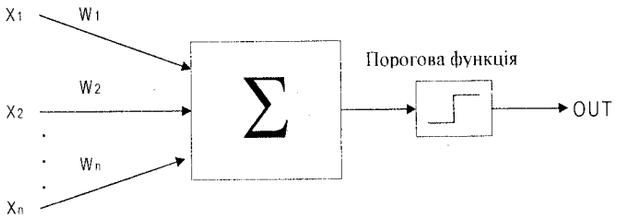


Рис. 8.14. Схема перцептрона.

Блок суматора перемножує кожен вхід X на ваговий коефіцієнт W і після цього сумує їх. Якщо ця сума є більшою, ніж певний визначений пороговий рівень, то на виході з'являється одиниця, в інакшому випадку – нуль. Такі системи (і багато подібних варіантів) було названо перцептронами. Як правило, вони утворюються одиничним шаром нейронів, зв'язаних ваговими коефіцієнтами на входах, хоча подібну назву мають і багато інших мереж різноманітної архітектури. Набір одиничних перцептронів утворює нейронну мережу, яка має назву багато-вихідного перцептрона, модель якого показана нижче.

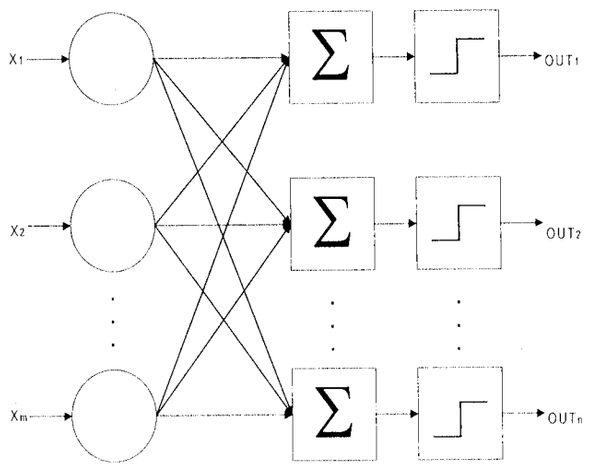


Рис. 8.15. Багатовихідний перцептрон.

У 60-х роках минулого століття перцептрони викликали велику зацікавленість і оптимізм. У 1962 р. Rosenblatt відкрив теорему про навчання перцептронів. Widrow (Widrow 1961, 1963; Widrow and Angell 1962; Widrow and Hoff 1960) створив значну кількість переконливих перцептроноподібних систем, потенціал яких відразу ж почали використовувати багато дослідників. Але початкова ейфорія швидко поступилася місцем розчаруванню, як тільки перцептрони зазнали невдачі на найпростіших навчальних задачах.

Minsky (Minsky and Papert 1969) надзвичайно уважно проаналізував цю проблему і встановив, що існує певна межа, до якої може навчатися однорівневий перцептрон, а отже, і продукувати певні вихідні дані. На той час ще не існувало знань про навчання багаторівневих мереж, і тому тодішні дослідники звернулися до перспективніших галузей, а розвиток напряму штучних нейронних мереж припинився на деякий період часу. Винахід нових методів навчання багаторівневих мереж, більш ніж будь-який інший фактор, мав великий вплив на відновлення інтересу та досліджень у цій галузі. Робота Мінського дещо знизила завзяття ентузіастів та прихильників перцептронів, але це дало необхідний час для консолідації і розвитку базової теорії. Відзначимо, що аналіз Мінського не був спростований; він залишається досить важливим аргументом і повинен бути до кінця вивченим.

8.4.5. Навчання перцептрона

Здатність штучних нейронних мереж до самонавчання є однією з найбільш інтригових властивостей. Подібно до біологічних систем, які вони наслідують, ці мережі здатні модифікувати себе таким чином, щоб продукувати на виході результат, що є найближчим до бажаного. У 1962 р. Rosenblatt розробив алгоритм для навчання перцептронів (perceptron trammg algorithm), довівши, що перцептрон може навчитися відображати будь-яку функцію.

Навчання може бути підконтрольним або неконтрольованим. Підконтрольне навчання передбачає наявність зовнішнього «вчителя», який оцінює характер системи і здійснює поточні корекції. Неконтрольоване навчання не вимагає вчителя; мережа самоорганізується для здійснення бажаних змін. Перцептронне навчання відноситься до підконтрольного типу навчання.

Перцептрон навчається на наборі вхідних зразків і змінює свої вагові коефіцієнти доти, поки не буде досягнуто бажаного вихідного результату.

Навчальний алгоритм має такий вигляд.

1. Застосувати вхідний зразок X і обчислити вихід Y: Y = XW.

2 а) якщо вихід є коректним, йти до кроку 1;

2 б) якщо вихід є неправильним і дорівнює «0», додати до кожного входу відповідний ваговий коефіцієнт;

2 в) якщо вихід не є істиною і дорівнює «1», відняти кожен вхід від відповідного вагового коефіцієнта.

3. Перейти до кроку 1.

Найбільш загальний перцептронний навчальний алгоритм, який поширює техніку навчання на неперервні входи і виходи, має назву правило дельта. Щоб побачити, як це відбувається, замінимо крок 2 алгоритму, використовуючи поняття терміну δ, який є різницею між бажаним T і дійсним A виходами:

T–A

$$\delta = (T - A).$$

Коли $\delta = 0$, переходимо до кроку 2а, де вихід є коректним і нічого не потрібно змінювати.

Якщо $\delta > 0$ переходимо до кроку 2б;

$\delta < 0$ переходимо до кроку 2в.

У кожному з цих випадків δ перемножається на відповідне значення входу X_i і результат додається до відповідного вагового коефіцієнта. У загальному випадку значення δ X_i множиться ще на коефіцієнт навчального приросту n, встановлюючи середній розмір змін вагових коефіцієнтів.

$$\Delta_i = \eta \delta x_i,$$

$$w_i(n+1) = w_i(n) + \Delta_i,$$

де Δ_i – корекція по входу x_i; $w_i(n+1)$ – значення і-го вагового коефіцієнта після корекції; $w_i(n)$ – значення і-го вагового коефіцієнта до корекції.

Дельта правило модифікує вагові коефіцієнти для всіх систем з полярними, неперервними і бінарними входами і виходами. Це дає можливість застосовувати подібні перцептронні системи для розв'язування багатьох задач. Але разом з тим перцептронний навчальний алгоритм має і ряд недоліків. Наприклад, в доказі про цей алгоритм не визначено кількості кроків, необхідних для навчання мережі. Хоча невеликою втіхою буде знання про кількість кроків, якщо час, необхідний для навчання, вимірюється космічними мірками. Більше того, нема доказів того, що перцептронний навчальний алгоритм є швидшим, аніж простий перебір всіх можливих варіантів вагових коефіцієнтів; у деяких випадках таке несвідоме наближення може дати кращі результати. Набагато кращими нейронними мережами є багаторівневі мережі, що використовують алгоритм зворотної похибки (backpropagation).

8.4.6. Алгоритм зворотньої похибки (backpropagation)

Винайдення алгоритму зворотньої похибки зіграло вирішальну роль у відновлені інтересу до штучних нейронних мереж. Backpropagation є систематичним методом для навчання багаторівневих нейронних мереж. Він має сильну математичну базу. Всупереч всім обмеженням, цей метод має широкий діапазон застосувань щодо проблем, які можуть бути подані за допомогою штучних нейронних мереж. Він може продемонструвати багато успішних прикладів використання своєї сили.

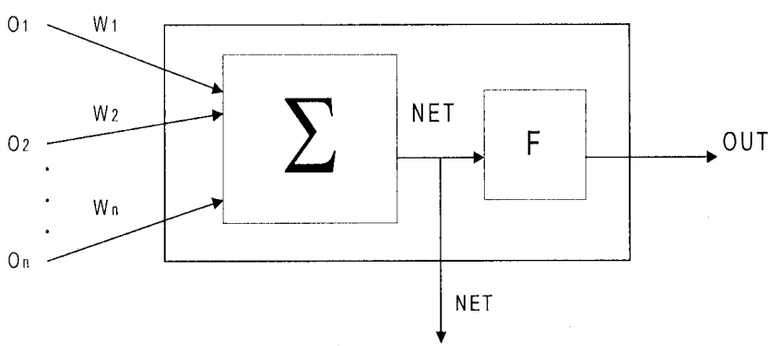


Рис. 8.16. Штучний нейрон з функцією активізації.

На рис. 8.16 показана загальна форма нейрона, який використовується як основний

блок для побудови мереж за методом зворотньої похибки.

Кожен вхід такого нейрона є виходом нейрона попереднього рівня. Всі входи перемножуються на відповідні вагові коефіцієнти і результати сумуються. У результаті отримуємо сигнал NET, який мусить бути обчислений для кожного нейрона в мережі. Після обчислень виходів NET до кожного з них застосовується активізаційна функція F продукуючи сигнал OUT.

$$NET = O_1 W_1 + O_2 W_2 + \ldots + O_n W_n = \sum_{i=1}^{n} O_i W_i,$$

$$OUT = F(NET).$$

Як правило для алгоритму backpropagation застосовується сигмоїдальна функція див. рис. 8.11:

$$OUT = F(NET) = \frac{1}{1+e^{-NET}},$$

$$F'(NET) = \frac{\partial\, OUT}{\partial\, NET} = OUT(1-OUT).$$

Як видно з рівнянь, така функція є досить прийнятною для алгоритму зворотньої похибки, оскільки має просту похідну. Її часто називають логічною або функцією, що стискає, оскільки вона стискає межі NET таким чином, що вихід OUT лежить в межах від 0 до 1. Багаторівнева мережа має більші репрезентаційні можливості в порівнянні з однорівневою, якщо подана нелінійність. Ця функція продукує бажану нелінійність. Серед великої кількості можливих функцій алгоритм зворотньої похибки вимагає такої, яка б постійно змінювалась. Наведена вище функція задовольняє цю вимогу. Крім того, вона ще й здійснює автоматичний контроль приросту. Для сигналів низького рівня (NET біля 0) нахил кривої входу/виходу буде крутим, продукуючи великий приріст; якщо ж величина сигналу збільшується, то приріст зменшується.

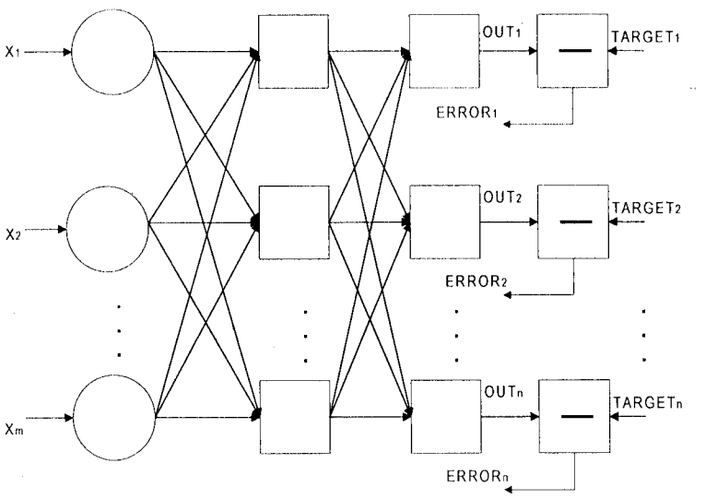


Рис. 8.17. Дворівнева мережа за методом зворотньої похибки.

На рис. 8.17 показана багаторівнева мережа, яка може бути використана для алгоритму зворотньої похибки.

Перші нейрони використовуються як вхідні; вони не виконують вхідних сумувань. Вхідний сигнал просто проходить через них на виходи Кожен нейрон в подальших рівнях продукує сигнали NET і OUT, як було описано вище.

У літературі не визначена кількість рівнів в таких мережах. Дехто рахує рівні за кількістю шарів нейронів (включаючи початковий рівень, що не сумує), інші – за кількістю рівнів вагових коефіцієнтів. Оскільки останнє визначення краще надається для функціонального опису, то скористаємося ним. Згідно з цим визначенням, мережа на рис. 8.17 складається із двох рівнів. Кожен нейрон асоціюється з відповідним набором вагових коефіцієнтів, які застосовуються до його входів. Тоді вагові коефіцієнти першого рівня закінчуються на нейронах першого рівня. Вхід (чи початковий рівень) визначений як нульовий шар.

Метою тренування мережі є встановлення вагових коефіцієнтів таким чином, щоб множина входів давала на виході бажані результати. Навчання передбачає існування тренувальних векторів, що репрезентують бажані результати, для кожного з векторів на вході системи. Вхідний і тренувальний вектори складають тренувальну пару. Як правило, мережа навчається з великою кількістю тренувальних пар.

Перед початком тренування всі вагові коефіцієнти мусять бути ініціалізовані довільними малими числами. Це запобігає перенасиченню нейронної мережі великими значеннями вагових коефіцієнтів і виникненню інших тренувальних патологій. Для прикладу, якщо початково були встановлені всі однакові вагові коефіцієнти, а бажані перетворення повинні репрезентувати значення, які не є рівними між собою, то мережа не буде навчатися.

Алгоритм методу backpropagation передбачає такі кроки.

1. Вибрати тренувальну пару з навчального набору і застосувати вхідний вектор до входу системи.
2. Обчислити значення на виході мережі.
3. Обчислити помилку між значенням, що отримане на виході мережі, та бажаним значенням (вектор-зразок із тренувальної пари).
4. Встановити вагові коефіцієнти мережі в напрямку, що мінімізує помилку.
5. Повторити кроки 1-4 для кожного вектора в тренувальному наборі доти, поки помилка на вході не буде достатньо малою.

Кроки 1–2 подібні до інших навчальних алгоритмів. Обчислення здійснюються по рівневих базисах. Використовуючи рис. 8.17, можна побачити, що спочатку обчислюються значення на виходах нейронів рівня j, які подаються на входи рівня k; обчислюються значення на виходах нейронів на рівні k, і вони утворюють вихідний вектор мережі. На кроці 3 кожне значення на виходах мережі (OUT) віднімається від відповідної складової вектора зразка з метою визначення похибки. Ця похибка використовується на кроці 4 при встановлені вагових коефіцієнтів, причому полярність і величина вагових змін залежать від навчального алгоритму. Після певної кількості повторень цих 4-х кроків помилка між значеннями реальних виходів і зразками буде мати прийнятне значення, і можна стверджувати, що мережа є навченою. Далі цю мережу можна застосовувати для роботи з певними даними, причому вагові коефіцієнти вже залишатимуться незмінними.

Легко побачити, що кроки 1-2 визначають прямий хід, оскільки при цьому сигнал поширюється від входу мережі до її виходу. Кроки 3-4 є зворотним ходом; тут обчислений сигнал похибки поширюється назад через мережу, встановлюючи відповідні вагові коефіцієнти. У математичному викладі ці два кроки будуть мати такий вигляд.

Прямий хід

Введемо позначення: вектор X – вхідна множина значень, Y – вихідна множина. Вектор-зразок для відповідного вхідного вектора X береться з навчального набору і позначається через T. Застосовуючи вектор X до входів, на виході мережі отримуємо певний вектор Y. Обчислення в багаторівневих мережах здійснюється від рівня до рівня. NET значення кожного нейрона в першому рівні обчислюється як вагове сумування по всіх входах. Далі активізаційна функція стискає значення NET, продукуючи сигнал OUT для кожного нейрона в цьому рівні. Виходи нейронів одного рівня служать входами для наступного. Такий процес повторюється рівень за рівнем, доки не буде знайдений вихідний набір мережі.

Зобразимо цей процес у векторній формі. Вагові коефіцієнти складають матрицю W. Для прикладу, ваговий коефіцієнт від нейрона 8 в шарі 2 до нейрона 5 в 3-му рівні позначається як $W_{8,5}$. Вектор NET для рівня N може бути зображений як

$$N = XW.$$

Застосовуючи функцію F до вектора N компонента за компонентою, продукуємо вихідний вектор O:

$$O = F(XW),$$

Як вже було зазначено раніше, такі обчислення здійснюються рівень за рівнем для всієї мережі.

Зворотний хід

1. Встановлення вагових коефіцієнтів вихідного рівня.

Оскільки для кожного нейрона у вихідному рівні є в наявності вектор-зразок, то встановлення вагових коефіцієнтів відбувається за правилом дельта. Внутрішні рівні мають назву «схованих», оскільки їх виходи не мають векторів-зразків для порівняння; а отже, процес навчання дещо ускладнюється. Рис. 8.18 показує навчальний процес для одного вагового коефіцієнта від нейрона p у схованому рівні і до нейрона q у вихідному рівні k.

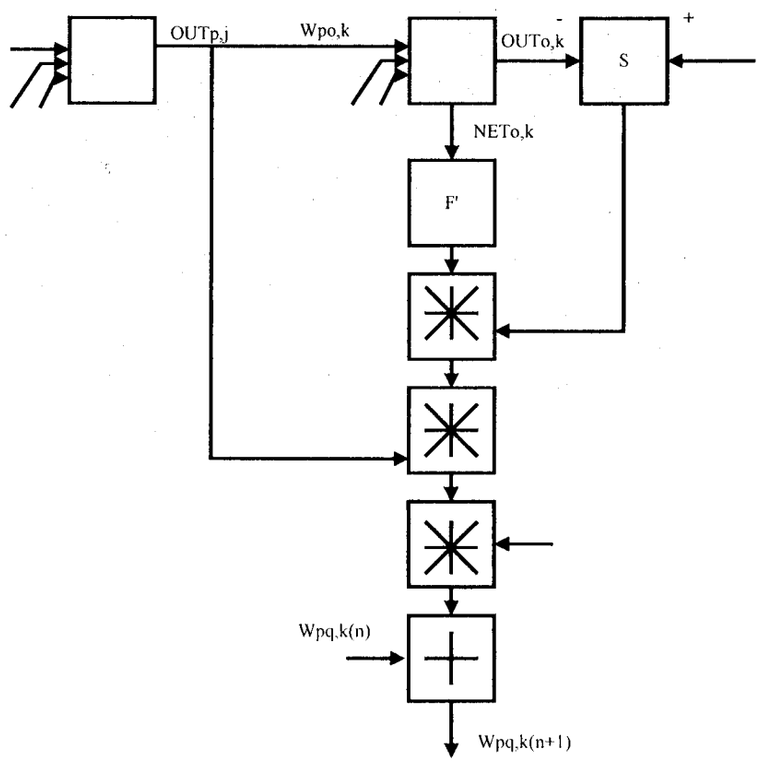


Рис. 8.18. Приклад навчального процесу.

Вихід нейрона рівня k віднімається від значення зразка, продукуючи тим самим сигнал похибки. Він, у свою чергу, множиться на функцію, що стискає OUT(1–OUT), яка обчислена для цього нейрона k-го рівня, і ми отримуємо значення δ :

$$\delta = OUT(1-OUT)(T\arg et - OUT). \qquad (11.1)$$

Далі δ перемножається на сигнал OUT від нейрона j та на коефіцієнт навчального приросту n (як правило від 0.01 до 1.0) і отриманий результат додається до вагового коефіцієнта.

$$\Delta W_{pq,k} = \eta\delta_{q,k} OUT_{p,j}, \qquad (11.2)$$

$$\Delta W_{pq,k}(n+1) = W_{pq,k}(n) + \Delta W_{pq,k}, \qquad (11.3)$$

де $\Delta W_{pq,k}(n)$ – значення вагового коефіцієнта від нейрона p в схованому рівні до нейрона q у вихідному рівні на кроці n (перед зміною); $\Delta W_{pq,k}(n+1)$ – значення вагового коефіцієнта на кроці n+1; $\delta_{q,k}$ – значення δ для нейрона q в вихідному рівні k ; $OUT_{p,j}$ – значення OUT для нейрона p в схованому рівні j.

Відзначимо, що позначки p,q використовуються для позначення специфічних нейронів, а j,k – для позначення рівнів. Подібні процеси здійснюються для кожного вагового коефіцієнта від нейрона в схованому рівні до нейрона у вихідному рівні.

2. Встановлення вагових коефіцієнтів у схованих рівнях.

Сховані рівні не мають вектора-зразка, а отже нема можливості скористатися наведеним вище тренувальним процесом. Відсутність тренувальних зразків викликала велику проблему при формуванні алгоритму зворотної похибки. Backpropaga-tion навчає нейрони у схованих рівнях, поширюючи вихідну помилку назад через мережу, встановлюючи вагові коефіцієнти для кожного рівня.

Вирази (8.2) і (8.3) використовуються як для вихідного, так і для схованого рівнів. Але для схованих рівнів значення обчислюється без допомоги вектора-зразка. На рис. 8.19 показано, яким чином це досягається.

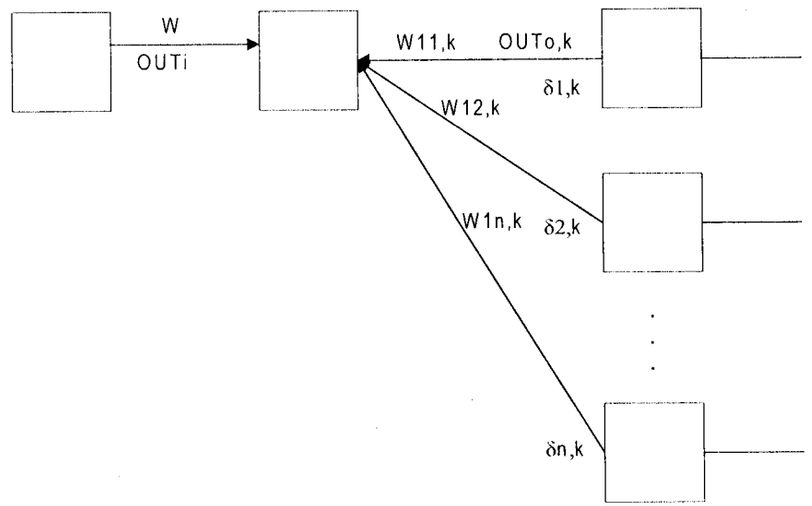


Рис. 8.19. Процес обчислення значень δ для схованих рівнів.

Спочатку для кожного нейрона за рівністю (8.1) обчислюється δ . Це використовується для встановлення коефіцієнтів у вихідному рівні. Значення δ для нейрона в схованому рівні отримується наступним чином :

$$\delta_{p,j} = OUT_{p,j}(1-OUT_{p,j})(\sum_{q}\delta_{q,k}W_{pq,k}). \qquad (8.35)$$

Маючи δ , можна обчислити вагові коефіцієнти в першому схованому рівні. Це здійснюється шляхом використання рівнянь (8.2) і (8.3), модифікуючи відповідні вказівники.

Значення δ мусить бути обчислене для всіх нейронів в схованому рівні і всі вагові коефіцієнти, які асоціюються з цим рівнем, повинні бути встановлені. Ця операція повторюється назад від рівня до рівня, аж поки не буде досягнуто вихідного рівня і всі вагові коефіцієнти не будуть встановлені.

У векторному поданні операція зворотного поширення помилки записується компактніше. Позначимо набір δ s вихідного рівня через D_k і набір вагових коефіцієнтів вихідного рівня як W_k. Щоб перейти до D_j необхідно здійснити два наступних кроки.

1. Перемножити δ вектор вихідного рівня D_k на вагові коефіцієнти t вихідного рівня W_k, що зв'язані зі схованим рівнем.
2. Перемножити кожен результат кроку 1 на функцію, що стискає відповідного нейрона у схованому рівні.

У символьному відображенні це має вигляд :

$$D_j = D_k W_k^t \otimes [O_j \otimes (I - O_j)],$$

де оператор $\otimes$ означає множення покомпонентно двох векторів; O_j – вихідний вектор рівня j; I – одиничний вектор.

8.4.7. Обмеження обчислювальних можливостей нейронних мереж

Однорівневі нейронні мережі мають обмеження щодо виду задач, які можна репрезентувати за їх допомогою. Є певний клас функцій, які не можуть подаватися за допомогою однорівневої мережі. Ці функції називаються лінійносепарабельними, і вони встановлюють межу можливостей однорівневих мереж. Оскільки лінійна сепарабельність обмежує можливості перцептрона, то важливо знати чи є певна функція лінійно-сепарабельною. На жаль, нема можливості встановити це, якщо кількість змінних досить велика. Для прикладу, для нейрона з n бінарними входами, що може мати 2" різних варіантів, які продукують «0» або «1», ми маємо 22 різних функцій від n змінних.

Як видно з наведеної нижче таблиці, ймовірність довільно вибраної функції бути лінійно-сепарабельною зменшується зі збільшенням кількості змінних. З цієї причини однорівневі перцептрони є практично обмежені для вирішення простих проблем.

n	2^{2^n}	Кількість лінійно-сепарабельних функцій
1	4	4
2	16	14
3	256	104
4	65536	1882
5	4.3E9	94572
6	1.8E19	5028134

Шляхом подолання лінійно-сепарабельного обмеження є збільшення рівнів нейронної мережі. Наприклад, утворюючи каскад з двох однорівневих нейронних мереж ми тим самим створюємо дворівневу нейронну мережу, яка вже має значно більші можливості застосування, і для якої можна вже використати навчальний алгоритм зворотньої похибки back-propagation.

Запитання для повторення та контролю знань

1. Що розуміють під терміном «машинне навчання»?
2. На досягнення яких результатів націлені навчальні алгоритми?
3. Які компоненти повинна містити навчальна система?

4. Які Ви знаєте рівні навчання?
5. Яке навчання називається самонавчанням?
6. Коли застосовуються автомати з лінійною тактикою?
7. Який алгоритм роботи автомата з лінійною тактикою?
8. Яке місце у процесі навчання займає узагальнення?
9. Опишіть схему Бонгарда.
10. Яке навчання називається параметричним?
11. Чим відрізняється індуктивна перевірка гіпотез від індуктивного формування гіпотез?
12. Як виглядає правило Мілля формування гіпотез?
13. У чому полягає парадокс Хемпеля?
14. Для чого використовуються генетичні алгоритми?
15. Що таке еволюційна теорія?
16. Дайте поняття природного відбору і генетичного спадкування.
17. Яка умовна схема кросінговеру?
18. Для чого використовується генетичний алгоритм?
19. Опишіть роботу генетичного алгоритму.
20. Як виглядає блок-схема генетичного алгоритму?
21. Коли застосовуються генетичні алгоритми?
22. Дайте постановку задачі класифікації даних.
23. Як виглядає метод класифікації на основі індукції дерева рішень?
24. Яка послідовність побудови класифікаційної моделі?
25. Як відбувається побудова дерева рішень та набору класифікаційних правил?
26. Наведіть приклад виявлення логічних закономірностей в даних.
27. Що таке штучні нейронні мережі?
28. Наведіть модель штучного нейрона.
29. Які Ви знаєте функції активізації?
30. Які штучні нейронні мережі називаються однорівневими?
31. Дайте поняття перцептрона.
32. Який перцептрон називається багатовихідним?
33. Як відбувається навчання перцептрона?
34. Опишіть алгоритм зворотної похибки (backpropagation).
35. Як працює двохрівнева мережа за методом зворотної похибки?
36. Які кроки методу backpropagation?
37. Які обмеження обчислювальних можливостей нейрон-них мереж?

Завдання для самостійного розв'язування

1. За вихідною матрицею ознак та об'єктів побудувати дерево рішень, де об'єкти 1,2,3 відносяться до 1-го класу, а 4,5,6 до 2-го класу.

	x_1	x_2	x_3	x_4	x_5	x_6
1	0	1	0	0	1	1
2	0	0	0	1	1	1
3	0	1	1	0	1	1
4	1	0	1	0	0	1
5	0	0	1	0	1	1
6	0	0	1	1	0	1

Написати правила, що випливають із побудованого дерева. До якого класу відноситься об'єкт із ознаками: 0 1 0 1 0 1?

РОЗДІЛ 9

ОНТОЛОГІЇ Й ОНТОЛОГІЧНІ СИСТЕМИ

- Основні визначення
- Моделі онтологій. Онтологічні системи
- Системи і засоби подання онтологічних знань

В даному розділі дається поняття онтології, розглядаються онтологічні системи. Описано моделі онтологій і їх використання під час розроблення інтелектуальних систем. У кінці розділу описано системи та засоби подання онтологічних знань.

9.1. Основні визначення

Онтологія (від грец. онтос – суще, логос – навчання, поняття) – термін, що визначає вчення про буття, про сутність, на відміну від гносеології – вчення про пізнання. Вже у Х. Вольфа (1679-1754), автора терміну «онтологія», вчення про буття було відокремлене від вчення про пізнання. Термін введений у філософську літературу німецьким філософом Р. Гокленіусом (1547-1628). До цього онтологія була частиною метафізики, наукою самостійною, незалежною і не пов'язаною з логікою, з «практичною філософією», з науками про природу. Її предмет складає вивчення абстрактних і загальних філософських категорій, таких як буття, субстанція, причина, дія, явище тощо, а сама онтологія як наука домагалася повного пояснення причин всіх явищ [157].

Зрозуміло, що таке визначення трохи недоречне для практичного використання, але дає поштовх для подальшої конкретизації й обговорення, виходячи з цілей цього видання. У цьому значенні цікавіше визначення онтології, запропоноване в межах розроблення системи стандартів на мультиагентні системи міжнародним співтовариством FIPA (Foundation for Intelligent Physical Agents). У філософському розумінні можна посилатися на онтологію як на певну систему категорій, що є наслідком певного погляду на світ.

При цьому сама система категорій не залежить від конкретної мови: онтологія Аристотеля завжди одна і та ж, незалежно від мови, використаної для її опису.

З іншої точки зору, ближчої до понять, пов'язаних зі штучним інтелектом (ШІ), онтологія – це формально відображені на базі концептуалізації знання. Концептуалізація, як вже обговорювалося вище, припускає опис безлічі об'єктів і понять, знань про них і зв'язків між ними. Отже, онтологією [249] називається експліцитна специфікація концептуалізації. Формально онтологія складається з термінів, організованих в таксономію, їх визначень і атрибутів, а також пов'язаних з ними аксіом і правил виведення.

Часто набір припущень, що становлять онтологію, має форму логічної теорії першого порядку, де терміни словника є іменами унарних і бінарних предикатів, званих

відповідно концептами і відношеннями. У простому випадку онтологія описує тільки ієрархію концептів, зв'язаних відношеннями категоризації. У складніших випадках в неї додаються відповідні аксіоми для виразу інших відношень між концептами і для того, щоб обмежити їх передбачувану інтерпретацію. Враховуючи вищесказане, онтологія є базою знань, що описує факти, які передбачаються завжди істинними в рамках певного співтовариства на основі загальноприйнятого значення використовуваного словника.

Ще конкретніше поняття онтології у відомому проекті Ontolingua [235], який активно розробляється в Стенфордському університеті. Тут передбачається, що онтологія – це експліцитна специфікація певної теми.

Такий підхід припускає формальне і декларативне уявлення деякої теми, яке включає словник (або список констант) для посилання до термінів предметної області, обмеження цілісності на терміни, логічні твердження, які обмежують інтерпретацію термінів і те, як вони поєднуються один з одним.

Резюмуючи вищесказане, можна констатувати, що на сьогодні розуміння терміну «онтологія» різне, залежно від контексту і цілей його використання. У роботі [253] дане достатньо змістовне і цікаве обговорення цих питань, яке зводиться до того, що тут виділяються наступні аспекти інтерпретації цього терміну:

1) онтологія як філософська дисципліна;
2) онтологія як неформальна концептуальна система;
3) онтологія як формальний погляд на семантику;
4) онтологія як специфікація «концептуалізації»;
5) онтологія як уявлення концептуальної системи через логічну теорію, що характеризується:
   - спеціальними формальними властивостями,
   - її призначенням;
6) онтологія як словник, використовуваний логічною теорією;
7) онтологія як метарівнева специфікація логічної теорії.

Зазначимо, що перша інтерпретація радикально відрізняється від інших і пов'язана, як пропонують автори вищезгаданої роботи, з тим, що тут ми говоримо про Онтологію (з великої літери) і маємо на увазі філософську дисципліну, що вивчає, за Аристотелем, природу й організацію сущого. У цьому значенні Онтологія намагається відповісти на запитання: «Що є суще?» або, в іншому формулюванні, на запитання: «Які властивості є загальними для всього сущого?». Коли ж ми говоримо про онтологію (з маленької літери), то посилаємося на об'єкт, природа якого може бути різною, залежно від вибору між інтерпретаціями 2-7. За другою інтерпретацією онтологія є концептуальною системою, яку ми можемо припускати як базис визначеної БЗ. Згідно з інтерпретацією 3 онтологія, на основі якої побудована БЗ, виражається в термінах відповідних формальних структур на семантичному рівні. Отже, ці дві інтерпретації розглядають онтологію як концептуальну «семантичну» сутність, неважливо – формальну або неформальну, тоді як інтерпретації 5-7 трактують онтологію як спеціальний «синтаксичний» об'єкт. Інтерпретація, що залишилася, четверта, яка була запропонована Грубером [249] як визначення онтології для використання в рамках ШІ-співтовариства, – одна з найпроблематичніших, оскільки точне значення її залежить від розуміння термінів «специфікація»

і «концептуалізація». І разом з тим, саме це визначення найчастіше і використовується сьогодні в роботах з проектування і дослідження онтології.

Для визначеності подальшого викладу ми вважатимемо, що онтології – це БЗ спеціального типу, які можуть «читатися» і розумітися, відчужуватися від розробника чи фізично розділятися їх користувачами.

При цьому онтологічний інжиніринг – гілка інженерії знань, яка використовує Онтологію (з великої букви) для побудови онтології (з маленької букви). Зрозуміло, що будь-яка онтологія має під собою концептуалізацію, але одна концептуалізація може бути основою різних онтологій, і дві різні БЗ можуть відображати одну онтологію.

9.2. Моделі онтології й онтологічної системи

Вище вже наголошувалося, що поняття онтології припускає визначення і використання взаємозв'язаної та взаємоузгодженої сукупності трьох компонентів: таксономії термінів, визначень термінів і правил їх опрацювання. Враховуючи це, введемо наступне визначення поняття моделі онтології.

Під *формальною моделлю онтології* **O** будемо розуміти впорядковану трійку такого вигляду:

$$O = <X, \Re, \Phi>,$$

де **X** – скінченна множина концептів (понять, термінів) предметної області, яку задає онтологія **O**; $\Re$ – скінченна множина відношення між концептами (поняттями, термінами) заданої предметної області; **Φ** – скінченна множина функцій інтерпретації (аксіоматизація), заданих на концептах чи відношеннях онтології **O**.

Зазначимо, що природним обмеженням, що накладається на множину **X**, є його скінченність і непустота. Інша справа з компонентами **Φ** і $\Re$ у визначенні онтології **O**. Зрозуміло, що і в цьому випадку **Φ** і $\Re$ повинні бути скінченними множинами. Розглянемо окремі випадки, коли ці множини порожні.

Нехай $\Re = \emptyset$ і $\Phi = \emptyset$. Тоді онтологія **O** трансформується в простий словник:

$$O = V = <X, \{\}, \{\}>.$$

Така вироджена онтологія може бути корисна для специфікації, поповнення і підтримки словників ПО, але онтологічні словники мають обмежене використання, оскільки не вводять експліцитно значення термінів. Хоча в деяких випадках, коли використовувані терміни належать до дуже вузького (наприклад, технічного) словника, і їх значення вже наперед добре узгоджені в межах певного (наприклад, наукового) об'єднання, такі онтології застосовуються на практиці. Відомими прикладами онтології цього типу є індекси машин пошуку інформації в мережі Інтернет.

Інша ситуація у разі використання термінів природної мови або в тих випадках, коли спілкуються програмні агенти. У цьому випадку необхідно характеризувати передбачуване значення елементів словника за допомогою відповідної аксіоматизації, мета використання якої – вилучення небажаних моделей і в тому, щоб інтерпретація була єдиною для всіх учасників спілкування.

Інший варіант відповідає випадку $\Re = \emptyset$, але $\Phi \neq \emptyset$. Тоді кожному елементу множини термінів з **X** може бути поставлена у відповідність функція інтерпретації **f** з **Φ**. Формально це твердження може бути записане таким чином.

Нехай

$$X = X_1 \cup X_2,$$

причому

$$X_1 \cap X_2 = \emptyset,$$

де $\mathbf{X}_1$ – множина термінів, що інтерпретуються; $\mathbf{X}_2$ – множина інтерпретуючих термінів, які інтерпретують (інтерпретаційних)

Тоді

$$\exists (x \in X_1, y^1, y^2, ..., y^k \in X_2),$$

такі що

$$x = f(y^1, y^2, \ldots, y^k),$$

де $f \in \Phi$

Те, що перетин множин X_1 і X_2 порожня множина виключає циклічні інтерпретації, а введення на розгляд функції k аргументів покликано забезпечити повнішу інтерпретацію. Тип відображення f з Φ визначає виразну потужність і практичну користь цього виду онтології. Так, якщо припустити, що функція інтерпретації задається оператором присвоєння значень ($X_1 := X_2$, де X_1 – ім'я інтерпретації X_2), то онтологія трансформується в пасивний словник V^p:

$$O = V^p = <X_1 \cup X_2, \{\}, \{:=\}>.$$

Такий словник пасивний, оскільки всі визначення термінів з X_1 беруться із вже наявної та фіксованої множини X_2. Практична цінність його вища, ніж простого словника, але недостатня, наприклад, для відображення знань в задачах опрацювання інформації в Інтернеті через динамічний характер цього середовища.

Щоб врахувати останню обставину, припустимо, що частина інтерпретаційних термінів із множини X_2 задається процедурно, а не декларативно. Значення таких термінів «обчислюється» кожного разу при їх інтерпретації.

Цінність такого словника для задач опрацювання інформації в середовищі Інтернет вища, ніж у попередньої моделі, але все ще недостатня, оскільки елементи, що інтерпретуються з X_1, ніяк не зв'язані між собою, отже, виконують лише роль ключів входження в онтологію.

Для відображення моделі онтології, яка потрібна для розв'язування задач опрацювання інформації в Інтернеті, очевидно, вимагається відмовитися від припущення $\Re = \emptyset$.

Отже, припустимо, що множина відношень на концептах онтології не порожня, і розглянемо можливі варіанти її формування.

Для цього введемо в розгляд спеціальний підклас онтології – просту таксономію, а отже:

$$O = T^0 = < X, \{is\_a\}, \{\} >.$$

Під таксономічною структурою будемо розуміти ієрархічну систему понять, зв'язаних між собою відношенням is_a («бути елементом класу»).

Відношення is_a має фіксовану наперед семантику і дозволяє організувати структуру понять онтології у вигляді дерева. Такий підхід має свої переваги і недоліки, але загалом є адекватним і зручним способом для відображення ієрархії понять.

Результати аналізу окремих випадків моделі онтології наведені в табл. 8.1.

Таблиця 9.1. Класифікація моделей онтології

Компоненти моделі	$\Re = \emptyset \Phi = \emptyset$	$\Re = \emptyset$ $\Phi \neq \emptyset$	$\Re = \emptyset$ $\Phi \neq \emptyset$	$\Re = \{is\_a\}$ $\Phi = \emptyset$
Формальне визначення	< X, {}, {} >	$< X_1 \cup X_2, \{\}, \Phi >$	$< X_1 < X_1 \cup (X_2' \cup \rho_2'), \{\}, \Phi >$	< X, {is_a}, {} >
Пояснення	Словник ПО	Пасивний словник ПО	Активний словник ПО	Таксономія понять ПО

Далі можна узагальнити окремі випадки моделі онтології таким чином, щоб забезпечити можливість:

- відображення множини концептів X у вигляді мережевої структури;
- використання достатньо великої множини відношень, яка включає не тільки таксономічні відношення, але і відношення, що відображають специфіку конкретної предметної області, а також засоби розширення множини $\Re$;
- використання декларативних і процедурних інтерпретацій і відношень, включаючи можливість визначення нових інтерпретацій.

Тоді можна ввести в розгляд модель розширюваної онтології і досліджувати її властивості. Проте, враховуючи технічну спрямованість цього підручника, ми не будемо робити цього тут, а охочих познайомитися з такою моделлю скеруємо до роботи [18]. Як показано в цій роботі, модель розширюваної онтології є досить могутньою для специфікації процесів формування просторів знань в середовищі Інтернет. Разом з тим, і ця модель є неповною через свою пасивність навіть там, де визначені відповідні процедурні інтерпретації та введені спеціальні функції поповнення онтології. Адже єдиною точкою управління активністю в такій моделі є запит на інтерпретацію певного концепту. Цей запит виконується завжди однаково й ініціює запускання відповідної процедури. А власне виведення відповіді на запит чи пошук необхідної для цього інформації залишається поза моделлю і повинен реалізовуватися іншими засобами.

Враховуючи вищесказане, а також необхідність експліцитної специфікації процесів функціонування онтології, введемо в розгляд поняття онтологічної системи.

Під *формальною моделлю онтологічної системи* $\sum^0$ будемо розуміти триплет вигляду:

$$\sum^0 = < O^{meta}, \{O^{d\&t}\}, \Xi^{inf} >,$$

де $\mathbf{O}^{meta}$ – онтологія верхнього рівня (метаонтологія); $\{\mathbf{O}^{d\&t}\}$ – множина предметних онтологій і онтологій задач предметної області; Ξ^{inf^k} – модель машини виведення, асоційованої з онтологічною системою $\sum^0$.

Використання системи онтологій і спеціальної машини виведення дозволяє розв'язувати в такій моделі різні задачі. Збагачуючи систему моделей $\{\mathbf{O}^{d\&t}\}$, можна враховувати

побажання користувача, а змінюючи модель машини виведення, вводити спеціалізовані критерії релевантності, що одержуються в процесі пошуку інформації, і формувати спеціальні репозиторії накопичених даних, а також поповнювати, при необхідності, використовувані онтології.

У моделі Σ^0 є три онтологічні компоненти:

- метаонтологія;
- предметна онтологія;
- онтологія задач.

Як показувалося вище, метаонтологія оперує загальними концептами і відношеннями, які не залежать від конкретної предметної області. Концептами метарівня є загальні поняття, такі як «об'єкт», «властивість», «значення» тощо. Тоді на рівні метаонтології ми отримуємо інтенсіональний опис властивостей предметної онтології й онтології задач. Онтологія метарівня є статичною, що дає можливість забезпечити тут ефективне виведення.

Предметна онтологія O^{domain} містить поняття, що описують конкретну предметну область, відношення, семантично значущі для заданої предметної області, і множину інтерпретацій цих понять і відношень (декларативних і процедурних). Поняття предметної області специфічні в кожній прикладній онтології, але відношення – універсальніші. Тому як базис зазвичай виділяють такі відношення моделі предметної онтології, як part_of, kind_of, contained_in, member_of, see_also і деякі інші.

Відношення part_of визначене на множині концептів, є відношенням належності і показує, що концепт може бути частиною інших концептів. Воно є відношенням виду «части-на-ціле» і за властивостями близьке до відношення is_a і може бути задане відповідними аксіомами. Аналогічним чином можна ввести й інші відношення виду «частина-ціле».

Відношення see_also володіє іншою семантикою й іншими властивостями. Тому доцільно вводити його не декларативно, а процедурно, подібно до того, як це робиться при визначенні нових типів у мовах програмування, де підтримуються абстрактні типи даних:

X see_also Y:

see_also member_of Relation {

if ((X is_a Notion) & (Y is_a Notion) & (X see_also Y)) if (Operation connected_with X) Operation connected_with Y}.

Зазначимо, що і відношення see_also «не цілком» транзитивне. Так, якщо припустити, що (X1 see_also X2) & (X2 see_also X3), то можна вважати, що (X1 see_also X3). Проте у міру збільшення довжини ланцюжка об'єктів, зв'язаних цими відношеннями, справедливість транзитивного перенесення властивості connected_with зменшується. Тому у разі відношення see_also ми маємо справу не з відношенням часткового порядку (як, наприклад, у разі відношення is_a), а з відношенням толерантності. Однак для простоти це обмеження може бути перенесене з визначення відношення у функцію його інтерпретації.

Аналіз різних предметних областей показує, що введений вище набір відношень є достатнім для початкового опису відповідних онтологій. Зрозуміло, що цей базис є відкритим і може поповнюватися залежно від предметної області і цілей, що стоять перед прикладною системою, в якій така онтологія використовується.

Онтологія задач як поняття містить типи вирішуваних задач, а відношення цієї онтології, як правило, специфікують декомпозицію задач на підзадачі. Разом з тим, якщо прикладною системою розв'язується єдиний тип задач (наприклад, задачі пошуку релевантного запиту інформації), то онтологія задач може в такому випадку описуватися словниковою моделлю, розглянутою вище. Отже, модель онтологічної системи дозволяє описувати необхідні для її функціонування онтології різних рівнів. Взаємозв'язок між онтологіями показаний на рис. 9.1.

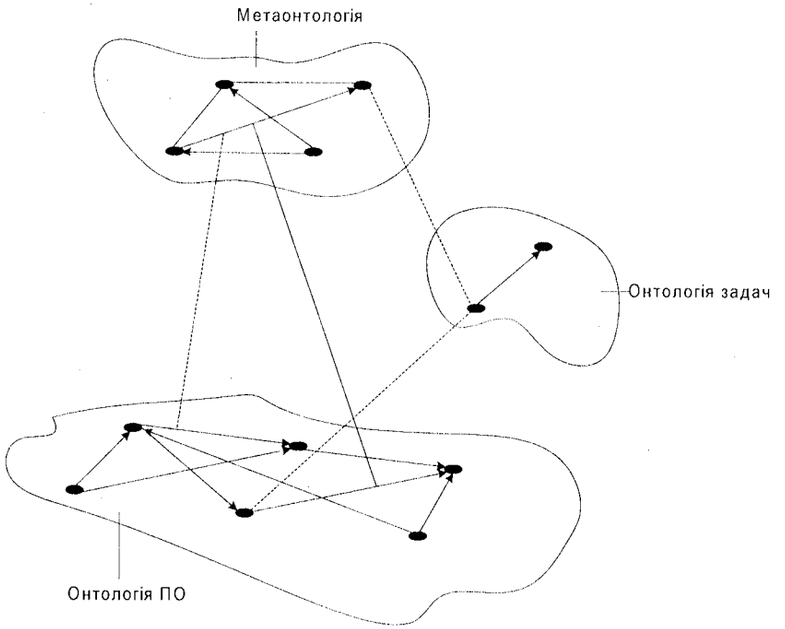


Рис. 9.1. Взаємозв'язок між онтологіями онтологічної системи

Машина виведення онтологічної системи в загальному випадку може спиратися на мережеве відображення онтології всіх рівнів. При цьому її функціонування буде пов'язане:

- з активацією понять чи відношень, що фіксують задачу, яка розв'язується (опис первинної ситуації);
- визначенням цільового стану (ситуації);
- виведенням на мережі; полягає у тому, що від вузлів первинної ситуації розповсюджуються хвилі активації, що використовують властивості відношень, пов'язаних з ними. Критерієм зупинки процесу є досягнення цільової ситуації або перевищення тривалості виконання (time-out).

9.3. Методології створення і «життєвий цикл» онтології

Як наголошувалося вище, розробники систем, заснованих на знаннях, стикаються з проблемою «вузької шийки» видобування знань. Аналогічна проблема існує і при створенні онтології. Але, на відміну від розробників інтелектуальних систем, творців онтології чекають і додаткові проблеми, пов'язані з відсутністю хоч якихось загальних і верифікованих методологій, що визначають, які «процедури» повинні виконуватися в

процесі розроблення, і на яких стадіях розроблення онтології вони повинні виконуватися. Сьогодні існує лише декілька незалежних методологій, орієнтованих на побудову онтології [250].

Відразу відзначимо, що ці підходи і методології базуються на наступних принципах проектування і реалізації онтології, запропонованих Грубером [249].

1. **Чіткість (Clarity)** – онтологія повинна ефективно передавати значення введених термінів. Визначення повинні бути об'єктивними, хоча мотивація введення термінів може визначатися ситуацією або вимогами обчислювальної ефективності. Для об'єктивізації визначень повинен використовуватися чітко фіксований формалізм, при цьому логічно задавати визначення у вигляді логічних аксіом.
2. **Узгодженість (Coherence)** – означає, що принаймні всі визначення повинні бути логічно несуперечливі, а всі твердження, що виводяться в онтології, не повинні суперечити аксіомам.
3. **Розширюваність (Extendibility)** – онтологія повинна бути спроектована так, щоб забезпечувати використання словників термінів, що розділяються, що допускають можливість монотонного розширення чи спеціалізації без необхідності ревізії вже наявних понять.
4. **Мінімум впливу кодування (Minimal encoding bias)** – концептуалізація, що лежить в основі створюваної онтології, повинна бути специфікована на рівні уявлення, а не символьного кодування. Цей принцип пов'язаний з тим, що агенти, що поділяють онтологію, можуть бути реалізовані в різних системах відображення знань.
5. **Мінімум онтологічних зобов'язань (Minimal ontological commitment)** – онтологія повинна містити тільки найістотніші припущення про модельований світ, щоб залишати свободу розширення і спеціалізації. Звідси витікає, що онтології базуються на «слабких» теоріях, оскільки мета їх створення і використання полягає, перш за все, в тому, щоб «говорити» про предметну область, на відміну від БЗ, які можуть містити знання, необхідні для розв'язання задач чи одержання відповідей на запитання.

9.4. Приклади онтологій

Сьогодні дослідження в області онтології та онтологічних систем є «гарячими точками» не лише в ШІ, але і в працях з інтелектуалізації інформаційного пошуку в середовищі Інтернет; у працях з мультиагентних систем; у проектах з автоматичного «виведення» знань з текстів на природній мові; в проектах, що проводяться в суміжних областях.

При цьому різні автори вводять різні типізації онтологій [20], сумуючи які можна виділити класифікації за:

- степенем залежності від конкретної задачі чи предметної області;
- рівнем деталізації аксіоматизації;
- «природою» предметної області тощо.

Додатково до цих вимірів можна ввести і класифікації, пов'язані з розробленням, реалізацією та супроводом онтологій, але така типізація доцільніша при обговоренні питань реалізації онтологічних систем.

За ступенем залежності від конкретної задачі чи предметної області часто розрізняють:

- онтології верхнього рівня;
- онтології, орієнтовані на предметну область;
- онтології, орієнтовані на конкретну задачу;
- прикладні онтології.

Онтології верхнього рівня описують дуже загальні концепти, такі як простір, час, матерія, об'єкт, подія, дія і т. ін., які незалежні від конкретної проблеми чи області. Тому здається розумним, принаймні в теорії, уніфікувати їх для більших товариств користувачів.

Наприклад, такою загальною онтологією є CYC® [277]. Однойменний проект – CYC® – орієнтований на створення мультиконтекстної бази знань і спеціальної машини виведення, яка розробляється фірмою Сусогр. Основна мета цього гігантського проекту – побудувати базу знань всіх загальних понять (починаючи з таких, як час, сутність і т. ін.), яка включає семантичну структуру термінів, зв'язків між ними й аксіом. Є наміри, що така база знань може бути доступна різним програмним засобам, що працюють зі знаннями, і грають роль бази «початкових знань». В онтології, за деякими даними, вже подані 106 концептів і 105 аксіом. Для відображення знань у рамках цього проекту розроблена спеціальна мова CYCL.

Іншим прикладом онтології верхнього рівня є онтологія системи Genrailized Upper Model [217], орієнтована на підтримку процесів опрацювання природної мови: англійської, німецької та італійської. Рівень абстракції цієї онтології перебуває між лексичними і концептуальними знаннями, що визначаються потребами спрощення інтерфейсно-лінгвістичними ресурсами. Модель Generalized Upper Model містить таксономію, організовану у вигляді ієрархій концептів (біля 250 понять) і окремої ієрархії зв'язків. Фрагмент системи понять цієї онтології показаний на рис. 9.2.

У цілому ж, можна констатувати, що, незважаючи на окремі успіхи, створення достатньо загальних онтологій верхнього рівня являє собою дуже вагоме завдання, яке ще не має задовільного рішення.

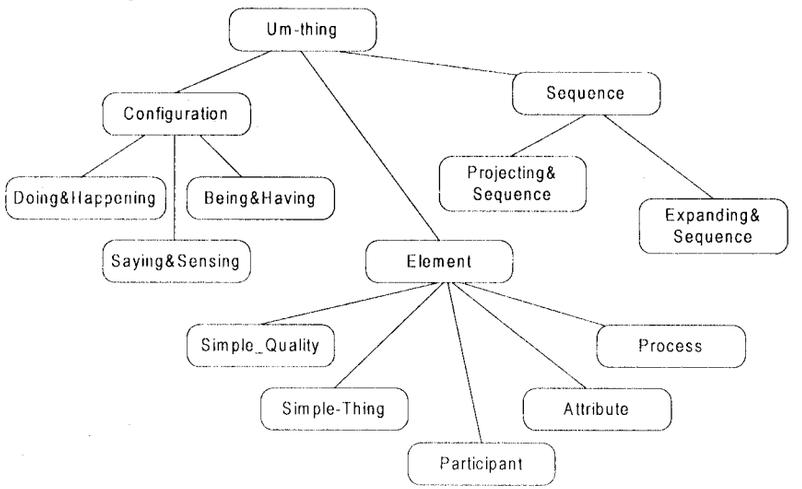


Рис. 9.2. Фрагмент системи понять онтології Generalized Upper Model

Предметні онтології й онтології задач описують, відповідно, словник, пов'язаний з предметною областю (медицина, комерція тощо) чи з конкретною задачею або діяльністю (діагностика, продажі тощо) за рахунок спеціалізації термінів, введених в онтологію верхнього рівня. Прикладами онтологій, орієнтованих на відповідну предметну область і конкретну задачу, є TOVE і Plinius відповідно [312].

Онтологія в системі TOVE (Toronto Virtual Enterprise Project) [312] предметно орієнтована на відображення моделі корпорації. Основна ціль її розроблення – відповідати на запитання користувачів з реінжинірингу бізнес-процесів, виводячи експліцитно відображення в онтології знання. При цьому система може здійснювати дедуктивне виведення відповідей. В онтології нема засобів для інтеграції з іншими онтологіями. Формально онтологія описується за допомогою фреймів. Таксономія понять онтології TOVE подана на рис. 9.3.

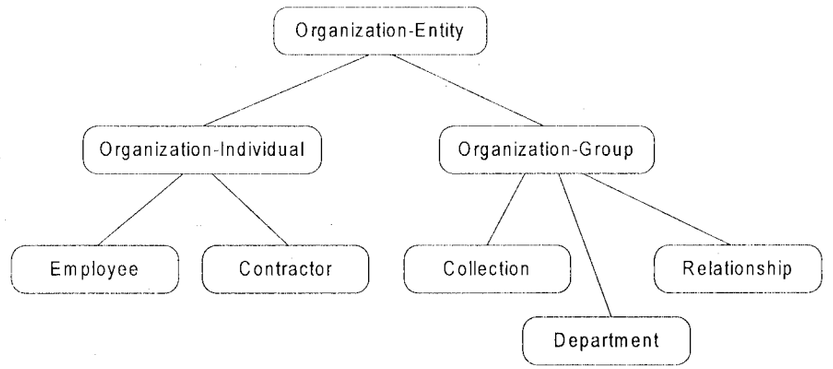


Рис. 9.3. Фрагмент таксономії понять онтології TOVE

Прикладні онтології описують концепти, які залежать як від конкретної предметної області, так і від задач, які в них розв'язуються. Концепти в таких онтологіях часто відповідають ролям, які відіграють об'єкти в предметній області в процесі здійснення відповідної діяльності. Приклад такої онтології – онтологія системи Plinius [317], призначена для напівавтоматичного виведенню знань з текстів в області хімії. На відміну від інших, згаданих вище онтологій, тут немає явної таксономії понять. Замість цього визначено декілька множин атомарних концептів, таких як, наприклад, хімічний елемент, ціле число тощо, і правила конструювання решти концептів. В онтології описано біля 150 концептів і 6 правил. Формально онтологія Plinius теж описується за допомогою фреймів.

Як показує аналіз робіт в цій області, науковими товариствами і колективами створюються онтології різних типів, але в цілому сьогодні найактивніше розроблюються і використовуються на практиці предметні онтології.

Разом з тим, незалежно від типу онтології, для її уявлення і використання потрібні спеціальні алгоритмічні засоби, до обговорення яких ми і переходимо в наступному параграфі.

9.5. Системи і засоби подання онтологічних знань

9.5.1. Основні підходи

Сьогодні у всьому світі дослідження використання знань в середовищі Інтернет [268] ведуться широким фронтом. І одним із головних аспектів в таких дослідженнях є алгоритмічні і програмні засоби уявлення онтологічних знань і роботи з онтологіями.

Як приклади дослідницьких проектів в цій тематиці можна вказати Co-operative Information Gathering Project із лабораторії Розподіленого ШІ університету Массачусет; проект видобування знань із гіпертекстів на основі використання методів машинного навчання, що виконується в університеті Карнегі Меллон; роботи Knowledge Technology Group лабораторії Sun Microsystems з технології опрацювання знань (проект «Precision Content Retrieval»), метою якого є побудова концептуальної таксономії фраз, виділених з індексованих матеріалів, і багато інших [278]. Загальною метою майже всіх таких проектів є розроблення нових підходів до побудови просторів знань і засобів роботи з ними, де б забезпечувалося:

- використання семантики для керівництва процесом відповіді на запити;
- можливість побудови відповідей з відповідною семантикою і простим синтаксисом, які могли б бути «зрозумілі» і опрацьовані програмними агентами чи іншими програмними засобами;
- можливість гомогенного доступу до інформації, яка фізично розподілена і гетерогенно відображена в Інтернеті;
- отримана інформація, яка явно не міститься серед фактів, отриманих з мережі, але може бути виведена з інших фактів і базових знань.

Вражаюча колекція посилань на такі проекти відображена в Інтернеті за адресою http://www.tzi.org/grp/i3/, але найцікавішими з огляду на тему цього розділу, напевно, є ініціатива (KA)2 [207] і проект SHOE [258], які розглядаються далі.

9.5.2. Ініціатива (KA)2 та інструментарій Ontobroker

Анотація знань в рамках ініціативи (KA)2

Анотація знань товариством набуття знань (Knowledge Annotation Initiative of the Knowledge Acquisition Community) – так розшифровується абревіатура (KA)2. Метою робіт цього міжнародного проекту є інтелектуальний пошук у середовищі Інтернет і автоматичне накопичення нових знань.

У рамках ініціативи (KA)2 виділяються три основних напрями досліджень:

- онтологічний інжиніринг (ontological engineering);
- анотація Web-сторінок;
- запити на інформацію на Web-сторінках і виведення відповідей на базі онтологічних знань.

Онтологічний інжиніринг – один з основних напрямів, в рамках якого товариство (KA)2 має наміри створити свою особисту і достатньо загальну систему онтології на основі використання засобів Ontolingua [235]. Сьогодні вже розроблено 8 онтологій, які можуть розглядатися як розділи загальної онтології – онтологія організації (organization ontology), проекту (project ontology), особистості (person ontology), напряму досліджень

(research-topic ontology), публікацій (publication ontology), подій (event ontology), дослідницьких продуктів (research-product ontology) і дослідних груп (research-group ontology). При цьому розроблення прикладів онтологій здійснюється і керується учасникам проекту – так званими провайдер-агентами («provider agents»), а розміщуються ці онтології на їх Web-сторінках. Такі сторінки анотуються з використанням нового типу HTML-тегів (ONTO), в рамках яких інформація розробляється спеціальним компонентом, що працює на основі онтології, – системою Ontocrawler. У рамках цього компоненту, в залежності від «багатства» використаної онтології, може виводитися нова інформація, релевантна запитам, але неявно присутня на Web-сторінках в мережах Інтернету.

Сама система Ontocrawler опрацьовується в рамках окремого проекту ініціативи (KA)2 – проекту Ontobroker [237], який, власне, і цікавий з точки зору обговорення засобів відображення й опрацювання онтологічних знань.

Засоби специфікації онтологій в проекті Ontobroker

У Ontobroker є три основні підсистеми: інтерфейс формулювання запитів (query interface), машина виведення відповідей (inference engine) і власне машина доступу до Інтернетресурсів – «хробак» (Webcrawler), що використовується для накопичення потрібних знань із цього середовища.

Для специфікації онтологій розроблена спеціальна мова відображення знань. Підмножина цієї мови служить і для формулювання запитів, а мова анотування – для «збагачення» Web-документів онтологічною інформацією. Всі ці компоненти обговорюються нижче.

Формалізм запитів

Формалізм запитів орієнтований на фреймове відображення онтологій, в межах якого, як і завжди, визначені поняття екземплярів, класів, атрибутів значень.

Схема O:C[A->>V] означає, що об'єкт O є екземпляром класу C з атрибутом A, що має значення V. Важливо, що в кожній позиції такої схеми можуть використовуватися не лише константи, але й змінні чи вирази. Для прикладу, запит вигляду

FORALL R <-R:Researcher передбачає пошук всіх об'єктів, що є екземплярами класу Researcher. Якщо припустити, що ідентифікатором об'єкту служить URL домашньої сторінки спеціаліста, як результат за цим запитом буде видано список відповідних Інтернет-посилань.

Зрозуміло, це елементарний запит. Зазвичай в запиті визначається пошуковий образ об'єктів, що мають певні властивості. Так, якщо необхідно знайти всіх спеціалістів на прізвище Іванов і при цьому видати як результат їх імена і електронні адреси, то наведений вище запит можна модифікувати наступним чином:

FORALL Obj, FN, EM <Obj :Researcher[firstName->>FN;
lastName->«Іванов»; email->EM].

Як відповіді при цьому можуть бути отримані значення змінних:

Obj = http://www.anywhere.ru/~ivanov/ FN = Іван

EM = mailto:ivanov@anywhere.ru

Є в мові Ontobroker і засоби виведення значень властивостей. Так, деякі з атрибутів об'єкту можуть задавати відношення, властивості яких відомі машині виведення. Для прикладу, в запиті вигляду

FORALL Obj.CP <Obj: Researcher[lastName ->>Іванов»; cooperatesWith->>CP], атрибут cooperatesWith є відношенням, що має властивість симетричності. Це означає, що

навіть якщо в об'єкта, що описує спеціаліста на прізвище Іванов, властивості cooperatesWith- ні, Ontobroker виведе його, якщо в онтології відображений об'єкт, що описує іншого спеціаліста, який має таку властивість зі значенням «Іванов».

У мові відображення онтологічних знань присутні й інші правила виведення значень атрибутів, експліцитно не відображених у Web-документах. При цьому зрозуміло, що мова запитів Ontobroker може використовуватись і для формування репозиторіїв з інформацією, що задовольняє задані обмеження. Більше того, за допомогою запитів можна отримувати й метаінформацію: запит вигляду

FORALL Att, T <— Researcher[Att=>>T]

поверне як результат імена всіх атрибутів класу Researcher і пов'язаних з ним класів.

У самій системі Ontobroker підтримуються два типи інтерфейсів при формуванні запитів – текстовий (для експертів) і графічний (для користувачів). Перший з них передбачає, що запити формуються безпосередньо у вхідній мові опису онтологій. Зрозуміло, що при цьому знання синтаксису мови Ontobroker і знайомство з онтологією, для якої формується запит, повинні бути наявні в експерта.

Проблема знання синтаксису вирішується в цьому випадку, як і в більшості інших інструментальних засобів нового покоління, за рахунок діалогів, що керують системою (system-driven dialogue). Користувачу видається відповідна панель, де можуть бути визначені (шляхом вибору з меню) компоненти запиту і зв'язки між ними. Такий підхід забезпечує синтаксичну коректність та однозначність інтерпретації запиту.

Складніше подолати розрив у знаннях експерта і користувача, особливо початківця, про онтологію, що використовується. Адже для правильного формулювання запиту потрібно знати, щонайменше, які концепти в онтології присутні та які атрибути наявні у концептів. Тому всі системи відображення онтологічних знань представляють своїм користувачам засоби візуалізації онтологій і навігації по онтології.

У Ontobroker для візуалізації онтологій використовується підхід, побудований на ідеях гіперболічної геометрії (Hyperbolic Geometry) [275]. У випадку системи, що обговорюється, ці ідеї реалізуються наступним чином: клас, що цікавить користувача в певний момент, подається «великим шаром», а класи, пов'язані безпосередньо з ним, – «маленькими шарами» і розміщуються вони по границі кола, «окреслюють» відповідний шар. Використовуючи цей інтерфейс, і експерт, і користувач можуть легко й ефективно включати у свій запит потрібні концепти та їх атрибути, а система Ontobroker перекладе їх у текстове відображення автоматично. Інтерфейс онтологій Ontobroker реалізований як Java-аплет, що забезпечує роботу з Web-броузерами на будь-яких платформах, де підтримується Java-технологія.

Формалізм подання і машина виведення

Як говорилося вище, онтологія визначається через концепти (класи), пов'язані відношеннями, атрибути й аксіоми. І адекватна мова відображення повинна забезпечувати зручні засоби для опису всіх перерахованих компонентів. У Ontobroker базисом відображення є так звані логіки фреймів (Frame-Logic) [269].

Базисними конструкціями у цьому підході є:

- підкласи (Subclassing) – запис C1 :: C2 означає, що клас C1 є підкласом C2;
- екземпляри (Instance of) – запис O : C означає, що O є екземпляром класу C;

- декларації атрибутів (Attribute Declaration) – запис C1 [A=>>C2] означає, що для екземпляру класу C1 визначений атрибут A, значення якого повинно бути екземпляром C2;
- значення атрибутів (Attribute Value) – запис O [A->>V] означає, що екземпляр O має атрибут A зі значенням V;
- частина-ціле (Part-of) – запис O1 <: O2 означає, що O1 є частиною O2;
- відношення (Relations) – предикати вигляду p (a1, a2) можуть використовуватися, як і у звичайних логічних формалізмах, але з тим розширенням, що як аргументи тут можуть виступати не лише терми, але й вирази.

З базисних конструкцій будуються складніші – факти (facts), правила (rules), «подвійні» правила (double rules) і запити (queries). Запити вже обговорювалися вище. Факти, за суттю, є елементарними виразами.

Правила, зазвичай, мають ліву і праву частини, причому ліва частина (тут вона називається «головою») є кон'юнкцією елементарних виразів, а права («тіло») – складна формула, термами якої є елементарні вирази, пов'язані звичайними предикатними символами типу **implies:** →, **implied by:** ←, **equivalent:** ↔, **AND, OR** і **NOT.** Різниця між звичайними і «подвійними» правилами в симетричності останніх. Важлива перевага формалізму – можливість використання змінних в «голові» правил (з квантором **FORALL**) чи в його «тілі» (з кванторами **FORALL і EXISTS).** Приклад фрагменту онтології в формалізмі Ontobroker, адаптований з роботи [237], наводиться нижче.

```
Визначення атрибутів
Person [firstName =>> STRING;
        lastName =>> STRING;
        eMail =>> STRING;
...
        publication =>> Publication].
Employee [affiliation =>> Organization;  ...].
Researcher [researchInterest =>> ResearchTopic;
            memberOf =>> ResearchGroup;
            cooperatesWith =>> Researcher].
Publication [ author =>> Person;
            title =>> STRING;
            year =>> NUMBER;
...         abstract =>> STRING].
Правила
FORALL Person1,  Person2
Person1:Researcher [cooperatesWith->> Person2] <-
Person2:Researcher [cooperatesWith->> Person1].
FORALL Person1,  Publication1
Publication1:Publication [author->> Person1] <->
Person1:Person [publication->> Publication1].
```

Напевне, пояснення тут потребують лише правила. Перше з них фіксує симетричність відношення cooperatesWith. Друге стверджує, що якщо конкретна особистість (екземпляр класу Person) має публікацію, то остання має автора, який теж є екземпляром класу Person, і навпаки.

Машина виведення Ontobroker складається з двох основних компонентів: транслятора з розширеної мови відображення в обмежений і власне обчислювача виразів обмеженої мови, яка є звичайною мовою логічного програмування.

Анотація Web-сторінок онтологічною інформацією

Оскільки, як вже зазначалося вище, Web-інформація частіше відображена мовою HTML, в рамках проекту

Ontobroker розроблено його просте розширення для анотації Web-сторінок. Основна ідея цього розширення полягає в наступному. В мову HTML додано декілька релевантних для вирішення поставлених задач тегів, використання яких дозволяє Ontobroker інтерпретувати анотовані фрагменти HTML-тексту як факти мови відображення онтологічних знань. При цьому Web-сторінки залишаються придатними для стандартних броузерів типу Netscape Navigator або MS Explorer.

У мову введені три епістемологічно різних примітиви:

- ідентифікація об'єкту, який може бути визначений як екземпляр визначеного класу, за допомогою URL;
- встановлення значення атрибуту об'єкту;
- визначення відношень між об'єктами.

Всі примітиви синтаксично розширяють тег <a ...> мови HTML. Так, наприклад, якщо спеціаліст Іванов захоче визначити себе як об'єкт онтології, що обговорювалася вище, він може на своїй домашній сторінці ввести конструкцію вигляду:

Тепер для об'єкту Іванов класу Researcher можна ввести атрибут email і його значення за допомогою наступної конструкції:

<a onto=" "http://www. anywhere.ru/~ivanov/"
[email]="mailto: ivanov@anywhere. ru"] ">

Аналогічно вводяться і відношення:

 ...

Наявні в мові і засоби, які забезпечують меншу складність анотування: наприклад, можливість називання «довгих» конструкцій і подальшого використання цих імен.

При такому підході Ontocrawler – компонент системи Ontobroker – простий CGI-скрипт, який періодично перевіряє анотовані сторінки на Web. Для пошуку таких сторінок він звертається до індексних сторінок провайдерів, які зареєстровані в рамках ініціативи (КА)².

9.5.3. Інші підходи і тенденції

На завершення цього розділу необхідно хоча б поверхнево розглянути зусилля World Wide Web Consortium (W3C) зі створення і впровадження засобів маркування Інтернет-ресурсів.

Донедавна в розпорядженні Інтернет-авторів для цього, майже виключно, використовувалася вже вказана вище мова HTML. Однак з точки зору семантичної розмітки Інтернет-документів ця мова має ряд недоліків, основними є такі:

- жорстка орієнтація на візуалізацію;
- єдина «точка зору» на дані;
- нерозширюваність;
- достатньо обмежені засоби специфікації семантичної структури документів.

Заради справедливості відзначимо, що ще в наприкінці 60-х років XX ст. в рамках досліджень з подання документів компанією IBM була розроблена мова SGML (Standard Generalized Markup Language), яка позбавлена багатьох із перерахованих недоліків. До середини 80-х років минулого століття ця мова стала стандартом для багатьох промислових компаній і керівних структур США, але, на думку спеціалістів робочої групи SGML W3C [215], вона надто складна для широкого використання Інтернет-авторами. Ось чому в рамках W3C, починаючи з 1996 року, докладаються зусилля для розроблення засобів розмітки документів, що порівнюються з потужністю SGML, а за простотою використання – з HTML. Серед робіт такого напряму, в першу чергу, відзначимо мову XML (extensible Markup Language).

У мові XML «знято» багато обмежень HTML, мова розмітки стала набагато потужнішою. І одночасно XML-тексти залишаються зрозумілими для всіх, хто працював з мовою HTML. Відмінні якості XML і в тому, що тут фіксується стандарт на визначення синтаксису й однотипність засобу введення в мови розмітки (Markup Language) нових тегів. А це, у свою чергу, дозволяє конструювати на основі XML нові мови маркування Web-документів і, крім того, забезпечує можливість різним додаткам (і в тому числі програмним агентам) «розуміти» й опрацювати XML-документи.

Кожний XML-документ має певну логічну і фізичну структуру. Фізично це композиція елементів, що називаються одиницями (entities), які можуть бути пов'язані взаємними посиланнями, документ складається з декларацій, одиниць, коментаріїв, самих текстів та інструкцій опрацювання, причому кожна конструкція XML маркується спеціальними тегами певним чином. Всі теги XML – парні, а конструкції можуть бути вкладені одна на одну, створюючи правильно побудоване дерево. Так, наприклад, конструкція <Item Attribute1 =«Value1»> </Item> визначає одиницю з іменем Item і списком пар атрибут-значення, який в нашому випадку поданий єдиним атрибутом з іменем Attribute1, що має значення «Value1».

Для ілюстрації можливостей цієї мови розглянемо змістовний приклад XML-документа, що описує домашню сторінку дослідника Іваненка.

```
<?xml version="1.0"?>
<Homepage>
 <Name>Домашня сторінка Іваненко</Name>
<Person>
<firstName>Ivan</firstName >
 <lastName>Ivanenko</lastName>
 <marriedTo Homepage="http://www.anywhere. ru">
Mariya  Ivanenko</marriedTo>
<employee  Homepage="http://www.ccas.ukr">
CCAS of Ukraine</employee>
 <publications>
     <book title="First Book"/>
    <book title="Second Book"/>
--------------------------------
               </publications>
               </Person>
</Homepage>
```


Цей XML-документ структурований значно краще, ніж аналогічний HTML-текст, але наразі не має «змісту», оскільки з нього не випливає, як інтерпретуються одиниці типу Person, publications, book і т.ін. Для вирішення цього питання використовується спеціальна специфікація визначення типу документа DTD (document type definition). За суттю, це граматика мови розмітки, в рамках якої визначається, які елементи можуть бути присутніми в документі, які атрибути вони мають і як елементи співвідносяться один з одним. Зрозуміло, що для стандарту XML такі специфікації вже розроблені самими авторами мови, але в нашому випадку використовується спеціальний його діалект, і тому саме ми повинні специфікувати DTD нашого документу. Така специфікація може бути наступною:

```
<! ELEMENT  Homepage  (Name,   Person)>
<! ELEMENT   Name  ( # PCDATA)>
<! ELEMENT Person  (firstName,   lastName,   marriedTo?,
employee?,   publications?,   Homepage?)>
<! ATTLIST  Person Homepage xml:link CDATA>
<! ELEMENT  firstName  ( #PCDATA)>
<! ELEMENT lastName  ( #PCDATA)>
<! ELEMENT marriedTo  (Person)>
<! ELEMENT employee  (organization)>
<! ATTLIST organization Homepage xml:link CDATA>
<! ELEMENT publications (book*,   paper*,   report*)>
<!  ATTLIST  book  title  CDATA    #REQUIRED,      coauthor
Person,publisher CDATA,   year CDATA)>
  <! ELEMENT paper (title,   coauthor*,   journal,   year,
vol?,number?)>
<! ELEMENT report    (title,    coauthor*,   organization,
year)
```

З наведеного опису випливає, що в DTD специфіковано «свідчення» конструкцій нашого XML-документа до стандартних XML-конструкцій, які є зрозумілими броузерам нового покоління.

На сьогодні вже розроблені DTD для різних предметних областей, і кожна така специфікація визначає нову мову розмітки. Відомим прикладом розвитку DTD для специфікації загальних ресурсів є RDF (Resource Description Framework), що розроблявся W3C. Цей формат може використовуватися для додавання в документи метаінформації, яка, в тому числі, може бути відображенням семантики документа.

Використання власних діалектів XML є важливим кроком на шляху формування просторів знань у середовищі Інтернет. Але це лише перший крок в цьому напрямі. Дійсно, які засоби дає мова XML для відображення знань? Очевидно, що це, в першу чергу, засоби специфікації декларативного компоненту розвинутих систем відображення знань. І то в обмеженому обсязі. Яким же чином автори цієї мови та її розширень передбачають підключення процедур опрацювання XML-конструкцій? На сьогодні в пропозиціях W3C явно прослідковується лише одна ідея: оскільки XML-документи не що інше, як пор-табельні дані, а мова Java має портабельний код, потрібно їх використовувати спільно. Для цього необхідні спеціальні інтерфейси, наприклад, SAX (Simple API for XML), які вже сьогодні можуть підтримувати багато Java-аналізаторів. Основна ідея тут достатньо проста – аналізатор переглядає вузли дерева документа, з XML-файла викликає відповідні

методи, визначені користувачем. Для того, щоб цей механізм працював, програміст повинен створити клас, що реалізує відповідний інтерфейс. Методи цього класу викликаються кожен раз, коли на вході розпізнавача з'являється потрібна конструкція (тег, вхідний рядок тощо). Саме опрацювання інформації при цьому цілком в руках програміста, а середовище лише підтримує загальне функціонування й опрацювання окремих ситуацій.

Такий підхід має багато спільного і з підходом Ontobroker, і з підходом SHOE. Автори обох цих проектів активно вітають зусилля W3C, але разом з тим відзначають, що в пропозиціях відповідних робочих груп ще багато недоліків. У першу чергу – це відсутність стандартів на інтелектуальне опрацювання XML-конструкцій, порівняно невеликий практичний досвід семантичної розмітки Інтернет-документів і достатньо обмежені засоби логічного опрацювання, використані при цьому.

От чому, як показує аналіз літератури й Інтернет-ресурсів за вказаною тематикою, сьогодні:

- ефективне опрацювання інформації на Web пов'язується, в першу чергу, з використанням ШІ-технологій;
- основні підходи в цій області орієнтовані на вирішення проблеми з видобування з Web-ресурсів експліцитних знань на основі семантичного маркування таких ресурсів;
- у всіх дослідницьких і багатьох комерційних проектах такого напряму активно використовуються (чи декларуються як використані) агентно-орієнтовані обчислення і технології.

Враховуючи вищесказане, в наступних розділах обговорюються системи інтелектуальних агентів і мультиагентні системи.

Запитання для повторення та контролю знань

1. Що таке онтологія?
2. Як задається формальна модель онтології?
3. Які бувають частинні випадки моделі онтологій?
4. Яка структура називається таксономічною?
5. Як класифікуються моделі онтологій?
6. Як задається формальна модель онтологічної системи?
7. Які компоненти входять у формальну модель онтологічної системи?
8. Відобразіть взаємозв'язок між онтологіями онтологічної системи.
9. Які Ви знаєте методології, що орієнтовані на побудову онтології? На яких принципах вони базуються?
10. Як класифікуються онтології?
11. Наведіть приклад онтології верхнього рівня.
12. Як онтологічні знання використовуються в Інтернеті?
13. Яка ціль міжнародного проекту (КА)2?
14. Що таке онтологічний інжиніринг?
15. Чому сьогодні ефективне опрацювання інформації в Інтернеті пов'язується, в першу чергу, з використанням інтелектуальних систем?

РОЗДІЛ 10

ІНТЕЛЕКТУАЛЬНІ АҐЕНТИ ТА МУЛЬТИАҐЕНТНІ СИСТЕМИ

- Аґенти і варіанти середовища
- Якісна поведінка: концепція раціональності.
- Визначення характеру середовища
- Структура агентів
- Основні поняття мультиагентних систем
- Аналіз сучасних досліджень у розробленнях мультиагентних систем
- Аналіз моделей, методів та алгоритмів, що використовуються у мультиагентних системах.
- Моделі мультиагентних систем

У цьому розділі дана груба класифікація варіантів середовища і показано, як властивості середовища впливають на проектування аґентів, що найбільш відповідають обраному середовищу. Тут описаний ряд основних, «скелетних» проектів аґентів. Почнемо з вивчення аґентів, варіантів середовища і зв'язків між ними. Спостерігаючи за тим, що деякі аґенти діють краще, ніж інші, можна цілком обґрунтовано висунути ідею раціонального аґента; таким є аґент, який діє настільки успішно, наскільки це можливо. Успіхи, яких може досягти аґент, залежать від характеру середовища.

10.1. Аґенти і варіанти середовища

Аґентом є все, що може розглядатися як таке, що сприймає своє середовище за допомогою давачів (сенсорів) і що впливає на це середовище за допомогою виконавчих механізмів. Ця проста ідея ілюструється на рис. 10.1. Людина, що розглядається в ролі аґента, має очі, вуха й інші органи чуття, а виконавчими механізмами для нього слугують руки, ноги, рот і інші частини тіла. Робот, що виконує функції аґента, як давачі може мати відеокамери й інфрачервоні далекоміри, а його виконавчими механізмами можуть бути різні двигуни. Програмне забезпечення, що виступає аґентом, вхідними сенсорними даними отримує коди натиснення клавіш, вміст файлів і мережеві пакети, а його дія на середовище виражається в тому, що програмне забезпечення виводить дані на екран, записує файли і передає мережеві пакети. Ми приймаємо загальне допущення, що кожен аґент може сприймати свої власні дії (але не завжди їх результати).

Ми використовуємо термін сприйняття для позначення отриманих аґентом сенсорних даних у будь-який конкретний момент часу. Послідовністю актів сприйняття аґента називається повна історія всього, що було коли-небудь ним сприйняте.

Загалом, вибір аґентом дії в будь-який конкретний момент часу може залежати від всієї послідовності актів сприйняття, що спостерігалися до цього моменту часу. Якщо

існує можливість визначити, яка дія буде вибрана аґентом у відповідь на будь-яку можливу послідовність актів сприйняття, то може бути дано більш-менш точне визначення аґента. Це рівносильне твердженню, що поведінка деякого аґента може бути описана за допомогою функції аґента, яка відображає будь-яку конкретну послідовність актів сприйняття на деяку дію.

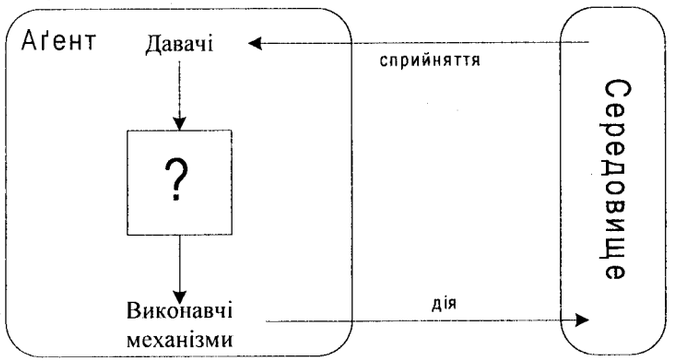


Рис. 10.1. Аґент взаємодіє із середовищем за допомогою давачів і виконавчих механізмів

Може розглядатися завдання табуляції функції аґента, яка описує будь-якого конкретного аґента; для більшості аґентів це була б дуже велика таблиця (фактично нескінченна), якщо не встановлюється межа довжини послідовностей актів сприйняття, які повинні враховуватися в таблиці. Проводячи експерименти з деяким аґентом, таку таблицю, в принципі, можна сконструювати, перевіряючи всі можливі послідовності актів сприйняття і реєструючи, які дії у відповідь виконує аґент. Така таблиця, безумовно, є зовнішнім описом аґента. Внутрішній опис полягає у визначенні того, яка функція аґента для цього штучного аґента реалізується за допомогою програми аґента. Важливо розрізняти два останні поняття. Функція аґента є абстрактним математичним описом, а програма аґента – це конкретна реалізація, що діє в рамках архітектури аґента.

Для ілюстрації викладених ідей скористаємося дуже простим прикладом: розглянемо показаний на рис. 10.2 світ, в якому працює порохотяг. Цей світ настільки простий, що існує можливість описати все, що в ньому відбувається; крім того, це світ, створений людиною, тому можна винайти безліч варіантів його організації. У такому конкретному світі є тільки два місцезнаходження: квадрати А і Б. Порохотяг, що виконує роль аґента, сприймає, в якому квадраті він знаходиться і чи є сміття в цьому квадраті. Аґент може вибрати такі дії, як перехід вліво, вправо, всмоктування сміття або бездіяльність. Одна з дуже простих функцій аґента полягає в наступному: якщо в поточному квадраті є сміття, то всмоктати його, інакше перейти в інший квадрат. Часткова табуляція такої функції аґента показана в табл. 10.1. Проста програма аґента для цієї функції аґента наведена нижче в цьому розділі, в лістингу 10.2.

Часткова табуляція функції простого аґента для світу порохотяга показана на рис. 10.2. На підставі табл. 10.1 можна зробити висновок, що для світу порохотяга можна визначати різних аґентів, заповнюючи різними способами правий стовпець цієї таблиці. Тому очевидне запитання полягає в наступному: «Який спосіб заповнення цієї таблиці є правильним?». Іншими словами: «Завдяки чому аґент стає хорошим або поганим, інтелектуальним або не відповідним критеріям інтелектуальності?». Відповідь на це запитання наведена в наступному підрозділі.

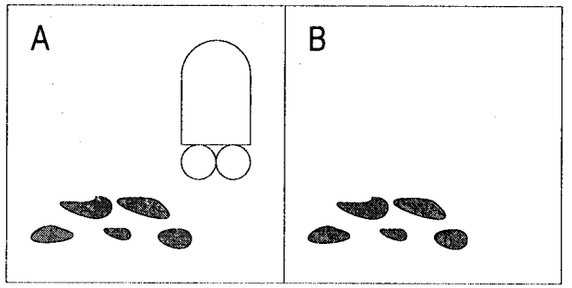


Рис. 10.2. Світ порохотяга, в якому є тільки два місцезнаходження

Відзначимо, що поняття аґента розглядається як інструмент для аналізу систем, а не як абсолютна класифікація, за якою світ ділиться на аґентів і неаґентів. Наприклад, як аґента можна було б розглядати кишеньковий калькулятор, який вибирає дію з відображенням «4» після отримання послідовності актів сприйняття «2 + 2 = ...», але подібний аналіз навряд чи допоможе зрозуміти роботу калькулятора.

Таблиця 10.1. Часткова табуляція функції простого аґента для світу порохотяга, наведеного на рис. 10.2

Послідовність актів сприйняття	Дія
[A,Clean]	*Right*
[A,Dirty]	*Suck*
[B,Clean]	*Left*
[B,Dirty]	*Suck*
[A,Clean], [A,Clean]	*Right*
[A,Clean], [A,Dirty]	*Suck*
...	...
[A,Clean], [A,Clean], [A,Clean]	*Right*
[A,Clean], [A,Clean], [A,Dirty]	*Suck*
...	...

10.2. Якісна поведінка: концепція раціональності

Раціональним аґентом є такий аґент, який виконує правильні дії; висловлюючись формальніше, таким є аґент, в якому кожен запис у таблиці для функції аґента заповнений правильно. Очевидно, що виконання правильних дій краще, ніж здійснення неправильних дій, але що розуміється під виразом «виконання правильних дій»? У першому наближенні можна сказати, що правильною дією є така дія, яка забезпечує найбільш успішне функціонування аґента. Тому потрібний певний спосіб вимірювання успіху. Критерії успіху, разом з описом середовища, а також давачів і виконавчих механізмів аґента, надають повну специфікацію завдання, з яким стикається аґент. Маючи ці компоненти, ми можемо визначити точніше, що розуміється під словом «раціональний».

Показники продуктивності

Показники продуктивності втілюють у собі критерії оцінки успішної поведінки аґента. Після занурення в середовище аґент виробляє послідовність дій, відповідних отриманим ним сприйняттям. Ця послідовність дій змушує середовище пройти через

послідовність станів. Якщо така послідовність відповідає очікуваному результату, то агент функціонує добре. Безумовно, що не може бути одного постійного показника, відповідного для всіх агентів. Можна було б дізнатися у агента його суб'єктивну думку про те, наскільки він задоволений своєю власною продуктивністю, але деякі агенти не будуть здатні відповісти, а інші схильні займатися самообманом. Тому необхідно наполегливо добиватися застосування об'єктивних показників продуктивності, і, як правило, проектувальник, що конструює агента, передбачає такі показники.

Розглянемо агента-порохотяга, описаного в попередньому підрозділі. Можна було б запропонувати вимірювати показники продуктивності за об'ємом сміття, прибраного за одну восьмигодинну зміну. Але, безумовно, маючи справу з раціональним агентом, Ви отримуєте те, що просите. Раціональний агент може максимізувати такий показник продуктивності, прибираючи сміття, потім висипаючи його на підлогу, потім знову прибираючи, і т. д. Тому прийнятніші критерії продуктивності повинні винагороджувати агента за те, що підлога залишається чистою. Наприклад, одне очко могло б присуджуватися за кожен чистий квадрат в кожному інтервалі часу (можливо, у поєднанні зі штрафом за споживану електроенергію і створюваний шум). Як загальне правило вкажемо, що краще за все розробляти показники продуктивності відповідно до того, чого дійсно необхідно добитися в заданому середовищі, а не відповідно до того, як, на думку проектувальника, повинен поводитися агент.

Задача вибору показників продуктивності не завжди є простою. Наприклад, поняття «Чистої підлоги», яке розглядалося вище, засноване на визначенні усередненої чистоти підлоги в часі. Але необхідно також враховувати, що одна і та ж чистота може бути досягнута двома різними агентами, один з яких постійно, але неквапливо виконує свою роботу, а інший час від часу енергійно займається очищенням, але робить довгі перерви. Може здатися, що визначення того способу дій, який в певному випадку має перевагу, належить до тонкощів домогосподарства, але фактично це глибоке філософське питання з далекосяжними наслідками. Що краще – відчайдушне життя зі зльотами й падіннями чи безпечне, але одноманітне існування? Що краще – економіка, в якій кожен живе в помірній бідності, чи така економіка, в якій одним нічого не бракує, а інші ледве зводять кінці з кінцями? Залишаємо завдання пошуку відповідей на ці запитання як вправу для допитливого читача.

Раціональність

У будь-який конкретний момент часу оцінка раціональності дій агента залежить від чотирьох перерахованих нижче чинників:

- показники продуктивності, які визначають критерії успіху;
- знання агента про середовище, набуті раніше;
- дії, які можуть бути виконані агентом;
- послідовність актів сприйняття агента, які відбулися раніше.

З урахуванням цих чинників можна сформулювати наступне визначення раціонального агента.

Для кожної можливої послідовності актів сприйняття раціональний агент повинен вибрати дію, яка, як очікується, максимізувала б його показники продуктивності, з урахуванням фактів, наданих певною послідовністю актів сприйняття і всіх вбудованих знань, якими володіє агент.

Розглянемо приклад простого агента-порохотяга, який очищає квадрат, якщо в ньому є сміття, і переходить в інший квадрат, якщо сміття в ньому немає; результати часткової табуляції такої функції агента наведені в табл. 9.1. Чи є цей агент раціональним? Відповідь на це запитання не така вже й проста. Спочатку потрібно визначити, в чому полягають показники продуктивності, що відомо про середовище, і які давачі й виконавчі механізми має агент. Приймемо перераховані нижче припущення.

- Вживані показники продуктивності передбачають винагороду в одне очко за кожен чистий квадрат в кожному інтервалі часу протягом «терміну існування» агента, що складається з 1000 інтервалів часу.
- «Географія» середовища відома наперед (див. рис. 10.2), але розподіл сміття і початкове місцезнаходження агента не визначені. Чисті квадрати залишаються чистими, а всмоктування сміття приводить до очищення поточного квадрата. Дії Left і Right приводять до переміщення агента відповідно вліво і вправо, за винятком тих випадків, коли вони могли б вивести агента за межі середовища, і в цих випадках агент залишається там, де він перебуває.
- Єдиними доступними діями є Left, Right, Suck (всмоктати сміття) і NoOp (нічого не робити).
- Агент правильно визначає своє місцезнаходження і сприймає свідчення давача, що дозволяють дізнатися, чи є сміття в цьому місці.

Автори стверджують, що в цих обставинах агент дійсно є раціональним; його очікувана продуктивність щонайменше не нижча, ніж у будь-яких інших агентів.

Можна легко уявити, що в інших обставинах той самий агент може стати нераціональним. Наприклад, після того, як все сміття буде прибране, агент почне здійснювати непотрібні періодичні переміщення вперед і назад; якщо показники продуктивності передбачають штраф в одне очко за кожне пересування в тому або іншому напрямку, то агент не зможе добре заробляти. У такому разі кращий агент повинен був би нічого не робити доти, поки він упевнений в тому, що всі квадрати залишаються чистими. Якщо ж чисті квадрати можуть знову стати брудними, то агент зобов'язаний час від часу здійснювати перевірку і знову очищати їх за потребою. А якщо географія середовища невідома, то агентові можливо потрібно досліджувати її, а не обмежуватися квадратами A і B.

Всезнайність, навчання і автономність

Потрібно відрізняти раціональність і всезнайність. Всезнайний агент знає фактичний результат своїх дій і може діяти відповідним чином; але всезнаючість насправді неможлива. Розглянемо такий приклад: якийсь пан одного разу прогулюється в Парижі Єлісейськими Полями і бачить на іншій стороні вулиці старого друга. Поблизу немає жодних машин, а наш пан нікуди не поспішає, тому, будучи раціональним агентом, він починає переходити через дорогу. Тим часом, на висоті 10 000 метрів у літака, що пролітає, відриваються двері вантажного відсіку, і перш ніж нещасний встигає досягти іншої сторони вулиці, двері розплющують його. Чи було нераціональним саме те, що цей пан вирішив перейти на іншу сторону вулиці? Малоймовірно, що в його некролозі написали б: «Жертва ідіотської спроби перейти вулицю».

Цей приклад показує, що раціональність не можна розглядати як рівнозначну досконалості. Раціональність – це максимізація очікуваної продуктивності, а досконалість – максимізація фактичної продуктивності. Відмовляючись від прагнення до досконалості,

ми не тільки застосовуємо до агентів справедливі критерії, але і враховуємо реальність. Річ у тому, що якщо від агента вимагають, щоб він виконував дії, які виявляються найкращими після їх здійснення, то завдання проектування агента, що відповідає цій специфікації, стає нездійсненним (принаймні доти, поки ми не зможемо підвищити ефективність машин часу або кришталевих куль, вживаних ворожками).

Тому наше визначення раціональності не вимагає усезнання, адже раціональний вибір залежить тільки від послідовності актів сприйняття, сформованої до заданого моменту. Необхідно також стежити за тим, щоб ми ненавмисно не дозволили агентові брати участь в діях, які, безумовно, не є інтелектуальними. Наприклад, якщо агент не озирається вліво і вправо, перш ніж перетнути дорогу з інтенсивним рухом, то отримана ним досі послідовність актів сприйняття не зможе підказати, що до нього на великій швидкості наближається величезна вантажівка. Чи вказує наше визначення раціональності, що тепер агент може перейти через дорогу? Ні. По-перше, агент не був би раціональним, якби спробував перейти на іншу сторону, отримавши таку неінформативну послідовність актів сприйняття: ризик нещасного випадку при подібній спробі перейти автомагістраль, не озирнувшись, дуже великий. По-друге, раціональний агент повинен вибрати дію «озирнутися», перш ніж ступити на дорогу, оскільки такий огляд дозволяє максимізувати очікувану продуктивність. Виконання в цілях модифікації майбутніх сприйнять певних дій (іноді званих збиранням інформації) складає важливу частину раціональності. Другий приклад збирання інформації виражається в тому досліджені ситуації, яке має бути зроблене агентом-порохотягом у середовищі, яке спочатку було для нього невідомим.

Наше визначення вимагає, щоб раціональний агент не тільки збирав інформацію, але також навчався максимально можливо на тих даних, які він сприймає. Початкова конфігурація агента може відображати деякі попередні знання про середовище, але, у міру придбання агентом досвіду, ці знання можуть модифікуватись і поповнюватись. Існують крайні випадки, в яких середовище повністю відоме наперед. У подібних випадках агентові не потрібно сприймати інформацію або навчатися; він просто відразу діє правильно. Безумовно, такі агенти є вельми вразливими. Розглянемо скромного гнійного жука. Викопавши кубло і відклавши яйця, він скачує кульку гною, набравши його з найближчої гнійної купи, щоб заткати вхід у кубло. Якщо кулька гною буде видалена безпосередньо перед тим, як жук його схопить, жук продовжує маніпулювати нею і зображає таку пантоміму, неначе він затикає кубло неіснуючою кулькою гною, навіть не помічаючи, що ця кулька відсутня. У результаті еволюції поведінка цього жука була сформована на підставі певного припущення, а якщо це припущення порушується, то за цим слідує безуспішна поведінка. Небагато інтелектуальнішими є оси-сфекси. Самка сфекса викопує нірку, виходить з неї, жалить гусеницю і затягає її в нірку, потім знову виходить з нірки, щоб перевірити, чи все в порядку, витягає гусеницю назовні й відкладає в неї яйця. Гусениця слугує джерелом живлення під час розвитку яєць. Досі все йде добре, але якщо ентомолог перемістить гусеницю на декілька дюймів убік, поки сфекс виконує свою перевірку, ця комаха знову повертається до етапу «перетягання» свого плану і продовжує виконувати план без змін, навіть після десятків втручань у процедуру переміщення гусениці. Оса-сфекс нездатна навчитися діяти в такій ситуації, коли її природжений план порушується, і тому не може його змінити.

У агентах, що успішно діють, завдання обчислення функції агента розбивається на три окремі періоди: при проектуванні агента деякі обчислення здійснюються його проектувальниками; додаткові обчислення агент здійснює, вибираючи одну зі своїх чергових дій; а у міру того, як агент вчиться на підставі досвіду, він здійснює інші допоміжні обчислення для ухвалення рішення про те, як модифікувати свою поведінку.

Якщо ступінь, в якому агент покладається на апріорні знання свого проектувальника, а не на свої сприйняття, дуже високий, то такий агент розглядається як такий, що володіє недостатньою автономністю. Раціональний агент має бути автономним – він повинен навчатися всьому, що може засвоїти, для компенсації неповних або неправильних апріорних знань. Наприклад, агент-порохотяг, який навчається прогнозуванню того, де і коли з'явиться додаткове сміття, безумовно, працюватиме краще, ніж той агент, який на це не здатний. На практиці агентові рідко виставляється вимога, щоб він був повністю автономним із самого початку: якщо агент має мало досвіду або взагалі не має досвіду, то вимушений діяти випадковим чином, якщо проектувальник не надав йому певної допомоги. Тому, як і еволюція надала тваринам достатню кількість природжених рефлексів, що дозволяють їм прожити після народження настільки довго, щоб встигнути навчитися самостійно, так і штучному інтелектуальному агентові було б розумно надати деякі початкові знання, а не тільки наділити його здатністю навчатися. Після достатнього досвіду існування у своєму середовищі поведінка раціонального агента може за суттю стати незалежною від його апріорних знань. Тому включення в проект здібності до навчання дозволяє проектувати простих раціональних агентів, які можуть діяти успішно у різноманітних варіантах середовища.

10.3. Визначення характеру середовища

Тепер, після визначення раціональності, ми майже готові приступити до створення раціональних агентів. Але спочатку необхідно визначити, чим є проблемне середовище, що по суті є «проблемою», для якої раціональний агент служить «рішенням». Почнемо з демонстрації того, як визначити проблемне середовище, і проілюструємо цей процес на ряді прикладів. Потім в цьому підрозділі буде показано, що проблемне середовище може мати цілий ряд різновидів. Вибір проекту, найбільш відповідного для програми конкретного агента, безпосередньо залежить від заданого різновиду проблемного середовища.

Визначення проблемного середовища

У наведеному вище дослідженні раціональності простого агента-порохотяга нам довелося визначити показники продуктивності, середовище, а також виконавчі механізми і давачі агента. Згрупуємо опис усіх цих чинників під заголовком проблемне середовище. Для тих, хто любить абревіатури, автори скорочено позначили відповідний опис як PEAS (Performance, Environment, Actuators, Sensors – продуктивність, середовище, виконавчі механізми, давачі). Перший етап проектування будь-якого агента завжди повинен полягати у визначенні проблемного середовища з найбільш можливою повнотою.

Приклад, в якому розглядався світ порохотяга, був нескладним; тепер розглянемо складнішу проблему – створення автоматизованого водія таксі. Цей приклад використовуватиметься далі у цьому підрозділі. Перш ніж читач відчує тривогу за безпеку майбутніх пасажирів, хочемо відразу ж відзначити, що завдання створення повністю автоматизованого водія таксі на сьогодні все ще виходить за межі можливостей наявної технології. Повне рішення проблеми водіння автомобіля є звичайно трудомістким,

оскільки немає межі появі все нових і нових комбінацій обставин, які можуть виникати у процесі водіння; це ще одна з причин, через яку ми обрали цю проблему для обговорення. У табл. 10.2 наведений підсумковий опис PEAS для проблемного середовища водіння таксі. Кожний з елементів цього опису розглядається далі детальніше.

Таблиця 10.2. Опис PEAS проблемного середовища для автоматизованого водія таксі

Тип агента	Показники продуктивності	Середовище	Виконавчі механізми	Давачі
Водій таксі	Безпечна, швидка, комфортна їзда в рамках правил дорожнього руху, максимізація прибутків	Дороги, інші транспортні засоби, пішоходи, клієнти	Засоби керування, акселератор, тормоз, клаксон, дисплей	Відеокамери, спідометр, одометр, акселерометр, давачі двигуна

Перш за все необхідно визначити показники продуктивності, якими ми могли б стимулювати діяльність нашого автоматизованого водія. До бажаних якостей належить успішне досягнення потрібного місця призначення; мінімізація споживання палива, зносу і старіння; мінімізація тривалості і/або вартості поїздки; мінімізація кількості порушень правил дорожнього руху і перешкод іншим водіям; максимізація безпеки і комфорту пасажирів; максимізація прибуття. Безумовно, що деякі з цих цілей конфліктують, тому повинні розглядатися можливі компроміси.

Потім розглянемо, в чому полягає середовище водіння, в якому діє таксі. Будь-якому водієві таксі доводиться мати справу із різними дорогами, починаючи з путівців і вузьких міських провулків і закінчуючи автострадами з дванадцятьма смугами руху. На дорозі зустрічаються інші транспортні засоби, безпритульні тварини, пішоходи, робочі, що здійснюють дорожні роботи, поліцейські автомобілі, калюжі і вибоїни. Водієві таксі доводиться також мати справу з потенційними і дійсними пасажирами. Крім того, є ще декілька важливих додаткових чинників. Таксистові може випасти доля працювати в Південній Каліфорнії, де рідко виникає така проблема, як сніг, або на Алясці, де снігу на дорогах не буває дуже рідко.

Може статися, що водієві все життя доведеться їздити правою стороною або від нього може вимагатися, щоб він зумів достатньо успішно пристосуватися їзди по лівою стороною під час перебування у Британії або в Японії. Безумовно, чим обмеженішим є середовище, тим простіше завдання проектування.

Виконавчі механізми, наявні в автоматизованому таксі, повинні бути більшою чи меншою мірою такими ж, як і ті, які перебувають у розпорядженні водія-людини: засоби керування двигуном за допомогою акселератора і засобу керування водінням за допомогою керма і гальм. Крім того, для нього можуть знадобитися засоби виведення на екран дисплея або синтезу мови для передавання повідомлень у відповідь пасажирам і, можливо, певні способи спілкування з водіями інших транспортних засобів, іноді ввічливого, а іноді й не зовсім.

Для досягнення своєї мети в такому середовищі водіння таксистові потрібно буде знати, де він перебуває, хто ще їде цією дорогою, і з якою швидкістю рухається він сам. Тому до числа його основних давачів повинні входити одна або декілька керованих телевізійних камер, спідометр і одометр. Для правильного керування автомобілем, особливо на поворотах, у ньому повинен передбачатися акселерометр; водієві буде потрібно

також знати механічний стан автомобіля, тому йому буде потрібний звичайний набір давачів для двигуна і електричної системи. Автоматизований водій може також мати прилади, недоступні для середнього водія-людини: супутникову глобальну систему навігації та визначення положення (Global Positioning System – GPS) для отримання точної інформації про місцезнаходження по відношенню до електронної карти, а також інфрачервоні або ультразвукові давачі для вимірювання відстаней до інших автомобілів і перешкод. Нарешті, йому будуть потрібні клавіатура або мікрофон для пасажирів, щоб вони могли вказати місце свого призначення.

Деякі програмні аґенти (звані також програмними роботами, або софтботами) існують у складних, необмежених проблемних областях. Уявіть собі програмний робот, призначений для керування тренажером, що імітує великий пасажирський літак. Цей тренажер є ретельно промодельованим, складним середовищем, в якому імітуються рухи інших літаків і робота наземних служб, а програмний аґент повинен вибирати в реальному часі найбільш доцільні дії із широкого діапазону дій.

Ще одним прикладом може слугувати програмний робот, призначений для переглядання джерел новин в Internet і показу клієнтам повідомлень, що цікавлять їх. Для успішної роботи йому потрібні певні здібності до опрацювання тексту на природній мові, він повинен у процесі навчання визначати, що цікавить кожного замовника, а також уміти змінювати свої плани динамічно, припустимо, коли з'єднання з яким-небудь із джерел новин переривається або в оперативний режим переходить нове джерело новин. Internet є середовищем, яке за своєю складністю змагається з фізичним світом, а до числа об'єктів цієї мережі входить багато штучних аґентів.

- **Властивості проблемного середовища**

Поза сумнівом, що різноманітність варіантів проблемного середовища, які можуть виникати у штучному інтелекті, вельми велика. Проте існує можливість визначити відносно невелику кількість вимірювань, за якими можуть бути класифіковані варіанти проблемного середовища. Ці вимірювання у значній мірі визначають найбільш прийнятний проект аґента і застосовність кожного з основних сімейств методів для реалізації аґента. Спочатку в цьому розділі буде наведений список вимірювань, а потім проаналізовано декілька варіантів проблемного середовища для ілюстрації цих ідей. Наведені тут визначення є неформальними; точніші твердження і приклади варіантів середовища кожного типу описані в наступних підрозділах.

- **Повністю спостережуване або частково спостережуване середовище**

Якщо давачі аґента надають йому доступ до повної інформації про стан середовища в кожен момент часу, то таке проблемне середовище називається повністю спостережуваним. За суттю, проблемне середовище є повністю спостережуваним, якщо давачі виявляють всі дані, які є релевантними для вибору аґентом дії; релевантність, у свою чергу, залежить від показників продуктивності. Повністю спостережувані варіанти середовища є зручними, оскільки аґентові не потрібно підтримувати який-небудь внутрішній стан для того, щоб бути в курсі того, що відбувається у світі. Середовище може виявитися частково спостережуваним через створюваний шум і неточність сенсорів або через те, що окремі характеристики його стану просто відсутні в інформації, отриманій від давачів; наприклад, аґент-порохотяг, в якому є тільки локальний давач сміття, не може визначити, чи є сміття в інших квадратах, а автоматизований водій таксі не має відомостей про те, які маневри мають намір виконати інші водії.

♦ **Детерміноване або стохастичне середовище**

Якщо наступний стан середовища повністю визначається поточним станом і дією, вчинено аґентом, то таке середовище називається детермінованим; інакше воно є стохастичним. У повністю спостережуваному детермінованому середовищі аґентові не доводиться діяти в умовах невизначеності. Але якщо середовище частково спостережуване, то може створитися враження, що воно є стохастичним. Це частково справедливо, якщо середовище складне і аґентові нелегко стежити за всіма його неспостережуваними аспектами. У зв'язку з цим часто буває зручніше класифікувати середовище як детерміноване або стохастичне з погляду аґента. Очевидно, що при такому трактуванні середовище водіння таксі є стохастичним, оскільки ніхто не може точно передбачити поведінку всіх інших транспортних засобів; більш того, в будь-якому автомобілі абсолютно несподівано може відбутися проколювання шини або зупинка двигуна.

Описаний тут світ порохотяга є детермінованим, але інші варіанти цього середовища можуть включати стохастичні елементи, такі як випадкова поява сміття і ненадійна робота механізму всмоктування. Якщо середовище є детермінованим в усіх відношеннях, окрім дій інших аґентів, то автори цієї книги називають це середовище стратегічним.

♦ **Епізодичне або послідовне середовище**

В епізодичному проблемному середовищі досвід аґента складається з нерозривних епізодів. Кожен епізод включає сприйняття середовища аґентом, а потім виконання однієї дії. При цьому вкрай важливе те, що наступний епізод не залежить від дій, вчинених в попередніх епізодах. У епізодичних варіантах середовища вибір дії в кожному епізоді залежить тільки від самого епізоду. Епізодичними є багато завдань класифікації. Наприклад, аґент, який повинен розпізнавати дефектні деталі на складальній лінії, формує кожне рішення стосовно поточної деталі незалежно від попередніх рішень; більш того, від поточного рішення не залежить, чи буде визначена як дефектна наступна деталь. З іншого боку, в послідовних варіантах середовища поточне рішення може вплинути на всі майбутні рішення. Послідовними є такі завдання, як гра в шахи і водіння таксі: у обох випадках короткочасні дії можуть мати довготривалі наслідки. Епізодичні варіанти середовища набагато простіші в порівнянні з послідовними, оскільки в них аґентові не потрібно думати наперед.

♦ **Статичне або динамічне середовище**

Якщо середовище може змінитися в ході того, як аґент вибирає чергову дію, то таке середовище називається динамічним для цього аґента; інакше воно є статичним. Діяти в умовах статичного середовища простіше, оскільки аґентові не потрібно спостерігати за світом у процесі вироблення рішення про виконання чергової дії, до того ж аґентові не доводиться турбуватися про те, що він витрачає на роздуми дуже багато час. Динамічні варіанти середовища, з іншого боку, як би безперервно питають аґента, що він збирається робити, а якщо він ще нічого не вирішив, то це розглядається як рішення нічого не робити. Якщо з часом саме середовище не змінюється, а змінюються показники продуктивності аґента, то таке середовище називається напівдинамічним. Очевидно, що середовище водіння таксі є динамічним, оскільки інші автомобілі й саме таксі продовжують рух і під час того, як алгоритм водіння визначає, що робити далі. Гра в шахи з контролем часу є напівдинамічною, а завдання розв'язування кросворду – статичною.

♦ **Дискретне або безперервне середовище**

Відмінність між дискретними і безперервними варіантами середовища може відноситися до стану середовища, способу обліку часу, а також сприйнять і дій аґента. На-

приклад, таке середовище з дискретними станами, як гра в шахи, має кінцеву кількість помітних станів. Крім того, гра в шахи пов'язана з дискретною множиною сприйнять і дій. Водіння таксі – це проблема з безперервно змінним станом і безперервно плинним часом, оскільки швидкість і місцезнаходження самого таксі й інших транспортних засобів змінюються в певному діапазоні безперервних значень, причому ці зміни відбуваються в часі плавно. Дії по водінню таксі також є безперервними (безперервне регулювання кута повороту кермом тощо.). Строго кажучи, вхідні дані від цифрових камер надходять дискретно, але зазвичай вони розглядаються як дані, що задають неперервність зміни швидкості та місцезнаходження.

♦ **Одноаґентне або мультиаґентне середовище**

Відмінність між одноаґентними і мультиаґентними варіантами середовища на перший погляд може здатися достатньо простою. Наприклад, очевидно, що аґент, який самостійно розв'язує кросворд, перебуває в одноаґентному середовищі, а аґент, що грає в шахи, діє у двохаґентному середовищі. Проте при аналізі цієї класифікаційної ознаки виникають деякі нюанси. Перш за все, вище було описано, на якій підставі деяка сутність може розглядатися як аґент, але не було вказано, яка сутність повинна розглядатися як аґент. Чи повинен аґент А (наприклад, водій таксі) вважати аґентом об'єкт В (інший автомобіль), чи може відноситися до нього просто як до об'єкту, що стохастично діє, який можна порівняти із хвилями, що набігають на берег, або з листям, що тріпоче на вітрі? Ключова відмінність полягає в тому, чи слід або не слід описувати поведінку об'єкту В як максимізацію особистих показників продуктивності, значення яких залежать від поведінки аґента А. Наприклад, у шахах сутність В, що прагне максимізувати свої показники продуктивності, а це, за правилами шахів, приводить до мінімізації показників продуктивності аґента А. Отже, шахи – це конкурентне мультиаґентне середовище. А в середовищі водіння таксі, з іншого боку, запобігання зіткненням максимізувало показники продуктивності всіх аґентів, тому вона може слугувати прикладом частково кооперативного мультиаґентного середовища. Воно є також частково конкурентним, оскільки, наприклад, паркувальний майданчик може зайняти тільки один автомобіль.

Проблеми проектування аґентів, що виникають у мультиаґентному середовищі, часто повністю відрізняються від тих, з якими доводиться стикатися в одноаґентних варіантах середовища; наприклад, однією з ознак раціональної поведінки в мультиаґентному середовищі часто буває підтримка зв'язку, а в деяких варіантах частково спостережуваного конкурентного середовища раціональною стає стохастична поведінка, оскільки вона дозволяє уникнути пасток передбаченості.

Як і слід чекати, найскладнішими варіантами середовища є частково спостережувані, стохастичні, послідовні, динамічні, безперервні й мультиаґентні. Крім того, часто виявляється, що багато реальних ситуацій є настільки складними, що незрозуміло навіть, чи дійсно їх можна вважати детермінованими. З практичної точки зору їх треба розглядати як стохастичні. Проблема водіння таксі є складною у всіх вказаних відношеннях.

У табл. 10.3 перераховані властивості багатьох відомих варіантів середовища. Варто зазначити, що в окремих випадках наведені в ній описи є дуже короткими і скупими. Наприклад, у ній вказано, що шахи – це повністю спостережуване середовище, але строго кажучи, це твердження є помилковим, оскільки деякі правила, що стосуються рокіровки, узяття пішака на проході і оголошення нічиї при повторенні ходів, вимагають запам'ятовування певних фактів про історію гри, які не можна виявити з аналізу позиції на дошці.

Але ці виключення з визначення спостережливості, безумовно, є незначними в порівнянні з тими ситуаціями, з якими стикається автоматизований водій таксі, інтерактивна система викладання англійської мови або медична діагностична система.

Табл. 10.3. Приклади варіантів проблемного середовища і їх характеристик

Проблемне середовище	Спостережувана повністю або частково	Детермінована, стратегічна або стохастична	Епізодична або послідовна	Статична, динамічна або напів-динамічна	Дискретна або неперервна	Одноагентна або мультиагентна
Розв'язування кросворда	Повністю	Детермінована	Послідовна	Статистична	Дискретна	Одноагентна
Гра в шахи з контролем часу	Повністю	Стохастична	Послідовна	Напівдинамічна	Дискретна	Мультиагентна
Гра в покер	Частково	Стохастична	Послідовна	Статистична	Дискретна	Мультиагентна
Гра в нарди	Повністю	Стохастична	Послідовна	Статистична	Дискретна	Мультиагентна
Водіння таксі	Частково	Стохастична	Послідовна	Динамічна	Неперервна	Мультиагентна
Медицинська діагностика	Частково	Стохастична	Послідовна	Динамічна	Неперервна	
Аналіз зображень	Повністю	Детермінована	Епізодична	Напівдинамічна	Неперервна	Одноагентна
Робот-сортувальник деталей	Частково	Стохастична	Епізодична	Динамічна	Неперервна	Одноагентна
Контролер очисних споруд	Частково	Стохастична	Послідовна	Динамічна	Неперервна	Одноагентна
Інтерактивна програма, що навчає англійську мову	Частково	Стохастична	Послідовна	Динамічна	Дискретна	Мультиагентна

Деякі інші відповіді в цій таблиці залежать від того, як визначено проблемне середовище. Наприклад, в ній завдання медичного діагнозу визначене як одноагентне, оскільки сам процес розвитку захворювання у пацієнта недоцільно моделювати як агента, але системі медичної діагностики іноді доводиться стикатися з пацієнтами, що не хочуть приймати її рекомендації, а також зі скептично настроєним персоналом, тому її середовище може мати мультиагентний аспект. Крім того, медична діагностика є епізодичною, якщо вона розглядається як завдання вибору діагнозу на основі аналізу переліку симптомів, але ця проблема стає послідовною, якщо розв'язувана при цьому задача може включати вироблення рекомендацій щодо виконання ряду лабораторних досліджень, оцінку прогресу в ході лікування і т.ін. До того ж, багато варіантів середовища є епізодичними на вищих рівнях у порівнянні з окремими діями агента. Наприклад, шаховий турнір складається з ряду ігор; кожна гра є епізодом, оскільки (загалом) від ходів, зроблених у попередній грі, не залежить те, як вплинуть на загальну продуктивність агента ходи, зроблені ним у поточній грі. З іншого боку, ухвалення рішень в одній і тій самій грі, безумовно, відбувається послідовно.

Експерименти часто застосовуються не до одного варіанту середовища, а до багатьох варіантів, сформованих на основі деякого класу варіантів середовища. Наприклад, щоб оцінити дії водія таксі в модельованій ситуації дорожнього руху, може потрібно буде здійснити багато сеансів моделювання з різними умовами трафіку, освітлення і погоди. Якби ми спроектували цього агента для одного сценарію, то могли б краще скористатися специфічними властивостями заданого конкретного випадку, але не мали б можливості визначити прийнятний проект розв'язування задачі автоматизованого водіння в цілому. З цієї причини репозиторій коду включає також генератор варіантів середовища для кожного класу варіантів середовища; цей генератор обирає певні варіанти середовища (з деякою вірогідністю), в яких виконується перевірка агента. Наприклад, генератор варіантів середовища порохотяга ініціалізував випадковим чином такі початкові дані, як розподіл сміття і місцезнаходження агента. Річ у тому, що найбільший інтерес викликає те, яку середню продуктивність матиме даний конкретний агент у деякому класі варіантів середовища. Раціональний агент для певного класу варіантів середовища максимізував свою середню продуктивність.

10.4. Структура агентів

Досі властивості агентів розглядалися на підставі аналізу їх поведінки — дій, що виконуються агентом після отримання будь-якої заданої послідовності актів сприйняття. Тепер нам мимоволі доведеться змінити тему і перейти до опису того, як організовано їх внутрішнє функціонування. Завдання штучного інтелекту полягає в розробленні програми агента, яка реалізує функцію агента, відображаючи сприйняття дій. Передбачається, що ця програма повинна працювати в обчислювальному пристрої з фізичними давачами і виконавчими механізмами; в цілому ці компоненти іменуються архітектурою, а структура агента умовно позначається такою формулою:

Агент = Архітектура + Програма.

Очевидно, що вибрана програма повинна бути відповідною для цієї архітектури. Наприклад, якщо в програмі здійснюється вироблення рекомендацій з виконання таких дій, як Walk (ходіння), то в архітектурі доцільно передбачити використання опорно-рухового апарату. Архітектура може бути звичайним персональним комп'ютером або може бути втілена у вигляді автомобіля з декількома бортовими комп'ютерами, відеокамерами й іншими давачами. Загалом, архітектура забезпечує передавання у програму результатів сприйняття, отриманих від давачів, виконання програми і передавання виконавчим механізмам варіантів дій, вибраних програмою, в міру їх вироблення.

Програми агентів

Агенти мають такі програми: вони приймають від давачів як вхідні дані результати поточного сприйняття і повертають виконавчим механізмам вибраний варіант дії. Необхідно вказати на відмінність між програмою агента, яка приймає як вхідні дані результати поточного сприйняття, і функцією агента, яка приймає на вході всю історію актів сприйняття. Програма агента отримує як вхідні дані тільки результати поточного сприй-

няття, оскільки більше нічого не може дізнатися зі свого середовища; якщо дії агента залежать від всієї послідовності актів сприйняття, то агент винен сам запам'ятовувати результати цих актів сприйняття.

Для опису програм агентів застосовуватиметься проста мова псевдокоду. Наприклад, у лістингу 10.1 показана досить нескладна програма агента, яка реєструє послідовність актів сприйняття, а потім використовує отриману послідовність для доступу за індексом до таблиці дій і визначення того, що потрібно зробити. Таблиця явно відображає функцію агента, що втілюється цією програмою агента. Щоб створити раціонального агента таким чином, проектувальники повинні сформувати таблицю, яка містить відповідну дію для будь-якої можливої послідовності актів сприйняття.

Лістинг 10.1. Програма Table-Driven-Agent, яка викликається після кожного сприйняття нових даних і кожного разу повертає варіант дії; програма реєструє послідовність актів сприйняття з використанням своєї власної закритої структури даних

```
function Table-Driven-Agent (percept) returns дія action
  static: percepts, послідовність актів сприйняття, спочатку
порожня table, таблиця дій, індексована за послідовностями актів
сприйняття і повністю задана із самого початку; її результати сприйняття percept додати до кінця послідовності percepts
  action ← Lookup (percepts, table)
  return action
```

Аналіз того, чому такий підхід до створення агента, заснований на використанні таблиці, приречений на невдачу, є вельми повчальним. Припустимо, що P – безліч можливих актів сприйняття, а T – термін існування агента (загальна кількість актів сприйняття, яка може бути ним отримана). Пошукова таблиця міститиме

$$\sum_{t=1}^{T} |P|^{t} \text{ записів}$$

Розглянемо автоматизоване таксі: візуальні вхідні дані від однієї телекамери надходять зі швидкістю 27 мегабайтів за секунду (30 кадрів за секунду, 640 x 480 пікселів з 24 бітами інформації про колір). Згідно із цими даними пошукова таблиця, розрахована на 1 годину водіння, повинна містити кількість записів, що перевищує $10^{250000000000}$. І навіть пошукова таблиця для шахів (крихітного, добре вивченого фрагмента реального світу) мала б щонайменше 10^{150} записів. Приголомшуючий розмір цих таблиць (при тому, що кількість атомів у спостережуваному всесвіті не перевищує 10^{80}) означає, що, по-перше, жоден фізичний агент у нашому всесвіті не має простору для зберігання такої таблиці; по-друге, проектувальник не зможе знайти достатньо часу для створення цієї таблиці; по-третє, жоден агент ніколи не зможе навчитися тому, що міститься у всіх правильних записах цієї таблиці, на підставі власного досвіду, і, по-четверте, навіть якщо середовище достатньо просте для того, щоб можна було створити таблицю прийнятних розмірів, все одно у проектувальника немає керівних відомостей про те, як треба заповнювати записи подібної таблиці.

Не дивлячись на все сказане, програма Table-Driven-Agent виконує саме те, що від неї потрібно: вона реалізує бажану функцію агента. Основна складність, що стоїть перед штучним інтелектом як науковим напрямом, полягає в тому, щоб дізнатися, як створювати програми, які в рамках можливого виробляють раціональну поведінку з використанням невеликого об'єму коду, а не великої кількості записів таблиці. Існує безліч прикладів, які показують, що таке завдання може бути виконане успішно в інших областях; наприклад, величезні таблиці квадратного кореня, що використовувалися інженерами і школярами до 1970-х років, тепер замінені програмою з п'яти рядків, що працює в електронних калькуляторах, в якій застосовується метод Ньютона. Запитання полягає в тому, чи може штучний інтелект зробити для інтелектуальної поведінки в цілому те, що Ньютон зробив для спрощення обчислення квадратного кореня? Автори цієї книги вважають, що відповідь на це запитання є позитивною.

У решті частини цього розділу розглядаються чотири основні види програм агентів, які втілюють принципи, які лежать в основі майже всіх інтелектуальних систем:

- прості рефлексні агенти;
- рефлексні агенти, засновані на моделі;
- агенти, що діють на основі мети;
- агенти, що діють на основі корисності.

Далі наведений опис у загальних термінах того, як перетворити агентів всіх цих типів в агентів, що навчаються.

Прості рефлексні агенти

Простим видом агента є простий рефлексний агент. Подібні агенти вибирають дії на основі поточного акту сприйняття, ігноруючи решту всієї історії актів сприйняття. Наприклад, агент-порохотяг, для якого результати табуляції функції агента наведені в табл. 10.1, є простий рефлексний агент, оскільки його рішення засновані тільки на інформації про поточне місцезнаходження і про те, чи містить воно сміття. Програма для такого агента подана в лістингу 10.2.

Лістинг 10.2. Програма простого рефлексного агента в середовищі порохотяга з двома станами. Ця програма реалізує функцію агента, яка табульована в табл. 10.1.

```
function Reflex-Vacuum-Agent ([location,status]) returns дія
action
  if status = Dirty then return Suck
  else if location = Athen return Right
  else if location = У then return Left
```

Зверніть увагу на те, що ця програма агента-порохотяга дійсно дуже мала в порівнянні з відповідною таблицею. Найбільш очевидне скорочення обумовлене тим, що в ній ігнорується історія актів сприйняття, внаслідок чого кількість можливих варіантів скорочується від 4^T просто до 4. Додаткове невелике скорочення обумовлене тим фактом, що якщо в поточному квадраті є сміття, то виконувана при цьому дія не залежить від місцезнаходження порохотяга.

Уявіть себе на місці водія автоматизованого таксі. Якщо автомобіль, що рухається попереду, гальмує і спалахують його гальмівні вогні, то ви повинні помітити це і теж почати гальмування. Іншими словами, виконується певне опрацювання візуальних вхідних даних для виявлення умови, яка позначається як «car-in-front-is-braking» (автомобіль, що рухається попереду, гальмує). Потім ця умова активізує деякий зв'язок з дією «initiate-braking» (почати гальмування), встановленою в програмі агента. Такий зв'язок називається правилом «умо-ва-дія» і записується таким чином:

```
if car-in-front-is-braking then initiate-braking.
```

Люди також використовують велику кількість таких зв'язків, причому деякі з них є складними відгуками, освоєними в результаті навчаннями (як при водінні автомобіля), а інші є природженими рефлексами (такими як моргання, яке відбувається при наближенні до ока стороннього предмету).

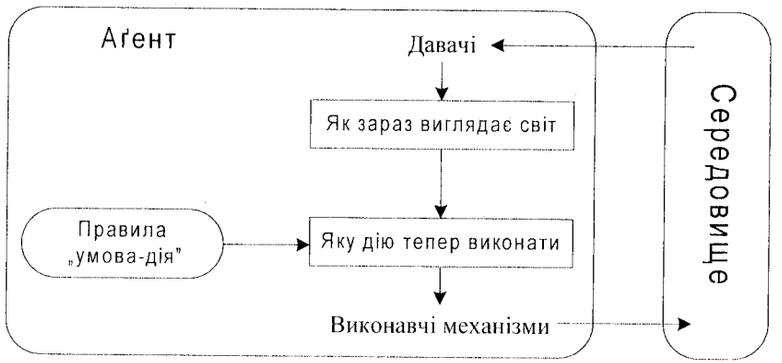


Рис. 10.3. Схематичне зображення структури простого рефлексного аґента

Програма, наведена в лістингу 10.2, спеціалізована для одного конкретного середовища порохотяга. Більш загальний і гнучкий підхід полягає в тому, щоб спочатку створити інтерпретатор загального призначення для правил «умова-дія», а потім визначити набори правил для конкретного проблемного середовища. На рис. 10.3 наведена структура такої загальної програми у схематичній формі й показано, яким чином правила «умова-дія» дозволяють аґентові створити зв'язок від сприйняття до дії (не треба турбуватися, якщо такий спосіб здається тривіальним; незабаром він виявить набагато цікавіші можливості). У подібних схемах для позначення поточного внутрішнього стану процесу ухвалення рішення аґентом використовуються прямокутники, а для подання фонової інформації, вживаної в цьому процесі, слугують овали. Програма цього аґента, яка також є дуже простою, наведена в лістингу 10.3. Функція Interpret-Input виробляє абстрагований опис поточного стану за наслідками сприйняття, а функція Rule-Match повертає перше правило у множині правил, яке відповідає заданому опису стану. Зазначимо, що поданий тут виклад у термінах «правил» і «відповідності» є чисто концептуальним; фактичні реалізації можуть бути настільки простими, як сукупність логічних елементів, що реалізовують логічну схему.

Лістинг 10.3. Програма простого рефлексного аґента, який діє згідно з правилами, умова якого відповідає поточному стану, визначуваному результатом сприйняття.

```
function Simple-Reflex-Agent(percept) returns дія action
static: rules, множина правил «умова-дія»
  state Interpret-Input(percept)
  rule Rule-Match(state, rules)
  action Rule-Action[rule]
  return action
```

Прості рефлексні аґенти характеризуються тією чудовою особливістю, що вони надзвичайно прості, та зате володіють вельми обмеженим інтелектом. Аґент, програма якого наведена в лістингу 10.3, працює, тільки якщо правильне рішення може бути ухвалене на основі виняткового поточного сприйняття, інакше кажучи, тільки якщо середовище є повністю спостережуваним. Внесення навіть невеликої частки неспостережності може викликати серйозне порушення його роботи. Наприклад, у наведеному вище правилі гальмування прийнято припущення, що умова car-in-front-is-braking може бути визначена з поточного сприйняття (поточного відеозображення), якщо рухомий автомобіль має гальмівний сигнал, розташований на центральному місці серед інших сигналів. На жаль, деякі старіші моделі мають інші конфігурації задніх фар, гальмівних сигналів, габаритних вогнів, сигналів гальмування і сигналів повороту, тому не завжди можливо визначити з єдиного зображення, чи гальмує цей автомобіль, чи ні. Простий рефлексний аґент, що веде свій автомобіль услід за таким автомобілем, або постійно гальмуватиме без будь-якої необхідності, або, що ще гірше, взагалі не буде гальмувати.

Виникнення аналогічної проблеми можна виявити й у світі порохотяга. Припустимо, що в простому рефлексному аґентові-порохотязі зіпсувався давач місцезнаходження і працює тільки давач сміття. Такий аґент отримує тільки два можливі сприйняття: [Dirty] і [Clean]. Він може виконати дію Suck у відповідь на сприйняття [Dirty], а що він повинен робити у відповідь на сприйняття [Clean]? Виконання руху Left завершиться відмовою (на невизначено довгий час), якщо виявиться, що він починає цей рух із квадрата A; а якщо він починає рух із квадрата B, то завершиться відмовою на невизначено довгий час рух Right. Для простих рефлексних аґентів, що діють у частково спостережуваних варіантах середовища, часто бувають неминучими нескінченні цикли.

Вихід з нескінченних циклів стає можливим, якщо аґент має здатність рандомізувати свої дії (вводити в них елемент випадковості). Наприклад, якщо аґент-порохотяг отримує результат сприйняття [Clean], то можна підкинути монету, щоб вибрати між рухами Left чи Right. Легко показати, що аґент досягне іншого квадрата в середньому за два етапи. Потім, якщо в цьому квадраті є сміття, то порохотяг його прибере, і завдання очищення буде виконано.

Тому рандомізований простий рефлексний аґент може перевершити за своєю продуктивністю детермінованого простого рефлексного аґента.

Вище вже згадувалося, що в деяких мультиаґентних варіантах середовища може виявитися раціональною рандомізована поведінка правильного типу. А в одноаґентних варіантах середовища рандомізація зазвичай не є раціональною. Це лише корисний трюк, який допомагає простому рефлексному аґентові в деяких ситуаціях, але в більшості випадків можна добитися набагато більшого за допомогою складніших детермінованих аґентів.

Рефлексні аґенти, засновані на моделі

Найефективніший спосіб організації роботи в умовах часткової спостережливості полягає в тому, щоб аґент відстежував ту частину світу, яка сприймається нами у поточний момент. Це означає, що аґент повинен підтримувати свого роду внутрішній стан, який залежить від історії актів сприйняття і тому відображає принаймні деякі з неспостережуваних аспектів поточного стану. Для вирішення завдання гальмування підтримка внутрішнього стану не вимагає дуже великих витрат – для цього досить зберегти попередній кадр, знятий відеокамерою, щоб аґент мав змогу визначити той момент, коли два червоні світлові сигнали з обох боків задньої частини автомобіля, що йде попереду, спалахують або гаснуть одночасно. Для вирішення інших завдань водіння, таких як перехід з однієї смуги руху на іншу, аґент повинен стежити за тим, де перебувають інші автомобілі, якщо він не може бачити всі ці автомобілі одночасно.

Для забезпечення можливості оновлення цієї внутрішньої інформації про стан протягом часу необхідно, щоб у програмі аґента були закодовані знання двох видів. По-перше, потрібна певна інформація про те, як світ змінюється незалежно від аґента, наприклад, про те, що автомобіль, який йде на обгін, зазвичай стає ближчим, ніж в якийсь попередній момент. По-друге, потрібна певна інформація про те, як впливають на світ власні дії аґента, наприклад, що при повороті аґентом рульового колеса за годинниковою стрілкою автомобіль повертає вправо або що після їзди автомагістраллю протягом п'яти хвилин на північ автомобіль перебуває на п'ять миль північніше від того місця, де він був п'ять хвилин тому. Ці знання про те, «як працює світ» (які можуть бути втілені в простих логічних схемах або у складних наукових теоріях), називаються моделлю світу. Аґент, в якому використовується така модель, називається аґентом, заснованим на моделі.

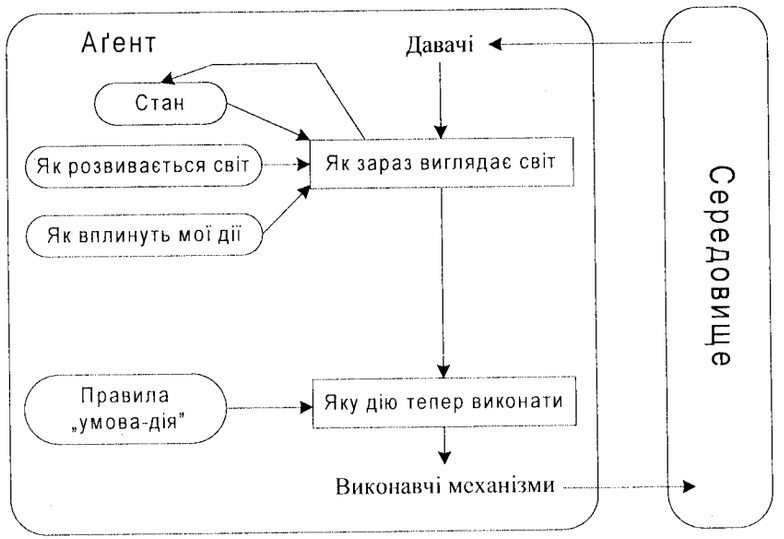


Рис. 10.4. Рефлексний аґент, який заснований на моделі

На рис. 10.4 наведена структура рефлексного аґента, який діє з урахуванням внутрішнього стану, і показано, як поточне сприйняття комбінується з попереднім внутрішнім станом для вироблення оновленого опису поточного стану. Програма такого аґента наведена в лістингу 10.4. У цьому лістингу цікавою є функція Update-State, яка відповідає за створення нового опису внутрішнього стану. Ця функція не тільки інтерпретує результати нового сприйняття у світлі наявних знань про стан, але й використовує інформацію про те, як змінюється світ, для стеження за невидимими частинами світу, тому повинна враховувати інформацію про те, як дії аґента впливають на стан світу.

Лістинг 10.4. Рефлексний аґент, заснований на моделі, який стежить за поточним станом світу з використанням внутрішньої моделі, потім обирає дію таким самим чином, як і простий рефлексний аґент.

```
function Reflex-Agent-With-State(percept)        returns дія
action
  static: state,   опис поточного стану світу
  rules,   множина правил умова-дія
```



```
  action,   остання за часом дія;
  спочатку не визначено
  state ← Update-State(state,   action,   percept)
   rule ← Rule-Match(state,   rules)
   action ← Rule-Action[rule]
  return action
```

Аґенти, засновані на меті

Знань про поточний стан середовища не завжди достатньо для ухвалення рішення про те, що робити. Наприклад, на перехресті доріг таксі може повернути наліво, повернути направо або їхати прямо. Правильне рішення залежить від того, куди має потрапити це таксі. Іншими словами, аґентові потрібний не тільки опис поточного стану, але і свого роду інформація про мету, яка описує бажані ситуації, такі як доставка пасажира в місце призначення. Програма аґента може комбінувати цю інформацію з інформацією про результати можливих дій (з такою самою інформацією, як і та, що використовувалася при оновленні внутрішнього стану рефлексного аґента) для вибору дій, що дозволяють досягти цієї мети. Структура аґента, що діє на основі мети, показана на рис. 10.5.

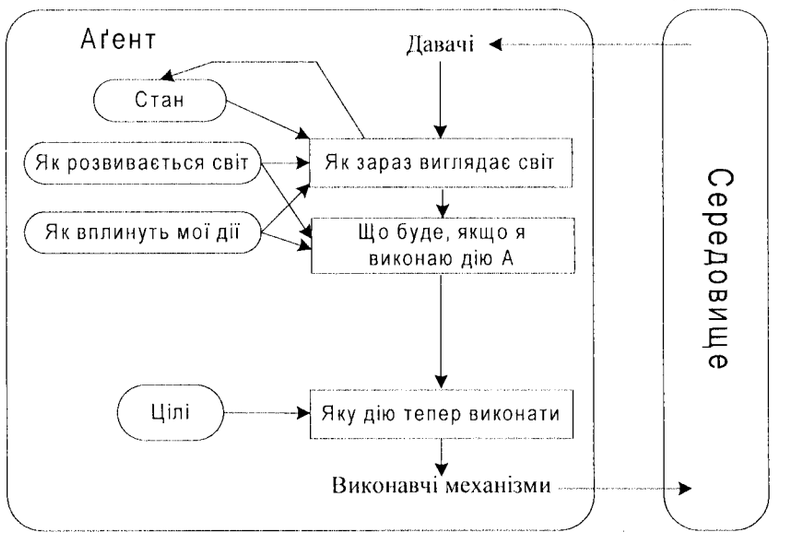


Рис. 10.5. Аґент, який заснований на моделі й цілях мети

Він стежить за станом світу, а також за множиною цілей, які він намагається досягнути, і вибирає дію, яка дозволяє (в кінцевому випадку) добитися досягнення цих цілей.

Іноді завдання вибору дії на основі мети вирішується просто, коли досягнення мети негайно стає результатом єдиної дії, а іноді це завдання стає складнішим, і аґентові потрібно розглянути довгі послідовності рухів і поворотів, щоб знайти спосіб досягнення мети. Підобластями штучного інтелекту, присвяченими виробленню послідовностей дій, що дозволяють аґентові досягти його цілей, є пошук і планування. Потрібно враховувати, що процедура ухвалення рішень такого роду має фундаментальні відмінності від описаної вище процедури застосування правил «умова-дія», оскільки в ній доводиться роздумувати про майбутнє, відповідаючи на два запитання: «Що відбудеться, якщо я

зроблю те і те?» і «Чи дозволить це мені досягти мети?». У проектах рефлексних агентів така інформація не подана явно, оскільки вбудовані правила встановлюють безпосередню відповідність між сприйняттями і діями. Рефлексний агент гальмує, побачивши сигнали гальмування, що рухається попереду, автомобіля, а агент, заснований на меті, може розсудити, що якщо на автомобілі, що рухається попереду, зажевріли гальмівні вогні, то він уповільнює свій рух. Враховуючи принцип, за яким зазвичай змінюється цей світ, для нього єдиною дією, що дозволяє досягти такої мети, як запобігання зіткненню з іншими автомобілями, є гальмування.

Хоча на перший погляд здається, що агент, заснований на меті, менш ефективний, він є гнучкішим, оскільки знання, на які спираються його рішення, подані явно і можуть бути модифіковані. Якщо починається дощ, агент може відновити свої знання про те, наскільки ефективно тепер працюватимуть його гальма; це автоматично викликає зміну всіх відповідних правил поведінки з урахуванням нових умов. Для рефлексного агента, з іншого боку, у такому разі довелося б переписати цілий ряд правил «умова-дія». Поведінку агента, заснованого на меті, можна легко змінити, щоб скерувати його в інше місце, а правила рефлексного агента, які вказують, де повертати і де їхати прямо, виявляться застосовними тільки для єдиного місця призначення; для того, щоб цього агента можна було скерувати в інше місце, всі ці правила мають бути замінені.

Агенти, засновані на корисності

Насправді, в більшості варіантів середовища, для вироблення високоякісної поведінки одного лише обліку цілей недостатньо. Наприклад, зазвичай існує багато послідовностей дій, що дозволяють таксі дістатися до місця призначення (і тим самим досягти поставленої мети), але деякі з цих послідовностей забезпечують швидшу, безпечнішу, надійнішу або дешевшу поїздку, ніж інші. Цілі дозволяють здійснити лише жорстку бінарну відмінність між станами «досягнення мети» і «недосягнення мети», тоді як загальніші показники продуктивності повинні забезпечувати порівняння різних станів світу в точній відповідності з тим, наскільки задоволеним стане агент, якщо їх вдасться досягти. Оскільки поняття «задоволеності» є не зовсім науковим, частіше застосовується термінологія, за якою стан світу, переважніший в порівнянні з іншим, розглядається, якщо має вищу корисність для агента.

Функція корисності відображає стан (або послідовність станів) на дійсне число, яке позначає відповідний ступінь задоволеності агента. Повна специфікація функції корисності забезпечує можливість ухвалювати раціональні рішення в описаних нижче двох випадках, коли цього не дозволяють зробити цілі. По-перше, якщо є конфліктні цілі, тобто можуть бути досягнуті тільки деякі з них (наприклад, або швидкість, або безпека), то функція корисності дозволяє знайти прийнятний компроміс. По-друге, якщо є декілька цілей, до яких може прагнути агент, але жодна з них не може бути досягнута з усією певністю, то функція корисності надає зручний спосіб зваженої оцінки вірогідності успіху з урахуванням важливості цілей.

Будь-який раціональний агент повинен поводитися так, ніби він володів функцією корисності, очікуване значення якої він намагається максимізувати. Тому агент, що володіє явно заданою функцією корисності, має можливість ухвалювати раціональні рішення і здатний робити це за допомогою алгоритму загального призначення, не залежного від конкретної максимізованої функції корисності. Завдяки цьому «глобальне» визначення раціональності (згідно з яким раціональними вважаються функції агента, що мають най-

вищу продуктивність) перетвориться в «локальне» обмеження на проекти раціональних агентів, яке може бути виражене у вигляді простої програми.

Структура агента, що діє з урахуванням корисності, показана на рис. 10.6.

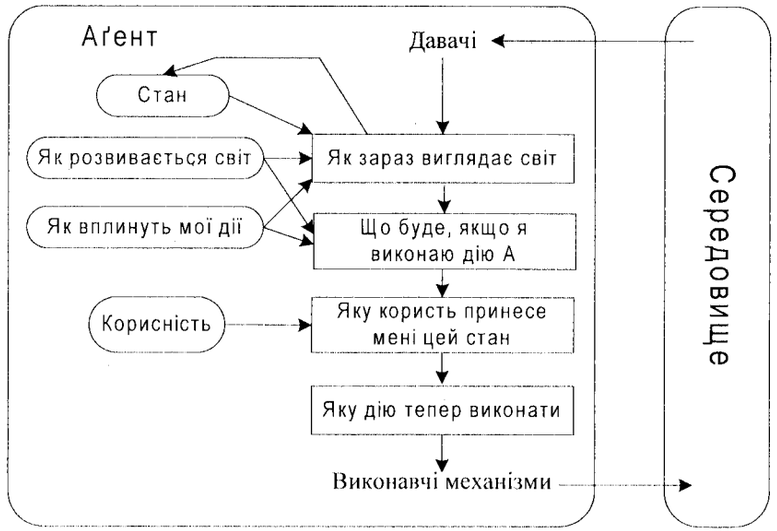


Рис. 10.6. Агент, заснований на моделі та корисності

У цьому модель світу використовується разом із функцією корисності, яка вимірює переваги агента стосовно станів світу. Потім агент обирає дію, яка веде до якнайкращої очікуваної корисності. Для обчислення очікуваної корисності виконується усереднення за всіма можливими результуючими станами з урахуванням коефіцієнта, що визначає вірогідність кожного результату.

Агенти, що навчаються

Вище були описані програми агентів, в яких застосовуються різні методи вибору дій. Але досі ще не були подані відомості про те, як створюються програми агентів. У своїй знаменитій ранній статті Тюринг [314] проаналізував ідею про те, як фактично має здійснюватися програмування запропонованих ним інтелектуальних машин вручну. Він оцінив об'єм роботи, який для цього буде потрібний, і прийшов до такого висновку: «Бажано було б мати якийсь продуктивніший метод». Запропонований ним метод полягав у тому, що необхідно створювати машини, що навчаються, а потім здійснювати їх навчання. Тепер цей метод став домінантним методом створення найсучасніших систем у багатьох областях штучного інтелекту. Як наголошувалося вище, навчання має ще одну перевагу: воно дозволяє агентові функціонувати у зазделегідь невідомих йому варіантах середовища і ставати компетентнішим у порівнянні з тим, що могли б дозволити тільки його початкові знання. У цьому розділі стисло подані основні відомості про агентів, що навчаються.

Як показано на рис. 10.7, структура агента, що навчається, може поділятися на чотири концептуальні компоненти. Найважливіша відмінність спостерігається між повчальним компонентом, який відповідає за внесення вдосконалень, і продуктивним компонентом, який забезпечує вибір зовнішніх дій. Продуктивним компонентом є те, що досі розглядалося, як агент отримує інформацію й ухвалює рішення про виконання дій. Повчаль-

ний компонент використовує інформацію зворотного зв'язку від критика з оцінкою того, як діє аґент, і визначає, яким чином має бути модифікований продуктивний компонент для того, щоб він успішніше діяв у майбутньому.

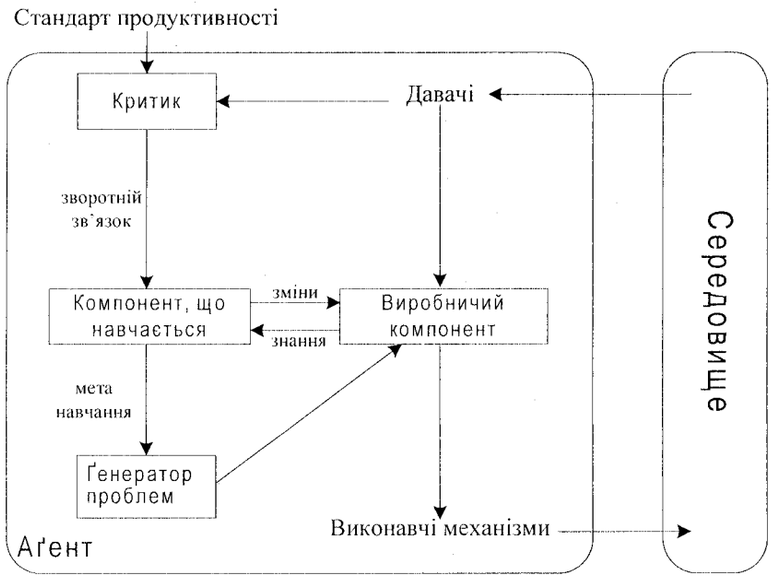


Рис. 10.7. Загальна модель аґентів, що навчаються

Проект повчального компоненту багато в чому залежить від проекту продуктивного компоненту. Здійснюючи спробу спроектувати аґента, який навчається певним здібностям, потрібно перш за все прагнути знайти відповідь на запитання: «Якого роду продуктивний компонент буде потрібний моєму аґентові після того, як він буде навчений тому, як виконувати свої функції?», а не на запитання: «Як приступити до розв'язування задачі навчання його для виконання цих функцій?». Після того, як спроектований сам аґент, можна приступати до конструювання повчальних механізмів, що дозволяють удосконалити будь-яку частину цього аґента.

Критик повідомляє повчальний компонент, наскільки добре діє аґент з урахуванням постійного стандарту продуктивності. Критик необхідний, оскільки самі результати сприйняття не дають ніяких вказівок на те, чи успішно діє аґент. Наприклад, шахова програма може отримати результати сприйняття, які вказують на те, що вона поставила мат своєму супротивникові, але їй потрібний стандарт продуктивності, який дозволив би визначити, що це – добрий результат; самі дані сприйняття нічого про це не говорять. Важливо, щоб стандарт продуктивності був постійним. У принципі, цей стандарт слід розглядати як повністю зовнішній по відношенню до аґента, оскільки аґент не повинен мати можливості його модифікувати так, щоб він більшою мірою відповідав його власній поведінці.

Останнім компонентом аґента, що навчається, є генератор проблем. Його завдання полягає в тому, щоб пропонувати дії, які повинні привести до отримання нового й інформативного досвіду. Річ у тому, що якщо продуктивний компонент наданий самому собі, то продовжує виконувати дії, які є якнайкращими з погляду того, що він знає. Але якщо аґент готовий до того, щоб небагато проекспериментувати і в короткочасній

перспективі виконувати дії, які, можливо, виявляться не зовсім оптимальними, то він може виявити набагато більш кращі дії з погляду довготривалої перспективи. Генератор проблем призначений саме для того, щоб пропонувати такі дослідницькі дії. Саме цим займаються вчені, проводячи експерименти. Галілей не вважав, що скидання каменів з вершини Пізанської башні є самоціллю. Він не старався просто вщент розбити ці камені або здійснити фізичну дію на голови невдалих перехожих. Його задум полягав у тому, щоб змінити погляди, що склалися в його власній голові, сформулювавши кращу теорію руху об'єктів.

Для того, щоб перевести весь цей проект на конкретний ґрунт, повернемося до прикладу автоматизованого таксі. Продуктивний компонент складається з тієї колекції знань і процедур, яка застосовується водієм таксі при виборі ним дій з водіння. Водій таксі за допомогою цього продуктивного компоненту виїжджає на дорогу і веде свою машину. Критик спостерігає за світом і в ході цього передає відповідну інформацію повчальному компоненту. Наприклад, після того, як таксі швидко виконує поворот наліво, перетинаючи три смуги руху, критик помічає, які шокуючі висловлювання використовують інші водії. На підставі цього досвіду повчальний компонент здатний сформулювати правило, яке свідчить, що це – неприпустима дія, а продуктивний компонент модифікується шляхом встановлення нового правила. Ґенератор проблем може визначити деякі області поведінки, що вимагають удосконалення, і запропонувати експерименти, такі як перевірка гальм на різних дорожніх покриттях і за різних умов.

Повчальний компонент може вносити зміни до будь-якого з компонентів «знань», показаних на схемах аґентів (див. рис. 10.3–10.6). У простих випадках навчання здійснюватиметься безпосередньо на підставі послідовності актів сприйняття. Спостереження за парами послідовних станів середовища дозволяє аґентові освоїти інформацію про те, «як змінюється світ», а спостереження за результатами своїх дій може дати аґентові можливість дізнатися, «який вплив мають мої дії». Наприклад, після того, як водій таксі прикладе певний гальмівний тиск під час їзди мокрою дорогою, він незабаром дізнається, яке зниження швидкості фактично було досягнуте. Очевидно, що ці два завдання навчання стають складнішими, якщо середовище спостережуване лише частково.

Ті форми навчання, які були описані в попередньому абзаці, не вимагають доступу до зовнішнього стандарту продуктивності, вірніше, в них застосовується універсальний стандарт, згідно з яким зроблені прогнози мають бути узгоджені з експериментом. Ситуація стає небагато складнішою, коли йдеться про аґента, заснованого на корисності, який прагне освоїти в процесі навчання інформацію про корисність. Наприклад, припустимо, що аґент, який займається водінням таксі, перестає отримувати чайові від пасажирів, які під час втомливої поїздки відчули себе повністю розбитими. Зовнішній стандарт продуктивності повинен інформувати аґента, що відсутність чайових – це негативний внесок в його загальну продуктивність; у такому разі аґент дістає можливість засвоїти в результаті навчання, що грубі маневри, втома пасажирів не дозволяють підвищити оцінку його власної функції корисності. У цьому сенсі стандарт продуктивності дозволяє виділити визначену частину вхідних результатів сприйняття як винагороду (або штраф), які безпосередньо надходять від даних зворотного зв'язку і впливають на якість поведінки аґента. Саме з цієї точки зору можуть розглядатися жорстко закріплені стандарти продуктивності, такі як біль або голод, якими характеризується життя тварин.

Підводячи підсумок, відзначимо, що агенти мають різноманітні компоненти, а самі ці компоненти можуть бути подані в програмі агента багатьма способами, тому створюється враження, що різноманітність методів навчання надзвичайно велика. Проте всі ці методи мають єдиний аспект що їх об'єднує. Процес навчання, здійснюваний в інтелектуальних агентах, можна в цілому охарактеризувати як процес модифікації кожного компоненту агента для забезпечення точнішої відповідності цих компонентів доступній інформації зворотного зв'язку і тим самим поліпшення загальної продуктивності агента.

10.5. Основні поняття мультиагентних систем

Вивчення мультиагентних систем почалося в галузі розподіленого штучного інтелекту (Distributed Artificial Intelligence, DAI) біля двадцяти років тому, і сьогодні ці системи є не тільки темами наукових досліджень, але починають бути предметом академічного вивчення, а також знаходять своє застосування у комерційних програмних продуктах.

З недавнього свого зародження (кінець 70-х років XX ст.), наука про розподілений штучний інтелект розвивалася динамічно, досягла досить суттєвих їх результатів та стала різноманітною. Сьогодні це вже галузь, що встановилася, але продовжує розвиватися. Вона є дуже перспективною дослідницькою галуззю, а також галуззю практичного застосування.

Вона увібрала в себе багато дисциплін, таких як: штучний інтелект, комп'ютерні науки, соціологія, економіка, наука організації та керування (менеджмент) і філософія. Така природа галузі, що включає багато дисциплін, ускладнює завдання характеризувати галузь – розподілений штучний інтелект (DAI) – у декількох тезах.

Розподілений штучний інтелект – це наука, структура та програмні продукти, які як апарат використовують мультиагентні системи (Multiagent Systems, MAC).

Мультиагентні системи, у свою чергу, це – системи, в яких функціонує деяка кількість інтелектуальних агентів (agents), що взаємодіють та намагаються досягти деякої множини цілей або розв'язати деяку множину задач.

Агент – це обчислювальна одиниця, така як комп'ютерна програма або робот; вона може розглядатися як така, що діє на основі навколишнього середовища і яка автономна, а також принаймні частина її поведінки ґрунтується на її власному досвіді.

Агент як інтелектуальна одиниця діє у розмаїтті навколишнього середовища гнучко та раціонально. Отримання інформації про навколишнє середовище та дія у ньому досягається за допомогою перцептуального та дійового апаратів, відповідно. Ця гнучкість та раціональність поведінки агента досягається за рахунок таких базових понять, як вирішення завдань (problem solving), планування (planning), прийняття рішень (decision making) та навчання (learning).

Агент як одиниця, що взаємодіє, може бути змінений у процесі своєї діяльності іншими агентами, або навіть людьми. Ключовим моментом взаємодії у мультиагентних системах є координація на досягнення цілей та розв'язання задач, причому як у ситуації кооперації, так і у ситуації суперництва. У ситуації кооперації деякі агенти намагаються об'єднати свої зусилля, щоб досягти групою того, чого вони не могли досягти поодинці. У ситуації суперництва агенти намагаються досягти того, що тільки деяка кількість з них може мати.

Як довготривалу мету досліджень у галузі розподіленого штучного інтелекту треба вважати розроблення механізмів та методів, які б дозволили агентам взаємодіяти так

добре, як людина (або навіть краще), щоб зрозуміти взаємодію між розумними істотами, будь-то обчислювальна одиниця, людина, або і те, і те. Ця мета призводить до цілої низки питань перспективних досліджень, які сконцентровані навколо одного простого запитання: коли, як і з ким взаємодіяти?

Головними силами, що стимулюють дослідження у галузі розподіленого штучного інтелекту, є дві причини.

Перша визначає те, що мультиагентні системи мають можливість відіграти головну роль у сучасній і майбутній комп'ютерній науці та її програмних продуктах. Сучасні обчислювальні системи є розподіленими, великими, відкритими та неоднорідними (різнотипними), а комп'ютерні системи вже більше не автономні системи, а тісно пов'язані між собою та своїми користувачами. Це збільшення складності комп'ютерних та інформаційних систем неминуче призводить до зростання складності їх програмних продуктів. Це приводить до зростання рівня синхронізованого, централізованого обчислення, тому що потрібне, наприклад, опрацювання величезної кількості інформації, або інформації, що з'являється у географічно віддалених місцях. Щоб вдало справлятися з такими завданнями, комп'ютерні системи повинні діяти більше як «індивідууми», тобто агенти, ніж тільки як «частини».

Технології, які розподілений штучний інтелект як наука буде впроваджувати, належать до тих, що терміново необхідні для керування високорівневими взаємодіями у складних програмних продуктах для сучасних обчислювальних систем та систем опрацювання інформації.

Друга причина полягає в тому, що мультиагентні системи мають можливість відіграти головну роль у розробленні та аналізі моделей і теорій взаємодії у людських співтовариствах. Люди взаємодіють різними способами та на різних рівнях. Наприклад, вони спостерігають та моделюють один одного, запитують та пропонують інформацію, ведуть переговори та обговорюють, досліджують спільне навколишнє середовище, визначають та вирішують конфлікти, створюють та розпускають такі організаційні утворення, як команди, комітети та економічні організації. Багато процесів взаємодії між людьми досі малозрозумілі, але вони – частина нашого повсякденного життя. Технології розподіленого штучного інтелекту дають нам змогу зрозуміти їх соціальне та психологічне підґрунтя.

Традиційний штучний інтелект використовує психологію та біхевіоризм (науку про поведінку) для ідей, натхнення та метафор, у той час як розподілений штучний інтелект використовує соціологію та економіку. Отже, розподілений штучний інтелект не є спеціалізацією традиційного штучного інтелекту, а є його узагальненням.

Наведемо перелік класів предметних областей, для яких можна будувати мультиагентні системи:

- електронна комерція та електронні ринки, де агенти-покупці та агенти-продавці замовляють або продають товари за дорученням своїх користувачів;
- моніторинг та керування телекомунікаційними мережами у реальному часі, де агенти відповідають, наприклад, за скерування дзвінків, а також перемикання та передавання сигналів;
- моделювання та оптимізація локальних, місцевих, регіональних, національних або всесвітніх систем постачання, де агенти, наприклад, надають транспортні засоби або товари чи клієнтів, яких необхідно перевезти;

- збирання інформації в інформаційних середовищах, таких як Інтернет, де багаточисельні аґенти відповідають за фільтрування та збирання інформації;
- вдосконалення міського або повітряного потоку транспортних засобів, де аґенти відповідають за правильну інтерпретацію інформації, що знімається з різного роду давачів (сенсорів);
- автоматизація графіку зустрічей, де аґенти діють за дорученням своїх користувачів з метою вирішення таких питань, як визначення деталей: місце, час та питання, що будуть вирішуватися на зустрічі;
- оптимізація промислового виробництва і процесів виробництва, таких як складання графіків роботи верстатів або керування ланцюгом постачань, де аґенти, наприклад, являють робочі одиниці, або навіть цілі підприємства;
- аналіз бізнес-процесів всередині підприємства або між підприємствами, де аґенти відображають людей або окремі відділи, що включені у ці процеси на різних стадіях і на різних рівнях;
- індустрія електронних розваг, таких як інтерактивні комп'ютерні ігри, що базуються на технологіях віртуальної реальності, де аґенти – це анімовані персонажі, які, використовуючи різні засоби створеного навколишнього середовища та його правила, грають один проти іншого або проти людей (аґентів, повністю керованих людиною);
- розроблення та реінжиніринґ структур протікання інформації або керівництва у великих природних, технічних або гібридних організаціях, де аґенти відображають сутності, з яких складаються такі структури;
- вивчення соціальних аспектів інтелекту та симуляція складних соціальних феноменів, таких як еволюція прав, норм та організаційних структур, де аґенти відіграють роль членів суспільства, яке розглядається.

10.6. Аналіз сучасних досліджень у розробленнях мультиаґентних систем

Як вже було сказано, розподілений штучний інтелект – це наука, що надзвичайно динамічно розвивається. Отже, розглянемо сучасні напрями її розвитку.

Наприкінці 80-х – початку 90-х років XX ст. виникли два цікавих, тісно пов'язаних між собою напрями кібернетичних досліджень: «Штучне життя» (англійська назва Artificial Life або ALife) і «Адаптивна поведінка» (Adaptive Behavior) [309].

Розглянемо напрям «Штучне життя».

Основною мотивацією досліджень штучного життя слугує бажання зрозуміти і промоделювати формальні принципи організації біологічного життя. Як сказав керівник першої міжнародної конференції зі штучного життя К. Лангтон «основне припущення штучного життя полягає в тому, що «логічна форма» організму може бути відокремлена від матеріальної основи його конструкції».

Прихильники напряму «Штучне життя» часто вважають, що вони досліджують більш загальні форми життя, ніж ті, які існують на Землі. Тобто вивчається життя, яким воно могло б у принципі бути («life-as-it-could-be»), а не обов'язково те життя, як ми його знаємо («life-as-we-know-it»).

Штучне життя – це синтетична біологія, яка за аналогією із синтетичною хімією намагається відтворити біологічну поведінку в різних середовищах. Це життя, створене людиною, а не природою («life made Man rather than Nature»). Дослідження штучного життя скеровані не лише на теоретичні дослідження властивостей життя, але і (аналогічно до синтетичної хімії) на практичні додатки, такі як рухливі роботи, медицина, нанотехнологія, «життя» соціальних систем та інші інженерні застосування.

Велику роль в дослідженнях штучного життя відіграє математичне і комп'ютерне моделювання. Дуже часто «організми» у штучному житті – це вигадані людьми об'єкти, що живуть у світі комп'ютерних програм.

Відзначимо, хоча гасло «Штучне життя» було проголошене наприкінці 80-х років минулого століття, насправді ідейно близькі моделі розроблялися у 50-70-ті роки XX ст. Подамо два приклади з історії.

У 60-х роках XX ст. кібернетик і математик М. Л. Цетлін запропонував і дослідив моделі автоматів, здатних адаптивно пристосовуватися до навколишнього середовища. Роботи М. Л. Цетліна ініціювали цілий науковий напрям, що отримав назву «колективна поведінка автоматів».

У 60-70-х роках минулого століття під керівництвом талановитого кібернетика М. М. Бонгарда була побудована вельми нетривіальна модель «Тварина», що характеризує адаптивну поведінку штучних організмів, які живуть на розбитій на клітинки площинах і мають низку потреб, що конкурують між собою.

Наведемо деякі приклади характерних досліджень штучного життя [309].

- Дослідження динаміки життєподібних структур у клітинних автоматах під керівництвом К. Лангтона.
- ПоліСвіт (PolyWorld) Л. Ягера: комп'ютерна модель штучних організмів, які мають структуровану нейронну мережу, володіють колірним зором, можуть рухатися, харчуватися (і збільшувати тим самим свою енергію), можуть схрещуватися і боротися один з одним. При моделюванні еволюції в ПоліСвіті виникала ціла низка нетривіальних стратегій поведінки організмів. Нижче буде наведено більш детальний опис цієї моделі.
- Тьєра (Tierra) Т. Рея: модель еволюції комп'ютерних програм, що самі репродукуються. "Організми" Тьєри містять геноми, які визначають інструкції виконавчих програм. Взаємодії між організмами приводять до еволюційного виникнення складної "біорізноманітності" програм, що самі репродукуються.
- Авіда (Avida) К. Адамі із співробітниками; ця модель може розглядатися як розвиток моделі Тьєра. У порівнянні з Тьєрою, ця модель простіша і володіє більшою спільністю. Модель аналізувалася аналітичними методами. Були отримані характеристики розподілу осіб в популяціях, що еволюціонують. Ця модель кількісно підтримує ту точку зору, що еволюція рухається стрибками, а не неперервно.

Аналіз взаємодії між навчанням і еволюцією виконаний Д. Еклі та М. Літтманом. Ця робота продемонструвала, що «навчання й еволюція разом успішніші у формуванні адаптивної популяції, ніж навчання або еволюція окремо». Нижче буде наведено детальніший опис цієї моделі.

- «ЛУНА (ECHO)» Дж. Холланда. Ця модель описує еволюцію простих аґентів, які взаємодіють між собою шляхом схрещування, боротьби і торгівлі. Взаємодії між

аґентами сприяють формуванню різних екологічних систем: «війни світів», симбіозів тощо.

- Модель еволюції двох конкуруючих популяцій, одна з яких є популяцією програм, що вирішують певну прикладну проблему (проблему сортування), а друга – популяція задач, що еволюціонують у напрямі ускладнення проблеми (Д. Хилліс (D. Hillis)). Перша з популяцій може розглядатися як популяція осіб-господарів, а друга – як популяція паразитів. Моделювання показало, коеволюція (coevolution) в системі «паразит-господар» призводить до знаходження значно кращих рішень проблеми в порівнянні з тим рішенням, яке можна знайти в результаті еволюції лише першої популяції.
- Моделі еволюції клітинних автоматів, наприклад, моделі М. Мітчела зі співробітниками, що описують еволюційний пошук клітинних автоматів, які можуть виконувати прості обчислення.
- «Мурашина ферма (AntFarm)» Р. Коллінза і Д. Джефферсона. Ця модель розроблена на базі «Конекшен-машини» (Connection-Machine). Модель імітує поведінку пошуку їжі у величезних популяціях штучних мурашок, що еволюціонують.
- Класифікаційні системи Дж. Холанда зі співробітниками. Це модель еволюції когнітивного процесу. Класифікаційна система є системою індуктивного виведення, яка заснована на наборі логічних продукційних правил. Кожне правило має наступну форму: «якщо <умова>, тоді <дія>». Система правил оптимізується як за допомогою навчання, так і еволюційним методом. У процесі навчання змінюються пріоритети використання правил (тобто змінюються коефіцієнти, що характеризують силу правил). При навчанні використовується так званий алгоритм «пожежної бригади»: при успіху нагороджуються не тільки ті правила, які безпосередньо привели до успішної дії, але і ті, які були попередниками успіху. Пошук нових правил здійснюється еволюційним методом.

Моделі штучного життя – область досліджень, що активно розвивається. Більшість моделей – дотепні комп'ютерні експерименти. Хороший приклад серйозного математичного дослідження – роботи К. Адамі по розподілу осіб у популяціях, що еволюціонують. Цей аналіз заснований на теорії критичності, що самоорганізується і розумно інтерпретує як комп'ютерні експерименти на моделях Тьєра і Авіда, так і реальні біологічні дані.

Дослідження штучного життя тісно пов'язані з іншими цікавими напрямами: моделями походження життя, автоматами С. А. Кауффмана, роботами з прикладного еволюційного моделювання, з теорії нейронних мереж. Еволюція популяцій штучних організмів – один з напрямів досліджень штучного життя. Моделі еволюції тут часто засновані на генетичному алгоритмі. Однак у моделях штучного життя часто не вводиться явно поняття пристосованості, як це робиться в генетичному алгоритмі. Пристосованість виявляється природним шляхом: особи народжуються, коли їх батьки готові дати нащадків, і гинуть, коли не вистачає їжі або коли їх вбиває і з'їдає хижак. У цьому випадку говорять, що пристосованість ендогенна. Приклад такої ендогенної пристосованості наведений нижче в описі моделі ПоліСвіт. Керування поведінкою штучних організмів часто моделюється за допомогою нейронних мереж.

Моделі штучного життя проливають нове світло на еволюційні та соціальні явища. Яскравий приклад, що ілюструє цю тезу – дослідження ефекту Балдвіна. За ефектом

Балдвіна (1896 рік), придбані при навчанні навички організмів можуть побічно передаватися подальшим поколінням. Ефект Балдвіна працює у два етапи. На першому етапі організми, що еволюціонують (завдяки відповідним мутаціям) набувають властивість навчитися деякій корисній навичці. Пристосованість таких організмів збільшується, отже, вони розповсюджуються по популяції. Але навчання має свої «невигідні витрати», так воно вимагає енергії та часу. Тому можливий другий етап (який називають генетичною асиміляцією): придбана корисна навичка може бути «повторно винайдена» генетично еволюцією, внаслідок чого він записується безпосередньо в геном і стає успадкованою. Другий етап триває безліч поколінь; стійке навколишнє середовище і висока кореляція між генотипом і фенотипом полегшують цей етап. Отже, корисна навичка, яка була спочатку придбана, може стати успадкованою, хоча еволюція має дарвінівський характер.

Низка дослідників (Г. Хинтон і С. Новлан, Д. Еклі та М. Літтман, Г., Мейлей і багато інших) аналізували ефект Балдвіна. Вони показали, що цей ефект може відіграти істотну роль у процесі еволюції штучних осіб. Конкретний приклад моделі, в якій виявляється ефект Балдвіна – модель Д. Еклі та М. Літтмана – описана нижче.

Модель ПоліСвіт Л. Ягера

Модель Ягера – одна з типових моделей штучного життя, в якій здійснено моделювання достатньо природної поведінки штучних організмів.

Уявимо собі деякий обмежений простір (скажімо: великий стіл), на якому можуть жити штучні організми. По краях стіл-світ обмежений бар'єрами так, щоб організми не падали зі столу. На столі можуть зростати галявинки зеленої їжі. Організми можуть рухатися прямолінійно, повертатися, поглинати їжу. Вони володіють колірним зором. Одні організми можуть вступати в боротьбу з іншими, при цьому переможені організми вмирають, і їх каркас перетворюється на їжу. Організми можуть схрещуватися, даючи нащадків. Якщо організм вступає в боротьбу, то він червоніє, якщо відчуває бажання схреститися, – він синіє.

Організми мають нервову систему, що складається зі штучних нейронів. Нейронна мережа організму керує його поведінкою: регулює швидкість руху, повороти, войовничу і статеву активність, забезпечує фокусування зору на оточуючих організм об'єктах. Поїдаючи їжу (зелені галявини або каркаси мертвих осіб), організми поповнюють свій ресурс енергії. Проявляючи активність (рух, повороти, боротьба, схрещування), організми витрачають енергію. Якщо ресурс організму стає нижчим від певної межі, то організм вмирає (і, природно, перетворюється на їжу).

Популяція організмів еволюціонує. Розмноження організмів відбувається в результаті схрещування, загибель – в результаті боротьби або голоду. Параметри організму (розмір, швидкість руху, бійцівська сила, основний колір і т. ін.), а також структура нейронної мережі визначаються геномом організму. Нащадки організмів успадковують гени батьків (частина генів від одного батька, частина – від іншого), при переході від батьків до нащадків гени переживають малі мутації.

Еволюція організмів у ПоліСвіті моделювалася комп'ютерною програмою, що містить 15 000 рядків на C++.

У процесі моделювання еволюції спостерігалося формування певних стратегій поведінки тварин.

Одну зі стратегій можна умовно назвати «тупа корова»: організм рухається прямолінійно з максимальною швидкістю, поїдає всі галявини їжі, що зустрічаються, і схрещується зі всіма, кого зустріне.

Друга стратегія – «лінивий канібал»: організм крутиться на місці, схрещуючись або вступаючи в боротьбу з кожним, хто наблизиться (поїдаючи каркас суперника у разі перемоги або гинучи у разі поразки).

У деяких комп'ютерних експериментах еволюція приводила до появи стратегії життя «на краю світу»: організми циркулювали за або проти годинникової стрілки уздовж бар'єрів, що обмежують стіл, і це приводило до певних переваг, оскільки тут організми часто знаходили особин, з якими можна схреститися або померятися силами.

Відзначимо переваги і недоліки моделі ПоліСвіт. До безперечних переваг цієї моделі належить продумана схема поведінки: природна реакція на події в навколишньому середовищі, природні дії організму, природні взаємодії між організмами. Але модель «переобтяжена» деталями, пов'язаними з колірним зором. Це обумовило й інший недолік моделі: дуже складна нервова система. Як наслідок результати моделювання виявилися досить туманними. Зокрема, хоча в модель була закладена можливість навчання, насправді навчання себе ніяк не проявило. Фактично модель показала, що:

- складний колірний зір цілком міг сформуватися в процесі еволюції;
- можливе еволюційне формування вказаних вище стратегій – цим основні результати моделі й вичерпуються.

На жаль, модель ПоліСвіт залишилася незавершеною (її автор «перемкнувся» на інші задачі), хоча вона має певний потенціал для подальшого розвитку.

Модель взаємодії навчання й еволюції Д. Еклі та М. Літтмана

Загальна схема моделі

Модель так само, як і ПоліСвіт, використовує достатньо природну схему поведінки штучних організмів (агентів), але не переобтяжена технічними деталями, пов'язаними зі специфікою зору. Це модель Д. Еклі та М. Літтмана.

У моделі передбачається, що агенти живуть у двовимірному світі, розбитому на клітинки. У клітинах можуть розташовуватися самі агенти, трава, хижаки, дерева, камені. Трава слугує їжею для агентів. Хижаки та інші агенти можуть битися з іншим агентом, послаблюючи його здоров'я. Залізаючи на дерева, агенти стають недоступними для хижаків. Дерева зростають і гинуть. Якщо дерево гине, і при цьому на дереві сидів агент, то дерево вбиває цього агента. Камені є перешкодами для агентів: якщо агент стикається з каменем, то він втрачає при цьому частину свого здоров'я. Агенти можуть схрещуватися, даючи нащадків. Агенти гинуть з голоду або втрачають здоров'я. Загиблі агенти слугують їжею для інших агентів і хижаків.

Популяція агентів еволюціонує: при схрещуванні народжуються нащадки агентів, які, загалом, відрізняються від своїх батьків. Агенти можуть навчатися: їх дії визначаються нейронною мережею, яка може удосконалюватися у процесі життєдіяльності агента. Популяція хижаків не еволюціонує: всі хижаки однакові, хижаки не навчаються, їх дії в будь-якій ситуації однозначно визначені.

Поведінка агентів керується їх нейронною мережею. Входами нейронної мережі є видима картина світу і внутрішній стан агентів (кількість енергії та здоров'я). Агенти бачать світ навколо себе на відстані до 4-х клітин у 4-х напрямках (північ, південь, схід, захід). Поведінка агентів визначається виходами їх нейронної мережі. Світ розвивається

в дискретному часі t. У кожний такт часу нейронна мережа визначає вибір дій агента. Вибір дій дуже простий: вибрати один з 4-х напрямків руху. Після вибору дії «локальна доля» агента в наступний такт часу буде однозначно визначена – вона залежить тільки від того, що є в цільовій клітинці (клітинці у напрямку руху).

Наприклад, агент може просто переміститися в цільову клітинку (якщо ця клітинка порожня), з'їсти в клітинці траву (якщо вона там є), залізти на дерево (якщо в цільовій клітинці є дерево, і на ньому немає іншого агента), ударитися об камінь, бути ударений іншим агентом або хижаком тощо.

Нейронна мережа змінюється як в ході еволюції, так і в індивідуальному процесі навчання. Геном агента містить вагу синапсів блоку оцінювання дій і початкову вагу синапсів блоку поведінки. Ця вага змінюється в ході еволюції.

Вага синапсів блоку поведінки змінюється в ході індивідуального розвитку агентів. Ця зміна і є навчання, яке відбувається під контролем блоку оцінки поведінки. Вага синапсів блоку оцінки не змінюється протягом життя агента. Блок оцінки відіграє роль вчителя для блоку поведінки. Сам «вчитель» при цьому оптимізується в процесі еволюції.

Навчання блоку поведінки здійснювалося спеціальним способом: «Комплементарним методом зворотного розповсюдження помилки» («Complementary reinforcement back propagation» (CRBP)) – цей метод можна розглядати як варіант методу зворотного розповсюдження. Основна ідея методу полягає в тому, що помилка на виході кожного нейрона блоку поведінки визначається за виходом цього нейрона і за сигналом навчання r, що надходить від блоку оцінки дій. Сигнал навчання r може бути позитивним, якщо блок оцінки «вважає», що команда блоку поведінки поліпшує життя агента, або негативним – якщо інакше. Відповідно до помилок на виходах нейронів коректується вага синапсів нейронів, аналогічно до того, як це робиться у звичайному методі зворотного розповсюдження помилок.

При моделюванні Д. Еклі та М. Літтман аналізували як повну модель агентів (ERL), що включає як навчання, так і еволюцію (так, як це описано вище), так і часткові моделі, що включають тільки еволюцію (E) або тільки навчання (L).

Початково задавали випадкові популяції та стежили, наскільки швидко ці популяції вимирають. На рис. 10.8 схематично подані результати моделювання.

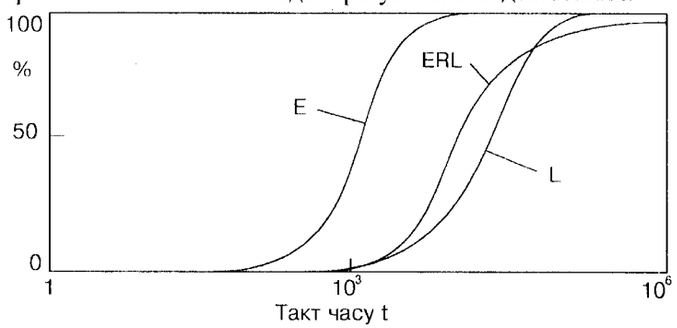


Рис. 10.8. Залежність відсотка вимерлих популяцій від часу t (схематично)

Рис. 10.8 демонструє, що для такої моделі агентів сама лише еволюція без навчання (E) погано забезпечує пристосовування популяції агентів. Навчання (L) приводить до кращого пристосовування, але найкращий результат спостерігається у разі спільної

взаємодії еволюції та навчання (ERL). У цьому останньому випадку перебували агенти, які не вимирали протягом мільйона тактів життя агентів.

В одному з комп'ютерних експериментів над повною версією програми (ERL) Д. Еклі та М. Літтман запустили програму на декілька днів, а потім зі здивуванням знайшли, що популяція не вимерла, хоча число тактів життя агентів і досягло $9*10^6$.

Д. Еклі та М. Літтман проаналізували на моделі й інші особливості взаємодії між навчанням і еволюцією, наприклад, вони знайшли ефект «екранування»: якщо вже є природжена сприятлива навичка, то навчання цій навичці вже не потрібне, і воно дійсно не відбувається.

Отже, модель Д. Еклі та М. Літтмана на прикладі агентів з дуже простою нейронною мережею характеризує деякі особливості взаємодії між навчанням і еволюцією.

Перевага цієї моделі – оптимальна схема розділення нейронної мережі на блок оцінки дій, що формує цілі поведінки, і блок поведінки, що дає команди на виконання дій, тобто формуючи саму поведінку. Проте в моделі є і недоліки: світ, в якому живуть агенти, переобтяжений зайвими деталями, остаточні результати недостатньо чіткі. Крім того, виділення окремого блоку оцінки дій (хоча і розумне для штучних організмів) виглядає дуже відірваним від реальних біологічних нейронних мереж.

Декілька років тому доктор фізико-математичних наук В. Г. Редько разом із Р. В. Гусаревим почали роботу над більш природною моделлю еволюційного виникнення цілеспрямованої адаптивної поведінки. Модель може розглядатися як розвиток робіт Л. Ягера, Д. Еклі та М. Літтмана. У вказаній моделі цілі зроблені більш природними в порівнянні з моделлю Д. Еклі та М. Літтмана і пов'язані з основними потребами організмів. Нижче подається коротка характеристика моделі.

Основні припущення моделі еволюційного виникнення цілеспрямованої адаптивної поведінки полягають у наступному:

1) є популяція агентів (штучних організмів), що мають природні потреби (Безпека, Енергія, Розмноження);
2) популяція агентів еволюціонує в простому середовищі, де ростуть галявинки трави (їжа агентів); агенти взаємодіють між собою: агенти можуть схрещуватися і боротися один з одним і зовнішнім середовищем; боротьба може привести до загибелі агента (загиблий агент перетворюється на їжу і може бути з'їдений переможцем); схрещування приводить до народження нових агентів;
3) кожна потреба характеризується кількісно мотивацією; наприклад, якщо енергетичний ресурс агента малий, то велика мотивація знайти їжу і поповнити енергетичний ресурс;
4) поведінка агента керується його нейронною мережею, яка має спеціальні входи від мотивацій; якщо є певна мотивація, то поведінка агента змінюється з тим, щоб задовольнити відповідну потребу; таку поведінку прийнято називати цілеспрямованою (є ціль досягти певну потребу).

Найцікавіше запитання: як така цілеспрямована поведінка могла виникнути в процесі еволюції? Мета моделі – досліджувати цю проблему настільки глибоко, наскільки це можливо.

10.7. Аналіз моделей, методів та алгоритмів, що використовуються у мультиагентних системах

Виконаємо аналіз моделей, методів та алгоритмів, що використовуються при проектуванні та побудові систем, що ґрунтуються на соціальних моделях, тобто мультиагентних систем.

10.7.1. Взаємодія агентів

Агенти у мультиагентних системах діють у так званому навколишньому середовищі. Оскільки агентів у таких системах може бути дуже багато, проектування навколишнього середовища має передбачати інфраструктуру для взаємодій агентів, щоб можна було оперувати з агентами не індивідуально, а колективно. Ця інфраструктура має включати протоколи взаємодії агентів та протоколи спілкування агентів між собою.

Протоколи спілкування дають змогу агентам обмінюватися повідомленнями та розуміти їх. Протоколи взаємодії дають змогу агентам власне спілкуватися, що являє собою структурований обмін повідомленнями.

Наприклад, протоколи спілкування можуть специфікувати наступні типи повідомлень, які один агент може передати іншому:

- пропозиція напрямку дій;
- згода з напрямком дій;
- відмова від напрямку дій;
- скорегувати напрямок дій;
- не погодитися з напрямком дій;
- видати контрпропозицію напрямку дій.

Агенти взаємодіють між собою з метою більшого наближення до їх цілей або цілей суспільства/системи, в якій вони діють. При чому цілі можуть і не бути явно відомі агентові. Це залежить від того, чи агент орієнтований на ціль (goal-based). Взаємодія може дати змогу агентам координувати свої дії та поведінку, перетворюючи систему у більш цілеспрямовану.

Координація – це властивість системи агентів виконувати деякі дії у спільному навколишньому середовищі. Ступінь координації – це міра усунення непотрібної активності у системі шляхом зменшення розбіжностей дій, уникнення тупикових ситуацій та циклічностей, а також підтримка прийнятного ступеня безпеки. Кооперація – це координація між неантагоністичними агентами, а переговори (домовленості, negotiation) – це координація простих самозацікавлених агентів, що конкурують між собою. Щоб кооперуватися успішно, типовим є те, що кожен агент має підтримувати модель інших агентів, а також розробляти модель майбутніх взаємодій. Це передбачає товариськість (здатність до спілкування).

Схематично систематику різних способів координації поведінки та активності агентів можна показати наступним чином (рис. 10.9)

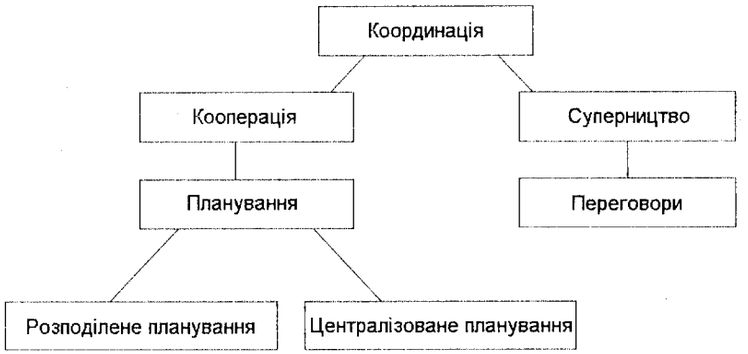


Рис. 10.9. Систематика різних способів координації поведінки та активності агентів.

Цілеспрямованість – це, як вдало система поводить себе як єдине ціле. Проблема мультиагентних систем – як вона може підтримувати глобальну цілеспрямованість без явного глобального контролю. Втім, агенти повинні мати можливість самі визначати цілі, які вони розділяють з іншими агентами, визначати спільні завдання, уникати непотрібних конфліктів, слабких знань.

Отже, агенти мають спілкуватися. Фундаментальним у цьому є розділення семантики комунікаційного протоколу (протоколу спілкування) від семантики повідомлень. Протокол спілкування має бути універсальним та спільним для всіх агентів. Він має бути лаконічним та мати обмежений набір примітивних комунікаційних актів.

Таким протоколом є мова запитів знань та маніпуляції (knowledge query and manipulation language, KQML) [260]. Це протокол обміну інформацією та знаннями, його елегантність полягає у тому, що вся інформація для розуміння змісту повідомлення міститься у самому спілкуванні.

Поряд зі спілкуванням, агенти потребують опису об'єктів світу. Опис на натуральній мові, такій як українська чи англійська, які можуть описати все різноманіття оточуючих об'єктів та ситуацій, є непрактичним через те, що може мати багатозначність, та є надзвичайно складним для реалізації. Символьна логіка є загальним математичним апаратом для опису речей. Точніше, проста логіка (наприклад, числення предикатів першого ступеня [127]) була визнана спроможною описати майже все корисне або таке, що цікавить людину або іншого інтелектуального агента. Це включає прості факти, визначення, абстракції, правила виведення, обмеження і навіть метазнання (знання про знання).

Як стандарт опису речей (об'єктів, дій, ситуацій) всередині експертних систем, баз даних, інтелектуальних агентів тощо, був запропонований так званий Стандарт Обміну Знаннями (Knowledge Interchange Format, KIF). Цей формат читабельний як для комп'ютерних систем, так і для людей. Більше того, цей стандарт був розроблений спеціально, щоб служити як «interlingua» (міжмова), тобто посередник у перекладах між різними мовами. KIF – це префіксна форма числення предикатів першого ступеня з розширенням для дозволу правил виведення та визначень з неповною інформацією.

Онтологія – специфікація об'єктів, концепцій та відношень. Концепції можуть бути відображені логікою першого ступеня як унарні предикати, а предикати більшої розмірності можуть зображати відношення. Онтологія – це більше ніж систематика, ієрархія класів (типів), вона має описувати відношення. Класи та відношення мають

бути подані в онтології, але екземпляри класів не мають там подаватися. Можна провести аналогію між онтологією та схемою бази даних, а не вмістом бази даних. Агент має подавати свої знання у словнику специфікованої онтології, яка є спільною для всіх агентів системи.

Описані вище механізми використовуються для обміну поодинокими повідомленнями між агентами. Протоколи взаємодії керують обміном серіями повідомлень між агентами – спілкуванням. Деякі протоколи взаємодії були розроблені для систем агентів. У випадках, коли агенти мають конкуруючі цілі, тобто вони зацікавлені в успіху тільки для себе, задача протоколів – максимізувати корисності агентів. У випадку спільних цілей або спільних задач у розподіленому розв'язанні задач (distributed problem solvig, DPS), завдання протоколів – покращити загальну цілеспрямовану продуктивність агентів без порушення їх автономності, тобто без явного глобального контролю.

10.7.2. Розподілене розв'язання задач та планування

Розподілене розв'язання задач та планування (distributed problem solving and planning) – це підгалузь розподіленого штучного інтелекту, практичне застосування якої – у спільній роботі агентів для отримання розв'язку, що потребує колективного зусилля [155]. Через властиве розділення між агентами ресурсів, таких як знання, спроможність, інформація, досвід, сам агент не може закінчити поставлену задачу краще (швидше, повніше, точніше або достовірніше), ніж працюючи з іншими разом. Розв'язуючи розподілену задачу, необхідні як груповий зв'язок (агент має «хотіти» працювати у групі), так і компетенція (агент має знати, як добре працювати у групі). Зрозуміло, що груповий зв'язок важко уявити між індивідуально-мотивованими агентами, тому агенти розробляються для розв'язку таких задач із вбудованою сталою мотивацією до роботи у групі, або «виплати» індивідуально-мотивованим агентам набагато більші у випадку роботи у групі, або соціальний інжиніринг представляє стримуючі фактори індивідуалізму агентів тощо.

10.7.3. Алгоритми пошуку

Оскільки дуже часто агенти діють у просторовому середовищі, для розв'язання цілої низки задач вони потребують алгоритмів пошуку. У цьому сенсі розподілений штучний інтелект, отже і мультиагентні системи, використовують всі математичні засоби, які використовує традиційний штучний інтелект, але це справедливо на мікрорівні, тобто рівні агента. Більшість пошукових проблем не може бути вирішена «a pri-ori» , тобто раніше, ніж почати діяти. Замість того вони вирішуються методом «спроб та помилок» альтернативних варіантів.

Фактично всі задачі, адресовані алгоритмам пошуку, можна поділити на три типи: задачі знаходження шляху (path-finding problems, типова задача: так звані «п'ятнашки»), задачі задоволення обмежень (constraint satisfaction problems, CSP, типова задача: задача про вісім ферзів), і ігри двох гравців (two-player games, типова задача: нарди, шахи).

Оскільки ігри двох гравців відображають ситуацію, в якій є два агенти, що конкурують, такі задачі мають дуже близьке відношення до розподіленого штучного інтелекту/мультиагентних систем з агентами, що конкурують.

З іншого боку, як було згадано раніше, більшість алгоритмів для перших двох класів задач (знаходження шляху та задоволення обмежень) були розроблені для одноагентного розв'язування задачі. Головною проблемою їх «розподіленого» застосування є те, що для більшості з них (наприклад, алгоритм A*) потрібно на кожній їх ітерації мати

повну інформацію про задачу, що робить беззмістовним розподілення такої задачі. Однак є деякі алгоритми (наприклад, деякі алгоритми пошуку на графі), які використовують принцип акумулювання результатів обчислення на кожній вершині графа для розв'язання поставленої задачі. Порядок виконання таких локальних обчислень не має значення або він дуже гнучкий, отже, може бути виконаний асинхронно та конкуруючи. Такі алгоритми називаються асинхронні алгоритми пошуку. Так, наприклад, є асинхронна версія відомого алгоритму бектрекінгу (asynchronous backtracking).

10.7.4. Використання генетичних алгоритмів у мультиагентних системах

Розглянемо генетичні алгоритми через призму мультиагентних систем. Тут агентів не можна назвати інтелектуальними, а розв'язок поставленої задачі досягається за рахунок еволюційних законів. Систему, що використовує суто генетичний алгоритм для максимізації функції мети, не можна назвати мультиагентною, але приклад генетичного алгоритму тут наведемо, оскільки він може бути пристосований для коригування поведінки інтелектуальних агентів. Ця тематика буде предметом подальших досліджень.

Уявимо собі мультиагентну систему, де існує певна кількість агентів, і кожен з них – це деяке розв'язання поставленої задачі. Будемо вважати, що агент тим більше пристосований, чим краще відповідає рішенню (чим більше значення цільової функції воно дає). Тоді задача максимізації цільової функції зводиться до пошуку найбільш пристосованого агента. Звичайно, ми не можемо помістити у систему відразу усіх агентів, що відповідають всім розв'язкам задачі. Їх дуже багато, а якщо б ми все ж таки це зробили, розв'язок задачі перетворився б у тривіальне повне перебирання. Натомість, ми будемо розглядати багато поколінь, що змінювалися. Тепер, якщо ми зуміємо ввести в дію природний відбір і генетичну спадковість, то отриманий світ буде підкорятися законам еволюції. Помітимо, що відповідно до визначення пристосованості, метою цієї штучної еволюції буде якраз створення якнайкращих рішень. Очевидно, еволюція – нескінченний процес, у ході якого пристосованість агентів поступово підвищується. Примусово зупинивши цей процес через достатньо довгий час після його початку і вибравши найбільш пристосованого агента в поточному поколінні, ми отримаємо не абсолютно точний, але близький до оптимального розв'язок задачі. Так можна коротко описати ідею генетичного алгоритму. Перейдемо тепер до точних визначень і опишемо роботу генетичного алгоритму детальніше. Щоб говорити про генетичну спадковість, потрібно забезпечити агентів хромосомами. У генетичному алгоритмі хромосома – це деякий числовий вектор, відповідний параметру, що підбирається, а набір хромосом певного агента визначає розв'язок задачі. Які саме вектори треба розглядати в конкретній задачі, вирішує проектувальник системи. Кожна з позицій вектора хромосоми називається ген.

Визначимо тепер поняття, що відповідають мутації та кросинговеру в генетичному алгоритмі. Мутація – це перетворення хромосоми, що випадково змінює одну або декілька її позицій (генів). Найбільш поширений вид мутацій – випадкова зміна тільки одного з генів хромосоми. Кросинговер (у літературі з генетичних алгоритмів також вживається назва кросовер або схрещування) – це операція, при якій із двох хромосом породжується одна або декілька нових хромосом. У найпростішому випадку кросинговер у генетичному алгоритмі реалізується так само, як і в біології. При цьому хромосоми розрізають у випадковій позиції та обмінюються частинами між собою. Наприклад, якщо хромосоми (1, 2, 3, 4, 5) і (0, 0, 0, 0, 0) розрізати між третім і четвертим генами

і обміняти їх частини, то вийдуть нащадки (1, 2, 3, 0, 0) і (0, 0, 0, 4, 5). Спочатку генерується початкова популяція агентів, тобто деякий набір розв'язків задачі. Як правило, це робиться випадковим чином. Потім необхідно змоделювати розмноження всередині цієї популяції. Для цього випадково відбираються декілька пар агентів, виробляється схрещування між хромосомами в кожній парі, а отримані нові хромосоми вміщуються в популяцію нового покоління. У генетичному алгоритмі зберігається основний принцип природного відбору – чим агент більш пристосований, тим з більшою ймовірністю він буде брати участь у схрещуванні. Тепер моделюються мутації – в декількох випадково вибраних агентів нового покоління змінюються деякі гени. Потім стара популяція частково або повністю знищується і ми переходимо до розгляду наступного покоління. Популяція наступного покоління в більшості реалізацій генетичних алгоритмів містить стільки ж агентів, скільки початкова, але через відбір пристосованість в ній в середньому вища. Тепер описані процеси відбору, схрещування і мутації повторюються вже для цієї популяції і так далі. У кожному наступному поколінні ми будемо спостерігати виникнення нових розв'язків нашої задачі. Серед них будуть як погані, так і гарні, але завдяки відбору число хороших рішень буде зростати.

10.7.5. Методи проектування структури мультиагентної системи

Побудувати мультиагентну систему, в якій агенти будуть «робити те, що вони мають робити», виявляється надзвичайно складно. Єдиний спосіб розв'язати таку задачу – це розробити таку систему взаємодій агентів, що буде максимально влаштовувати у конкретній ситуації. У такому випадку запитання «коли, як і з ким взаємодіяти?» перетворюється на запитання «коли, як і які агенти мають взаємодіяти, кооперуватися чи змагатися, щоб успішно досягти завдань, для вирішення яких система була спроектована?». Ґрунтуючись на загальному розмежуванні «мікрорівня» системи, чи рівня агента, та «макрорівня», чи рівня групи агентів, можна виділити два різні шляхи до відповіді на поставлене запитання:

- «знизу-догори»: знаходження специфічних для поставлених цілей можливостей мікрорівня, тобто знаходження методів наділити агентів здібностями правильно взаємодіяти на загальному макрорівні, тобто у групі;
- «згори-донизу»: знаходження специфічних групових правил, так званих правил поведінки, норм і так далі, які дозволяють обмежити набір можливих взаємодій між агентами, тобто на мікрорівні.

Такі запитання як «рівень агент – індивідуум – діяльність» та «рівень групи – суспільство – правила та структури» пов'язані між собою, відоме як «мікро-, макропроблема» у соціології.

Незважаючи на те, який шлях з вищезгаданих був обраний, загальне запитання розростається до набору перспективних пов'язаних запитань.

1. Як дати можливість агентам розкладати на частини свої цілі та задачі з метою визначення підцілей і підзадач та їх призначення іншим агентам, а також синтезування часткових результатів та рішень?
2. Як дати можливість агентам спілкуватися? Які мови спілкування та протоколи використовувати?
3. Як дати можливість агентам формулювати та розмірковувати про дії, плани та знання інших агентів з метою відповідно з ними взаємодіяти?

4. Як дати можливість аґентам формулювати та розмірковувати про стан процесу їхньої взаємодії? Як їм дати можливість визначити, чи вони досягли прогресу в їх погоджених зусиллях, та як дати можливість аґентам покращувати стан їх погодженості та діяти синхронно?
5. Як дати можливість аґентам визначати та зводити до погодженості точки зору те, що не можна співставити? Як синтезувати погляди та результати?
6. Як розробити та обмежити практичну мультиаґентну систему? Як розробити технологічну платформу та розробницькі технології для розподіленого штучного інтелекту?
7. Як ефективно балансувати між локальними обрахунками та взаємодією?
8. Як уникнути або пом'якшити згубну поведінку всієї системи (наприклад, хаотичну або таку, що коливається)?
9. Як дати можливість аґентам вести переговори та контактувати? Які протоколи переговорів та контактів використовувати?
10. Як дати можливість аґентам формувати та розбивати організаційні структури – команди, альянси тощо, які б підходили для досягання цілей і для завершення задач?
11. Як формально описати мультиаґентну систему і взаємодію між аґентами? Як упевнитися, що вони коректно специфіковані?
12. Як реалізувати такі «інтелектуальні процеси», як рішення завдань (problem solving), планування (planning), прийняття рішень (decision making) та навчання (learning)?

Знаходити відповіді на поставлені запитання – це основне призначення розподіленого штучного інтелекту.

10.8. Моделі мультиаґентних систем

Термін «мультиаґентні системи» використовується для позначення систем, що складаються із множини автономних модулів – аґентів, з наступними властивостями:

- кожен аґент має здатність мислити;
- немає центрального керування аґентами;
- джерела даних і доступ до них децентралізовані;
- робота аґентів асинхронна.

Одним із чинників інтересу до мультиаґентних систем став розвиток мережі Інтернет, яка є відповідним середовищем виконання для розподілених автономних програмних систем. Для успішного функціонування в такому середовищі аґенти мають бути здатні вирішувати два основні завдання: уміти знаходити один одного й уміти взаємодіяти.

Мультиаґентні системи створені для вирішення різних завдань штучного інтелекту, в яких присутні декілька учасників.

Сучасний підхід до штучного інтелекту заснований на понятті раціонального аґента, який завжди прагне оптимізувати відповідну міру корисності своїх дій. Наприклад, люди можуть розглядатися як аґенти з очима – сенсорами, які здатні виконувати певні дії за допомогою рук. Роботи здатні сприймати світ через камери і пересуватися за допомогою коліс. Для програм графічний інтерфейс є засобом і сприйняття, і дії. Проте, аґенти рідко є поодинокими системами. Частіше вони взаємодіють один з одним. Системи, що містять групу аґентів, які можуть взаємодіяти між собою, і називаються мультиаґентними системами.

10.8.1. Характеристики мультиаґентних систем

Які характеристики відрізняють мультиаґентні системи від одноаґентних? Можна виділити ряд відмінностей.

Структура аґента. Часто аґенти сконструйовані по-різному, наприклад, програми, написані різними людьми. Відмінності можуть полягати в техніці та програмному забезпеченні. Часто таких аґентів називають гетерогенними в протилежність гомогенним, які сконструйовані ідентичним чином і мають спочатку однакові можливості. Проте відмінність не так очевидна: аґенти, засновані на однаковій техніці та програмах, але такі, що поводять себе по-різному, теж можуть називатися гетерогенними.

Оточення аґентів може бути статичним (незалежним від часу) або динамічним (нестаціонарним). Через простоту опрацювання і строгіший математичний опис більшість методів штучного інтелекту в одноаґентному випадку були розроблені для статичного оточення. У мультиаґентних системах сама присутність декількох аґентів робить оточення динамічним.

Сприйняття. Інформація, якою володіє система аґентів, є зазвичай розподіленою: аґенти можуть стежити за оточенням з різних положень, отримувати інформацію в різні моменти часу або інтерпретувати цю інформацію по-різному. Отже, стан оточення є частково осяжним для кожного аґента.

10.8.2. Раціональний аґент

Тут буде описана проблема прийняття оптимального рішення. Це означає, що аґентові на кожному кроці слід вибрати якнайкращу дію, виходячи з того, що йому відомо про навколишній світ. Вводиться міра корисності дій аґента, яку він постійно намагається оптимізувати. Раціонального аґента також називають розумним. Надалі ми, головним чином, розглядатимемо лише обчислювальних аґентів, тобто тих, що створені для вирішення спеціальних завдань і виконуються на обчислювальних пристроях.

Прийняття рішення аґентом

Припустимо, що на кожному часовому кроці $t = 1, 2..., \infty$ аґент може з кінцевого набору можливих дій A вибрати якусь дію a_t. Інтуїтивно зрозуміло, що, щоб діяти раціонально, аґент повинен оцінювати і минуле, і майбутнє під час вибору подальших дій. Під минулим розуміємо те, що аґент сприйняв і які дії зробив до моменту часу t, а під майбутнім – чого він чекає і що збирається потім робити. Позначимо o_t спостереження аґента у момент часу T, тоді для вибору оптимальної дії у момент часу t в загальному випадку необхідно використовувати всю історію спостережень o_t і історію дій a_t, що передує моменту часу t.

Функція $p(o_1, a_1, o_2, a_2, ..., o_t) = a_t$, яка відображає набір пар «спостереження–дія» до моменту часу t в оптимальну дію a_t називається **стратегією аґента**.

Якщо зможемо знайти функцію p, що здійснює таке відображення, то частина завдання про відшукання оптимального рішення на основі минулого буде вирішена. Проте, визначення і реалізація такої функції вельми проблематичні; складна історія може

містити велику кількість пар спостережень, які можуть змінюватися від одної задачі до іншої. Більш того, збереження всіх спостережень вимагає дуже великого об'єму пам'яті і, відповідно, приводить до зростання складності обчислень. Цей факт приводить до необхідності використання простіших стратегій. Наприклад, агент може ігнорувати всю історію спостережень, за винятком останнього. У цьому випадку стратегія приймає вигляд $p(o_t) = a_t$, що відображає поточне сприйняття агента в дію a_t.

Агент, який відображає поточне сприйняття o_t в нову дію a_t, називається **рефлексією**, а його стратегію називають **реактивною** або **стратегією без пам'яті**.

Виникає природне питання: наскільки хорошим може бути такий рефлексійний агент? Як ми побачимо далі, він може бути доволі непоганим.

Оточення і властивість Маркова

Із сказаного вище виходить, що поняття оточення і агента є спареними, так що одне поняття не може бути визначене без іншого. Фактично, відмінність між агентом і його оточенням не завжди виразна, і це іноді ускладнює проведення чіткої межі між ними. Для простоти припустимо, що існує світ, в якому є один або більше агентів, в якому вони сприймають, думають і діють.

Колективна інформація, яка міститься в навколишньому світі у момент часу t і яка важлива для виконуваного завдання, називається **станом світу** і позначається s_t. Множину станів світу позначимо через S.

Залежно від природи задачі, світ може бути дискретним або неперервним. Дискретний світ характеризується кінцевим числом станів. Прикладом може служити гра в шахи. З іншого боку, прикладом неперервного світу може служити мобільний робот, який може вільно пересуватися в декартовій системі координат. Більшість з наявних технологій штучного інтелекту створені для дискретного світу, тому ми надалі розглядатимемо саме його.

Спостереження

Важлива властивість, що характеризує світ з погляду агента, пов'язана зі сприйняттям. Ми говоритимемо, що світ повністю спостережуваний, якщо поточне сприйняття агента o_t повністю описує стан світу s_t. У протилежність цьому, в частково спостережуваному світі поточне сприйняття o_t описує лише частину інформації про стан світу, що задає імовірнісний розподіл $P(s_t \mid o_t)$ між дійсними станами світу. Отже, s_t можна розглядати як випадкову величину, розподілену на S. Часткове спостереження може бути викликане двома фактами. Перший з них – це перешкоди в сенсорній інформації агента. Наприклад, внаслідок недосконалості сенсорів, в різні моменти часу агент може сприймати один і той самий світ по-різному. Можливо також, що для агента деякі стани визначити неможливо.

Властивість Маркова

Розглянемо знову рефлексивного агента з реактивною тактикою $p(o_t) = a_t$ у світі, який повністю можна оглянути. Припущення оглядовості означає, що $s_t = o_t$, і, отже, тактика агента $p(s_t) = a_t$. Іншими словами, в оглядовому світі тактика рефлексивного агента, є відображенням із станів світу в дію. Вигода полягає в тому, що у багатьох завданнях стан світу у момент часу t дає повний опис історії до моменту часу t.

Про такий стан світу, який містить всю важливу інформацію про минуле у конкретному завданні, говорять, що воно є марківським або володіє властивістю Маркова.

З вищесказаного ми можемо зробити висновок, що у марківському світі агент може безпечно використовувати стратегію без пам'яті для ухвалення рішення замість теоретично оптимальної стратегії, яка може вимагати багато пам'яті.

Досі ми розглядали, як стратегія агента може залежати від останньої події і окремих характеристик оточення. Проте, як ми говорили спочатку, ухвалення оптимального рішення може також залежати від оцінки майбутнього.

Стохастичні переходи

У кожен момент часу t агент вибирає дію a_t з кінцевої множини дій A. Як наслідок після вибору агентом дії світ змінюється. Модель переходу (іноді називають моделлю світу) визначає, як змінюється світ після виконання дії. Якщо поточний стан світу s_t, і агент здійснює дію a_t, ми можемо виділити два випадки.

1. У детермінованому світі модель переходу відображає єдиним чином пару «стан–дія» (s_t, a_t) у новий стан s_{t+1}. У шахах, наприклад, кожен хід змінює ігрову позицію.
2. У стохастичному світі модель переходу відображає пару «стан–дія» у розподіл ймовірності $p(s_{t+1} | s_t, a_t)$ станів. Як і в розглянутому вище випадку з частковою оглядовістю, s_{t+1} – це випадкова величина, яка може приймати всі можливі значення з множини S з відповідною ймовірністю $p(s_{t+1} | s_t, a_t)$. Найпрактичніші застосування включають стохастичні моделі переходу, наприклад, рух робота неточний через те, що його колеса ковзають.

Іноді часткова оглядовість є наслідком неточності сприйняття агентом його оточення. Тут ми бачимо інший приклад, де важлива неточність, а саме, світ змінюється, коли здійснюється дія. У стохастичному світі ефект дії агента не відомий наперед. Натомість є випадковий елемент, який визначає, як змінюється світ внаслідок дії. Зрозуміло, що стохастичність під час переходу з одного стану в інший вносить додаткові складнощі в завдання ухвалення оптимального рішення агентом.

Для класичного штучного інтелекту метою окремої задачі є досягнення бажаного стану світу. Отже, **планування** визначається як пошук оптимальної стратегії. Коли світ детермінований, планування переходить в пошук за графом для кожного варіанту наявного методу. У стохастичному ж світі може не відбуватися перехід в простий пошук за графом, оскільки перехід зі стану в стан не є детермінованим. Тепер агент під час планування може враховувати неточності переходів. Щоб зрозуміти, як це може бути використано, відзначимо, що в детермінованому світі агент віддає перевагу за замовчанням кінцевому стану. У загальному випадку, агент зберігає настройки при переході зі стану в стан. Наприклад, робот, що грає у футбол, прагнутиме забити гол, прагнутиме менше стояти з м'ячем перед порожніми воротами і т.ін. Щоб формалізувати це поняття, пов'яжемо з кожним станом s дійсне число $U(s)$, зване корисністю стану цього агента. Формально, для двох станів s і s^* виконується $U(s) > U(s^*)$ тоді і лише тоді, коли агент віддає перевагу стану s стану s^*, а $U(s) = U(s^*)$, якщо для агента ці стани невиразні. Інтуїтивно зрозуміло, що чим вища корисність, тим вигідніший стан, в якому перебуває агент. Відзначимо, що в мультиагентних системах бажаний стан для одного з агентів може бути не бажаним для інших. Наприклад, у футболі забивання гола є небажаним для протилежної команди.

Прийняття рішення в стохастичному світі

Тепер виникає запитання, як агент може ефективно використовувати функції корисності для ухвалення рішення. Припустимо, що світ стохастичний з моделлю переходу

$p(s_{t-1}|s_t, a_t)$, що знаходиться у стані s_t доти, поки агент обдумує, яку дію йому зробити. Нехай $U(s)$ – функція корисності стану якогось агента. Припустимо, що у нас тільки один агент. Тоді прийняття рішення засноване на припущенні, що оптимальна дія агента повинна бути максимумом корисності, тобто

$$a^*_t = \max\sum p(s_{t+1}|s_t, a_t)\cdot U(s_{t+1}),$$

де сумування виконується по всіх станах s_{t+1}. Щоб побачити, наскільки корисна дія, агент повинен помножити корисність кожного можливого кінцевого стану на ймовірність потрапляння в цей стан, і потім підсумувати отриманий результат. Тоді агент може вибрати дію a^*_t з максимальною сумою. Якщо кожен стан світу має величину корисності, агент може провести обчислення і вибрати оптимальну дію для кожного можливого стану. Тоді агент зі стратегією може оптимально переводити стан в дію.

10.8.3. Теорія ігор

Тут ми обговоримо прийняття рішення в мультиагентних системах, де група агентів взаємодіє і одночасно ухвалює рішення. Ми використаємо теорію ігор для аналізу завдання, зокрема, стратегічних ігор, де ми вивчимо ітеративне виключення точно певних дій і рівновагу Неша. Дія агента на оточення може залишатися невизначеною, і це може накладати невизначеність на прийняття рішення. У мультиагентних системах, де агенти ухвалюють рішення одночасно, агент може не знати про рішення інших задіяних агентів. Зрозуміло, що агент повинен діяти залежно від дій інших агентів.

Мультиагентне прийняття рішення є предметом теорії ігор. Теорія намагається передбачити поведінку агентів, що взаємодіють, у стані невизначеності і ґрунтується на двох припущеннях. По-перше, агенти, що взаємодіють раціональні, а по-друге, прийняття рішення відбувається залежно від рішення інших агентів. Залежно від способу прийняття рішення агентами, ми можемо виділити два типи гри. У стратегічній грі кожен агент вибирає свою стратегію тільки один раз на початку гри, а потім всі агенти діють одночасно. У грі узагальненої форми агенти можуть змінювати стратегію впродовж всієї гри. Інша відмінність полягає у повноті інформації агента про інших агентів. Розглядатимемо тільки ігри, де агент володіє всією інформацією, що стосується інших агентів.

Стратегічні ігри

Стратегічна гра (також звана нормальною формою) є простою моделлю взаємодії агентів в теорії ігор. Її можна розглядати як мультиагентне розширення попередньої моделі, що характеризується такими властивостями.

1. Агентів не менше одного ($n > 1$).
2. Кожен агент i вибирає дію a_i, звану стратегією, з власної множини дій A_i. Вектор $(a_1, ..., a_n)$, тобто дія, називається сумісною дією і позначається (a_i). Ми користуватимемося позначенням a_{-i} для позначення всіх дій агента, окрім i-ї, а також (a_{-i}, a_i) для позначення сумісної дії, де агент i здійснює дію a_i.
3. Гра відбувається у фіксованому стані s. Стан може бути визначений як такий, що містить n агентів, їх множину дій A_i і їх виграші.
4. Кожний агент i має власне значення функції дії $Q^*_{i(s, a)}$, яка обчислює корисність сумісної дії для агента i. Кожний агент може віддавати різні переваги різним спільним діям. Ми припускаємо, що функція виграшу наперед визначена і фіксована.

5. Стан повністю оглядовий для агентів. Тобто, вони знають один про одного, множини дій один одного і виграші один одного. Все це в грі і є загальним знанням агентів.
6. Кожний агент вибирає єдину дію. Більш того, агенти вибирають свої дії одночасно і незалежно. Жоден агент не знає про прийняте рішення іншого агента доти, поки рішення не буде прийнято.

У результаті, в стратегічних іграх кожний агент вибирає одну дію і отримує результат залежно від вибору загальної дії. Ця спільна дія називається результатом гри. Хоча функція виграшу агента є загальним знанням, агент не знає наперед про вибір дії іншим агентом. Найкраще, що він може, – спробувати вгадати дію іншого агента. Рішенням гри є прогноз результату, використовуючи припущення, що всі агенти раціональні і стратегічні.

Ітеративне виключення дій, що строго домінують

Перше важливе поняття засноване на припущенні, що раціональний агент ніколи не вибирає субоптимальне рішення. Під субоптимальним рішенням ми розуміємо дію, яка незалежно від того, що роблять інші агенти, завжди буде в результаті менш виграшною для агента, ніж якась інша дія.

Говоритимемо, що дія a_i агента i строго домінує над іншою дією a'_i, якщо $u_i(a_{-i}, a'_i) > u_i(a_{-i}, a_i)$ для всіх дій a_{-i} інших агентів.

У цьому визначенні $u_i(a_{-i}, a_i)$ є виграшем агента i, якщо він вибере дію a_i, поки решта агентів виконують дію a_{-i}.

Характеристикою ітеративного виключення дій, що строго домінують є те, що агенти можуть не зберігати віру в стратегії інших агентів для того, щоб обчислити їх оптимальну дію. Єдине, що потрібно – це припущення про те, що всі агенти раціональні.

Рівновага Неша

Рівновага Неша є важливішим поняттям, ніж ітеративне виключення дій, що строго домінують в тому сенсі, що воно породжує точніший прогноз результатів у широкому класі ігор. Це формально може бути визначено таким чином.

Рівновага Неша – це загальна дія a^* така, що для кожного агента i виконується $u_i(a^*_{-i}, a^*_i) \geq u_i(a^*_{-i}, a_i)$ для всіх дій $a_i \in A_i$.

Відзначимо, що на відміну від ітеративного виключення дій, що строго домінують, яке описує рішення за допомогою алгоритму, рівновагу Неша описує рішення в термінах можливих станів. Рівновага Неша кожного агента є оптимальною відповіддю на дію іншого агента.

Спільна дія a називається оптимальною дією Парето, якщо ні для якої з дій a не виконується $u_i(a') > u_i(a)$ для будь-якого агента i.

Вище ми припустили, що під час гри агент i вибиратиме дію з множини його дій A_i. Проте, це не завжди так. У багатьох випадках у агента є причина проявляти стохастичність у своїй поведінці. У загальному випадку ми можемо припустити, що агент i вибирає дію a_i у процесі з деяким розподілом ймовірності $p_i(a_i)$, який може бути різним для різних агентів.

Змішана стратегія агента i – це розподіл ймовірності дій $a_i \in A_i$.

У своїй відомій теоремі Неш показав, що стратегічна гра з кінцевою кількістю агентів і скінченною кількістю дій завжди досягає рівноваги у змішаних стратегіях.

10.8.4. Координація

Розглянемо завдання координації, тобто як окреме рішення агента може вплинути на прийняття загального рішення. Майбутнє мультиагентних систем фактично означає, що процес ухвалення рішення може бути поширеним. Це означає, що немає центрального агента, який контролює, що робить кожний агент у будь-який момент часу. Кожний агент сам вирішує, яку дію він виконає. Координація реалізується за допомогою розподілу агентів за ролями. Тобто агент, розуміючи, що він найбільше підходить для виконання певної ролі, привласнює її собі. Крім того, може існувати наперед певний набір правил, якими повинні керуватися агенти. Наприклад, таким правилом є пропускання перешкоди з правої сторони у правилах дорожнього руху. Іншим прикладом може слугувати гра роботів у футбол. Команда роботів прагне забити м'яч у ворота протилежної команди. Тут координація забезпечує те, що, наприклад, два роботи з однієї команди не намагаються вдарити м'яч одночасно. Тут немає агента, який контролює і міг би інструктувати роботів у реальному часі бити або не бити м'яч. Вони самі повинні це вирішувати.

10.8.5. Загальне знання

Раніше ми припускали, що світ повністю оглядовий. Тепер послабимо це припущення і розглянемо випадок, коли деякі стани невидимі для агента. У світі, що частково оглядовий, агент повинен завжди розмірковувати про те, що він знає про навколишній світ, і про те, що знають інші агенти, перш ніж прийняти рішення. Щоб діяти раціонально, агент повинен завжди реагувати на те, що він дізнається про поточний стан світу. Якщо світ повністю оглядовий, то агент може діяти, практично не роздумуючи. У світі, що частково оглядовий, агент повинен уважно розглядати, що він знає і що не знає, перед вибором дії. Інтуїтивно зрозуміло, що чим більше агент знає про стан, тим краще рішення він може прийняти. У мультиагентних системах раціональний агент може враховувати знання інших агентів про стан (тобто інформацію, якою вони володіють про стан). Також він повинен розглядати те, що інші агенти знають про його дії, і чи знають вони про те, що він знає про них. Те, що два раціональні агенти завжди хочуть укласти парі, не є загальним знанням.

Як приклад наведемо головоломку про шапки. Троє агентів (наприклад, дівчата) сидять навколо столу, на кожній з них одягнена шапка, яка може бути або білою, або червоною. Кожен агент знає колір шапки двох інших, але не знає кольору своєї. Людина, яка спостерігає за ними з боку, просить їх обернутися і запитує, чи знають вони, якого кольору їх шапка. Кожний агент дає негативну відповідь. Тоді ведучий оголошує, що, принаймні, на одному з них червона шапка, а потім знову просить їх обернутися. Перший агент говорить «Ні», другий агент говорить «Ні», але, коли він питає третього агента, то той відповідає «Так». Як сталося, що третій агент нарешті може визначити колір своєї шапки?

Якщо у агентів однакові стратегії та їм відомо про те, що роблять агенти в поточному стані, то агенти повинні однаково діяти.

10.8.6. Комунікація

Мультиагентна взаємодія часто асоціюється з деякою формою спілкування. Ми використовуємо спілкування у повсякденному житті для сумісного вирішення завдань, для того, щоб домовлятися з іншими, або просто для обміну інформацією з іншими. Як ми

бачили, у головоломці про капелюхи шляхом відповідей на запитання агенти змогли сформулювати точніше твердження задачі і, врешті-решт, вирішити її.

Коли ми говоримо про обчислювальних агентів, спілкування розглядається на деякому рівні абстракції. На нижчому рівні (network) можна бути впевненим, що повідомлення, що пересилаються агентами один одному, безпечно і вчасно дійдуть до адресатів. Для цього існують протоколи. На середньому мовному рівні можна ґрунтуватися на безлічі основних слів мови та їх стандартних змінах, щоб агенти, що говорять однією мовою, могли розуміти один одного.

І останній рівень – рівень застосування, тобто можна ефективно використовувати спілкування для вирішення стандартних мультиагентних задач, наприклад, для координації або узгодження.

Як ми вже зрозуміли, стан світу характеризується кількістю агентів, їх можливими діями і вигодою агентів, якщо вони залучені у стратегічну гру, й іншими аспектами навколишнього оточення. У таких випадках деякий агент може бути дезактивований в деяких станах, якщо йому присвоєна деяка роль.

Аналогічно, якщо стан світу частково оглядовий для агентів, кожний агент володіє різними рівнями знання про справжній стан світу. Формально можна описати комунікацію розглядом кожної первісної комунікації як дії, яка оновлює знання агентів про стан. Найзагальніші типи процесу комунікації (вони використовуються, наприклад, у правилах дорожнього руху):

- інформативні;
- застережливі;
- підтверджувальні;
- заборонні;
- вказівні.

Кожний процес комунікації може вплинути на знання агентів по-різному. З іншого боку, він може бути використаний для інформування про вибір дій агентом. Можна дати просту інтерпретацію іншому процесу комунікації. Наприклад, заборона може відіграти роль, що забороняє деактивацію деякої дії в окремій ситуації. Вказівний процес може виявлятися в організованій групі агентів, в якій агент з великим авторитетом може давати команди іншим агентам. Деякі мови комунікації агентів, спрямовані на стандартизацію процесу комунікації, можуть застосовуватися серед декількох агентів.

Дію можна розглядати як таку, що змінює знання залученого агента про стан. Розглянувши множину можливих комунікаційних актів, агент повинен задуматися над питанням, який акт використовувати і кому передавати інформацію. Ми приписуємо телекомунікаційному акту індикатор корисності. Але звідки ми візьмемо це значення? Структура, яка дозволяє належним чином визначити значення комунікації, – це Байєсова гра. Ця модель є стратегічною грою з частковою оглядовістю. Ми маємо деяку кількість агентів, безліч станів світу, різну для кожного агента функцію інформації. Також припускаємо, що кожне потрапляння в стан відбувається з деякою ймовірністю, рівною для всіх агентів, і що це визначає стратегічну гру з відповідним виграшем. Отже, агенти можуть обчислювати значення всіх можливих комунікаційних актів в деякій ситуації та потім вибрати одну, у якої значення найбільше. На практиці прийняття оптимального рішення, що передавати і кому, може зажадати великих обчислювальних витрат, які можуть зменшити здібності агента.

Перевагою використання комунікації є те, що більше не потрібно намагатися вгадувати дії інших агентів. Шляхом комунікації агенти можуть визначити здібності один одного і розподілити ролі в грі. На практиці, проте, це не завжди можливо. Наприклад, у світі, що частково є оглядовим. Коли спілкування між агентами можливе, агенти самі обчислюють свої здібності до тієї або іншої ролі, а потім посилають цю інформацію решті агентів.

10.8.7. Структура агента

Розглянемо мультиагентну систему, в якій агенти можуть кооперуватися. Той факт, що агенти об'єднюються для досягнення загальної мети, приводить нас до розроблення алгоритму, подібного до координаційного алгоритму, в якому агенти прагнуть передати якомога більше інформації про себе і поводитися за інструкцією. Футбольний робот, наприклад, ніколи не почне привласнювати чужу роль, оскільки це може потенційно завдати шкоди всій команді. На практиці, проте, ми часто маємо справу із самозацікавленими агентами, наприклад, агентами, які захищають інтереси власника, що хоче максимізувати свій прибуток. Типовий приклад подібного програмного агента – це електронний аукціон в Інтернеті. Розроблення алгоритму або протоколу для подібної системи – це завдання, яке зустрічається частіше, ніж в кооперативному випадку.

По-перше, ми повинні керувати агентом, беручи участь у протоколі, який наперед теж не відомий. По-друге, потрібно приймати до уваги факт, що агент може спробувати почати використовувати протокол у власних інтересах, приводячи до субоптимального результату. Це не виключає можливості того, що агент може почати обманювати, якщо це необхідно.

10.8.8. Навчання

Припустимо, що наш світ стохастичний, і нехай для кожного стану світу визначена якась функція $R(s)$, яка визначає корисність стану s для нашого агента. Агент хоче знайти таку послідовність дій $\{a_i\}$ (політику, позначену як p), щоб у результаті він прийшов в максимально корисний стан. Тобто він хоче знайти максимум по всіх можливих політиках величини математичного сподівання суми ряду з $R(s_t)$ при t від одиниці до нескінченності зі зменшувальними коефіцієнтами g^t (для того, щоб ряд зійшовся).

Математичне сподівання використовується, оскільки світ не детермінований. Інакше нам були б однозначно відомі всі переходи світу між станами після якоїсь нашої дії, і ми б пошуком у графі переходів знайшли оптимальний шлях.

Простим і ефективним способом обчислення найбільшої корисності є метод ітерації привласнення. Наприклад, використовуючи його в лабіринті, де ями позначені малою корисністю –10, всі клітинки – корисністю –1/30, а фінішні клітинки (яких потрібно досягти) – корисністю 10, отримаємо оптимальну корисність і оптимальну політику.

Щоб порахувати $R(s)$ і $U^*(p)$, потрібно брати як початкові якісь більш-менш розумні значення $R(s)$ і за ними рахувати $U(s)$ – корисність станів – в процесі діяльності агента. Ці $U(s)$ обчислюються покроковим присвоєнням:

$$U(s) := R(s) + g \cdot \max \{\sum p(s' \mid s, a) \cdot U(s')\}$$

де максимум береться по всіх a, а сума – по всіх s'. Присвоєння відбувається доти, поки з деякою точністю вони не збіжаться до чогось. Тоді й отримаємо відповідь – найкраща політика.

Q-learning-алгоритм

У Q-learning-алгоритмі замість корисності стану для агента $U(s)$ обчислюються величини корисності дії a в стані світу s – $Q(a,s)$. Ці величини теж спочатку вважаються випадковими, а потім збігаються до деяких значень в ході ітерацій присвоєння після кожної дії агента в процесі навчання.

$$Q(s, a) := (1 - \lambda) \cdot Q(s, a) + \lambda \cdot (R + \gamma \max Q(s', a')),$$

де $\lambda \in (0, 1)$ – частка, на яку ми дозволяємо кожному присвоєнню змінювати значення $Q(s,a)$.

Отже, наука «розподілений штучний інтелект», у тому числі напрям «мультиагентні системи», дуже динамічно та продуктивно розвивається, хоча це молода галузь. Як було відзначено, існує ціла низка напрямів сучасної науки для застосування розроблених та адаптованих методів та моделей. Актуальність досліджень у цій галузі безумовно підтверджується потребою в таких інструментах з боку згаданих напрямів наук. Дослідження соціальних процесів, причин їх виникнення та модифікації в процесі еволюції суспільств та спільнот, відіграють важливу роль для розвитку науки. Також існують практично корисні застосування цих методів, адже є такі задачі, які за своєю природою є розподіленими, наприклад оптимізація руху транспорту в певному районі міста.

З іншого боку, актуальність досліджень обумовлена ще й тим, що залишається багато невирішених або неформалізованих питань.

Запитання для повторення та контролю знань

1. Що таке агент?
2. Як агент взаємодіє із середовищем?
3. Чим відрізняються функції агента від програми агента?
4. Який агент називається раціональним?
5. Що таке проблемне середовище?
6. Які властивості проблемного середовища?
7. Наведіть приклади проблемних середовищ і дайте їх характеристику.
8. Яка структура агента?
9. Які агенти називаються простими рефлексними?
10. Який агент називається таким, що заснований на моделі?
11. Який агент називається таким, що заснований на меті?
12. Який агент називається таким, що заснований на корисності?
13. Які агенти називаються такими, що навчаються?
14. Що таке розподілений штучний інтелект?
15. Які причини здійснення досліджень в області розподіленого ШІ?
16. Наведіть приклади ПО, для яких будують МАС.
17. Зробіть аналіз сучасних досліджень у розробленнях мультиагентних систем. Наведіть приклади систем.

18. Які Ви знаєте моделі «штучного життя»?
19. Наведіть модель ПоліСвіт Л. Ягера. Які переваги і недоліки?
20. Опишіть проект ERL.
21. Яка існує систематика різних способів координації поведінки і активності агентів?
22. У чому полягає розподілене розв'язування задач та планування?
23. Як використовуються генетичні алгоритми в мультиагентних системах?
24. Які Ви знаєте методи проектування структури мультиагентної системи?
25. Наведіть характеристики мультиагентних систем.
26. Як відбувається прийняття рішення агентам?
27. Що таке оточення і властивість Маркова?
28. Наведіть приклади випадків світів, у яких агент приймає рішення.
29. Як відбувається прийняття рішення у стохастичному світі?
30. Як використовують МАС в теорії ігор?
31. Як відбувається ітеративне виключення дій, що суворо домінують?
32. Як здійснюється координація в МАС?
33. Які види комунікацій агентів в МАС?
34. Опишіть алгоритми навчання МАС.

РОЗДІЛ 11

ПРИКЛАДНЕ ВИКОРИСТАННЯ ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

- Інтелектуальний пошук в мережі Інтернет
- Онтологія як засіб формалізації та алгоритмізації знань в інтелектуальній системі
- Технологія розроблення онтологій в редакторі Protégé
- Створення та експлуатація онтології
- Основні задачі, пов'язані з опрацюванням природної мови

У цьому розділі розглядається прикладне використання інтелектуальних систем, зокрема під час пошуку необхідної інформації, для формалізації та алгоритмізації знань, опрацювання природної мови.

11.1. Інтелектуальний пошук в мережі Інтернет

11.1.1. Стандарти подання документів в Інтернет

Говорячи про структуру і семантику тексту, підкреслимо, що HTML-мова описує документи в Інтернеті й була створена для того, щоб структурувати інформацію в документі, але в міру своєї еволюції зараз HTML-мова є засобом опису вигляду, в якому документ повинен бути наданий користувачу, тим самим здійснивши перехід від рівня зберігання до рівня уявлення. Оскільки більшість документів, поданих в мережі, зберігається у форматі HTML, завдання видобування знань стає нетривіальним. Іншими словами, сучасний Інтернет неінтелектуальний за своєю природою, оскільки зберігає не чисті знання, а дані, з яких знання треба одержувати евристичними методами. Проте, зараз існує декілька проектів щодо «інтелектуалізації» Інтернету – впровадження новітніх інформаційнійних технологій та інтелектуальних систем. Перший з них полягає в розширенні стандартної HTML спеціальними семантичними тегами, які дозволять зберігати знання в чистому вигляді безпосередньо в документах. Такі документи всередині себе вже мають інформацію про взаємозв'язки понять у семантичних атрибутах. Ідея створення розширеного HTML знайшла втілення в такому стандарті, як розроблений в W3C (інтернаціональний всесвітній Web-консорціум) мову XML (Extensible Markup Language). XML – мова для розмітки синтаксичної структури документів, що дозволяє завдяки специфікації синтаксису використовувати такі документи безлічі агентів, для яких цей формат є загальним. Щоб анотувати документи за допомогою XML, розроблений формат опису ресурсів RDF (Resource Description Framework). Метаінформація, яка визначена форматом RDF, розміщується як додаткова сторінка або блок всередині кожної Web-сторінки (елементи Web-сторінки не можуть бути анотовані прямо в тексті початкового документа, а мають бути повторені з додатковою метаінформацією). Такий

спосіб має багато труднощів через дублювання інформації. Інший цікавий підхід описаний в роботах [217] і [219]. Автори пропонують розширити HTML з метою отримання семантичних індексів до інформації, організованої у вигляді так званих Lightweight Deductive Databases, де зв'язки між окремими сторінками визначаються гіпертекстовими посиланнями з атрибутами. Дедуктивні бази даних є розширенням реляційних за рахунок вживання правил логічного програмування для складнішого подання даних [206]. Ще одну парадигму пропонує Тім Бернерс-Лі, винахідник World Wide Web, директор консорціуму W3C. Його проект називається «Semantic Web» і описує концепцію нового Web, в якому документи не тільки посилатимуться один на одного, але при цьому також розпізнаватиметься значення інформації в тих документах, на які поставлено посилання.

11.1.2. Використання онтологій

За визначенням онтологія – це специфікація концептуалізації, яка складається із словника і теорії. Онтології включають абстрактний опис як дуже загальних, так і специфічних для конкретної предметної області термінів. Питання про коректний спосіб аналізу знань з метою визначення термінів залишається наразі відкритим, і його обговоренню присвячено безліч робіт [135]. Однією із сильних сторін онтологій є їх потенційні здібності до вирішення таких важливих завдань як розділення знань і їх повторне використання.

Цей висновок ґрунтується на припущенні про те, що якщо загальна схема (уявлення і використання знань, тобто онтологія) явно визначена для агентів, що працюють з нею, як загальний ресурс, то цей ресурс можна розділяти між агентами і багато разів використовувати [55]. Якби у прикладі, коли користувач намагається знайти інформацію за запитом «Астрологічне сузір'я Оскара Уайльда», була онтологія, на основі якої система змогла б визначити контекстну синонімічність понять «астрологічне сузір'я» і «знак зодіаку», і одночасно оцінити контексти двох інших ресурсів, як невідповідні запиту, релевантність одержаного відгуку склала б 100%. Це підтверджує важливість онтології, як елемента інтелектуального пошуку. Оскільки словники онтологій повинні включати гігантську кількість фактів, що описують реальний світ, і при цьому можуть бути багато разів використані різними агентами, найраціональнішим видається мати лише одну, але якнайповнішу базу онтологій для кожної наочної області. Звичайно, що при цьому повинна бути визначена мова спілкування між базою онтологій і всією множиною агентів. Ця проблема має більш загальну постановку, яка свідчить про те, що всі агенти повинні мати нагоду спілкування зі всіма агентами. Але про це ми поговоримо дещо нижче.

Окрім баз онтологій, що містять факти про певні предметні області, в Інтернеті зараз відображені також цікаві й амбітні проекти, які прагнуть у своїй базі онтологій описати весь реальний світ, тим самим роблячи непотрібним існування будь-яких інших онтологій. Авторам відомі наступні проекти, що служать цій меті: Ontobroker, MindPixel, CYC. Незважаючи на відмінності між ними, мета кожного з проектів – створити величезну базу знань, що включають всі факти, доступні людському мозку. Розглянемо проект CYC, який, як заявляють його творці, покликаний раз і назавжди побудувати базу знань всіх загальних понять, що включають семантичну структуру термінів, зв'язки між ними і правила виведення.

Проект CYC складається з бази знань (CYC Knowledge Base), механізму логічного виведення (CYC Inference Engine), мови подання CYCL (CYC Representation Language) і підсистеми опрацювання природної мови.

База знань CYC є формалізованим відображенням величезної кількості фундаментальних людських знань: фактів і правил виведення. Засобом відображення знань в CYC є спеціалізована мова CYCL. База знань складається з термів, які утворюють словник онтології та тверджень, що зв'язують ці терми. Твердження, у свою чергу, складаються з простих основних тверджень і правил виведення. База знань CYC розділена на окремі «мікротеорії», кожна з яких містить частину загальної кількості тверджень. Таких мікротеорій зараз налічується декілька сотень. Мікротеорії концентруються на окремих областях знань, різних рівнях деталізації, на різних тимчасових проміжках. Механізм мікротеорій дозволяє CYC одночасно зберігати твердження, які з першого погляду суперечать одне одному.

На сьогодні CYC містить десятки тисяч термів і по декілька тверджень для кожного терму. Механізм логічного виведення CYC працює на основі логічної дедукції та навішування кванторів існування і загальності, а також застосовує добре відомі у штучному інтелекті механізми логічного виведення (спадкоємство, автоматична класифікація тощо). Опрацювання природної мови – це одна з найподатливіших проблем, що вивчаються в сучасному інжинірингу програм. Розглянемо, як така проблема розв'язується в проекті CYC. Але спершу порівняємо нижченаведену пару пропозицій (приклад, що наводиться розробниками CYC).

Fred saw plane flying over Zurich. – (Фред бачив літак, що пролітав над Цюріхом).

Fred saw mountains flying over Zurich. – (Фред бачив гори, пролітаючи над Цюріхом).

Хоча пропозиції і виглядають дуже схожими, людський мозок все ж таки вловлює маленьку відмінність, розуміючи, що в першій пропозиції flying відноситься до літака, а в другому – до Фреда. Подібні виведення зазвичай спричиняють надзвичайні труднощі для машинних систем. CYC вирішує завдання сенсового розділення цих двох пропозицій, використовуючи свої знання про те, що літак може літати, а гори – ні.

Отже, цей невеликий приклад добре ілюструє важливість онтології в проблемі розпізнавання природної мови, рішення якої надзвичайно важливе для побудови людино-машинних інтерфейсів, і зокрема, для подальшого розвитку інтелектуального пошуку.

Мультиагент, що розглядається в будь-якій системі, – це апаратна або програмна сутність, здатна діяти на користь досягнення цілей, поставлених перед ним власником і/або користувачем. Отже, в рамках мультиагентних систем ми розглядаємо агенти, як автономні компоненти, які діють за певним сценарієм. Класифікуються агенти за чотирма основними типами: прості, розумні (smart), інтелектуальні (intelligent) і дійсно інтелектуальні (truly intelligent).

Перераховані вище і багато інших завдань агенти можуть виконувати без використання методів штучного інтелекту. Проте ряд завдань просто не може бути розв'язаний без них.

Інтелектуальний агент – агент, який володіє рядом знань про себе та навколишній світ, і поведінка якого визначається цими знаннями.

Програмні інтелектуальні агенти – це новий клас систем програмного забезпечення, яке діє або від імені користувача, або від імені системи, яка делегувала агенту повно-

важення на виконання тих або інших дій. Вони є, за суттю, новим рівнем абстракції, відмінним від звичних абстракцій типу класів, методів і функцій. Але при цьому, розроблення МАС дозволяє створювати системи, які володіють розширюваністю, тобто масштабованістю, мобільністю, або переносимістю, інтероперабельністю, що поза сумнівом дуже важливо при розробленні систем, заснованих на знаннях.

Зазначимо, що поняття агента є деяким парафразом визначення об'єкта. Дійсно, якщо взяти загальноприйняту формулу «клас = дані + методи (функції для роботи з ними)», то можна сказати, що агент – це об'єкт, що володіє функціями для приймання інформації із зовнішнього світу, впливу на зовнішній світ і прийняття рішення про рід впливу. Прийняття рішення може відбуватися рефлективно (тобто буде строго задане) і описуватися програмою.

Поняття інтелектуального агента є розвитком методів ООП. Простіше, можна сказати, що «інтелектуальний агент = дані + методи + знання», причому методи тут – це не тільки функції роботи з даними, але і функції роботи зі знаннями, і можливі методи впливу на навколишнє середовище агентом. Проте багато розроблювачів стверджують, що використовують у своїх продуктах технології інтелектуальних агентів, тоді як основна їхня відмітна риса – знання – у розробленнях не відображені.

Наявність штучного інтелекту означає, що агент повинен деяким чином зберігати свої знання. Згідно з історією штучного інтелекту було розроблено значну кількість методів відображення знань. Однак найбільш розповсюдженими на сьогодні є продукції (правила вигляду «якщо..., то...») і нейронні мережі. Перші здобули свій успіх завдяки простоті розуміння, формалізації та реалізації. Другі – тим, що немає необхідності формалізувати знання і заносити їх у базу, а досить навчити мережі. І ті, й інші дають цілком непогані результати.

Зазначимо, що як джерело інформації агент може використовувати інформацію, одержану від іншого агента. Системи, в яких передбачена взаємодія агентів, називаються мультиагентними.

11.1.3. Суть, структура та властивості мультиагентної системи

Мультиагентна система (МАС) – це множина деяких об'єктів (агентів), незалежних, але здатних створювати комунікативне співтовариство і спільно розв'язувати загальні завдання. У додатку до середовища Інтернет це визначення можна проілюструвати таким чином. Хай існує деяка множина U Інтернет-вузлів, на кожному з яких розташовується один або декілька агентів (програм, процесів), що входять у співтовариство Ai агентів. Якщо деяким агентом Ai одержана від користувача задача, то він аналізує її та намагається налагодити спілкування з тими агентами (або ланцюжками агентів), які можуть допомогти йому в розв'язанні поставленої задачі. При цьому мається на увазі, що йому відомі всі реквізити агентів системи (адреси вузлів, ідентифікатори) і він «володіє» спільною для всієї системи мовою взаємодії Ь. Зауважимо, що як користувач також може виступати агент, у тому числі й такий, що належить до іншої системи агентів. Така організація дозволяє створювати мережі МАС.

Сьогодні можна зустріти два основні типи структури мультиагентних систем. Перший з них – це рівнозначна (рівно-рангова) структура. Незважаючи на те, що структура, зображена на рис. 11.1, виглядає як ієрархічна організація агентів, це не так. Агент BR не є узагальнюючим або головним агентом, що дозволило б зробити його вищим за всі інші агенти за ієрархією, він лише містить відомості про всіх агентів, присутніх у системі, не

маючи ніяких інших функцій, окрім реєстрації та зберігання даних про агентів. Агент такого типу звичайно є спеціально виділеним в мультиагентних системах і називається агентом-брокером (але для різних МАС ця назва може видозмінюватися, не змінюючи значення). Іншим спеціальним агентом зазвичай є агент користувача (агент US на рис. 11.1). Він призначений для взаємодії з кінцевим користувачем, і зазвичай зберігає інформацію про призначені для користувача переваги і обмеження, різні налагодження тощо.

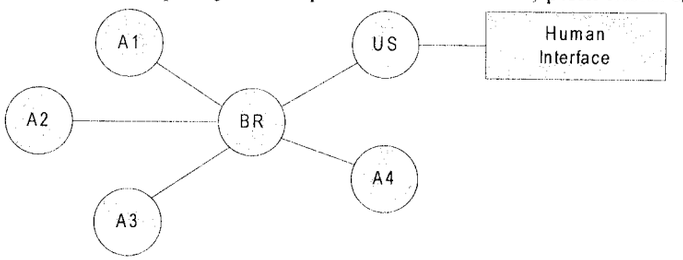


Рис. 11.1 Структура рівнозначної МАС.

Рівнозначна структура є найвідомішою в сучасних МАС Це можна пояснити простішим і природнішим способом взаємодії між агентами в такій системі, а також тим важливим фактом, що сучасний Інтернет організований саме як множина рівнорангових вузлів, незважаючи на явну ієрархічність, яка лежить в основі його адресації DNS.

Другий тип структури мультиагентної системи, менш відомий на практиці, – це структура ієрархічна (рис. 11.2).

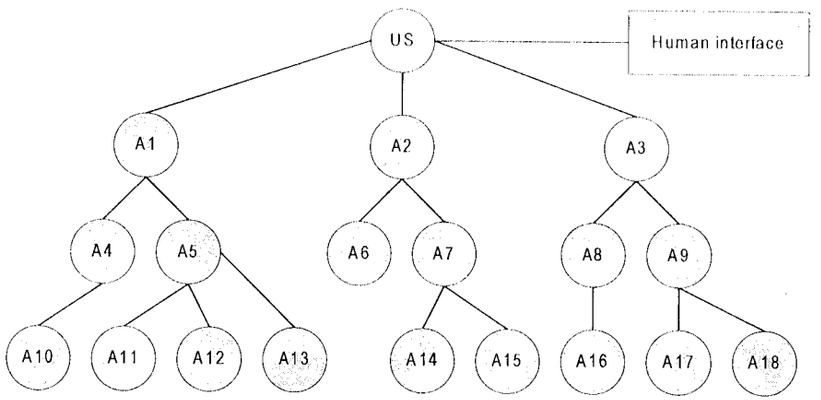


Рис. 11.2. Структура ієрархічної МАС

З рис. 11.2 видно, що в цьому випадку верхнім рівнем ієрархії є агент користувача, який формулює постановку задачі. Нижчі рівня є рівнями деталізації задачі. Внаслідок невеликого розповсюдження МАС з такою структурою, автори не мають докладної інформації про функціонування таких систем. Проте, можна припустити, що агентом, який виконує функції, схожі з функціями агента-брокера в рівнозначній МАС, тут є кожний вузол графа (не лист) по відношенню до вузлів, що лежать за ієрархією на один рівень

нижче від нього. Можливий і інший підхід, коли у всій ієрархії є лише один агент-брокер, при реєстрації у якого агенти повинні вказувати той рівень, на якому вони бажають працювати. Зауважимо, що кожний з агентів – це зовсім не обов'язково програма, жорстко пов'язана з певним вузлом Інтернет. Існують мобільні агенти, які можуть передавати себе на інші вузли у вигляді даних, а потім, влаштувавшись на новому вузлі і скоректувавши свої реквізити у агента-брокера, працюють в колишньому режимі. Агенти такого типу часто реалізуються на основі Java – аплетів – мобільних, платформозалежних програм. Мобільність агентів має немало переваг; так, неважко уявити собі ситуацію, коли в результаті виникнення проблем на деякому сервері А, агенти, що працюють на ньому, починають мігрувати на резервний сервер Б, швидко і прозоро для користувача відновлюючи колишній режим роботи.

Основна властивість «мультиагентної системи» полягає в розподіленні штучного інтелекту, який розглядає розв'язання однієї задачі на декількох інтелектуальних підсистемах. При цьому задача розбивається на декілька підзадач, які розподіляються між агентами. Ще однією областю вживання МАС є забезпечення взаємодії між агентами, коли один агент може виробити запит до іншого агента на передавання деяких даних або виконання певних дій. Також в МАС є можливість передавати знання. Побудова програмних систем за принципом МАС може бути обумовлена наступними чинниками:

- деякі предметні області застосовують МАС у тих випадках, коли логічно кожного з учасників процесу можна уявити у вигляді агента. Наприклад, соціальні процеси, в яких кожний з учасників відіграє свою роль;
- паралельним виконанням задач, тобто якщо предметна область легко подається у вигляді сукупності агентів, то незалежні задачі можуть виконуватися різними агентами;
- стійкістю роботи системи: коли контроль і відповідальність за виконувані дії розподілені між декількома агентами; при відмові одного агента система не перестає функціонувати; отже, логічно розмістити агентів на різних комп'ютерах;
- модульністю МАС, що дозволяє легко нарощувати і видозмінювати систему, тобто легше додати агента, ніж змінити властивості єдиної програми. Системи, які змінюють свої параметри, з часом можуть бути подані сукупністю агентів. Модульність обумовлює легкість програмування МАС.

Мультиагентні системи поділяються на кооперативні, конкуруючі та змішані. Агенти в кооперативних системах є частинами єдиної системи і розв'язують підзадачі однієї загальної задачі. Зрозуміло, що при цьому агент не може працювати поза системою і виконувати самостійні задачі. Конкуруючі агенти є самостійними системами, хоча для досягнення певної мети вони можуть об'єднувати свої зусилля, приймати цілі і команди від інших агентів, але при цьому підтримка зв'язку з іншими агентами не обов'язкова. Під змішаними агентами розуміються конкуруючі агенти, підсистеми яких також реалізуються за агентною технологією, тобто окрім спілкування з іншими агентами повинна бути реалізована можливість спілкування з користувачем [48].

Типи агентів

У МАС, призначених для пошуку релевантної інформації, кожний агент відповідає за реалізацію певного елемента інтелектуального пошуку. Існують агенти онтологій, агенти фільтрів, агенти видобування знань, агенти метапошуку тощо.

Розглянемо схему взаємодії всіх вищеперелічених типів агентів в деякій пошуковій МАС.

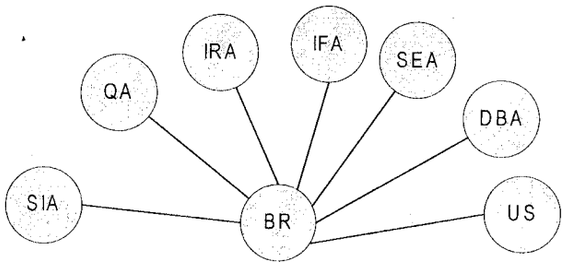


Рис. 11.3. Схема взємодії всіх типів агентів у пошуковій МАС

Використовуються наступні позначення:

- OA – агент онтології;
- IRA (Intelligent Retrieval Agent) – агент видобування інформації. Видобуває заданий документ і опрацьовує його, оцінюючи ступінь релевантності його запиту, і формулюючи пояснення цієї оцінки;
- IFA (Information Filtering Agent) – агент фільтрації, зменшує кількість документів, визнаних релевантними, враховуючи налагодження користувача, вибрані тематичні розділи і т. ін.;
- SEA (Search Engine Agent) – агент зв'язку з пошуковими машинами Інтернет;
- BR – агент-брокер;
- US – агент, призначений для користувача;
- DBA – агент обслуговування бази даних;
- SIA – агент-індексатор.

Види комунікаційних моделей між агентами

Чинником, який активно впливає на час відгуку системи, є комунікаційна модель, що визначає, яким чином віддалені частини програми спільно працюють над запитом користувача. Найпоширенішим підходом тут вважається пересилання повідомлень, що дозволяє досягти більш високого ступеня автономності між частинами програми, ніж якби вони викликалися директивно за допомогою RPC (Віддаленого Запуску Процесів) [206]. Пересилання повідомлень є концепцією, природно відповідною агентним системам, оскільки в її рамках агент робиться достатньо незалежним, виконує лише своє завдання і лише іноді відповідає на запити інших агентів або самостійно робить запити, якщо це буде потрібно для його роботи. У рамках цієї концепції в європейському проекті Grasshopper були виділені наступні моделі комунікації між агентами.

- Синхронна комунікація. Зазвичай, коли клієнт запускає метод на сервері, сервер виконує відповідний метод і повертає результат клієнту, який продовжує свою роботу. Цей стиль називається синхронним, тому що клієнт блокується доти, поки результати методу не будуть повернені.
- Асинхронна комунікація. При використанні асинхронної комунікації клієнту немає необхідності чекати, поки сервер виконає метод, натомість клієнт продовжує виконувати своє власне завдання. У такому разі у клієнта є декілька можливостей

одержати результати реалізованого методу. Він може періодично опитувати сервер про те, чи закінчено виконання методу, він може чекати результату в тих випадках, коли це потрібно, або підписатися на повідомлення про те, що результат буде доступний.

- Динамічна комунікація. Цей механізм зручний у тому випадку, коли у клієнта немає доступу до проксі-серверу. Клієнт може сконструювати повідомлення динамічно, шляхом вказання сигнатури того серверного методу, який повинен бути запущений. Динамічна генерація повідомлень може бути використана як синхронно, так і асинхронно.
- Багатозв'язкова комунікація. Багатозв'язкова комунікація дозволяє клієнтам використовувати паралелізм у процесі спілкування із серверними об'єктами. Використовуючи механізм багатозв'язкової комунікації, клієнт може запускати один і той самий метод на декількох серверах паралельно.

Сучасна система доступу до розподілених інформаційних ресурсів, що працює в онлайн-режимі в Інтернеті, повинна бути готова до приймання й опрацювання декількох різних запитів одночасно, тому не можна допустити, щоб проходження запитів затримувалося, поки така система опрацьовує інший запит. Тому комунікацію між системними агентами в такій розподіленій системі доцільно організовувати за асинхронним принципом з орієнтацією на створення багатопотокових програм. Проте, при створенні агентів, що відповідають за роботу з кінцевим користувачем системи, можуть бути застосовані різні підходи.

11.1.4. Мови спілкування між агентами

Є два підходи до розроблення мови спілкування між агентами.

Перший підхід процедурний, тобто комунікації базуються на виконанні інструкцій. Така мова може бути спроектована і запрограмована на Java. Другий підхід декларативний, тобто комунікації виникають на базі описів.

Другий підхід набув більшого поширення для створення мов спілкування агентів. Однією з найвідоміших мов є KQML – Knowledge Query and Manipulation Language.

Ця мова дозволяє описувати як формат повідомлення, так і протокол передачі повідомлень. KQML має три рівні: комунікативний рівень (описує такі параметри як відправник, одержувач, різні ідентифікатори повідомлення), рівень повідомлення (описує запити, керівні дії, а також протокол для інтерпретації повідомлення), рівень змісту (містить інформацію, що супроводить запити рівня повідомлення).

11.1.5. Неспеціалізовані пошукові агенти

У найпростішому випадку агент може шукати інформацію за ключовими словами і стійкими словосполученнями. Західних аналогів таких систем дуже багато. Це, в основному, комерційні програми, такі як AnchorPage (Iconovex); AppControl (Seagate Software); EchoSearch (Iconovex); Freeloader (Individual); Headliner (Lanacom); Hotbot (Inktomi); MagnetSearch (CompassWare Development); Net Attache (Tympani Development); Netriever 2.0 (Metz Software); Personalized Commerce Agents (Personal); SEARCH'97 Information Server #@; Secret Agent #@; Smart Bookmarks 3.0 Beta2 #@. Всі програми можна визначити як надбудови над звичайними машинами пошуку. Вони об'єднують посилання на сайти, знайдені множиною машин пошуку, виключаючи повторні й непрацюючі посилання, показуючи непогані результати. Але жодна система цього класу не в змозі

самостійно відібрати тільки корисну інформацію і залишає це користувачу. Звичайно, використовуючи ці програми, можна значно знизити кількість результуючих документів від сотні тисяч, що постачаються звичайними машинами пошуку, до сотень і навіть декілька десятків. Але ступінь «потрапляння в запит» все ж низький.

Основні недоліки таких систем можна звести до таких:

- результати трохи кращі, ніж при пошуку без агента;
- навчання агента зазвичай відбувається в інтерактивному режимі, алгоритми навчання не оптимальні й навчання займає багато часу;
- агент не накопичує знання про області пошуку.

11.1.6. Спеціалізовані пошукові агенти

Інший різновид цієї категорії агентів – спеціалізовані агенти пошуку. Вони пристосовані шукати інформацію, наприклад, тільки про музику, книги, акції тощо. Більшість цих агентів дуже жорстко налаштовані. Вони «вміють» дуже добре працювати на певних сайтах з фіксованим форматом даних (адреси цих сайтів можуть бути «зашиті» в агента), наприклад, сайти з котируваннями акцій на біржах світу. У агентів із систем цієї категорії недоліки такі ж, але є один плюс: чітке спрацьовування на добре структурованих даних у відомому форматі.

Приклади таких систем: BullsEye, WARREN, ShopBot, Kazbah й інші.

11.1.7. Системи з використанням методів і засобів штучного інтелекту

Серед агентних систем є продукти з використанням засобів ШІ, таких як подання знань, правила виведення нових знань, механізми навчання, лінгвістичне опрацювання природно-мовних текстів.

Складні інтелектуальні агенти розробляються переважно в дослідницьких лабораторіях університетів та існують у вигляді демонстраційних версій з обмеженими можливостями.

Приклади таких систем: BIG (MAC Lab., University Massachusetts at Amherst); ADEPT (QMW, University London.); CBR Agents (IIIA, Spanish Scientific Research Council).

Агентна система Autonomy компанії AgentWare є успішним прикладом реалізації інтелектуальної системи. Система інтелектуального пошуку з використанням технології нейрон-них мереж наприкінці 90-х років минулого століття перетворилася на могутню пошукову систему, а потім – систему керування знаннями на рівні великих корпорацій.

Загальними недоліками інтелектуальних агентів є слабке навчання. Результати пошуку використовуються, щоб поліпшити наступний пошук за цими ж темами, але недостатньо повно й ефективно. Такі системи є корисними інструментами при пошуку інформації в Інтернеті, але не можуть зробити цей пошук повністю автоматичним. Часто такі системи є тільки прототипами реальних систем і мають ряд істотних обмежень.

Враховуючи вищесказане, на авансцену розвитку агентних технологій виходять проблеми подання знань, механізми виведення нових знань, опис моделі світу, моделювання міркувань в рамках агентного підходу. Для вирішення комплексу цих і супутніх завдань у першу чергу необхідно застосувати методи роботи зі знаннями, прийняті у класичному і розподіленому ШІ, стосовно агентних технологій.

Отже, розібравши правила формування запитів, порівнявши алгоритми пошуку найвідоміших пошукових серверів та алгоритми роботи агентних систем виявлено, що для підвищення якості знайденої інформації в Інтернеті, розроблення інтелектуальної мультиагентної системи в пошуковій галузі інформації є дуже доцільною.

11.2. Онтологія як засіб формалізації та алгоритмізації знань в інтелектуальній системі

На сьогодні отримання знань залишається найвужчим місцем процесу проектування інтелектуальних систем. Систему, наділену вмінням визначати, зберігати та використовувати в потрібний момент необхідні релевантні знання, будемо називати інтелектуальною. Зазвичай, інтелектуальні системи застосовуються для розв'язання складних задач; основна складність їх розв'язку пов'язана з використанням слабоформалізованих знань спеціалістів-практиків, і логічне (або змістове) опрацювання інформації переважає над обчислювальним. Тому в основі архітектури сучасних інтелектуальних систем лежать бази знань (БЗ), які формуються відповідно до предметної області (ПО), в якій застосовується певна інтелектуальна система.

На етапі проектування та реалізації інтелектуальних систем виділяють ряд методологічних та технологічних проблем, з якими безпосередньо стикаються їх розробники [11]. Зокрема в Україні такі проблеми полягають у відсутності концептуальної цілісності й узгодженості між окремими прийомами та методами інженерії знань; нестачі кваліфікованих фахівців у цій області; жорсткості розроблених програмних засобів та їх низької адаптивної здатності; складності впровадження експертних та інтелектуальних систем, що зумовлені психологічними аспектами, неприйняття персоналом нових технологій; відсутності в Україні техніко-економічних показників оцінювання ефективності таких систем; емпіричності процедури вибору програмного інструментарію і процесу тестування (відсутність єдиних критеріїв). Перерахунок, звичайно, можна продовжувати, але тим не менше, перспективи розвитку і впровадження інтелектуальних систем у більшість галузей науки очевидні.

Успіх у розв'язанні задачі побудови ефективної спеціалізованої інтелектуальної системи визначається відповідністю її бази знань та онтології (ядра бази знань) до особливостей предметної області.

Тому дослідження онтологій стає все популярнішим серед науковців в області інформаційних технологій. Більше того, на сьогодні розроблення онтологій – явних формальних описів понять предметної області та зв'язків між ними [249] – переходить зі світу лабораторій штучного інтелекту на робочі столи експертів з певних предметних областей, інженерів та користувачів. На сьогодні онтології широко застосовуються в інформаційних технологіях (робота пошукових машин, електронна комерція, системи опрацювання інформації), матеріалознавстві (системи аналізу стану матеріалів), машинобудуванні та інших галузях науки та промисловості (рис. 10.4).

Онтологію можна розглядати як загальний словник понять для вчених, який спільно використовується в певній предметній області. Вона включає машинно-інтерпретовані формулювання основних понять предметної області та зв'язків між ними.

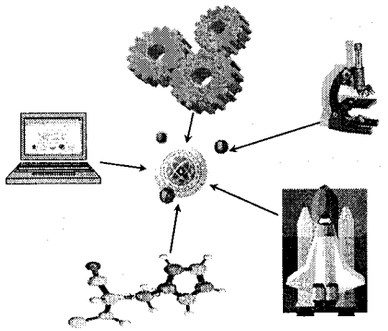


Рис. 11.4. Застосування онтології в системі знань

Цей підрозділ книги присвячений онтології – базовому компоненту інтелектуальної системи. У наступних параграфах розглянемо основні методики навчання онтологій, формати та стандарти відображення та наявні інструменти для їх створення. Особлива увага приділена технології розроблення онтології в редакторі Protege, найбільш зручному та адаптованому для побудови спеціалізованих предметно-орієнтованих онтологій. Авторами також розроблено інструкцію користувача з цього пакету на прикладі онтології в області матеріалознавства. В інструкції описано процедури створення, редагування, збереження та перегляду онтологій.

11.2.1. Аналіз підходів навчання онтологій

Поруч з очевидними перевагами застосування онтологій існують проблеми їх створення та навчання. Навчання онтології – це процес наповнення її новими знаннями з певної предметної області. За останні два десятиліття було розроблено ряд проектів у галузі навчання онтологій. У деяких з них, наприклад, Mikrokosmos [291] та CYC [276] знання одержували вручну, а вже після побудови великої базової онтології переводили цей процес у напівавтоматичний режим, використовуючи за основу цю онтологію.

Онтологію можна створювати з нуля, з наявних онтологій (як із глобальних, так і локальних), з інформаційних даних різних джерел або комбінації останніх двох підходів [315]. Онтології будуються з різною мірою автоматизації: вручну (інформація вноситься в БЗ інженером зі знань та за допомогою експерта з певної предметної області), напівавтоматично (обмін інформацією відбувається в діалоговому режимі між машиною та експертом) та повністю автоматично (БЗ наповнюється з використанням підходів інтелектуального аналізу інформації). Оскільки ручний режим вимагає значних часових і фінансових затрат, а автоматичний метод внаслідок слабкої розвиненості підходів семантичного аналізу текстів на сьогодні можна застосовувати лише для побудови дуже спрощених онтологій в обмежених випадках, тому основну увагу приділимо напівавтоматичному способу із застосуванням спеціалізованих редакторів.

З іншого боку, програми навчання онтологій працюють на різному структурному рівні та топології, використовуючи різноманітні підходи: статистичні методи [319], логічні підходи, такі як індуктивне логічне програмування (inductive logic programming – ILP) [323], методи виділення кластерів [209], лінгвістичні підходи, такі як синтаксичний аналіз [236] та морфологічно-синтаксичний аналіз [203]. Також існує декілька проектів, які використовують комбінації вищенаведених підходів для навчання різних компонентів онтологій [282, 225].

У цілому, основними факторами, що відрізняють одну систему навчання онтологій від іншої, є такі:

- елементи навчання (поняття, зв'язки між ними, аксіоми, екземпляри класів, синтаксичні категорії та тематичні ролі);
- початкова точка наповнення (стан наповненості та мова на вході);
- попередня обробка (лінгвістичне опрацювання, глибоке або поверхневе опрацювання тексту);
- методи навчання: а) підхід (статистичний, логічний, лінгвістичний, порівняння зі взірцем, методи шаблону та гібридні); б) задача навчання (класифікація, кластеризація, навчання правилам, формування понять, створення онтології); в) категорія навчання (контрольована/неконтрольована); г) ступінь автоматизації (ручний, напівавтоматичний, автоматичний);
- результатна онтологія (ступінь покриття предметної області, структура, топологія і мова відображення);
- методи оцінки (оцінювання навчальних методів або оцінювання результуючої онтології).

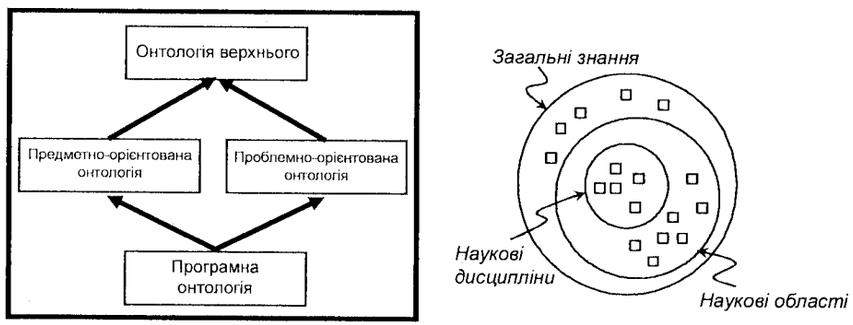


Рис. 11.5. Рівні абстракції онтологій.

У побудованій онтології розрізняють декілька рівнів абстракції, на кожному з яких можуть бути визначені окремі онтології (рис. 11.5). Наприклад, в області кожної наукової дисципліни можна визначити онтології, вище можна описати онтології наукових областей, що міститься на стику окремих наукових дисциплін. Найвищим рівнем абстракції ми поставимо загальні категорії структур знань. Для роботи ІС в певній предметній області необхідна онтологія тільки з відповідної наукової дисципліни.

11.2.2. Загальні принципи проектування онтологій

Існує декілька основних принципів побудови онтологій, запропоновані різними конструкторами в різний час.

- Під впливом філософських досліджень Гуаріно [251] запропонував методологію побудови онтології, відому як «Формальна онтологія» (Formal Ontology). Цей принцип містить у собі теорію частин, теорію цілісності, теорію рівності, теорію залежності та теорію узагальнень. Він об'єднав основні принципи побудови, які включають у себе: 1) необхідність розуміння всієї предметної області; 2) чітку ідентифікацію; 3) виділення класифікованої структури; 4) детальне визначення ролей.

- Усолд та Грунінгер [316] запропонували скелетну методологію побудови онтологій вручну: 1) визначення мети і границі (обмеження); 2) безпосередньо побудова онтології: заголовок онтології (визначення ключових понять та зв'язків між ними, забезпечення визначень таких понять і зв'язків); кодування онтології (встановлення основних її термінів – клас, об'єкт, зв'язок; вибір мови відображення; написання програмного коду); об'єднання наявних онтологій; 3) оцінка; 4) розроблення документації; 5) визначення основних принципів керування для кожного з попередніх етапів. Як результат онтологія повинна бути максимально чіткою (визначення мають бути максимально однозначними), внутрішньо і зовнішньо цілісною та зв'язною, і спроектованою таким чином, щоб її можна було послідовно максимально розширювати і повторно використовувати.
- Ontological Design Patterns (ODPs) [298] використовувались для визначення онтологічних структур, термінів і семантичного змісту. За допомогою цих методик можна виділити конструкції та визначення складних виразів з їх відображення. Цей метод успішно використовується при об'єднанні молекулярної біологічної інформації.

Узагальнюючи, можна виділити наступний алгоритм побудови онтологій.

1. **Накопичення знань про предметну область.** Здійснюється експертиза відповідних інформаційних ресурсів, при цьому формально визначаються основні терміни якими буде описана певна предметна область. Ці визначення мають бути зібрані із врахуванням можливості їх подання на спільній мові, обраній для онтології.
2. **Формування онтології.** Для цього розробляють повну понятійну структуру предметної області. Ймовірно, це вимагатиме розпізнавання головних, ключових понять предметної області та їх властивостей, визначення зв'язків між поняттями, створення абстрактних понять, виділення понять, що містять екземпляри, а також, можливо, залучення допоміжних онтологій.
3. **Розширення та конкретизація онтології.** Поняття, зв'язки, атрибути, екземпляри додаються доти, поки рівень деталізації забезпечить задоволення цілей онтології.
4. **Перевірка виконаної роботи.** На цьому етапі усуваються синтаксичні, логічні та семантичні неузгодженості елементів онтології та відбувається перевірка достовірності інформації.
5. **Впровадження онтології.** На кінцевій стадії розроблення онтології виконується її перевірка експертами з конкретної предметної області. Після чого онтологія впроваджується в робоче середовище.

Як бачимо, основні етапи побудови онтології є достатньо простими. Існують різноманітні методології, які описують теоретичні підходи, а також доступні численні засоби побудови онтологій. Проте, проблема полягає у відсутності єдиного підходу, а інструменти побудови онтологій не розроблені до рівня інструкцій. Також немає узгодженості у стандартах відображення інформації та мовах подання онтологій. Нижче розглянуто основні стандарти та мови подання онтологій.

11.2.3. Формати та стандарти подання інформації

У світі в цілому і наукових дослідженнях зокрема існує ряд факторів, що приводять до появи загальних вимог до інтелектуальних систем. До них належать: збільшення мас-

штабу завдань, які ставлять у сфері інформаційних систем; зростання кількості сторін, які створюють і використовують інформацію; збільшення об'єму інформації та частоти її змін; розподіленість інформації. При цьому ця система повинна бути гнучкою, масштабованою і відкритою.

Під відкритістю розуміється побудова інтелектуальної інформаційної системи на основі світових інформаційних стандартів. Масштабованість – це можливість побудови (в тому числі поетапного) системи з готових або стандартних компонентів. Це дозволяє легко змінювати масштаби кола задач, розв'язаних системою, змінювати і поетапно нарощувати систему. Гнучкість означає здатність системи до налаштування під інформаційні потреби користувача, можливість перенесення з одного середовища експлуатації в інше та незалежність від конкретних компонентів і програм, залучених у систему.

Що ж необхідно для розроблення інтелектуальних систем, які б задовольняли ці вимоги? Відповідь – стандарти. Їх завдання полягає в тому, щоб надати можливість користувачам і програмам спілкуватися між собою і один з одним, не обмежуючись конкретною предметною областю. Сьогодні кілька загальноприйнятих відкритих Інтернет-стандартів становлять базу для побудови таких систем. До стандартів, що задовольняють вищеописані вимоги, належать [21]:

- інтернет-стандарти, розроблені консорціумами IETF, WWW та WS-I (наприклад, XML);
- Web-служби як метод забезпечення обміну інформацією між різними системами різних розробників;
- єдиний підхід до організації метаданих (Дублінське ядро, стандарт RDF).

Вже розроблені та програмно підтримані десятки стандартів в області обміну інформацією. Серед інших можна виділити фундаментальні стандарти: InfoSet, Namespace, XML [88].

InfoSet – (XML Information set), специфікація цього стандарту визначає набір абстрактних інформаційних елементів, що використовуються в правильно побудованих XML-документах.

Namespace – (Namespaces in XML), простір імен. Використовується для усунення неоднозначності в іменах елементів і атрибутів при розподіленому розробленні.

XML – (Extensible markup language), мова, що описує клас об'єктів XML document, а також частково роботу комп'ютерних програм, котрі опрацьовують об'єкти з даними, що реалізують цей клас. XML став стандартом відображення структурованих електронних документів для більшості сучасних програмних засобів, зокрема загальновживаних пакетів Microsoft Office та OpenOffice.org.

Найсуттєвіша заслуга XML – відображення даних однією мовою. Універсальний синтаксис відкрив дорогу появі ряду важливих XML-технологій. Це мови XSL і XPath, призначені для роботи з деревовидною структурою документів; XML Schema – стандарт опису конкретних мов розмітки, використовуючи синтаксис XML; це самостійна специфікація, яка підтримує більш сувору типізацію даних, надає ширші можливості для визначення типів даних елементів та їх атрибутів; XLink і XPointer – засоби зв'язку розподілених блоків інформації в один загальний документ.

Окрім фундаментальних стандартів відображення даних, розглянемо й інші. Необхідне для уніфікації відображення даних та знань сімейство стандартів вже розроблено і широко використовується [300]. Зокрема формат RDF (Resource Descriptiption

Framework – інструмент опису ресурсів) є різновидом формату XML. Він призначений для формування електронного інформаційного середовища між джерелами та споживачами інформації, об'єднаними в єдину мережу. Формат RDF – набір інструментів для роботи з метаданими, який забезпечує єдине (стандартизоване) середовище взаємодії програм, які обмінюються інформацією у Web на зрозумілій мові. RDF робить наголос на легкість автоматизованого опрацювання Web-ресурсів. Загалом, RDF забезпечує основу для елементарних інструментів авторизації, пошуку і редагування даних, що сприяє трансформації Web в апаратно опрацьовуване сховище інформації. У лютому 2004 р. був прийнятий цілий ряд специфікацій до нього, які суттєво полегшують опис ресурсів. Серед них важливо виділити такі: Concepts and abstract syntax; RDF Semantics; RDF Primer; RDF Vocabulary description language 1.0: RDF schema; RDF/XML Syntax specification (revised); RDF Test cases.

Як стандарт відображення онтологій в Інтернеті консорціумом W3C (http://www.w3.org) пропонується мова DAML (DARPA markup language), що є надбудовою над іншими мовами: RDF та RDFS, які, у свою чергу, є розширеннями XML. DAML базована на фреймах, в якій концептуалізація задається у форматі об'єктів та властивостей. Ряд DAML-орієнтованих (або підтримуваних) інструментів досить широкий. Серед них можна виділити: редактори онтологій, візуалізатори у вигляді графів, броузери, транслятори на інші мови, машини виведення тощо.

Зараз перспективною є ініціатива консорціуму W3C під назвою Семантичний Веб (Semantic Web) [305], метою якого є інтелектуалізація всієї мережі Інтернет. У рамках проекту Semantic Web, спрямованого на аналіз семантики інформаційних ресурсів, здійснюється робота групи Web Ontology, оскільки саме онтологічний підхід є основою для подання знань про різні предметні області. Нещодавно ця група опублікувала першу версію робочого проекту, в якому викладені основні вимоги до нової мови подання онтологій – OWL (Ontology Web Language) [321].

Закономірно виникає запитання: навіщо нам ще одна мова опису даних?

Українською мовою абревіатуру OWL можна перекласти як «мова опису онтологій у Web». Під онтологією в цьому випадку розуміється сукупність термінів і понять, що використовуються в певній області знань або діяльності (наприклад, машинобудуванні, біології, матеріалознавстві тощо). Онтологія також містить формальні, а значить, зрозумілі для комп'ютерів описи ключових понять і взаємозв'язків між ними. Зараз широко використовувана мова XML та деякі інші механізми дозволяють забезпечити гнучкість при обміні даними між різними програмами, сервісами, базами даних тощо. Проте, вони ефективні тільки в тих випадках, коли контрагенти обміну використовують для відображення інформації однакові системи змістових координат. Засоби опису й опрацювання онтологій OWL дозволять «порозумітися» найрізноманітнішим прикладним програмам, основне завдання яких полягає не в поданні інформації для людини, а безпосередньо в опрацюванні вмісту інформаційних ресурсів. При створенні цієї мови був використаний і узагальнений досвід попередніх розроблень подібних мов. Прототипом для розроблення OWL є мова опису онтологій DAML+OIL (друга версія мови в рамках проекту DAML), розроблена Агентством перспективних досліджень Міністерства оборони США DARPA. У цій версії мови були використані результати проекту OIL (Ontology Inference Layer або Ontology Interchange Language), який виконувався в рамках програми Information Society Technologies за підтримки Європейського Союзу. Створена

зусиллями консорціуму W3C мова OWL являє собою мову для визначення структурованих, підтримуваних у Web-онтологіях, які забезпечують багатшу інтеграцію даних, ніж попередні мови, що не були орієнтовані на Web і, зокрема, на Семантичний Web (див. рис. 10.6).

Стандарт OWL визначає три рівні мови або три підмови, що відрізняються рівнем своїх виразних можливостей: OWL Lite, OWL DL і OWL Full, які детально будуть описані в наступних параграфах.

Важливою якістю всіх описаних стандартів є їх відкритість і незалежність від конкретних областей застосування і розділів знань.

Рис. 11.6. Дерево технологій Семантичного Вебу.

11.2.4. Засоби для створення онтології

Для ефективного і швидкого створення онтології необхідно мати достатньо потужні і, водночас, прості інструменти. Тому розробляються редактори, які здатні полегшити процес наповнення онтології потрібними знаннями, здійснювати їх обробку та приводити ці знання до формального вигляду. Здатність організовувати і керувати онтологією – ключ до практичності редактора.

Типовими компонентом редакторів є інтерфейсна оболонка, за допомогою якої користувач має змогу вносити необхідні знання, зберігати їх у сховищах та здійснювати вибірку та редагування вже наявних знань. Сучасні редактори містять не тільки механізми накопичення і класифікації знань, а й процедури їх опрацювання та прийняття рішень. За допомогою розроблених інтерфейсів користувач може також формувати запити системі та отримувати відповідь на них в діалоговому режимі.

На сьогодні кількість інструментів для створення та редагування онтологій налічує десятки.

Кожен із програмних засобів відрізняється один від одного метою їх розроблення, технічними особливостями, мовою відображення, допустимими форматами імпорту та експорту і, безумовно, способом їх розповсюдження. Серед них є такі, що розповсюджуються відкрито, комерційні продукти та інструменти, обмежені в застосуванні.

Комерційні продукти – це автономні редактори, що розробляються виключно для побудови онтологій в будь-якій предметній області і є частиною наборів комерційного програмного забезпечення для підтримки процесу прийняття рішень на підприємстві, фірмі тощо. Інше програмне забезпечення для редагування онтологій – це результати науково-дослідних проектів технічного застосування онтологій, як правило, фінансованих урядом та академічними структурами. Такі пакети зазвичай є відкритими для всіх користувачів з можливістю їх редагування та доповнення. Ще інші редактори призначені для побудови онтологій у специфічній предметній області. Вони мають обмежене застосування, і розробляються виключно для розв'язання особливих задач в цій області. Також існують редактори, орієнтовані на розроблення загальних онтологій. Прикладом може бути продукт під назвою CYC [226]. Проект включає створення розширеної онтологічної системи, що описує понад 106 концептів і 105 аксіом. Для відображення знань фірма розробила спеціальну мову CYCL. Для виведення по онтологічній базі знань розроблена спеціальна машина виведення. Основна мета цього проекту – побудова розширеної бази знань про загальні поняття практично у всіх областях людської діяльності (common knowledge).

Серед розглянутих інструментів для побудови предметно-орієнтованої онтології, зокрема в області матеріалознавства, виділимо редактор Protege . Protege – це гнучке, незалежне від платформи середовище для створення і редагування онтологій та баз знань. Воно відрізняється від інших інтелектуальних інструментів наступними особливостями та перевагами:

- забезпечує наочність та зручний у використанні графічний інтерфейс користувача;
- реалізує масштабованість, тобто модульне нарощування системи в рамках уніфікованої архітектури;
- на відміну від інших аналогічних програм, не спостерігається сповільнення роботи при значній кількості опрацьовуваної інформації;
- дозволяє нарощувати архітектуру за допомогою додатково розроблених підпрограм – плагінів (plug-in). Можна легко розширити Protege плагінами, зробленими на замовлення для будь-якої предметної області та конкретної задачі.

11.3. Технологія розроблення онтологій в редакторі Protégé

Protégé – це інструмент, який дозволяє користувачам конструювати онтології бази знань предметної області, вводити дані та настроювати форми їх виведення. Ця платформа може легко розширюватися шляхом включення в неї графічних компонентів: графи, таблиці, медіа (звук, зображення, відео). Програма дозволяє зберігати дані в найбільш розповсюджених форматах: *OWL*, *RDF*, *XML*, *HTML*.

11.3.1. Еволюція Protégé

Розвиток Protégé триває вже понад півтора десятків років. З невеликої прикладної програми для використання в медицині при плануванні терапії Protégé перетворився в універсальний набір інструментів для створення баз знань.

Першочергова мета Protégé – усувати вузьке місце в побудові інтелектуальних систем, а саме: мінімізувати роль інженера зі знань при конструюванні баз знань шляхом

створення діалогового інтерфейсу та спрощення введення даних в систему. При цьому появляється можливість структурувати інформацію на різних стадіях наповнення бази знань. Тому перша версія програми Protégé-I являла собою інструмент для спрощення процесу набуття знань. Марк Мусен ініціатор і розробник Protégé, дав йому наступний опис:

Protégé – це не експертна система, а також не є програмою, котра безпосередньо будує експертні системи; натомість Protégé – це інструмент, що допомагає розробникам експертних систем створювати свої інструменти, спеціально пристосовані для набуття знань в конкретних прикладних областях.

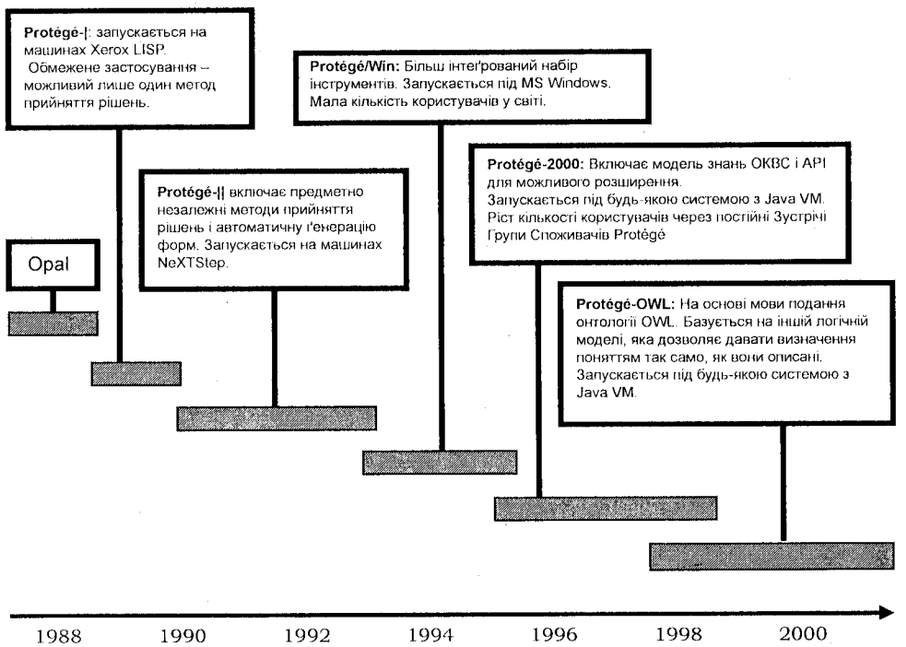


Рис. 11.7. Хронологія розвитку Protégé.

Protégé-II розширив первинний двокроковий процес: генерування інструменту набуття знань і його використання для реалізації бази знань, включивши додаткові кроки, пов'язані з методами пошуку розв'язку задач.

З часом коло споживачів Protégé зростало, надходили нові ідеї та пропозиції, і тому настав час знову перебудувати Protégé. Отримавши нові вимоги до змін, розробники усвідомили, що революційна перебудова системи буде ефективнішою, ніж подальший розвиток Protégé/Win. У Protégé-2000 було введено три основні покращення. По-перше, Protégé-2000 включав реконструкцію основної моделі знань Protégé. Щоб поліпшити виразність баз знань, розробники Protégé співпрацювали з іншими розробниками експертних систем, щоб узгодити більш консенсусну модель відображення знань. Метою було дозволити системам Protégé, на основі знань, взаємодіяти з іншими системами. По-друге, щоб покращити практичність і для кращої відповідності нової моделі знань, Protégé-2000 був побудований як єдина цілісна прикладна програма. І по-третє, щоб забезпечити більшу гнучкість і краще розподілити зусилля для розвитку, спроектований

Protégé-2000 був заснований на змінній архітектурі, підтримуваній мовою програмування Java. Нова архітектура Protégé-2000 наведена на рис. 11.8. В основі цієї архітектури лежить модель знань, котра взаємодіє з усіма об'єктами бази знань (екземпляри, класи і т.ін.), використовуючи інтерфейс прикладного програмування (Application Programming Interface – API) Protégé. Це дозволило незалежним розробникам вбудовувати окремі компоненти, котрі розширюють та змінюють функціональні можливості Protégé.

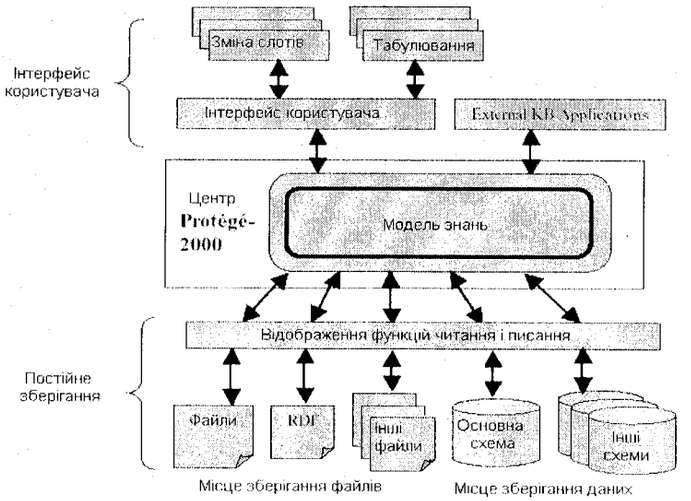


Рис. 11.8. Архітектура Protégé-2000.

Окрім того, розробники Protégé-2000 створили новий інтерфейс, який спрощував споживачам роботу зі створення бази. Якісною зміною, що відбулася в Protégé-2000 порівняно з попередніми версіями Protégé, є *масштабованість* – можливість нарощування системи в рамках уніфікованої архітектури та краща пристосованість до розроблення великих баз знань.

Для розширення кола споживачів і для забезпечення постійного розвитку системи було вирішено зробити Protégé-2000 загальнодоступною. Користь від такого кроку полягала у тому, що Protégé стала привабливою для розробників, які в іншому випадку, можливо, будували б подібні системи з нуля.

Фактично, найбільші перешкоди для подальшого розвитку Protégé мають, швидше, організаційний, ніж технічний характер. Для підтримки співтовариств користувачів та дописувачів Protégé, розробники створили і підтримують жваве „Protégé-обговорення" за допомогою електронного листування, витрачають значну кількість часу та зусиль, відповідаючи на запитання і забезпечуючи технічну підтримку.

11.3.2. Protégé-OWL. Мова web онтологій OWL

Чергова модернізація Protégé-2000 полягала у введенні в основу Protégé нової мови подання знань – OWL. У результаті споживачі отримали потужний інструмент побудови онтологій баз знань з новими можливостями – Protégé-OWL.

OWL (*Ontology Web Language* – мова опису онтологій в Web) – це найновіше розроблення стандартних мов для онтологій, зокрема Web-онтологій. Вона розроблялась як альтернатива DAML, отже, позбавлена деяких її недоліків. Ця мова орієнтована саме на

відображення онтологій, а не семантики в загальному. OWL будується на основі стандартів RDF і RDFS і збагачує останні новими можливостями для опису властивостей і класів. Наприклад, застосовуючи OWL, можна вказувати, що класи не перетинаються, вказувати кардинальність та визначати еквівалентність заданих класів. OWL має у своєму розпорядженні ширшу систему типів порівняно з іншими подібними мовами.

Редактор Protégé-OWL характеризується гнучкістю та модульністю (наявна система плагінів (plug-in) і застосовується в області інженерії онтологій (Ontology Engineering), здобуття знань (Knowledge Acquisition) та автоматичного виведення (Automated Reasoning). Protégé підтримує введення бази знань на будь-якій природній мові, але сам інструмент пропонується лише англійською. Редактор також підтримує будь-яку базу даних з драйвером JDBC 1.0, що дозволяє підтримку переважної більшості реляційних баз даних (Oracle, MySQL, Microsoft SQL Server, Microsoft Access).

Protégé-OWL дає змогу описувати класи з використанням нових можливостей. Зокрема, мова OWL має великий набір операторів і базується на логічній моделі, яка дозволяє давати визначення поняттям так, як вони описані. Тому складні комплексні поняття у визначеннях можуть бути створені з простіших. До того ж, логічна модель дозволяє використовувати механізм міркувань (reasoner), котрий може перевірити, чи твердження і визначення в онтології є взаємно послідовними, а також розпізнати відповідність визначень певним поняттям. Завдяки цьому механізму підтримується правильність ієрархії бази знань.

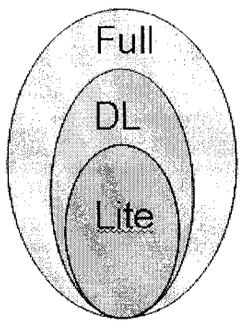


Рис. 11.9. Взаємозв'язок трьох підмов OWL.

У редакторі Protégé-OWL забезпечена можливість вибору однієї з трьох розроблених зараз версій мови OWL: OWL-Lite, OWL-DL, OWL-Full (рис. 11.9). Визначальною особливістю кожної з версій є міра їх виразності. OWL-Lite є найменш виразною підмовою, проте найпростішою з точки зору синтаксису. Її найкраще використовувати при побудові простої ієрархії класів та застосуванні обмежень. OWL-DL є значно виразнішою підмовою порівняно з OWL-Lite. Вона базується на логіці опису (description logics), і підтримує тих користувачів, які хочуть максимальної виразності без втрати повноти обчислень. Завдяки цьому можливо здійснювати автоматичне виведення висновків (процес міркування). Також можна обчислити ієрархію класифікації і здійснити перевірку неузгодженостей в онтології. OWL DL включає в себе всі мовні конструкції OWL з обмеженнями, на зразок розділення типу (клас не може бути властивістю, а властивість не може бути індивідом або класом). OWL-Full використовується в ситуаціях, де дуже висока виразність важливіша, ніж потреба у обчислювальних можливостях мови. Вона забезпечує максимальну синтаксичну свободу стандарту RDF без обчислювальних гарантій.

Разом з тим вона використовує всі переваги попередніх двох підмов. Отже, розробники онтологій, використовуючи OWL, повинні вирішити, яку з підмов краще використати для вирішення конкретних завдань.

У цілому OWL, як мова Web-онтологій вирішує наступні задачі і забезпечує:

- синтаксис опису понять, зручний для всіх користувачів мережі Інтернет – людей та програмних агентів;
- максимальну виразність механізмів опису понять та зв'язків між ними;
- механізми еволюції описів та спільного використання онтологій в середовищі Інтернету.

Ще одна з переваг редактора Protégé – безкоштовне розповсюдження (freeware). Інструмент доступний для завантаження на офіційному сайті проекту: http://protege.stanford.edu.

11.3.3. Основні терміни та поняття в Protégé-OWL

У центрі більшості онтологій перебувають класи. Останні версії Protégé та інші фреймові системи описують онтології декларативним чином, явно визначаючи класову ієрархію та приналежність індивідних концептів до відповідних класів. Онтології, побудовані в OWL, мають подібні компоненти до онтологій на основі фреймів. Проте, на відміну від інших, термінологія OWL базується на поняттях індивідних концептів або об'єктів (найчастіше використовується саме це поняття) та властивостей, які в цілому відповідають в Protégé, відповідно, екземплярам класів і слотам.

Для уникнення неузгодженості, в табл. 11.1 наведені основні терміни (перший рядок) та їх синоніми (другий рядок), які найчастіше зустрічаються в літературі для опису онтологій.

Табл. 11.1. Терміни ті їх синоніми.

Клас	Властивість	Об'єкт
Концепт, категорія, тип, терм, сутність.	Зв'язок, слот, атрибут, роль, обмеження, асоціація.	Індивідний концепт, екземпляр класу, ресурс.

Об'єкти – це окремі екземпляри предметної області. Важлива різниця між Protégé і OWL полягає в тому, що OWL не використовує однозначне присвоєння імен – Unique Name Assumption (UNA). Це означає, що дві різні назви можуть, фактично, посилатися на той самий об'єкт. Наприклад, назви „Королева Елізабет", „Королева" та „Елізабет Віндзор" можуть означати той самий об'єкт. В OWL має бути чітко визначено, що об'єкти є однаковими чи відмінними один від одного – в іншому випадку назви можуть відноситись до тих самих або різних об'єктів.

Властивості являють собою бінарні зв'язки між об'єктами (властивості зв'язують разом два окремі об'єкти). Наприклад, властивість „мати колір" зв'язує об'єкт „Золото" з об'єктом „Жовтий", або властивість „використовується в" зв'язує об'єкт „Золото" з об'єктом „електротехніка" (рис. 11.10).

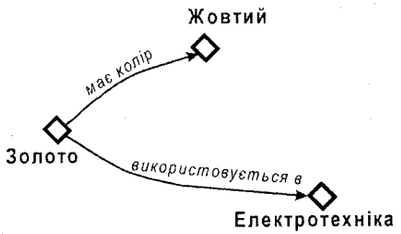


Рис. 11.10. Подання властивостей об'єкту.

У Protégé властивості подаються слотами, в описовій логіці – ролями, в UML та інших об'єктно-орієнтованих поданнях – зв'язками. Властивості можуть бути оберненими, тобто мати інверсію. Наприклад, інверсія до властивості об'єкту „має колір" – „бути кольором". Властивості можуть бути функціональними (обмежені єдиним значенням), транзитивними або симетричними. Ці характеристики детально розглянемо нижче.

Класи в OWL можна розглядати як множини, що містять об'єкти, котрі описуються формально (математично) для точного подання їх членства в певному класі. Класи можна організовувати в ієрархію клас-підклас – таксономія. OWL підклас означає необхідність включення. Наприклад, об'єкти „Чавун" і „Сталь" (рис. 11.11) відносяться до класу „Залізовуглецеві сплави", котрий, в свою чергу, разом з „Металокерамічними сплавами" та „Сплавами на основі кольорових металів" є підкласом „Сплавів". У випадку побудови глибшої ієрархії, об'єкти „Чавун" і „Сталь" можна розглядати як окремі класи зі своїми підкласами та об'єктами.

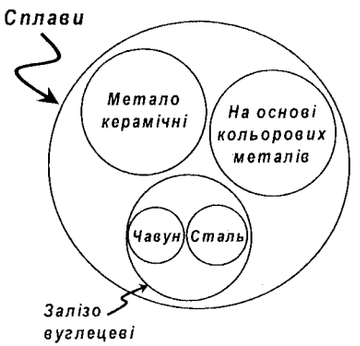


Рис. 11.11. Подання структури класів.

У OWL класах створюються описи, які конкретизують умови, котрим повинен відповідати об'єкт, щоб увійти до складу екземплярів класу. Детальніше про створення описів належності об'єктів до певних класів засобами Protégé-OWL описано в наступних параграфах.

11.3.4. Методика розроблення онтології засобами Protégé

Перед тим як приступити безпосередньо до проектування онтології, варто звернути увагу на деякі фундаментальні правила її розроблення, які у багатьох випадках можуть допомогти при ухваленні проектних рішень:

1) онтологія – це модель реального світу і поняття в ній повинні відображати цю реальність;

2) не існує єдиного правильного способу моделювання предметної області – завжди є життєздатні альтернативи; краще рішення часто залежить від вимог до кінцевого продукту та очікуваних модернізацій;
3) розроблення онтології – це обов'язково ітеративний процес, тобто постійно відбувається наповнення і уточнення;
4) поняття в онтології повинні бути максимально близькими до реальних об'єктів (фізичних чи логічних) та їх властивостей.

Нижче наведено загальну методику побудови онтологій, котру можна реалізувати засобами Protégé. Ця методика складається з семи кроків.

Крок 1. Визначення області та масштабу онтології. Роботу над розробленням онтології слід почати з визначення її обсягу та області застосування. Для цього на початку розробляють питання компетентності для перевірки відповідності онтології заданій предметній області, які надалі будуть виконувати функції лакмусового папірця, даючи уявлення про повноту поданої інформації та рівень її деталізації.

Крок 2. Можливість використання наявних онтологій. Варто врахувати, що над задачею створення онтології, наприклад, в області матеріалознавства, працював ще хтось. Тоді потрібно перевірити можливість адаптації існуючих онтологічних систем для нашої конкретної предметної області. В іншому випадку роботу треба розпочинати з нуля. На сьогодні є доступними багато розроблених онтологій в різних предметних областях. Вони можуть бути успішно імпортовані в середовище проектування, вибране розробником.

Крок 3. Перелік важливих термінів в онтології. Корисно скласти список всіх термінів та їх властивостей, які несуть основну інформацію про задану предметну область. Наприклад, у число важливих термінів, пов'язаних з матеріалознавством, входять металознавство, залізовуглецеві сплави, метали, їх пластична деформація; матеріали, напівпровідники, донорні домішки та хімічна чистота елемента; пористість порошкових матеріалів тощо. На початку важливо отримати повний список термінів, не турбуючись про те, наприклад, чи є поняття класом чи властивістю.

Наступні два кроки найважливіші у процесі проектування онтології: це розроблення ієрархії класів і визначення властивостей понять (слотів), які тісно між собою переплетені. Тому їх можна виконувати паралельно.

Крок 4. Визначення класів та їх ієрархії. Існує декілька підходів для побудови ієрархії класів: зверху-вниз, знизу-вгору та комбінований процес. На основі власного досвіду та ряду інструкцій з розроблення онтологій зазначимо типові помилки, які допускають розробники при проектуванні ієрархії понять.

1. Включати в ієрархію одне і те ж поняття як в однині, так і у множині, зробивши перше підкласом другого. Кращий спосіб уникнути такої помилки – завжди використовувати імена класів або в однині, або у множині.
2. Не розрізняти клас і його ім'я: класи відображають поняття предметної області, а не слова, які позначають ці поняття.
3. Сприймати синоніми як представників різних класів.
4. Створювати цикли в ієрархії класів. Вважається, що в ієрархії є цикл, коли в деякого класу A є підклас B, і в той же час B – надклас A.

Крок 5. Визначення властивостей класів. Після визначення певної кількості класів необхідно описати внутрішню структуру понять. На кроці 3 були вибрані класи зі ство-

реного списку термінів. Більшість термінів, що залишилися, ймовірно, будуть властивостями цих класів. Всі підкласи класу успадковують властивість цього класу.

Крок 6. Визначення фацетів властивостей. Властивості можуть мати різні фацети, які описують тип і коефіцієнт (потужність) значення властивості, діапазон та інші характеристики, які може мати властивість.

Потужність властивості визначає, скільки значень може мати ця властивість. У деяких системах розрізняються тільки одиничну потужність (лише одне значення) і множинну потужність (будь-яку кількість значень). Можна встановити мінімальну потужність – 0. Це встановлення означатиме, що для певного підкласу слот не може мати значень. Фацет описує тип значення слота, які типи значень (рядок, число, число з плаваючою комою, ціле число) можна присвоїти властивості. Нумеровані властивості визначають список конкретних дозволених значень властивості.

Крок 7. Створення екземплярів. Останній крок – це створення окремих екземплярів класів в ієрархії. Для визначення окремого екземпляра класу необхідно: 1) вибрати клас; 2) створити окремий екземпляр цього класу і 3) ввести значення слотів.

Розроблення онтологій відрізняється від проектування класів і зв'язків в об'єктно-орієнтованому програмуванні. Об'єктно-орієнтоване програмування зосереджується, головним чином, на методах класів – програміст ухвалює проектні рішення на основі операторних властивостей класу, тоді як розробник онтології ухвалює ці рішення, ґрунтуючись на структурних властивостях класу. Як результат структура класу і відношення між класами в онтології відрізняються від структури подібної предметної області в об'єктно-орієнтованій програмі.

11.4. Створення та експлуатація онтології

На сьогодні у світі гостро стоїть питання створення інтелектуальних систем в певних галузях науки та промисловості, які могли б у значній мірі спростити інженерам та науковцям пошук та доступ до величезних об'ємів інформації. Багато наявних онтологій призначені для використання у промисловості у складі інтелектуальних систем, функцією яких є прийняття рішень під час вирішення конкретних завдань. Як приклад можна навести побудову та використання онтології у складі системи керування знаннями в металопромисловості в Тайвані. Ця онтологія використовується в системі керування знаннями на етапі маніпулювання і підтримки задачі керування та пошуку. Однак вона охоплює лише певну частину області матеріалознавства – металургію, а також не забезпечує повного подання властивостей матеріалів, хоча досить в повній мірі містить знання про методи та технології їх виготовлення. Існує мало універсальних інтелектуальних систем, онтологія яких охоплювала б всю предметну область науки, в якій працює ця система, а в наукових дослідженнях потрібно мати якнайповнішу інформацію про об'єкт досліджень. Тому інтелектуальна система має забезпечити користувачу вичерпну інформацію, при цьому необхідно передбачити своєчасне поновлення бази знань новими знаннями.

Розроблена онтологія слугуватиме ядром бази знань для інтелектуальної системи, яка буде забезпечувати пошук та семантичне розпізнавання інформації, що надходить, зокрема з мережі Інтернет, накопичення і класифікацію отриманої інформації та її використання в наукових дослідженнях. Використання такої системи в поєднанні з можли-

востями Інтернету дозволить науковцям та інженерам з певної ПО різних наукових установ мати доступ до потрібної інформації, здійснювати швидкий та ефективний пошук та отримувати експертні висновки від інтелектуальної системи на основі запитів. Особливістю бази знань розроблюваної інтелектуальної системи є можливість динамічного наповнення її новими знаннями та наявність алгоритмів оптимізації структури знань. Це дозволить мінімізувати час та затрати на розвиток та модернізацію системи в процесі її використання.

Онтологія як основа бази знань такої інтелектуальної системи, розробляється з використанням інструментів редактора Protege , і перша її версія була написана українською мовою. Але, оскільки переважна частина інформації отримується системою з мережі Інтернет, було вирішено перекласти онтологію англійською мовою.

Далі буде детальніше описано етапи процесу створення онтології з використанням інструментів Protege OWL як найбільш гнучкого та придатного засобу для розв'язування такого класу задач.

11.4.1. Створення онтології

Створення та збереження нового проекту Protégé -OWL

Для створення нового проекту необхідно увійти в середовище Protege (запустити програму Protégé.exe) та у відкритому вікні вибрати закладку «Create New Project» (рис. 11.12). Після цього у списку «Project Format» вибрати тип створюваної онтології «OWL Files» та натиснути клавішу «Finish». Для відкриття вже існуючого проекту потрібно натиснути клавішу «Open Existing Project», вибрати потрібний файл збереженої онтології та натиснути «OK». Для швидкого відкриття нещодавно створеної або редагованої онтології є можливість її відкрити, скориставшись вікном «Recently Accessed Projects» (рис. 11.12).

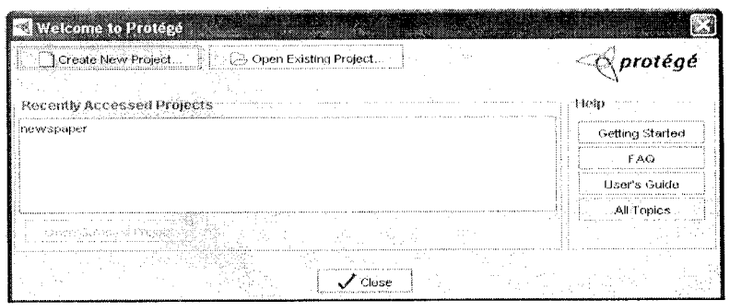


Рис. 11.12. Стартове вікно програми Protégé.

Після цього відкривається головне вікно проекту, в якому безпосередньо відбувається створення та редагування онтології. Для збереження нової онтології необхідно вибрати в головному меню File/Save Project As, після чого задати ім'я проекту в полі Project вказавши повний шлях розміщення його на диску. При цьому автоматично створюються назви файлів класів та екземплярів. У випадку внесення змін в існуючий проект, для його збереження достатньо натиснути клавішу на панелі інструментів або вибрати в меню File/Save Project.

Створення нових класів та присвоєння їм імен

При створенні нової онтології або відкритті існуючої відкривається головне вікно проекту Protégé-OWL на закладці OWLClasses (рис. 11.13), де створюються та дода-

ються нові класи та підкласи онтології. Для переходу в діалогові вікна введення властивостей, форм, об'єктів та метаданих слід скористатись відповідними закладками у верхній частині вікна.

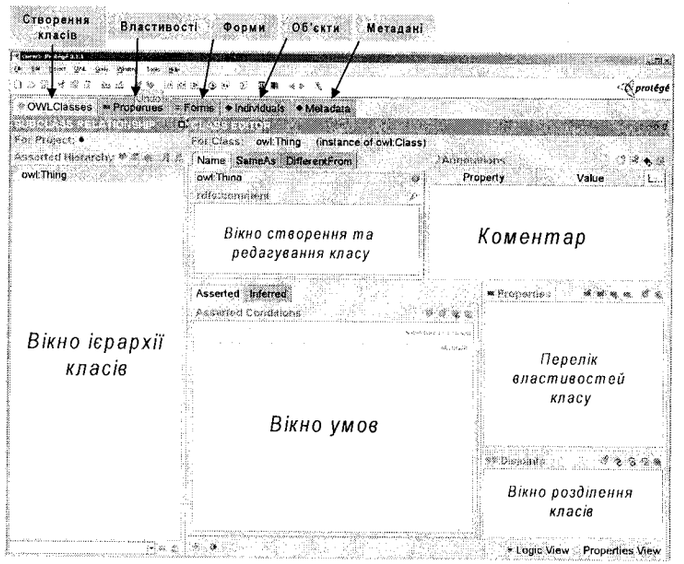


Рис. 11.13. Головне вікно редактора Protégé – створення класів.

Закладка OWLClasses складається із двох частин. Перша – це вікно ієрархії класів (Subclass Relationship), де відображаються у вигляді дерева усі введені класи та підкласи онтології. Друга частина (Class Editor) містить вікна, в яких відображається інформація про вибраний клас або підклас онтології – назва, властивості, умови, коментарі.

Порожня онтологія містить лише один клас owl: Thing – цей клас відображає множину всіх об'єктів, оскільки всі наступні класи є його підкласами. Це системний клас, котрий встановлюється за замовчуванням і в процесі створення онтології не змінюється. Для додавання в онтологію нового класу у вікні ієрархії класів необхідно виділити вже існуючий клас, і натиснути клавішу «Create Subclass» (рис. 11.14). При цьому створиться підклас до виділеного класу. У вікні введення назви класу, розміщеному в частині «Class Editor» (рис. 11.15) справа від ієрархії класів, змінюємо ім'я, встановлене за замовчуванням, на потрібне, наприклад, «Метал». Нижче є можливість введення додаткового коментаря для цього класу.

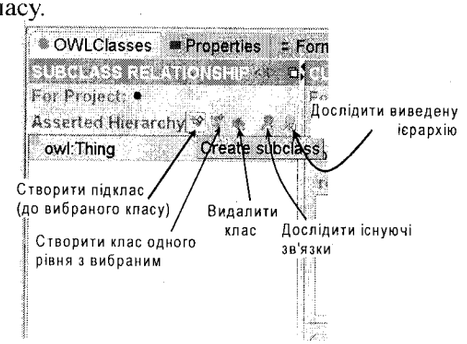


Рис. 11.14. Вікно ієрархії класів.

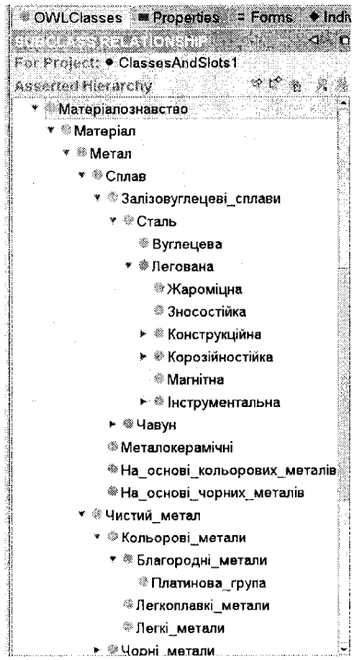


Рис. 11.15. Вікно введення назви класу.

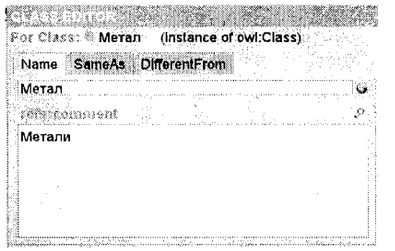


Рис. 11.16. Ієрархія класів онтології матеріалів.

Для введення класу одного рівня з вибраним, необхідно натиснути клавішу «Create sibling class». Для видалення класу з ієрархії існує клавіша «Delete Class». Однак перед видаленням необхідно видалити будь-які об'єкти (екземпляри класу), котрі мають відношення до цього класу. Отже, після введення класу «Матеріалознавство» як основного для онтології матеріалів, використовуючи вищенаведені підходи, можна ввести усі її підкласи (рис. 11.16).

Спростити процедуру створення класів можна за допомогою майстра, який дозволяє одночасно вводити декілька класів. Запускається майстер шляхом вибору в меню команд Tools/Quick OWL/Create multiply subclasses.

Коли створено декілька класів онтології, можна вказати, що вони розділені (disjoint), тобто об'єкти одного з них не можуть бути об'єктами іншого. Для цього існує вікно «Disjoints Widget» у правому нижньому куті закладки «OWLClasses» (див. рис. 11.13). У цьому вікні шляхом додавання класів з ієрархії, вибираються ті класи, які необхідно розділити.

OWL-властивості. Створення властивостей класів

Після ведення нового класу в онтологію потрібно задати його властивості для того, щоб поняття предметної області були наповнені певним змістом і перебували в певних зв'язках один з одним. Існують два основні види властивостей: властивості об'єктів (зв'язують об'єкт з об'єктом) та властивості типів даних (зв'язують об'єкт зі значенням типу даних XML-схеми або rdf літералом). Мова OWL забезпечує також третій тип властивостей, які називаються властивостями анотацій. Вони використовуються для додавання інформації до класів, екземплярів класів і властивостей об'єкт/тип-даних. Приклади трьох типів властивостей подано на рис. 11.17.

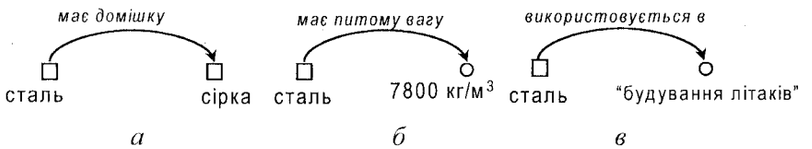


Рис. 11.17. Різні типи OWL властивостей:

Так, на рис. 11.17 а властивість «має домішку» зв'язує об'єкт «сталь» класу «метали» з об'єктом «сірка» класу «неметали»; на рис. 11.17 б властивість «має питому вагу» зв'язує об'єкт «сталь» із числовим значенням «7 800 кг/м3»; а на рис. 11.17 в як додаткова інформація застосовується описова властивість «використовується», яка зв'язує клас «сталь» з текстовими даними, наприклад, «будування літаків». При створенні нової властивості класу або об'єкта в Protégé потрібно вибрати відповідний їх тип (рис. 11.17).

Інверсія властивостей. Властивість кожного об'єкту може мати відповідну їй обернену властивість. Якщо якась властивість з'єднує об'єкт a з об'єктом b, тоді об'єкт b з об'єктом a буде зв'язувати обернена до неї властивість (рис. 11.18).

Рис. 11.18. Приклад інверсії властивостей.

Характеристики OWL властивостей.

Зміст та значення властивостей можна конкретизувати шляхом використання відповідних характеристик властивостей. У Protégé-OWL проекті виділяють наступні властивості: функціональні, обернено-функціональні, транзитивні та симетричні.

1. **Функціональна властивість** (особливість) деякого об'єкту використовуються для встановлення зв'язку з не більш, ніж одним іншим об'єктом або типом даних. За допомогою функціональних властивостей задаються характерні особливості об'єкту, що відрізняють його від інших. Якщо об'єкт пов'язаний однією функціональною властивістю з різними об'єктами або даними, тоді їх можна вважати однаковими, або такими самими. В іншому випадку це призведе до неузгодженості в базі знань, що не допустимо.
2. **Обернено-функціональні властивості.** Функціональні властивості також можуть бути оберненими. Тобто, якщо два окремі об'єкти зв'язані однією функціональною

властивістю з деяким третім об'єктом, то можна стверджувати, що ці два об'єкти є ідентичними або однаковими.

3. **Транзитивні властивості.** Якщо властивість є транзитивною, і вона зв'язує деякий об'єкт a з об'єктом b, а об'єкт b, у свою чергу, зв'язаний такою ж властивістю з об'єктом c, тоді можна стверджувати, що об'єкт a зв'язаний цією властивістю з об'єктом c. Так, відомо, що залізо за питомою вагою важче за мідь, а мідь, у свою чергу, важча за алюміній. Звідси можна стверджувати, що залізо є важчим за алюміній (рис. 11.19).

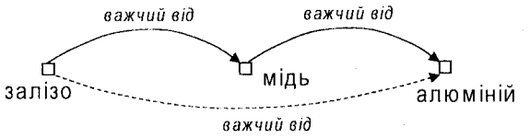


Рис. 11.19. Приклад застосування транзитивної властивості.

Зауважимо, що обернена до транзитивної властивість також транзитивна. А також, транзитивна властивість не може одночасно бути функціональною.

4. **Симетричні властивості.** Якщо об'єкт *a* пов'язаний з об'єктом *b* симетричною властивістю, тоді об'єкт *b* також зв'язаний з об'єктом *a* через таку саму властивість. Наприклад, залізо взаємодіє з сірчаною кислотою, але ми можемо сказати, що і сірчана кислота взаємодіє з залізом (рис. 11.20). У цьому випадку властивість *„взаємодіє з"* — симетрична.

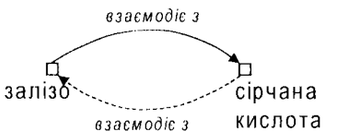


Рис. 11.20. Приклад застосування симетричної властивості.

Створення властивостей і присвоєння їх класам.

Для створення нової властивості використовується закладка „Properties" головного вікна програми (рис. 11.21).

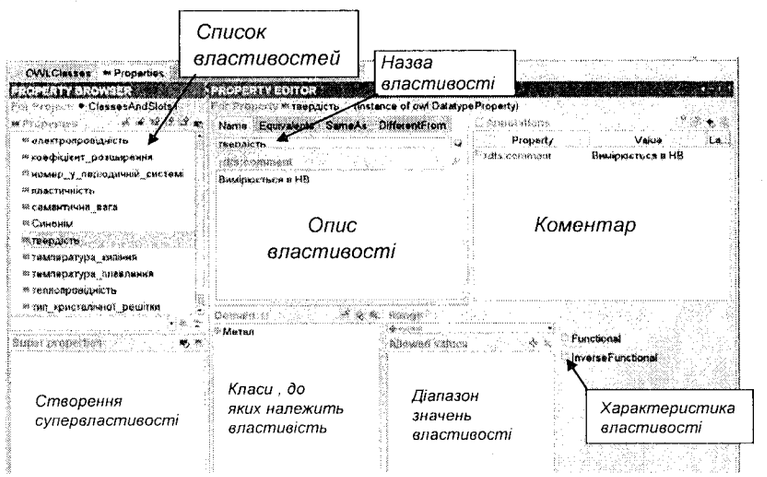


Рис. 11.21. Вікно створення властивостей.

Як і попереднє, вікно створення властивостей складається із двох частин – Списку властивостей (Property Browser) та Редактора властивостей (Property Editor). У вікні Списку властивостей відображаються усі створені властивості, а за допомогою клавіш керування, розташованих у верхній частині цього вікна, є можливість створювати нові (див. рис. 11.22).

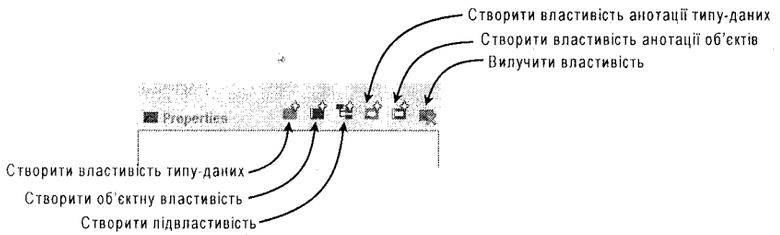


Рис. 11.22. Клавіші керування властивостями.

Так, для додавання нової властивості в онтологію, залежно від її типу, необхідно натиснути відповідну клавішу, після чого у вікнах редактора властивостей ввести її назву, опис, коментар та діапазон можливих значень. Для вибору характеристики властивості (функціональна, обернена функціональна, транзитивна, симетрична) потрібно поставити галочку навпроти відповідної характеристики в нижньому правому куті вікна. Зауважимо, що при створенні властивості типу даних в ній не можна задати характеристики транзитивності, симетрії та інверсії. У випадку необхідності введення інверсії для об'єктної властивості використовується вікно введення інверсії (рис. 11.23), в якому за допомогою керівних клавіш додаємо або видаляємо інверсію до вибраної властивості.

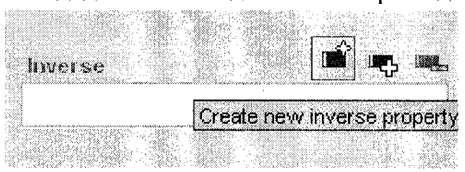
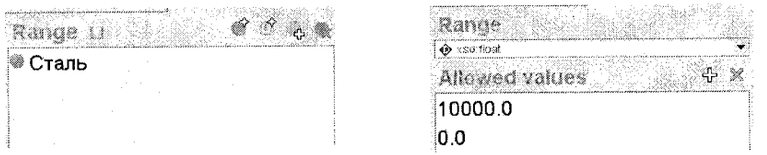


Рис. 11.23. Вікно інверсії властивостей

Присвоєння властивості певному класу відбувається у вікні класів (Domain U) на закладці «Properties», в якому вказується перший клас, до якого належить ця властивість (див. рис. 11.21). Згідно із принципом успадкування така властивість присвоюється всім підкласам цього класу. При натисканні клавіш «Create named class...» або «Add named class(es)» можна відповідно створити новий або додати існуючий клас до списку. За допомогою клавіш «Remove selected class(es) from Domain» вибрані класи видаляться зі списку.

Також для кожної властивості можна задати діапазон об'єктів або значень, з якими вона буде зв'язувати клас, до якого ця властивість належить. Так, у випадку, якщо деякий клас має об'єктну властивість, то можна задати клас (або класи), об'єкти якого (яких) будуть зв'язані цією властивістю з об'єктами цього класу. Для цього використовується вікно діапазону значень (рис. 11.24 а). За допомогою керівних клавіш створюються нові або додаються з ієрархії класів ті класи, об'єкти яких будуть зв'язані такою властивістю. Так, наприклад, якщо ми в цьому вікні для деякої об'єктної властивості виберемо клас «Сталь», то вона буде зв'язувати об'єкти класу, до якого вона належить, тільки з об'єктами класу «Сталь». У випадку властивостей типу даних у вікні діапазону

значень вказуються дозволені значення та їх тип, які може приймати ця властивість. Наприклад, значення деякої властивості, що описує певний клас, повинно бути 10 000 або 0 (рис. 11.24 б).



а *б*

Рис. 11.24. Застосування діапазону значень властивостей.

В OWL властивості можуть мати свої підвластивості, таким чином можна також будувати ієрархію властивостей (підвластивість до вибраної властивості створюється за допомогою відповідної клавіші керування (рис. 11.22). При цьому супер-властивість до вибраної підвластивості виводиться у вікні «Super Properties» (див. рис. 11.21).

Властивості вибраному класу можна присвоювати також безпосередньо з вікна створення класів. Для цього у вікні переліку властивостей (Properties) вибраного класу (див. рис. 11.13) існують відповідні клавіші керування, при натисканні яких відкривається вікно редактора властивостей (Property Editor), де задаються параметри властивості.

Опис та визначення класів

Після того, як було введено певну кількість класів та властивостей, ми можемо використовувати ці властивості для опису та визначення класів нашої онтології. Це робиться для встановлення зв'язків між об'єктами різних класів онтології.

Обмеження дії властивості. В OWL властивості використовуються для створення обмежень. Обмеження поділяються на три групи:

- кванторні обмеження;
- числові обмеження;
- обмеження типу «має_значення».

Для початку розглянемо кванторні обмеження. Цей тип обмежень складається із квантора, властивості та деякого класу об'єктів. У кванторних обмеженнях використовуються два квантори – квантор існування (∃), котрий можна прочитати як «принаймні один» або «деякі», та універсальний квантор (∀), котрий означає «тільки». Наприклад, вираз типу «∃ Використовують Конструкційні сталі» складається із квантора існування (∃), властивості «Використовують» та класу об'єктів «Конструкційні сталі» і визначає набір об'єктів або клас, які є зв'язані та можуть використовувати об'єкти із класу «Конструкційні сталі». А вираз типу «∀ має_легуючий_елемент Чистийметал» визначає набір об'єктів або клас, всі елементи якого є зв'язаними з елементами класу «Чистий метал». Цей вираз застосований, наприклад, до класу «Леговані сталі», означає, що всі об'єкти цього класу мають легуючі елементи, які відносяться до класу «Чисті метали».

Обмеження, присвоєні об'єктам деякого класу, виводяться у вікні умов (рис. 11.25), котре розташоване на закладці Створення класів (OWL Classes) (див. рис. 11.13).

Рис. 11.25. Вікно умов (введення та редагування обмежень).

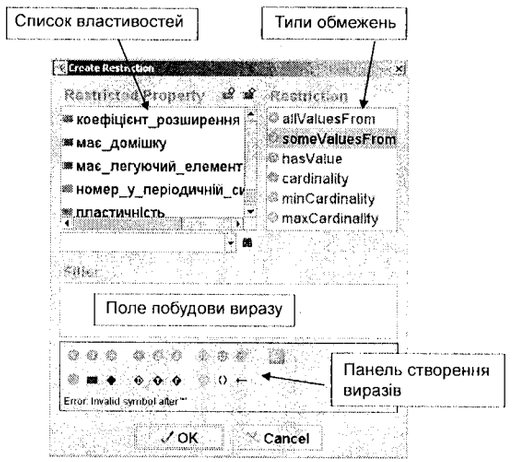


Рис. 11.26. Вікно введення виразу обмежень.

Вікно умов містить дві закладки – Введені Умови (Asserted) та Виведені Умови (Inferred). У вікні Asserted виводиться список усіх введених умов та обмежень для вибраного класу. У свою чергу список умов розділений на дві частини: Necessary – список умов, що належать саме вибраному класу онтології; Inferred – в цій частині виводяться умови, присвоєні надкласам (суперкласам) до вибраного, які згідно із принципом успадкування присвоюються й об'єктам цього класу. У правій частині рядка зі списку Inferred можна побачити, до об'єктів якого надкласу була присвоєна ця умова. Як і більшість вікон Protege, вікно умов має клавіші керування, за допомогою яких можна вводити вирази умов. Введення умов можливо здійснити двома способами: перший – натиснувши на кнопку введення за допомогою рядкового редактора «Create New Expression». У цьому випадку весь вираз умови необхідно буде вводити вручну, що в більшості випадків є незручним. Другий спосіб введення умови дозволяє зробити це в окремому вікні, де є можливість вибрати потрібний квантор, властивість зі списку властивостей, потрібний клас об'єктів та додаткові символи, необхідні для формування виразу.

Для введення обмеження для деякого класу об'єктів в окремому вікні потрібно у вікні ієрархії класів виділити потрібний клас, після чого у вікні умов натиснути клавішу «Create Restriction...». Далі у вікні (рис. 11.26) у списку властивостей виділити потрібну властивість (наприклад, «має_

Домішку), а у сусідньому віконечку – вибрати тип обмеження (наприклад, someValuesFrom). Після цього набрати в полі побудови назву класу, об'єкти якого будуть зв'язані вибраною властивістю (наприклад, «Неметал»), або натиснути клавішу «Insert class» на панелі створення виразу і вибрати потрібний клас з ієрархії. За допомогою панелі створення виразу є можливість створювати вирази, що містять логічні операції (об'єднання, перетин, логічне заперечення), булеві оператори та дужки. Після натискання клавіші «OK» вікно закриється і сформований вираз обмеження автоматично додасться до списку умов.

Числові обмеження. В OWL є можливість описати клас об'єктів, що мають не менш ніж, не більш ніж або чітко визначену кількість зв'язків між іншими об'єктами або даними. Обмеження, за допомогою яких встановлюються такі умови, називаються числовими (Cardinality restrictions). Тобто для певної властивості можна задати мінімальне, максимальне або визначене число зв'язків, якими вона буде зв'язана з об'єктами деякого класу. Наприклад, за допомогою такої умови можна задати процентний вміст та максимальну кількість домішок в чистому металі, при якому цей метал ще можна вважати чистим (рис. 11.27). Принцип створення таких обмежень аналогічний попередньому, тільки у цьому випадку для опису використовуються типи «Cardinality», «minCardinality» і «maxCardinality», а в полі побудови виразу вказується певне числове значення.

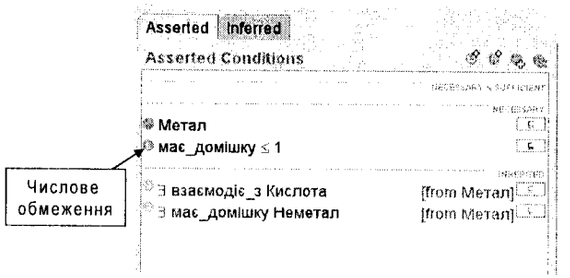


Рис. 11.27. Приклад застосування числового обмеження для класу „Чисті метали".

Обмеження типу «має значення». Ці обмеження позначаються символом з і визначають клас або набір об'єктів, що, використовуючи деяку властивість, мають принаймні один зв'язок з конкретним об'єктом. Наприклад, вираз «має_Домішку з Сірка» (де «Сірка» – об'єкт класу «Неметали») визначає набір об'єктів (наприклад, металів), які можуть мати домішку саме сірки. Тобто, якщо попередніми обмеженнями ми зв'язували за допомогою властивостей один набір об'єктів (клас) з іншим набором (класом) або деякими даними, то в цьому випадку ми вказуємо конкретний об'єкт, який безпосередньо пов'язаний такою властивістю з деяким класом або набором об'єктів.

Механізм міркувань

Однією з важливих особливостей використання мови OWL при побудові онтологій є можливість опрацьовувати їх за допомогою механізму міркування (reasoner). Під опрацьовуванням онтології в такому випадку розуміється перевірка правильності присвоєння підкласів певним класам, внаслідок чого автоматично формується виведена

ієрархія класів онтології. У Protege-OWL ієрархія класів, внесена вручну, називається заявленою ієрархією, а вирахувана механізмом міркувань – виведеною ієрархією. Також за допомогою механізму міркувань перевіряється логічна узгодженість (несуперечність) класів і підкласів онтології. На основі введених умов для кожного класу механізм міркування дає можливість встановити, чи певні об'єкти (екземпляри) можуть належати вказаним класам.

Використання механізму міркувань є особливо зручним при роботі з дуже великими онтологіями (декілька тисяч класів). При цьому виникає реальна необхідність автоматично визначати і відслідковувати взаємозв'язки в ієрархії «клас – суперклас». У таких випадках без використання механізму міркування надзвичайно важко підтримувати онтологію великих розмірів логічно узгодженою. Цей механізм забезпечує відповідність створеної онтології іншим онтологіям та прикладним програмам, а також мінімізує вплив людських помилок.

Для реалізації механізму міркувань в середовищі Protege-OWL необхідно встановити і запустити додатковий модуль, зокрема це може бути RACER, розроблений групою розробників інструментів описової логіки (DIG – Description Logic

Implementers Group). Цей модуль та інструкція з його встановлення доступні на сайті http://www.sts.tu-harburg.de/~r.f.-moeller/racer/.

Після того, як встановлено і налагоджено відповідний механізм міркування, введену вручну онтологію можна «відіслати до механізму міркування» для автоматичної класифікації та перевірки узгодженості. Це можливо здійснити, вибравши в головному меню програми пункт OWL, а в ньому – дію «Classify taxsonomy... » (рис. 11.28). Аналогічно, за допомогою дії «Check consistency...» перевіряється онтологія на суперечність. Ці дії також можна здійснити натисканням відповідних кнопок на панелі інструментів.

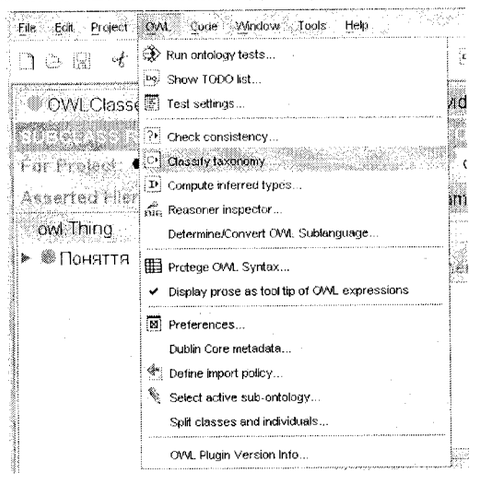


Рис. 11.28. OWL меню.

Коли виведена ієрархія обчислена, то вікно виведеної ієрархії відкриється поряд із вікном заявленої ієрархії під назвою «Inferred Hierarchy». При цьому структура класів може значно відрізнятись від введеної вручну. Ця різниця буде залежати, в першу чергу, від умов, присвоєних певним класам та їх екземплярам. Деякі класи, внаслідок автоматичної класифікації, можуть змінити свій суперклас. Тоді цей клас у виведеній ієрархії буде виділено синім кольором.

При виявленні суперечливого класу в онтології, він буде виділений у заявленій ієрархії червоним кольором. Це може бути у випадку, якщо попередньо було вказано, що об'єкти одного класу не можуть належати іншому класу (задається у вікні «Disjoint» (див. рис. 11.13). Тобто, якщо механізм міркування виявив, що деякий підклас належить одночасно різним класам, що були розділені, то він виділить цей клас як суперечливий. Для видалення суперечності необхідно відмінити розділення між класами, в яких виникла ця суперечність, шляхом видалення суперечливого класу у вікні «Disjoint».

Введення екземплярів класів

Після того, як введено основні класи нашої онтології, визначено їх властивості та встановлено умови їх взаємозв'язку, для забезпечення конкретизації предметної області вводяться екземпляри (об'єкти) класів. Об'єкти вважаються найнижчим рівнем онтології, вони успадковують всі властивості класів, до яких вони належать. Прикладами об'єктів в онтології матеріалів можуть бути конкретні марки сталей або назви чистих металів (наприклад: Ст3, сталь 20, 15Х2МФА, мідь, залізо). У випадку, якщо об'єкт має під собою інші об'єкти, він автоматично стає класом.

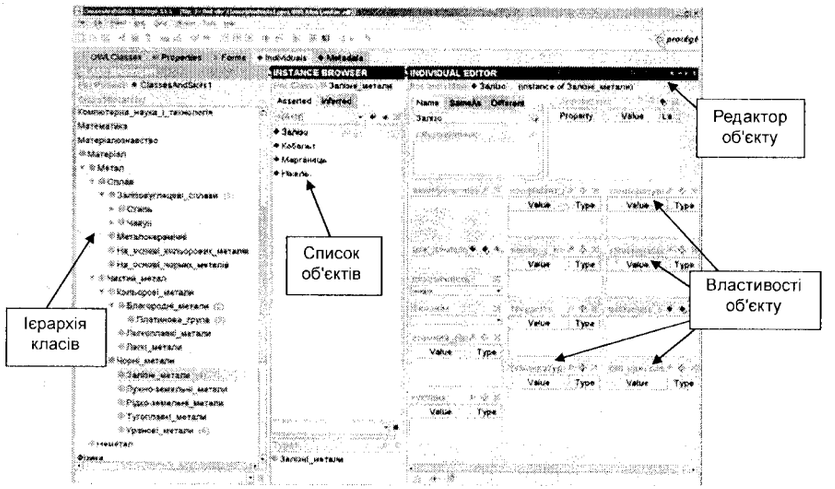


Рис. 11.29. Вікно екземплярів.

Об'єкти в Protégé-OWL можна створювати і редагувати у вікні об'єктів на закладці „Individuals" головного вікна програми (рис. 11.29).

Вікно екземплярів складається із трьох основних частин. Зліва розташоване вікно ієрархії класів онтології (Class Browser), в якому вибирається клас, екземпляр якого створюється. При цьому навпроти кожного класу в дужках вказується кількість об'єктів, що належать саме цьому класу. При виділенні певного класу в центральному вікні (Instance Browser) виводиться список об'єктів, що належать такому класу. За допомогою клавіш керування, що розташовані над списком екземплярів, можна створити новий екземпляр у виділеному класі, скопіювати уже існуючий екземпляр або видалити вибраний екземпляр зі списку. При виборі певного об'єкту зі списку або при створенні нового в крайній правій частині вікна відкривається редактор екземплярів (Individual Editor), за допомогою якого можна змінити ім'я об'єкта, додати опис та коментар, а також внести значення властивостей об'єкта. У редакторі об'єкта виводяться всі властивості, унаслідувані ним від батьківського класу. Також є можливість додавати або видаляти певні властивості

цього класу (самі об'єкти власних властивостей не мають). Кожна властивість має окреме поле, в якому вноситься її значення для конкретного об'єкту. Введення нової або видалення існуючої властивості відбувається за допомогою відповідних клавіш керування у верхній частині вікна, а внесення значення властивості – за допомогою клавіш, розташованих біля її назви або з випадного списку.

Описавши всі класи, властивості, обмеження і об'єкти предметної області, одержуємо базу знань, що є основою для побудови інтелектуальних систем, здатних здійснювати операції над інформацією. Створену онтологію в редакторі Protégé -OWL можна тепер експортувати у формат, зрозумілий для інших редакторів, які займаються безпосередньо розробленням інтелектуальних систем (CLIPS, HTML, RDF, OWL). У складі інтелектуальної системи базова онтологія повинна постійно розширюватись і модифікуватись. Це може відбуватись в автоматичному режимі, завдяки спеціально розробленим алгоритмам; в іншому випадку онтологію необхідно редагувати вручну (використовуючи редактори типу Protégé -OWL).

11.4.2. Автоматичний розвиток онтології у складі інтелектуальної системи

Кінцевою метою побудови бази знань є створення на її основі інтелектуальної системи, яка може передбачати у своїй архітектурі наявність природомовного інтерфейсу користувача, процедур опрацювання документів за змістом, алгоритмів пошуку та реферування документів тощо. У цьому випадку база знань не може бути статичною, а повинна динамічно розвиватися в процесі експлуатації системи. Одним з найефективніших підходів до навчання бази знань є її автоматичне навчання природомовними текстами, тобто необхідні знання отримувати з тексту, де під текстом розуміємо множину текстових документів (книг, статей, патентів, звітів тощо) в електронній формі (DOC, PDF, RDF, HTML, XML, TXT та ін.).

Отже, одним зі способів автоматичного розвитку бази знань новими знаннями на основі вже створеної онтології в редакторі Protégé-OWL, є метод її навчання за допомогою аналізу природомовних текстів. Зокрема, одним з підходів до автоматичного навчання бази знань є аналіз текстових документів шляхом застосування процесора знань (рис. 11.30).

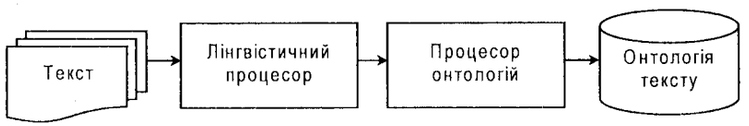


Рис. 11.30. Структурно-функціональна схема процесора знань.

У поданій схемі задача лінгвістичного процесора – переформатувати текст, виконати його лексичний, лексико-граматичний, синтаксичний та семантичний аналіз. У результаті цього база знань поповнюється: поняттями, САО-трійками (суб'єкт – акція – об'єкт) і причинно-наслідковими зв'язками між САО-трійками. Іншу частину важливих зв'язків між поняттями та їх властивостями встановлює процесор онтологій, котрий будує онтологію для кожного концепту C_i, отриманого після аналізу тексту. Це онтологія типу: 1) структура C_i (головне/атрибут); 2) властивості C_i, наприклад, C_i – складається з...; C_i – має параметри...; C_i – має недоліки...; 3) семантичне визначення C_i; 4) відношення C_i (ієрархічні, синонімічні, функціональні). Робота процесора онтологій підтримується

відповідною БЗ, основними компонентами якої є, по-перше, множина правил, по-друге, універсальна логічна база даних ModWN типу WordNet.

Процесор знань застосовується в системі автоматичного набуття знань з текстових документів, яка, у свою чергу, може ефективно вирішувати завдання семантичного пошуку в повнотекстових базах даних.

Серед систем, розроблених в Україні, відзначимо розроблення кафедри математичної інформатики Київського національного університету імені Тараса Шевченка – систему опрацювання текстів природною мовою. Система створена для розв'язування таких задач, як аналіз і синтез текстів на природній мові, автоматична генерація реферату тексту, автоматична індексація (визначення тематики) тексту.

Як база знань використовується онтологічно-семантична мережа – «Семантична онтологія» – напрямлений гіперграф, кожна вершина якого являє собою концепт і має набір зв'язків-відношень, які з'єднують цю вершину-концепт з іншими вершинами-концептами, тобто зміст концепту відображається його релятивною позицією. У базі знань, в основному, реалізовані зв'язки «бути "/" is_a», які утворюють онтологічний ієрархічний граф концептів природної мови (ієрархічність забезпечує ефективне використання механізму успадкування, що дозволяє уникнути надлишковості), а також «мати властивість», «робити». Така мережа побудована на базах даних лексичної та семантичної інформації словоформ англійської мови.

Найвагомішим технічним рішенням у системі є можливість «зважувати» вершини семантичної мережі тексту. При цьому найважливішими вершинами мережі вважаються вершини, які мають найбільшу кількість зв'язків з іншими. Така процедура може застосовуватись при побудові образу реферату шляхом зважування вершин і відкидання найлегших – «маргінальних». Порівняльний аналіз концептів і зв'язків отриманого образу з мережами проблемних областей, які містяться в семантичній онтології системи, забезпечує індексацію вхідного тексту.

Серед позитивних властивостей системи потрібно відзначити можливість легкого додавання модулів нових мов. Очевидний і ряд недоліків: по-перше, процедура зважування не враховує вагу зв'язків-відношень між вершинами-поняттями, по-друге, відсутні чітко сформульовані критерії, на основі яких в системі будується оптимізований граф тексту, де вершини і зв'язки мають свою тимчасову оцінку.

У цьому підрозділі ми детально розглянули лише два підходи до автоматичної побудови бази знань за допомогою природомовних текстів та системи на їх основі, проте різноманітні існуючі підходи до розв'язання такої задачі не вичерпуються. Зокрема на щорічній Європейській конференції зі Штучного Інтелекту існує секція, де дослідники постійно доповідають про свої досягнення в області автоматичного навчання онтологій та баз знань. Також ряд таких підходів детально описано в роботах [42, 251], зокрема деякі цікаві методики та системи на їх основі в області автоматичного створення онтологій і баз знань.

11.5. Основні задачі, пов'язані з опрацюванням природної мови

Розуміння природної мови є однією з найважливіших та найскладніших проблем теорії та практики штучного інтелекту. Ця задача найчастіше пов'язується з проблемою

створення систем штучного інтелекту, які здатні вести осмислений діалог з людиною. Відомо, що за останнє десятиліття в організації людино-машинного інтерфейсу було досягнено величезного прогресу, але цей прогрес був, в основному, досягнений іншим, обхідним шляхом – на основі систем підказок, меню тощо. Успіхи ж у власне розумінні природної мови залишаються скромними: природна мова є надто складною і неоднозначною. І справа тут не лише в розумінні. У процесі діалогу машина повинна реагувати на фрази людини, і ця відповідь повинна бути максимально адекватною і соціально бажаною; вона повинна враховувати мету діалогу, індивідуальні особливості співрозмовника тощо. На сучасному етапі неможливо сподіватися на повне вирішення цих проблем. Водночас можна і потрібно говорити про створення обмежених природних мов, які, з одного боку, припускатимуть ефективні методи аналізу, а з іншого – вони повинні бути достатньо природними для людини, і їх вживання не має становити для неї особливих труднощів.

Але задача оброблення природної мови виникає не тільки в процесі діалогу з людиною. Можна згадати, наприклад, системи автоматизованого перекладу, які дозволяють здійснювати переклад з однієї мови на іншу (дуже відомими є, зокрема, програми Stylus, ПРОМТ та ін). Якість такого перекладу все ще дуже недосконала, і отриманий переклад вимагає серйозних подальших доопрацювань. Серед професійних перекладачів немає єдності з приводу доцільності застосування таких систем. Дехто переконаний, що вони істотно полегшують роботу перекладача; інші ж вважають, що якісний переклад простіше відразу набирати вручну.

У цьому розділі ми розглядаємо оброблення речень, отриманих у вигляді готових текстів. Якщо ж мову доводиться сприймати на слух, то, виникає ще одна задача – задача розпізнавання мови [32].

11.5.1. Типова схема опрацювання природної мови

При всьому розмаїтті задач, пов'язаних з опрацюванням природної мови, і підходів до їх розв'язування роботу систем штучного інтелекту з природною мовою можна уявляти собі у вигляді певної загальної схеми.

1. Сприйняття тексту, написаного природною мовою (на практиці – як правило, обмеженою, речення якої прості за своєю структурою і відносяться до обмеженої предметної області.
2. Співставлення тексту зі своїми знаннями і переклад його на внутрішню мову системи – саме цей етап, власне кажучи, і складає основу розуміння.
3. Планування відповіді на основі тих чи інших алгоритмів, специфічних для певної предметної області або загальноінтелектуальних метапроцедур.
4. Генерація відповіді – у вигляді зв'язного тексту або, можливо, в деякій іншій формі.

Для розв'язування деяких практичних завдань не обов'язково прагнути до глибокого розуміння. Так, в основі роботи ряду інформаційно-пошукових систем, зокрема інтелектуальних агентів (автономних мобільних програмних засобів, призначених, у першу чергу, для роботи в Інтернеті), лежать продукційні правила, які дуже нагадують ідею фатичного діалогу.

Слід розрізняти такі види аналізу речень природної мови:

- морфологічний, який виділяє окремі слова та форми слів;

- синтаксичний, метою якого є з'ясування структури речення;
- семантичний, в процесі якого з'ясовується значення фрази та співставлення його зі знаннями системи;
- прагматичний – з'ясування, з якою метою було сказане речення.

Для сучасних методик характерне значне взаємопроникнення усіх видів аналізу. Крім цього, необхідним є розуміння не тільки окремих речень, але й усього тексту.

11.5.2. Рівні розуміння

Можна вважати, що система розуміє текст, якщо вона дає правильні відповіді на будь-які запитання, що мають відношення до тексту. Інтуїтивно зрозуміло, що «розуміти» тексти можна в різній мірі та з різною глибиною. Наводиться така класифікація рівнів розуміння [71].

Нехай на вході системи маємо текст **B = <T, E, P>**, де **T** – власне текст; **E** – розширений текст, що включає умови його породження; **P** – розширений текст, який враховує інформацію про суб'єкта, який породжує текст.

Крім цього, введемо такі позначення: **TR** – формальні правила поповнення тексту, що базуються на його внутрішній структурі та на псевдофізичних логіках; **ER** – правила поповнення тексту, що базуються на знаннях системи про світ; **PR** – правила поповнення тексту, що базуються на комунікативних та психологічних знаннях, а також на знаннях про особистість того, хто говорить; **A** – відповідь, що генерується системою.

Перший рівень характеризується схемою **T → A**, відповіді генеруються лише на основі прямого та безпосереднього сприйняття інформації. Нехай, наприклад, система сприймає текст.

"О восьмій ранку Петро пішов на роботу. Він повернувся о восьмій вечора, після чого повечеряв і ліг спати".

При першому рівні розуміння система в змозі відповідати на запитання *"Коли Петро пішов на роботу?"* або *"Коли він повернувся?"*.

Другий рівень характеризується схемою **<T, TR> → A.** Система в змозі відповідати на запитання типу *"Що було раніше: повернення Петра чи його вечеря?"*

Третій рівень характеризується схемою **<T, TR, ER> → A.** Система може відповісти на запитання типу *"Де був Петро о четвертій дня?"*. Легко зрозуміти, що відповідь може істотно залежати від наявних знань.

Четвертий рівень характеризується схемою **<E, TR, ER> → A.** Крім власне тексту, беруться до уваги інші джерела інформації. Наявність таких джерел дає можливість правильно сприймати висловлювання типу *"Подивись, яка погода! Чи варто їхати на пікнік?"*.

П'ятий рівень розуміння, який характеризується схемою **<P, TR, PR> → A,** бере до уваги інформацію про конкретну особистість, що є джерелом тексту, а також загальну інформацію про психологію спілкування, про мету учасників спілкування тощо. Так, наприклад, якщо система сприймає слова Василя про те, що *"Таня обіцяла незабаром повернутися"*, слід брати до уваги:

- чи слід взагалі довіряти Василю?
- який зміст вкладає Василь у слово "незабаром?";
- що ми знаємо про Таню та про те, куди вона пішла?

Крім цього, розглядаються ***метарівні розуміння***, які визначають можливі зміни у базах знань та сприйняття і породження метафор та асоціацій.

11.5.3. Глибинні відмінки

Розглянемо один з найвідоміших і класичних підходів до розуміння речень, написаних природною мовою. Він полягає в аналізі структури речення на основі глибинних відмінків. Під глибинним відмінком слід розуміти роль, яку відіграє певне слово в реченні; ця роль визначає відношення слова до основної ситуації, яка описується в реченні.

Виділяють різні системи глибинних відмінків. Однією з найвідоміших є система відмінків Філмора. Основними відмінками Філмора є такі [186]: агент (суб'єкт, що є ініціатором події); контрагент (сила, проти якої спрямовано дію), пацієнт (суб'єкт, що відчуває на собі ефект дії), джерело (місце, з якого починається дія) тощо.

Звичайно, виділення структури глобальних відмінків має сенс лише тоді, коли вдається розробити формальні правила, за допомогою яких можна отримувати конкретні значення цих відмінків. Ці правила можуть ґрунтуватися на формальній структурі речення (наприклад, деякий відмінок може застосовуватися лише після визначеного прийменника), на специфічних знаннях про предметну область тощо.

Однією з перших програм, побудованих на цій основі, була відома програма Т. Віноградa [31].

11.5.4. Типова схема аналізу речень на основі глибинних відмінків

Уїнстон [170] описує, яким чином можна використовувати теорію глибинних відмінків при аналізі речень, написаних простою мовою, і які відносяться до обмеженої предметної області.

У більшості речень йдеться про певну дію або про перехід з одного стану в інший. Це описується дієсловом, яке в такому випадку виступає ключовим елементом речення. Отже, аналіз речення повинен починатися з виділення основного дієслова.

Далі з'ясовуються значення глибинних відмінків, які показують, як саме іменники, які входять до речення, пов'язані з ключовим дієсловом. Уїнстон [170] описує такі відмінки:

1. **Агент** – те, що викликає дію або зміну стану. Наприклад, в реченні «Дівчина нагодувала цуцика» як «агент» виступає слово «дівчина».
2. **Об'єкт** – те, на що спрямована дія. У попередньому реченні «об'єкт» – це «цуцик».
3. **Інструмент** – засіб, який використовується агентом.
4. **Співагент** – той, хто допомагає агентові у здійсненні дії.
5. **Пункт відправки** – початковий стан.
6. **Пункт призначення** – кінцевий стан. Наприклад: «Дівчина пішла зі школи додому».
7. **Засіб доставки**. Спосіб, яким здійснюється переміщення з пункту відправки до пункту призначення.
8. **Траєкторія.** Траєкторія, за якою здійснюється переміщення.
9. **Місцезнаходження**. Місце, де здійснюється дія.
10. **Споживач**. Той, для кого виконується дія.

11. **Сировина.** Відмінок «сировина» виникає, якщо деякий матеріал зникає, перетворюючись на продукт. Наприклад: «Таня зробила компот з яблук».
12. **Час.** Час виконання дії.

Отже, аналіз речення можна розглядати як заповнення деякої фреймоподібної структури. Слотами цієї структури виступає ключове дієслово та глибинні відмінки, які розкривають ролі іменників по відношенню до дієслова. Можна також вводити слоти, які описують ознаки іменників, спосіб виконання дії («сильно», «слабко» і т. ін.).

Після того, як аналіз речення здійснений, система може відповідати на запитання, пов'язані з цим реченням. Для відповіді на запитання, очевидно, достатньо звернутися до слотів, асоційованих з певним типом запитань.

Наприклад, до системи вводиться текст «Володя написав вірус», після чого її запитують «Хто написав вірус?». Якщо із запитанням «Хто?» асоційований відмінок «Агент», система звертається до відповідного слоту і відповідає «Володя».

Запитання для повторення та контролю знань

1. Які кроки здійснені для інтелектуалізації Інтернету?
2. У чому важливість онтології під час інтелектуального пошуку?
3. Що таке програмні інтелектуальні агенти?
4. Які суть, структура та властивості мультиагентної системи?
5. Як задається структура рівнозначної МАС?
6. Як задається структура ієрархічної МАС?
7. Які існують агенти в МАС інтелектуального пошуку?
8. Які існують моделі комунікації між агентами?
9. Які є два підходи до розроблення мови спілкування між агентами?
10. Які класи архітектур агентів Вам відомі?
11. Наведіть приклад реактивної архітектури.
12. Які агенти є спеціалізованими і неспеціалізованими?
13. У чому полягає успіх у побудові ефективної спеціалізованої інтелектуальної системи?
14. Який процес називається навчанням онтологій?
15. Які Ви знаєте системи навчання онтологій?
16. За якими ознаками класифікуються онтології?
17. Які принципи побудови онтологій Ви знаєте?
18. Наведіть алгоритм побудови онтологій.
19. Які Ви знаєте стандарти, що використовуються під час побудови інтелектуальних систем?
20. Які мови подання онтологій Ви знаєте?
21. мЯкі Ви знаєте засоби для створення онтології?
22. Що таке Protégé?

23. Як виглядає архітектура Protégé -2000?
24. Які характеристики редактора Protégé -OWL?
25. Які є версії мови OWL?
26. Які завдання вирішує мова OWL?
27. Охарактеризуйте термінологію мови OWL.
28. Як відбувається подання структури класів в OWL?
29. Яка методика розроблення онтології засобами Protégé?
30. Як здійснювати створення та збереження нового проекту Protégé -OWL?
31. Як створювати властивості класів?
32. Які характеристики OWL властивостей? Наведіть приклади.
33. Як здійснити введення екземплярів класів у Protégé?
34. Як здійснюється автоматичний розвиток онтології у складі інтелектуальної системи?
35. Як виглядає структурно-функціональна схема лінгвістичного процесора знань?
36. Які основні задачі, пов'язані з опрацюванням природної мови?
37. Яка типова схема опрацювання природної мови?
38. Які існують рівні розуміння природної мови?
39. Яка типова схема аналізу речень на основі глибинних відмінків?

додатки. мови програмування інтелектуальних систем

Додаток А.
Моделювання нейронних мереж за допомогою NNML

Протягом останніх 10 років штучні нейронні мережі одержали широке поширення як інструмент для розв'язування різних задач аналізу даних і розпізнавання образів. На сьогодні відома значна кількість програмних імітаторів нейромереж. Але уявимо, що нам необхідно перенести нейромережну модель, побудовану за допомогою одного нейроімітатора, у формат іншого. Задача тяжко реалізовувана в зв'язку з тим, що поки немає загальновизнаного обмінного формату для нейромережних моделей.

Необхідність у подібному обміні може виникнути з багатьох причин. Одна з таких причин – обмін *нейромережними рішеннями*. Під *нейромережним рішенням* ми будемо розуміти обчислювальну модель, побудовану відповідно до методів моделювання нейронних мереж, що видає прогнозоване значення цільового параметра задачі на основі набору вхідних даних, що описують поточну ситуацію. Нейромережне вирішення в загальному випадку складається із системи вхідних і вихідних параметрів, що визначають задачу для нейромережі, навченої нейронної мережі (або системи нейронних мереж), модуля попереднього перетворення вхідних даних і модуля інтерпретації відповідей мережі. Вичерпний опис цих компонентів дозволяє однозначно відновити нейромережне вирішення і коректно його використовувати.

Основна задача цього проекту – розроблення обмінного формату для опису нейромережних вирішень. Ми пропонуємо використовувати для цієї мети мову, засновану на нотації XML. Робоча назва мови: Мова Розмітки для Нейронних Мереж (Neural Network Markup Language, NNML). Опис нейромережного рішення за допомогою NNML являє собою структурований текстовий документ, сформований за допомогою ієрархії вкладених елементів. NNML-документ містить повний опис нейромережної моделі, включаючи словник даних, опис методів попереднього опрацювання, структури і параметрів самої нейронної мережі, методів інтерпретації відповідей моделі і т.ін. NNML може застосовуватися як для обміну нейромережними моделями, так і для їхнього документування, збереження і керування ними незалежно від конкретної системи моделювання нейронних мереж.

NNML є XML-похідною мовою. Правила для формування NNML-документів задані у вигляді DTD-визначення.

Опис NNML

Neural Network Markup Language – мову для опису нейромережних моделей за допомогою нотації XML. Вона дозволяє цілком задавати нейромережну модель, включаючи словник даних, процедури попереднього і подальшого опрацювання, структуру і параметри нейронної мережі, а також різну допоміжну інформацію. Пропонований підхід до опису функціональних елементів моделі дозволяє записувати широкий спектр ма-

тематичних перетворень, використовуваних у процедурах попереднього опрацювання даних і моделях нейронних мереж. За допомогою NNML можуть бути описані гетерогенні нейронні мережі довільної структури. Передбачено засоби для компактного опису найбільш розповсюджених типів нейромереж (таких, як багатошаровий персептрон або мережі радіальних базисних функцій). Користувач має можливість задавати власні елементи для конструювання функціональних елементів моделі.

Область нейромережних досліджень допускає різні точки зору на штучні нейронні мережі. Вони можуть розглядатися як системи розпізнавання образів, як моделі реальних нейронних мереж, як база для рівнобіжних обчислень, як моделі фізичних процесів. У цьому додатку під нейромережними моделями ми розуміємо обчислювальні моделі (які відображають вхідні змінні у вихідні), створені відповідно до методів моделювання нейронних мереж і призначені для розв'язування практичних задач класифікації і розпізнавання. При цьому нейромережна модель містить не тільки саму нейронну мережу, але і задає набір вхідних і вихідних параметрів, що описують розв'язувану задачу, методи їхнього попереднього опрацювання й інтерпретації відповідей нейронної мережі.

Цей додаток присвячений розробленню електронного обмінного формату для нейромережних моделей. Зараз найбільш поширені програмні моделі нейронних мереж. Внаслідок цього проблема обмінного формату для нейронних мереж є актуальною.

Під обмінним форматом розуміється формалізована мова, яка б дозволила записувати в явному вигляді всю інформацію про нейромережну модель, необхідну для її коректного відновлення і наступного використання. Важливими властивостями обмінного формату є її незалежність від конкретної системи моделювання і переносимість на рівні обчислювальних платформ. На сьогодні відсутній стандартний спосіб, що дозволив би нейронну мережу, створену за допомогою одного нейроімітатора, використовувати в іншому нейроімітаторі, наприклад, для тестування, донавчання або візуалізації. Більш того, через відсутність єдиної термінології для опису моделей нейронних мереж не завжди вдається відновити принципи побудови тієї або іншої нейромережі навіть за вербальним описом. Причиною цього частково є згадана розмаїтість підходів до штучних нейронних мереж. Обговорення проблем опису нейронних мереж можна знайти в [243], [273].

Розроблення обмінного формату тісно пов'язане, з одного боку, з розробленням спеціалізованих мов для опису нейронних мереж, і, з іншого боку, із введенням формалізації і стандартизації в області нейромережного моделювання. Було запропоновано кілька спеціалізованих мов, призначених для опису нейромережних моделей, наприклад Aspirin [296], PlaNet v.5 [322], AXON [257], CuPit [254], EpsiloNN [200], CONNECT [242], Nspec [241]. Подібні мови використовують нотацію, наближену до мов програмування високого рівня, подібних до C++ або Паскалю. Вони дозволяють з більшим або меншим ступенем деталізації описати методи моделювання нейронної мережі в термінах структур даних, класів, змінних і процедур. На основі такого опису генерується відповідний програмний код мовою програмування загального призначення або використовується спеціалізований імітатор.

Подібні мови призначені скоріше для опису процесу моделювання нейронної мережі, ніж для обміну вже побудованими моделями. Їхнє застосування вимагає наявності спеціальних компіляторів. І, нарешті, вони передбачають запис параметрів і динаміки тільки нейронної мережі, тоді як для коректного відновлення моделі необхідні також

зведення про структуру даних, методах опрацювання вхідних і інтерпретації вихідних сигналів мережі. Інший тип мови опису нейронних мереж поданий у [310] як складова частина PMML. PMML дозволяє записувати інформацію про поточний проект, структуру вхідних і вихідних даних, статистичні параметри навчальної вибірки. Однак він дозволяє описувати тільки мережі зворотного розповсюдження помилки з фіксованим типом нейронів і заздалегідь визначеним набором з декількох найпростіших правил попереднього опрацювання і інтерпретації.

Формалізації і стандартизації нейронних мереж присвячені роботи [119], [199], [231], [280]. Fiesleretaly [231] пропонує ієрархічну специфікацію компонентів нейронної мережі і вводить математичну нотацію для їхнього опису. Smith [273], Atencia et al [280] використовують більш загальні математичні конструкції для опису нейромережних структур і їхньої динаміки. У [199] дається розширений аналіз нейромережних структур, що захоплює моделі біологічних нейронних мереж. Найбільш розгорнутий опис усіх компонентів системи нейромережного моделювання наведений в [119]. Зазначені роботи внесли значний вклад у систематизацію області моделювання нейронних мереж, однак наведені в них результати також не можуть бути безпосередньо використані для обміну конкретними нейромережними моделями.

У цьому додатку пропонується напрям для розроблення обмінного формату для нейроімітаторів, що використовують штучні нейронні мережі для розв'язання практичних задач з прогнозування або класифікації. При цьому об'єктом для опису є цілком навчені нейронні мережі. Нейронна мережа після завершення процесу навчання розглядається як статичний об'єкт із фіксованою структурою і внутрішніми параметрами. Відповідно, її опис не містить явних зведень про метод навчання мережі. Іншими словами, передається готова обчислювальна схема одержання значень цільових параметрів на основі відомих. Основне призначення пропонованого формату – запис всієї інформації, необхідної для однозначного і точного відтворення цієї нейромережної обчислювальної моделі нейроімітатором. Сказане не виключає можливості подальшого навчання цієї мережі нейроімітатором. Після відновлення з опису з нейромережною моделлю можуть бути зроблені ті ж дії, що і з нейронної мережею, згенерованою самим нейроімітатором.

Під час розроблення цього формату прагнули зробити його відкритим для розширень, легко інтерпретовуваним і незалежним від конкретної системи або мови програмування. У результаті була розроблена мова, заснована на нотації XML [234]. За аналогією з PMML її можна назвати Neural Network Markup Language (NNML). Надалі виклад терміну "формат" і "мова" використовуються, як синоніми. Також використовується наступна угода: для імен тегів NNML використовується шрифт `courier`.

Короткий огляд NNML

Основою для розроблення опису нейромережної моделі послужив аналіз складу інформації, необхідної для коректного відновлення обчислювальної моделі нейронної мережі. Нейронну мережу не можна цілком описати у відриві від оточення, у якому вона була створена. Іншими словами, опис нейромережного методу розв'язання тієї або іншої задачі повинен містити в собі не тільки інформацію про структуру і параметри нейронної мережі, але також інформацію про структуру словника даних, про методи попереднього опрацювання вхідних змінних перед подачею їхньої мережі, про методи одержання змістовної відповіді на основі вихідних сигналів мережі (методи подаль-

шого опрацювання). NNML-документ являє собою текстовий файл, що містить ієрархію вкладених тегів, відповідно до принципів побудови XML-документів. Кожен елемент верхнього рівня описує один базовий аспект нейромережної моделі.

У термінах XML секція або елемент є базовими одиницями опису. Елемент задається парою іменованих тегів і може містити деякі дані або інші елементи. Опис елемента може бути розширено за допомогою набору атрибутів.

Структура і зміст кожного розділу опису розроблялися на основі декомпозиції нейромережної моделі з врахуванням робіт, що торкаються цього питання, зокрема [119], [175], [302].

Для опису структури нейромережної моделі обраний наступний підхід: модель розглядається як сукупність пов'язаних між собою функціональних елементів, що передають один одному (можливо, і самим собі) вихідні сигнали, отримані на основі вхідних сигналів і внутрішнього функціонування. Отже, блок попереднього опрацювання, нейронна мережа і блок подальшого опрацювання подається як набори подібних елементів. Під функціональним елементом розуміється функція, що перетворить вектор дійсних вхідних значень у скалярне вихідне значення.

Завжди мова призначалася для опису нейронних мереж, навчених із учителем. У наш час вивчається питання про її застосовність для опису мереж, тих, яких навчають без учителя, наприклад SOM, LVQ і т.ін.

Структура NNML-документа

Кожен NNML-документ містить один кореневий елемент `neuralmodel`, що визначає границі NNML-опису. `neuralmodel` може містити базові елементи або розділи опису, такі як `header`, `declarations`, `data` або `neural_network`.

Заголовок містить інформацію, призначену, в основному, для читання людиною. Він включає три підрозділи `problem`, `creator` і `project`. Підрозділ `problem` описує задачу, для розв'язування якої створювалася нейромережна модель. Поля опису включають: найменування предметної області, наприклад "пророкування фінансових тимчасових рядів", найменування конкретної задачі і вільний опис задачі з поясненням необхідних деталей. Підрозділ `creator` містить зведення про нейроімітатор, за допомогою якого ця модель була створена (найменування і версія), а також ім'я автора, контактні дані, копірайт. Підрозділ `project` містить найменування поточної моделі, поле ідентифікатора для заповнення користувачем моделі, дату і час створення, коментарі творця.

Приклад:

```
<header>
<problem>
<task_name>XOR problem</task_name>
<domain>Logic computation</domain>
<task_description>Simple illustration of nonlinear neural net
mapping</task_description>
</problem>
<creator>
<author>
<name>D.Rubtsov</name>
```



```
<contact>E-mail: rubtsov@rbcmail.ru</contact>
</author>
<simulator name="NNET" version="0.2"/>
</creator>
<project>
<datestamp>25.04.2001</datestamp>
<model_id>1</model_id>
<model_name>xor2_01</model_name>
<model_comment>multilayer perceptron for separation two
classes based on xor(x1,x2) function</model_comment>
</project>
</header>
```

Розділ `data` містить опис використовуваного словника даних. Він розділений на описи полів даних `data_field`. Кожне поле даних може бути описане за допомогою наступних атрибутів:

`id` – унікальний ідентифікатор кожного поля чи даних, · `name` – найменування поля чи даних, що використовується в діалозі з користувачем,

`type` – тип змінної, котрій відповідає поле даних: неперервна (continuous), номінальна (nominal), впорядкована (ordered), спосіб використання змінної в моделі: як вхідна, вихідна або допоміжна.

Кожне поле має внутрішній тег `data_field_properties`, що містить опис властивостей змінної, наприклад інтервал припустимих значень. Для дискретних змінних (номінальних і упорядкованих) подається опис їх дискретних станів: ідентифікатор, найменування, інтервал чисельних значень для кожного можливого дискретного стану.

Приклад:

```
<data>
<data_field id="1" name="target class" type="nominal"
usage="output">
<comments>target class: 1 or 2</comments>
<datafield_properties>
<discrete_states>
<discrete_state id="1">
<ds_name>first class</ds_name>
<ds_interval min="1" max="1,5" brackets="10"/>
</discrete_state>
<discrete_state id="2">
<ds_name>second class</ds_name>
<ds_interval min="1,5" max="2" brackets="01"/>
</discrete_state>
</discrete_states>
</datafield_properties>
</data_field>
<data_field id="3" name="x2" type="nominal" usage="input">
<comments>nominal input variable with two discrete states: 0
and 1</comments>
```

```
<datafield_properties>
<discrete_states>
<discrete_state id="1">
<ds_name>0</ds_name>
<ds_interval min="0" max="0,5" brackets="10"/>
</discrete_state>
<discrete_state id="2">
<ds_name>1</ds_name>
<ds_interval min="0,5" max="1" brackets="01"/>
</discrete_state>
</discrete_states>
</datafield_properties>
</data_field>
<data_field id="2" name="x1" type="real" usage="input">
<comments>continuous input variable</comments>
<datafield_properties>
<value_range min="0" max="1" brackets="11"/>
<insertion type="parameter" name="mean">0,5</insertion>
</datafield_properties>
</data_field>
</data>
```

Розділ `preprocessing` призначений для опису методів попереднього опрацювання вхідних даних перед подаванням їх до нейронної мережі. Для попереднього опрацювання в практиці нейромоделювання найчастіше використовуються прості математичні перетворення типу скаляр – скаляр, такі як нормалізація (для неперервних змінних), або скаляр – вектор: кодування вектором бінарних ознак (для дискретних змінних). В останньому випадку одному полю вхідних даних буде відповідати певна кількість (за кількістю дискретних станів змінної) вихідних сигналів попереднього опрацювання. Однак можуть бути корисні і більш складні методи наведені в [119].

Для забезпечення гнучкості обраний наступний підхід: для кожного сигналу, що надходить від блоку попереднього опрацювання до нейронної мережі задається окремий функціональний елемент (тег `preprocessing_field`). Він описується відповідно до єдиного для усіх функціональних елементів нейромережної моделі правилами опису функціональних перетворювачів, що відносяться до нейронів, елементів попереднього або подальшого опрацювання. Кожен елемент попереднього опрацювання одержує сигнал з одного або декількох полів даних, і на підставі заданого правила генерує вихідний сигнал, що подається на вхід нейронної мережі. Для нормалізації таке правило буде полягати у відомих формулах, для кодування – у кусочній функції, результат якої дорівнює, наприклад, 1 при визначеному значенні вхідної змінної, і-1 в інших випадках.

Приклад:

```
<preprocessing>
<preproc_field id="1">
<scaling>
<source_min>
<number>0</number>
```



```
</source_min>
<source_max>
<number>1</number>
</source_max>
<source_current>
<connect object_type="data_field" object_id="2"/>
</source_current>
<target_min>
<number>-1</number>
</target_min>
<target_max>
<number>1</number>
</target_max>
</scaling>
</preproc_field>
<preproc_field id="2">
<encoding>
<connect object_type="data_field" object_id="3"/>
<encoding_present_value>1</encoding_present_value>
<encoding_absent_value>-1</encoding_absent_value>
<encoding_present_interval min="0" max="0,5" brackets="10"/>
</encoding>
</preproc_field>
<preproc_field id="3">
<encoding>
<connect object_type="data_field" object_id="3"/>
<encoding_present_value>1</encoding_present_value>
<encoding_absent_value>-1</encoding_absent_value>
<encoding_present_interval min="0,5" max="1" brackets="01"/>
</encoding>
</preproc_field>
</preprocessing>
```

Нейронна мережа

Розділ `neural_network` є основним і містить опис безпосередньої нейронної мережі і її параметрів. Нейронна мережа розглядається як набір обчислювальних модулів, що посилають один одному сигнали. У зв'язку з тим, що мета опису нейронної мережі – по можливості спростити її інтерпретацію і виділення математичних процедур, що лежать в основі моделі а також у відсутності сталої термінології (і способів визначення) для елементів мережі, було вирішено не використовувати поняття типу "синапс", "аксон" і т.ін. Структура нейромережі записується за допомогою загальних математичних операцій.

Нейронна мережа описується як математичний об'єкт, що складається зі зв'язаних процесорних елементів. Для наочності введений тег `layer`, що групує нейрони, які функціонують паралельно. Для повнозв'язаної мережі шар буде один. Кожен нейрон описується як пара функцій: комбінаційної і функції активації. Їхній зміст зберігається в тегах `combination_function` і, відповідно, `activation_function`. Ніяких

спеціальних обмежень на структуру цих функцій не накладається, тобто нейрон може мати будь-яке функціонування, що може бути описане в рамках підходу. Вихідне значення комбінаційної функції автоматично є вхідним для функції активації. В описі нейрона ваги і зв'язки не вказуються в явному вигляді. Вони складають аргументи функцій нейрона. Кожен нейрон може одержувати вхідні сигнали від полів даних, полів попереднього опрацювання або інших нейронів, включаючи самого себе. Отже, структура нейронної мережі задається неявно, через посилання до відповідних функцій нейрона. Такий підхід дозволяє задавати мережі практично довільної топології, у тому числі повнозв'язані або проріджені. Як функціональні складові описи нейрона може використовуватися також одна з визначених функцій або функція, сконструйована користувачем, що дає можливість описувати гетерогенні мережі.

Для скорочення опису використовується наступний підхід. Якщо, наприклад, усі нейрони певного шару мають ту саму функцію активації, допускається її вказати як перший тег усередині тега `layer`. Аналогічний підхід справедливий і для комбінаційної функції.

Поряд зі структурою мережі, у тегу `neural_network` наводиться інформація про правила її функціонування. Для мереж зі зворотними зв'язками передбачені наступні умови зупинки: під час досягнення рівноважного стану і після закінчення заданого числа тактів обміну сигналами. Указуються використовувані при навчанні обмеження на значення ваг зв'язків, параметрів функцій активації.

У підрозділі `neural_network_properties` дається загальна інформація про нейромережну модель: її тип (зі зворотними зв'язками, без зворотних зв'язків), парадигма (багатошаровий персептрон, мережі РБФ і т.ін.), коментарі розроблювача.

Приклад:

```
<neural_network>
<nn_structure>
<layer id="1">
<neuron id="1">
<combination_function>
<explicit_record>
<sum>
...
</sum>
</explicit_record>
</combination_function>
<activation_function>
<rational_sigmoid>
<connect object_type="combination_function"/>
<sigmoid_slope>
<number>0,1</number>
</sigmoid_slope>
</rational_sigmoid>
</activation_function>
</neuron>
...
<neuron id="2">
```



```
</neuron>
</layer>
<layer id="2">
<neuron id="1">
...
</neuron>
<neuron id="2">
...
</neuron>
</layer>
</nn_structure>
<nn_activity>
<nn_constraints>
<nn_constraint>
<nn_parameter>weight</nn_parameter>
<nn_parameter_range min="-1" max="1" brackets="11"/>
</nn_constraint>
<nn_constraint>
<nn_parameter>sigmoid slope</nn_parameter>
<nn_parameter_range min="0,01" max="2" brackets="11"/>
</nn_constraint>
</nn_constraints>
</nn_activity>
<nn_decription>
<nn_type>multilayer perceptron with fully interconnected
layers</nn_type>
<nn_class>feedforward supervised</nn_class>
</nn_decription>
</neural_network>
```

Розділ `postprocessing` призначений для запису інформації про методи одержання змістовної відповіді на основі вихідних сигналів мережі. Нейронна мережа може бути сконструйована для отримання декількох відповідей. Тому методи попереднього опрацювання згруповані по тегах `output_field`, кожний з яких відповідає одному вихідному параметру моделі. Тег `output_field` містить посилання на номер відповідного вихідного параметра, описаного в розділі Дані. Метод перетворення вихідних сигналів мережі задається виходячи з того ж підходу, що і методи попереднього опрацювання. Як подальше опрацювання найчастіше використовується денормалізація (для неперервних вихідних параметрів), або правило "переможець забирає все" (для дискретних вхідних параметрів), однак припустимі і більш складні методи [119]. Як вхідні сигнали для блоків подальшого опрацювання зазвичай використовуються сигнали вихідних нейронів мережі.

Для кожного вихідного поля введено тег Вербалізація. Він призначений для зіставлення визначених значень або груп значень вихідного параметра з деякою текстовою інформацією, що може допомогти інтерпретувати відповідь моделі, наприклад: якщо значення такого вихідного параметра лежить в інтервалі [2.5, 3.5] то відповідь мережі – "задовільно".

Приклад:

```
<postprocessing>
<postproc_field id="1" datafield_id="1">
<postproc_function>
<winner_takes_all>
<target_class discrete_state_id="1">
<connect object_type="neuron" object_id="1" nn_layer="2"/>
</target_class>
<target_class discrete_state_id="2">
<connect object_type="neuron" object_id="2" nn_layer="2"/>
</target_class>
</winner_takes_all>
</postproc_function>
<postproc_verbalization>
<postproc_case>
<postproc_interval min="1" max="1" brackets="11"/>
<interpretation>This is first class with centers in (0,0) and
(1,1)</interpretation>
</postproc_case>
<postproc_case>
<postproc_interval min="2" max="2" brackets="11"/>
<interpretation>This is second class with centers in (1,0)
and (0,1)</interpretation>
</postproc_case>
</postproc_verbalization>
</postproc_field>
</postprocessing>
```

Опис процесу налагодження нейронної мережі

Розділ `training_process` призначений для збереження різноманітної інформації, що характеризує процедуру навчання певного екземпляра нейронної мережі. Основні пункти опису: Метод Навчання (`training_method`), Функція Помилки (`error_function`), Вибірка (`sample`).

Тег `training_method` містить найменування методу, наприклад "сполучені градієнти з оптимізацією величини кроку", опис методу в довільній формі, посилання на першоджерела щодо цього методу, стратегію навчання (на прикладах, пакетно, по сторінках і т.ін.). Окремо вказується кількість епох (ітерацій). Функція помилки описується найменуванням і набором параметрів. Тег `sample` містить інформацію про набір даних, на якому навчалася і тестувалася нейронна мережа. Вибірка може містити найменування, вказання конкретних файлів, кількість прикладів у цілому. Також вона містить тегів `training_set`, `validation_set`, `testing_set`. Кожний з перерахованих тегів відповідно описує навчальний, валідаційний і тестовий набори даних. В описі кожного набору даних вказується кількість прикладів, можливо, номери цих прикладів у загальній вибірці, ім'я відповідного файла, а також значення функції помилки для набору на момент закінчення навчання, кількість правильно (із заданою точністю) розв'язаних прикладів.

Приклад:

```
<training_process>
<training_method>
<training_method_name>BFGS</training_method_name>
<training_method_description>Quasi-Newton method for nonlinear
optimization
</training_method_description>
<training_method_references>Gill, P., W. Murray and M.H.Wright,
Practical optimization, Academic Press, 1981, California, USA
</training_method_references>
</training_method>
<error_function>
<error_function_name>Mean squared error</error_function_
name>
</error_function>
<sample>
<sample_name>xor2</sample_name>
<sample_description>sample contains four cases with two
input and one output variables and defines xor function</sample_
description>
<volume>4</volume>
<training_set>
<volume>4</volume>
<file format="ascii text file">xor2.txt</file>
<insertion type="parameter" name="number of correct net
answers">4</insertion>
</training_set>
</sample>
<training_epochs>22</training_epochs>
<training_mode>batch training</training_mode>
</training_process>
```

Опис функціональних елементів у NNML

Усі базові компоненти моделі: попереднє опрацювання, нейронна мережа, інтерпретатор – описуються за допомогою функціональних елементів, і відображення математичних перетворень, що лежать в їх основі, є ключовим питанням для формату NNML. зазначимо, що розроблення раціонального підходу до цього питання не є тривіальною задачею. З одного боку, область нейромережного моделювання стрімко розвивається. Кількість моделей нейронів росте безупинно. Тому ми не можемо обмежитися стандартним набором математичних функцій, використовуваних у найпопулярніших мережах, таких як багатошаровий персептрон. Необхідно надати розроблювачам можливість подати широкий набір функцій, що може бути реалізований за допомогою стандартних засобів мов програмування. З іншого боку, найчастіше використовується обмежений набір нейромережних парадигм, тому було б розумно передбачити компактний запис для використовуваних у них елементів.

Підхід, прийнятий у НННМЛ для подання функціональних елементів полягає в наступному. Визначено базові операнди (аргументи), над якими задаються всі перетворення. Базових аргументів може бути два типи: число або посилання на один з об'єктів моделі.

1. Число задається тегом `number`, що має необов'язковий атрибут `name` – для відображення семантики аргумента, наприклад "weight" або "bias". Тег містить у загальному випадку дійсне число. Усі числові параметри нейромережної моделі – ваги зв'язків нейронів, параметри функцій активації, параметри елементів попереднього і подальшого опрацювання – записуються за допомогою цього тега.

   Приклади:

   ```
   <number>-1</number>
    <number name="weight">0,582004</number>
   ```

2. Посилання задається порожнім тегом `connect`. Має атрибути: `object` – містить назву типу об'єкта – джерела сигналу (назва одного з тегів: `preprocessing_field`, `neuron`, `data_field`), `id` – ідентифікаційний номер конкретного об'єкту зазначеного типу, `layer` – номер шару для об'єкту типу нейрон. Посилання служить для опису зв'язків між функціональними елементами моделі, з яких складаються блоки попереднього, подальшого опрацювання і нейронна мережа.

   Приклад:

   ```
   <connect object_type="data_field" object_id="3"/>
   <connect object_type="preproc_field" object_id="1/>
   <connect object_type="neuron" object_id="2" nn_layer="1"/>
   ```

   Базові елементи можуть бути згруповані в структури, наприклад, за допомогою тега `vector`.

Передбачено три рівні опису змісту математичних перетворень, що розглядаються далі.

1. **З використанням базових функцій.** Для опису того або іншого математичного перетворення введений набір елементарних базових функцій. Він містить у собі арифметичні й алгебраїчні операції: додавання, множення, піднесення до степеня, розподіл, взяття модуля, логарифм, експонента і т.ін., тригонометричні операції, відношення: більше, менше, дорівнює, умова Якщо-То, матричні операції – додавання, скалярний добуток і т.ін., а також кускову функцію для задання вибору значення результату за умовою. Вираз, записаний за допомогою базових функцій, утримується в тегу `explicit_record`. Теги базових функцій можуть містити інші базові функції або базові аргументи `number` або `connect`. Розбір таких конструкцій здійснюється шляхом побудови дерева обчислення. Листові вузли дерева завжди повинні містити один з елементарних аргументів. Пріоритет операцій визначається за вкладеністю – у першу чергу обчислюються всі "внутрішні" функції, незалежно від їхнього типу.

   Наприклад, стандартна комбінаційна функція для багатошарового персептрона – скалярний добуток векторів вхідних сигналів і ваг зв'язків – може бути записане в такий спосіб:

   ```
   <explicit_record>
    <sum>
   ```



```
<mult>
<connect object_type="preproc_field" object_id="1"/>
<number>1</number>
</mult>
<mult>
<connect object_type="preproc_field" object_id="2"/>
<number>0,5833240</number>
</mult>
<mult>
<connect object_type="preproc_field" object_id="3"/>
<number>-0,5844554</number>
</mult>
<mult>
<number name="bias">-1</number>
<number>-1</number>
</mult>
</sum>
</explicit_record>
```

2. **З використанням визначеної функції.** Для компактного опису найчастіше використовуваних функцій заданий набір так званих попередньо визначених функцій. Він включає функції попереднього опрацювання (нормалізація, кодування), комбінаційні функції нейронів (адаптивний суматор, квадратичний суматор, евклідова відстань і ін.), функції активації (різні вигляди сигмоїд, гранична функція, лінійна і кусково-лінійна функція, експонента і т.ін.), методи подальшого опрацювання (денормалізація, переможець забирає все).

   Приклад.

   Для опису лінійного масштабування може бути використана наступна визначена функція:

```
<scaling>
 <source_min>
 <number>0</number>
 </source_min>
 <source_max>
 <number>1</number>
 </source_max>
 <source_current>
 <connect object_type="data_field" object_id="2"/>
 </source_current>
 <target_min>
 <number>-1</number>
 </target_min>
 <target_max>
 <number>1</number>
 </target_max>
 </scaling>
```

Кодування номінальних змінних вектором числових значень може бути задане так:

```
<encoding>
 <connect object_type="data_field" object_id="3"/>
 <encoding_present_value>1</encoding_present_value>
 <encoding_absent_value>-1</encoding_absent_value>
      <encoding_present_interval      min="0"      max="0,5"
brackets="10"/>
 </encoding>
```

Стандартна комбінаційна функція багатошарового персептрона:

```
<adaptive_summator>
 <vector>
 <number name="weight">1</number>
 <number name="weight">0,582004</number>
 <number name="weight">-0,585775</number>
 <number name="weight">1</number>
 </vector>
 <vector>
 <connect object_type="preproc_field" object_id="1"/>
 <connect object_type="preproc_field" object_id="2"/>
 <connect object_type="preproc_field" object_id="3"/>
 <number name="bias">-1</number>
 </vector>
 </adaptive_summator>
```

3. **За допомогою функцій, заданих користувачем.** Користувач має можливість задавати довільні функції і використовувати їх в описі. У тексті опису моделі використовується тег `user_function` з атрибутами `name` – найменування функції, `source` – адреса в Інтернеті, що містить інформацію про зміст перетворення. Усередині цього тега припустимим є тільки використання спеціальних тегів – контейнерів `arg`, що можуть містити базові аргументи number і connect. Кожен тег `arg` має атрибут `id` і служить для установлення відповідності між формальними параметрами функції користувача і фактичними, котрі будуть використовуватися для обчислень. Зміст перетворення, заданого користувачем, на етапі аналізу NNML-документа не інтерпретується. Інша можливість описати нове перетворення: задати в спеціальному тегу `declaration` опис власної функції. Для цього необхідно вказувати її ім'я і за допомогою базових функцій задати операції, що будуть вчинені над параметрами.

Приклад.

У цьому прикладі `my_sigmoid` задає сигмоїдну функцію, що використовується в нейронній мережі.

Задання нової функції:

```
<declarations>
 <declare_function name="my_sigmoid">
 <relation>
 <farg id="1"/>
 <sum>
 <farg id="2"/>
```



```
 <abs>
 <farg id="1"/>
 </abs>
 </sum>
 </relation>
 </declare_function>
 </declarations>
```

...

....і використання її в описі нейронної мережі:

```
<activation_function>
<user_function name="my_sigmoid">
<arg id="1">
<connect object_type="combination_function"/>
</arg>
<arg id="2">
<number name="sigmoid slope">0,1</number>
</arg>
</user_function>
</activation_function>
```

Розширення можливостей NNML

Забезпечення відкритості і розширюваності опису нейромережної моделі також є істотним питанням. На сучасному етапі розвитку NNML підтримуються наступні можливості розширення NNML:

Використання визначених користувачем функцій (подано вище).

Введення нових тегів опису. Для використання конструкцій, не передбачених базовим набором тегів, введений тег `insertion`. Він має необов'язкові атрибути: тип, найменування, автор, джерело (`type`, `name`, `author`, і `source`). Тип може дорівнювати або "`parameter`", або "`construct`" (за замовчуванням). Значення "`parameter`" вказує на те, що за допомогою такого включення користувач задав додатковий елемент опису типу "ім'я параметра – значення параметра". Тоді атрибут `name` інтерпретується як найменування параметра, а зміст тега – як значення. Якщо тип зазначений як "`construct`", значить тег може мати довільний зміст, наприклад, структуру вкладених тегів.

Приклад.

Допустимо, користувачеві треба було розширити опис поля даних і додати новий параметр – математичне очікування значень змінної у вибірці. Коректний спосіб введення нового параметра – використання тега `insertion`:

```
<datafield_properties>
...
<insertion type="parameter" name="mean">0,5</insertion>
...
</datafield_properties>
```

Опрацювання NNML-документів

Оскільки NNML є мовою, заснованою на XML, правила формування файлів (документів) на NNML задаються у вигляді DTD визначень. DTD використовується також для

перевірки правильності NNML документа. Треба сказати, що DTD не може повністю контролювати всі параметри опису нейронної мережі, тобто формально правильний опис може бути проте не коректним з погляду нейромережної моделі, тому програма – аналізатор NNML має забезпечувати додатковий контроль за структурою і змістом тегів. Робота з документами NNML здійснюється за допомогою стандартних модулів маніпулювання з XML-документами (XML-парсерами).

Створення документа NNML містить наступні кроки:

- засобами відповідного нейроімітатора ґенерується і навчається нейронна мережа;
- за допомогою інтерфейсного модуля на основі внутрішнього подання створюється структура опису моделі, заповнюються її поля;
- викликаються методи універсального XML-парсера, що ґенерує файл NNML.

Відповідно, для використання готового файла NNML дії відбуваються в зворотному порядку:

- за допомогою методів XML-парсера на основі файла NNML будується дерево об'єктів;
- інтерфейсний модуль на основі цього дерева ініціалізує нейромережну модель у внутрішньому поданні імітатора;
- нейроімітатор працює з отриманою моделлю, використовуючи свої внутрішні можливості.

Отже, основна задача для розроблювачів програмного забезпечення – створення відповідних інтерфейсних модулів між внутрішнім поданням нейронної мережі і деревом об'єктів для ґенерації NNML-файла.

Застосування NNML

На сьогодні нам доступні наступні програмні засоби NNML:

Використання як обмінний формат у великих інформаційних системах. Штучні нейронні мережі широко застосовуються в практиці розв'язування задач аналізу даних. Тому необхідний технологічний підхід для ефективного використання нейронних мереж у прикладних інформаційних системах [0]. NNML дозволяє реалізувати такі технологічні рішення як спеціалізація програм моделювання нейронних мереж: виділення ґенераторів нейронних мереж, інтерпретаторів, засобів візуалізації; можливість використання різноманітних програмних засобів у єдиному комплексі; ефективне документування і ведення сховищ моделей; впровадження нейромережних засобів в наявні програмні системи.

Обмін моделями через мережу Інтернет. NNML може застосуються для поширення нейромережних моделей через глобальні комп'ютерні мережі. У перспективі можлива реалізація таких рішень, як використання вилучених машин для налагодження нейромережних моделей "за замовленням", ведення глобальних архівів нейромережних розв'язків тих або інших задач.

Додаток Б.
Мова програмування Clips.

CLIPS (C Language Integrated Production System) почала розроблятися в космічному центрі Джонсона NASA в 1984 році. Зараз CLIPS і документація на цей програмний засіб вільно розповсюджується через інтернет (http://www.ghg.net/clips/clips.html). Мова *CLIPS* не має недоліків попередніх інструментальних засобів для створення інтелектуальних систем, заснованих на мові LISP. Мова CLIPS набула великого поширення в державних організаціях і навчальних закладах завдяки низькій вартості, потужності, ефективності і переносимості з платформи на платформу. Наприклад, навіть Web-орієнтований інструментарій JESS (Java Expert System Shell), що використовує мову подання знань CLIPS, набув зараз широкої популярності.

Зазначимо, що не зважаючи на численні переваги функціонального програмування, деякі задачі краще розв'язувати в термінах об'єктно-орієнтованого програмування (ООП), для якого характерні три основні можливості: ІНКАПСУЛЯЦІЯ (робота з класами), ПОЛІМОРФІЗМ (робота з родовими функціями, що підтримують різну поведінку функції залежно від типу аргументів), УСПАДКУВАННЯ (підтримка абстрактних класів). ООП підтримує багато мов, зокрема Smalltalk, C++, Java, Common LISP Object System (CLOS). Мова *CLIPS*, у свою чергу, увібрала в себе основні переваги C++ і CLOS.

Читач може познайомитися з мовою *CLIPS*, отримавши через інтернет повний комплект документації англійською мовою або прочитавши книгу [180]. У запропонованому додатку дається коротке неформальне введення в CLIPS, необхідне для програмування простих завдань.

Відмінною особливістю *CLIPS* є конструктори для створення баз знань (БЗ):

`defrule`	визначення правил
`deffacts`	визначення фактів
`deftemplate`	визначення шаблону факту
`defglobal`	визначення глобальних змінних
`deffunction`	визначення функцій
`defmodule`	визначення модулів (сукупності правил)
`defclass`	визначення класів
`defintances`	визначення об'єктів за шаблоном, заданим `defclass`
`defmessagehandler`	визначення повідомлень для об'єктів
`defgeneric`	створення заголовка родової функції
`defmethod`	визначення методу родової функції

Конструктори не повертають ніяких значень, у відмінності від функцій, наприклад:

```
(deftemplate person
(slot name)
(slot age)
(multislot friends))
(deffacts people
(person (name Joe) (age 20))
```

```
  (person (name Bob) (age 20))
  (person (name Joe) (age 34))
  (person (name Sue) (age 34))
  (person (name Sue) (age 20)))
Приклад функції:
(deffunction factorial (?a)
(if (or (not (integerp ? a)) (< ? a0)) then
(printout t "Factorial Error!" crlf)
else
(if (= ? a0) then 1
else
(*? a (factorial ($-$ ? a1))))))
```

Правила в *CLIPS* складаються з передумов і висновку. Передумови також називають ЯКЩО-ЧАСТИНОЮ правила, лівою частиною правила або LHS правила (left-hand side of rule). Висновок називають ТО-ЧАСТИНОЮ правила, правою частиною правила або RHS правила (right-hand side of rule).

Приклад правила подано нижче:

```
(deftemplate data (slot x) (slot y))
(defrule twice
(data (x ? x) (y =(*2 ? x)))
⇒)
(assert (data (x2) (y4)); f-0
   (data (x3) (y9))); f-1
```

Тут найпоширеніша в *CLIPS* функція `assert` додає нові факти в список правил. Протилежною до assert є функція retract, яка видаляє факти із списку фактів, наприклад:

```
(defrule vis11
   ?doors < - (fit ? wdfit)
   (test (eq ? wdfit no))
 ⇒
   (assert (EVIDENCE OF MAJOR ACCIDENT))
   (retract ? doors))
```

У цьому правилі перевіряється наявність факту `doors`, і, у разі його відсутності, факт doors видаляється зі списку фактів завдання.

Також поширеною є функція `modify`. Вона дозволяє в певному факті поміняти значення слоту, наприклад

```
(deftemplate age (slot value))
(assert (age (value young)))
(modify 0 (value old))
```

Наступний приклад описує подання даних у вигляді фактів, об'єктів і глобальних змінних. Приклади фактів:

```
(voltage is 220 volt)
(meeting (subject "AI") (chief "Lytvyn") (Room "3240"))
```

У першому рядку наведений впорядкований факт, в другій – неврегульований, в якому порядок слотів не важливий.

CLIPS підтримує наступні типи даних: `integer`, `float`, `string`, `symbol`, `external-address`, `fact-address`, `instance-name`, `instance-address`.

Приклад `integer`: 594, 23, +51, S17.

Приклад `float`: 594e2, 23.45, +51.0, a17.5eb5.

`String` – це рядок символів, взятих у подвійні лапки.

Приклад `string`: "expert", "Phil Blake", "стан -0", "quote=\".

CLIPS підтримує наступні процедурні функції, що реалізовують можливості розгалуження, організації циклів в програмах і т.ін.:

`If`	оператор розгалуження
`While`	цикл з передумовою
`loop-for-count`	ітеративний цикл
`prong`	об'єднання дій в одній логічній команді
`prong$`	виконання множини дій над кожним елементом поля
`return`	переривання функції, циклу, правила і т.ін.
`break`	те ж, що і `return`, але без повернення параметрів
`switch`	оператор множинного розгалуження
`bind`	створення і об'єднання змінних

Функції *CLIPS* описується в [180]. Серед **логічних функцій** (що повертають значення `true` або `false`) виділимо наступні групи:

- функції булевої логіки: `and`, `or`, `not`;
- функції порівняння чисел: $=, \neq, >, \geq, <, \leq$;
- предикативні функції для перевірки належності типу, що перевіряється: `integerp`, `floatp`, `stringp`, `symbolp`, `pointerp` (чи відноситься аргумент до `xternal-address`), `numberp` (чи відноситься аргумент до `integer` або `float`), `lexemerp` (чи відноситься аргумент до `string` або `symbol`), `evenp` (перевірка цілого числа на парність), `oddp` (перевірка цілого числа на непарність), `multifildp` (чи є аргумент складеним полем);
- функції порівняння за типом і за значенням: `eq`, `neq`.

Серед математичних функцій виділимо наступні групи:

- стандартні: `+`, `-` `*`, `/`, `max`, `min`, `div` (цілочисельне ділення), `abs` (абсолютне значення), `float` (перетворення в тип `float`), `integer` (перетворення в тип `integer`);
- розширені: `sqrt` (добування кореня), `round` (округлення числа), `mod` (обчислення залишку від ділення);
- тригонометричні: `sin`, `sinh`, `cos`, `cosh`, `tan`, `tanh`, `acos`, `acosh`, `acot`, `acoth`, `acsc`, `acsch`, `asec`, `asech`, `asin`, `asinh`, `atan`, `atanh`, `cot`, `coth`, `csc`, `csch`, `sec`, `sech`, `deg-grad` (перетворення з градусів в сектори), `deg-rad` (перетворення з градусів в радіани), `grad-deg` (перетворення з секторів у градуси), `rad-deg` (перетворення з радіан у градуси);
- логарифмічні: `log`, `log10`, `exp`, `pi`.

Серед функцій роботи з рядками виділимо такі функції:

str-cat	об'єднання рядків
sym-cat	об'єднання рядків в значення типу symbol
sub-string	виділення підрядка
str-index	пошук підрядка
eval	виконання рядка як команди *CLIPS*
build	виконання рядка як конструктора *CLIPS*
upcase	перетворення символів у символи верхнього регістра
lowcase	перетворення символів у символи нижнього регістра
str-compare	порівняння рядків
str-length	визначення довжини рядка
check-syntax	перевірка синтаксису рядка
string-to-field	повернення першого поля рядка

Функції роботи зі складеними величинами є однією з відмінних особливостей мови *CLIPS*. До їх числа входять:

create$	створення складеної величини
nth$	отримання елемента складеної величини
members	пошук елемента складеної величини
subset$	перевірка однієї величини на підмножину інших
delete$	видалення елемента складеної величини
explode$	створення складеної величини з рядка
implode$	створення рядка зі складеної величини
subseq$	утворення підпослідовності зі складеної величини
replace$	заміна елемента складеної величини
insert$	додавання нових елементів у складену величину
first$	отримання першого елемента складеної величини
rest$	отримання залишку складеної величини
length$	визначення числа елементів складеної величини
delete-member$	видалення елементів складеної величини
replace-member$	заміна елементів складеної величини.

Функції введення-висновку використовують наступні логічні імена пристроїв:

stdin	пристрій введення
stdout	пристрій висновку
wclips	пристрій, використовуваний як довідковий
wdialog	пристрій для відсилання користувачеві повідомлень
wdisplay	пристрій для відображення правил, фактів і т.ін.
werror	пристрій виведення повідомлень про помилки
wwarning	пристрій для виведення попереджень
wtrase	пристрій для висновку налагоджувальної інформації

Власне функції введення-висновку наступні:

open	відкриття файла (види доступу r, w, r+, a, wb)
close	закриття файла
printout	виведення інформації на заданий пристрій
read	введення даних із заданого пристрою
readline	введення рядка із заданого пристрою
format	форматований висновок на заданий пристрій
rename	перейменування файла
remove	видалення файла.

Серед двох десяток команд *CLIPS* назвемо основні команди під час роботи з середовищем CLIPS:

load	завантаження конструкторів з текстового файла
load+	завантаження конструкторів з текстового файла без відображення
reset	скидання робочої пам'яті системи *CLIPS*
clear	очищення робочої пам'яті системи
run	виконання завантажених конструкторів
save	збереження створених конструкторів в текстовий файл
exit	вихід з *CLIPS*.

У рамках нашого короткого опису опустимо список функцій для роботи з методами родових функцій і список функцій для роботи з класами, об'єктами, слотами, обробниками повідомлень. З цим можна ознайомитися в документації. Список повідомлень про помилки наведений у [261].

На завершення слід мати на увазі, що *CLIPS* може не коректно працювати в реальному часі, коли буде потрібен час реакції менше 0,1 с. У цьому випадку треба досліджувати на розробленому прототипі механізми виведення для всієї множини правил предметної області на різних за продуктивністю комп'ютерах. Як правило, сучасні потужні комп'ютери Intel забезпечують роботу з продукційними системами об'ємом 1000-2000 правил у реальному часі. Web-орієнтовані засоби на базі JAVA (системи Exsys Corvid, JESS) є повільнішими, ніж, наприклад, CLIPS 6.0 або OPS-2000. Тому CLIPS – кращий на сьогодні вибір для роботи в реальному часі серед поширюваних безкоштовних оболонок для розроблення інтелектуальних систем, розроблених на C++.

Додаток В.
Мова програмування newlisp

Мова newLISP – спрощений діалект мови LISP, інтерпретатор якої працює в будь-якій операційній системі і який можна використовувати для "повсякденних завдань" на рівні мови Perl.

Для чого потрібний LISP?

Більшість мов програмування (за винятком BASICa) розроблялися з метою спрощення вирішення певних завдань. Назва мови LISP розшифровується як "LISt Processor", "опрацьовувач списків" – це і є основна область його застосування.

Які дані може містити список? У LISP – практично будь-які. Можна, без особливого перебільшення, сказати, що список у LISP – єдиний структурний тип даних. Записів – аналогів struct або record у LISP немає. Звичайно, сучасні реалізації мають вбудовану підтримку масивів і хешів, проте реально вони використовуються тільки у випадках, коли для алгоритму дійсно потрібні масиви або хеші. Зазначимо, що замість хешів у LISP зазвичай використовуються "асоціативні списки", про них – трохи далі.

Під час запису на LISP, список обмежується круглими дужками, а його елементи розділяються пропусками. Елементами списку можуть бути будь-які значення – константи, імена змінних і функцій (у термінології LISP – символи) і, звичайно, інші списки:

```
(1 2 "abc" var (34 "w")).
```

Асоціативний список – це "список списків", в якому перший елемент використовується як ключ для пошуку:

```
((ключ1 значення1_1 значення1_2) (ключ2 значення2_1 значення2_2)).
```

Для пошуку в асоціативних списках використовуються функції "assoc" і "lookup". З одного боку, такий підхід (теоретично) веде до зниження продуктивності пошуку (на практиці – це питання спірне...), зате з іншої сторони – ви можете мати декілька елементів з однаковим ключем і у вас завжди фіксований порядок проходження записів.

Читати програму дуже просто – все, що розташоване поза круглими дужками – це виклик функції, причому перше слово після відкривальної дужки – це ім'я функції, а інші – її параметри. Те, що математик записав би:

```
f1(x, y),
```

на LISP виглядатиме як:

```
(f1 x y).
```

А складніший запис:

```
f1(x, f2(y, z))
```

перетвориться на:

```
(f1 x (f2 y z)).
```

Приклад, додавання:"Математичний" запис:

```
1 + 2 -> 3
```

LISP-запис:

```
(+ 1 2) -> 3
```

(тут і далі знак "->" передуватиме результату обчислення виразу)

Скажете "Незручно!"? – Тоді подивіться на це:

```
(+ 1 2 3 4 5) -> 15.
```

Є і цікавіші варіанти такого підходу, але про них – нижче.

Як обчислюється LISP-вираз?

Так само, як і в математиці: під час обчислення функції спочатку відбувається обчислення її аргументів, а потім – над ними вчиняються дії, визначені за даною функцією.

```
(+ 10 (sqrt 25)) -> 15
```

Спочатку функція "sqrt" (квадратний корінь) обчислюється, результат (5) передається функції "+", яка, у свою чергу, обчислюється і повертає результат: 10 + 5 = 15.

Декілька корисних прикладів:

```
(setq a "test") -> "test"
```

"setq" – функція присвоєння, вона повертає значення останнього аргумента, і, крім того, також присвоює символу "a" значення "test":

```
a -> "test"
```

обчислення символу повертає його значення.

```
(setq b (sqrt 25)) -> 5,
(setq b '(sqrt 25)) -> (sqrt 25).
```

Апостроф – спеціальний символ, що запобігає обчисленню виразу, результат – звичайний список. Насправді, апостроф – це просто короткий запис для функції "quote". Єдине розширення LISP-СИНТАКСИСу, яке введене у нього у зв'язку з частим використанням:

```
(setq b (quote (sqrt 25))) -> (sqrt 25),
```

"quote", у свою чергу – простий приклад "макро-функції" – спеціальної функції, чиї аргументи автоматично не обчислюються. Докладніше про макро-функції – нижче.

```
(setq b '(("first" 1) ("second" 2) ("third" 3)))
  -> (("first" 1) ("second" 2) ("third" 3))
 (b 1) -> ("second" 2)
```

отримання елемента за індексом, перший елемент має індекс 0. Такий синтаксис в newLISP називається "Implicit indexing" *(примітка: у Common LISP такого синтаксису немає, індексація здійснюється функцією "nth").*

```
(1 b) -> (("second" 2) ("third" 3))
```

зріз з другого елемента і до кінця списку.

```
(0 2 b) -> (("first" 1) ("second" 2))
```

зріз з першого елементу списку завдовжки 2 елементи. Є і традиційніший для LISP спосіб – функції "nth" (елемент за індексом) і "slice" – зріз.

```
(assoc "second" b) -> ("second" 2)
(lookup "second" b) -> 2
(lookup "second" b 0) -> "second"
(lookup "second" b 1) -> 2
```

пошук в асоціативному списку: "assoc" повертає весь підсписок, а "lookup" – останній елемент підсписку, або елемент з індексом, вказаним у третьому параметрі.

Що відразу впадає в очі у коді програми на LISP, то велика кількість круглих дужок. І перше питання, яке зазвичай виникає – навіщо використовувати стільки дужок, коли їх можна замінити такою ж кількістю ком, крапок з комою, фігурних дужок і інших зручних синтаксичних символів.

Відповідь така: Ви, ймовірно, вже звернули увагу на зовнішню схожість LISP-списків і LISP-виразів. Дійсно, LISP-вираз синтаксично є LISP-списком. Більш того, він може опрацьовуватися як звичайний список – зберігатися в змінних, піддаватися перетворенням, передаватися параметром функції у вигляді списку, і, звичайно ж – виконуватись!

Це – одна з основних властивостей LISP – використання коду як даних.

Нескладна програма як приклад: випадкове переміщення по чотирьох напрямках

```
(setq x 0 y 0) ; ініціалізація позиції
(define (up moves) (dec 'y moves))
; визначення функції з ім'ям "up", параметром "moves", яка зменшує
; значення символа "y" на величину "moves"
(define (down moves) (inc 'y moves)) ; аналогічно
(define (left moves) (dec 'x moves))
(define (right moves) (inc 'x moves))
(setq doings (list up down left right))
; це лише список символів - щойно визначених функцій
; функція list - створює список
(seed (date-value)) ; ініціалізуємо давач випадкових чисел
; безпосередній вираз переміщення
(dotimes (i 100)
 ((doings (rand 4)) (rand 5))
 (println x ":" y))
```

Розберемо вираз переміщення докладніше:

`(dotimes (i 100) вираз1 вираз2 ...)` – обчислює "вирази" 100 разів, при цьому символ "i" приймає значення від 0 до 99.

`(rand 4)` – генерує випадкове ціле в діапазоні від 0 до 3.

`(doings (rand 4))` – ця форма була описана вище, як "отримання елемента списку за індексом". Відповідно: "doings" – список, а "(rand 4)" служить індексом.

Тут варто подивитися на список doings уважніше:

```
doings ->
  ((lambda (moves) (dec 'y moves))
   (lambda (moves) (inc 'y moves))
   (lambda (moves) (dec 'x moves))
   (lambda (moves) (inc 'x moves)))
```

Звичайний список може містити код програми. Слово "lambda", яке стоїть на початку кожної функції – елемента doings, не є елементом списку, а означає, що цей список – функція, яка може бути викликана. Такі функції називаються "лямбда-функціями", або "безіменними функціями". У той же час лямбда-функція – це звичайний список, і першим (нульовим) елементом лямбда-списків є елемент "(moves)".

Можливо, у вас виникло запитання: а як же імена – "up", "down", "left" і "right"? Насправді, ці імена всього лише символи-змінні, яким присвоєні як значення відповідні лямбда-списки:

```
up -> (lambda (moves) (dec 'y moves))
(up 1) -> -1 ; відбувається виклик функції з параметром 1
(nth 0 up) -> (moves) ;оскільки implicit indexing тут не працює
;використовуємо функцію nth для отримання 0-го елемента
(setq up-new up)
```



```
up-new -> (lambda (moves) (dec 'y moves))
(up-new 1) -> -2 ; було -1, зменшуємо ще на одиницю...
Фактично, наступні два вирази ідентичні:
(define (up moves) (dec 'y moves))
(setq up '(lambda (moves) (dec 'y moves)))
```

Проте, йдемо далі з нашим прикладом...

`((doings (rand4)) (rand 5))` – оскільки елемент списку "doings" – це функція, ми можемо використовувати її, як звичайний виклик функції, тобто, підставити в LISP-вираз замість імені функції. Відповідно, випадкове число (rand 5) буде аргументом функції.

`(println вираз1 вираз2 ...)` – друкує на екран результати обчислення виразів-аргументів. Крім того, функція "println" повертає як результат значення обчислення останнього виразу-аргумента.

На закінчення розгляду нашого прикладу зазначимо, що з не меншим успіхом ми могли б скласти список "doings" не з коду функцій, а з символів, яким вони присвоєні:

```
(setq doings '(up down left right))
(dotimes (i 100)
(apply (doings (rand 4)) (list (rand 5)))
(println x ":" y))
```

Єдина нова функція тут – "apply" – перший її аргумент – ім'я функції, яку вона повинна викликати, а другий – список аргументів, з якими повинна бути викликана ця функція.

Подібні можливості присутні у процедурних мовах, проте вони використовуються, в основному, висококваліфікованими фахівцями. Щоб зробити їх більш дружніми, в процедурні мови довелося додати об'єктно-орієнтоване програмування, яке, у свою чергу, зажадало ширших об'єктних декомпозицій і внесло порядну плутанину до освоєння програмування.

Проте в LISP використання коду як даних є "повсякденною" практикою.

У мові Common LISP система об'єктно-орієнтованого програмування (CLOS) реалізована як звичайна бібліотека, написана на тому ж Common LISP, – без яких-небудь змін самої мови. У мові newLISP для програмування з об'єктами пропонується вбудована система "контекстів" ("context") – ізольованих просторів імен, що реалізують основні принципи об'єктно-орієнтованого програмування у формі, максимально пристосованій для швидкого "скриптингу".

Винятки, що підтверджують правила

Вище ми вже зустрічалися з символом апострофа, скороченому запису для макрофункції "quote". Дійсно, ця функція – єдиний спосіб в LISP задавати як аргумент функції список-константу. Що робить її такою особливою?

```
(define (test1 arg) (println arg))
; визначимо звичайну функцію
(test1 (+ 1 2)) -> 3
; і випробуємо...
(define-macro (test2 arg) (println arg))
; тепер визначимо "незвичайну" макро-функцію
```

```
(test2 (+ 1 2)) -> (+ 1 2)
; і випробуємо...
```

На відміну від звичайних функцій, аргументи яких автоматично обчислюються перед викликом, під час виклику "макро-функцій" використовується так званий "ледачий" (lazy) порядок обчислення, в якому аргументи не обчислюються взагалі.

Щоб уникнути зайвої плутанини, тут варто відзначити, що слово "макро" в LISP не означає, що функція обчислюється препроцесором до інтерпретації (компіляції) основного коду, як це відбувається в мові Сі. "Макро"-функції, так само як і звичайні, обчислюються безпосередньо під час виконання програми.

Функція "quote" є простою макро-функцією. Якби вона була відсутня в мові, її можна було б визначити так:

```
(define-macro (quote a) a)
; функція приймає один аргумент
; і повертає його в незмінному вигляді (без обчислення) як результат.
```

Якщо ви передаєте аргументом макро-функції LISP-вираз, то цей вираз сам по собі буде доступний як змінна під час обчислення макро-функції, що викликається. Якщо ви захочете його обчислити, необхідно використати функцію "eval":

```
(define-macro (test3 arg) (println (eval arg)))
(test3 (+ 1 2) -> 3
Такий фокус породжує цікаву можливість:
(define-macro (my-if condition result-true result-false)
  (let (_c (eval condition))
    (and _c (eval result-true))
    (or _c (eval result-false))))
(my-if (= 1 1) (println "ture") (println "false")) -> "true"
(my-if (= 1 2) (println "ture") (println "false")) -> "false"
Функція "let" має синтаксис:
(let (символ1 вираз-значення1 символ2 вираз-значення2 ...)
  вираз-дія
....)
```

Вона створює і ініціалізує символи, які діятимуть у момент обчислення виразів-дій, і припинять своє існування (звільняє пам'ять) у момент повернення із функції "let". Результатом функції "let" є результат обчислення останнього виразу-дії.

Функції "and" і "or" – логічні і діють за очевидною схемою: для "or" – якщо перший вираз хибний, обчислюється другий і т.д., повертається результат першого істинного виразу. Для "and" – відповідно.

Отже, ми визначили і випробували нову синтаксичну конструкцію – звичайний оператор умовного розгалуження. І найдивовижніше в цьому операторові те, що зовні він не відмінний від будь-яких, зокрема вбудованих, синтаксичних конструкцій LISP.

Завдяки цій можливості LISP є "мета-мовою", або "мовою для створення мов": одним з рекомендованих методів програмування на LISP є створення власної мови з синтаксисом, зручним для вирішення певного завдання, а потім використання цієї мови для отримання необхідного результату.

Доречно зауважити, що під час створення компіляторів Common LISP велика частина стандартного синтаксису мови реалізується не на мові створення компілятора, а на самій мові Common LISP як бібліотеки.

Звичайно, є і складніші, і продуктивніші способи побудови макро-функцій, засновані на модифікації коду, що передається, і складання на його основі нових виразів. Під час написання макросів на newLISP не забудьте познайомитись з функцією "letex".

Дещо про умовні обчислення

Розглянемо логічні обчислення в newLISP (традиційний LISP має деякі відмінності, зокрема, інше трактування поняття "nil" і вельми незвичні правила рівності).

Основа логічних обчислень, двійкова логіка, побудована на значеннях "істина" і "хибність". У newLISP як значення "хибність" використовується символ "nil". Друге призначення цього символу – "порожнє" значення, яке приймають символи, що не ініціалізувались.

Решта всіх значень в newLISP трактується як "істина" (в т.ч. 0, порожній рядок і порожній список). Для зручності написання програм в newLISP є також спеціальний символ "true" – його можуть повертати деякі логічні функції.

```
(if умова вираз
    умова вираз
    ....
    інакше-вираз)
```

Ось так виглядає справжня функція умовного розгалуження. Якщо "умова" істинна, виконується відповідний "вираз" і обчислення функції закінчується. Якщо всі "умови" помилкові – виконується "інакше-вираз". Якщо який-небудь "вираз" обчислювався, його результат повертається як результат функції, інакше повертається nil.

Зверніть увагу, що "виразом" може бути тільки одиничний вираз:

```
(if вправо (+ x 1))
```

Якщо необхідно виконати послідовність з декількох виразів, їх потрібно "обернути" функцією "begin":

```
(if вправо-вниз (begin
                  (+ x 1)
                  (+ y 1)))
```

"begin" - проста функція для скріплення послідовності виразів. Звичайно, при необхідності, замість неї можна використовувати вже знайому нам функцію "let", інший умовний оператор або ще що-небудь, що відповідає певному випадку.

Зверніть увагу, що в прикладі використані "кириличні" імена символів "вправо" і "вправо-вниз" – у LISP це допустимо.

Альтернативний варіант функції "if" – функція "cond":

```
(cond (умова вираз)
      (умова вираз)
      ...)
```

За рахунок додаткових дужок "вирази" можуть бути послідовністю декількох виразів без додаткових "обгорток".

І, нарешті, насамкінець, до функцій умовного розгалуження – належить функція "case":

```
(case символ
  (тестова-константа вирази)
  (тестова-константа вирази)
  (true вирази))
```

Приклад:

```
(define (translate n)
  (case n
    (1 "one")
    (2 "two")
    (3 "three")
    (4 "four")
    (true "Can't translate this")))
(translate 3)  -> "three"
(translate 10) -> "Can't translate this"
```

У поданому прикладі значення символу "n" послідовно порівнюватиметься з "тестовими константами" 1, 2, 3 і т.д. і, при співпадінні, буде обчислено відповідний вираз. Зверніть увагу, що у цьому прикладі як результатні вирази використовуються рядкові константи. Тестова константа "true" використовується для позначення дії за замовчуванням.

Тепер за допомогою функції "translate" чисельне значення може бути перетворене в слово-числівник.

Зазначимо, що у функції "case" є особливість: "тестові константи" – це саме константи, вони не можуть бути обчислюваними виразами. Тобто, такий запис синтаксично допустимий, але не принесе бажаного результату:

```
(case n
  (a "n рівне значенню a")
  ((+ a 1) "n на одиницю більше значення a"))
```

Перевагою такої поведінки є підвищена (оптимізована) швидкість роботи функції "case".

Але як бути, якщо зручність важливіша? Дуже просто – написати свій макрос:

```
(define-macro (ecase _v)
  (eval (append
          (list 'case _v)
          (map (fn (_i) (set-nth 0 _i (eval (_i 0))))
               (args)))))
```

Випробуємо:

```
(setq a 1 n 2)
; привласнює a=1, n=2
(ecase n
  (a "n рівне значенню a")
  ((+ a 1) "n на одиницю більше значення a"))
  -> "n на одиницю більше значення a"
```

Макро-функція "ecase" працює таким чином: у списку своїх аргументів, такому ж, як для "case", вона замінює всі вирази, що стоять на місцях "тестових констант" на

результат їх обчислення, тобто на константи. Потім на початок перетвореного списку додається символ "case" – ім'я тестованої змінної. Вираз, що утворився, обчислюється за допомогою функції "eval".

Текст такого макросу поки що трохи складний – він стане зрозумілим після пунктів "Перетворення списків" і "Анонімні функції".

Такий "незручний" функціональний запис

Багато зі стандартних функцій LISP можуть опрацьовувати довільну кількість аргументів. Тут, звичайно, можна сказати, що незначна перевага – можливість статично вказати довільний список аргументів. Проте, не поспішайте... Легко відзначити, що аргументи функції "+" є списком. Запишемо його:

```
(setq l '(1 2 3 4 5)) -> (1 2 3 4 5)
l -> (1 2 3 4 5)
```

Тепер у нас є список. Якби ми могли сконструювати вираз з необхідним іменем функції і нашим списком, і обчислити його, то опрацювання функціями довільної кількості аргументів отримала б більше сенсу...

```
(define-macro (my-apply fun lst)
  (eval (cons fun (eval lst))))
(my-apply + l) -> 15
```

Нова для нас функція "cons" створює список, додаючи новий елемент (перший аргумент) на початок того, що існує (другий аргумент). Насправді, ми вже знайомі з вбудованою макро-функцією "apply", яка робить те ж, що і наш макрос.

```
(setq a (apply + l)) -> 15
```

Функція "apply" обчислює функцію, вказану першим аргументом, передаючи їй як аргументи список, вказаний другим аргументом. Звичайно, на місці функції "+" може бути будь-яка інша функція:

```
(define (average) (div (apply add (args)) (length (args))))
  -> (lambda () (div (apply add (args)) (length (args))))
(apply average l) -> 3
```

"add" і "div" - аналоги функцій "+" і "/", але вони працюють з числами з плаваючою крапкою. "args" – функція, що повертає всі незв'язані аргументи, передані функції. Тобто:

```
(define (f x y) (println "x=" x " y=" y " args=" (args)))
  -> (lambda (x y) (println "x=" x " y=" y " args=" (args)))
(f 1 2 3 4) -> x=1 y=2 args=(3 4)
```

Анонімні функції

Функції, які, подібно "apply", аргументами яких є інші функції, в LISP досить багато. Ви навіть можете написати свої власні.

Одною з найбільш розповсюджених є функція "map".

```
(map pow '(1 2 3 4)) -> (1 4 9 16)
; pow - піднесення до квадрату
(map first '((1 2 3) (4 5 6) (7 8 9))) -> '(1 4 7)
; first - повертає перший елемент списку
```

Функція "map" служить для перетворення елементів списку. Функція, вказана в першому аргументі "map" послідовно застосовується до всіх елементів списку, вказаного в другому аргументі. Результати обчислення формують новий список, який і повертається функцією "map". Проте, не все тут здається зручним:

```
(define (third lst) (lst 2))
; функції "third" в мові немає
(map third '((1 2 3) (4 5 6) (7 8 9))) -> '(3 6 9)
```

Ми лише хотіли взяти третій елемент кожного підсписку, а для цього нам довелося заздалегідь визначати функцію. Хоча іноді це буває потрібно, в загальному випадку хотілося б мати можливість без цього обійтися. На допомогу приходить функція "fn" – конструктор анонімних лямбда-списків:

```
(map (fn (lst) (lst 2))
     '((1 2 3) (4 5 6) (7 8 9))) -> '(3 6 9)
```

Робота функції "fn" аналогічна функції "define", за винятком того, що "fn" не здійснює присвоєння створеного лямбда-списку якому-небудь символу:

```
(define (third lst) (lst 2)) -> (lambda (lst) (lst 2))
;крім того, символ "third" набув значення - цей же лямбда-список
(fn (lst) (lst 2)) -> (lambda (lst) (lst 2))
; тільки лямбда список в результаті; ніякого додаткового ефекту
```

У парі до функції "fn" існує "fn-macro", призначена для створення анонімних макросів.

Ми вже бачили використання подібних технічних прийомів вище, тільки у цих прикладах використовувалася не функція-конструктор "fn", а безпосередньо готовий лямбда-список, захищений символом апострофа.

```
(fn (lst) (lst 2)) -> (lambda (lst) (lst 2))
'(lambda (lst) (lst 2)) -> (lambda (lst) (lst 2))
```

За результатом ці вирази еквівалентні. Різниця лише в стислості запису.

Перетворення списків

У прикладі з макросом "ecase" ми вже зустрічали функцію "list", яка дозволяє створювати списки з окремих елементів:

```
(setq a 25)
(list 1 2 3 a) -> (1 2 3 25)
```

Там же ми бачили і функцію "append", яка об'єднує списки:

```
(append '(1 2 3) '(4 5 6)) -> (1 2 3 4 5 6)
```

А в прикладі з "my-append" – функцію "cons", яка додає елемент в початок списку:

```
(cons 1 '(2 3 4)) -> (1 2 3 4)
(cons '(1 2) '(3 4)) -> ((1 2) 3 4)
```

У традиційному LISP, де списки програмно подані у вигляді "голови" і "хвоста", "cons" відіграє набагато істотнішу роль, об'єднуючи ці два компоненти. У newLISP списки будуються за "лінійним" принципом.

Набагато більший інтерес викликають функції, призначені для перетворення одного списку в інший. Найпростішою і очікуваною є функція "filter", яка фільтрує список, залишаючи у ньому тільки значення, що задовольняють задану умову:

```
; (filter тестова_функція список)
(filter (fn (x) (not (empty? x))) '("abc" "" "def" "jhi" ""))
  -> ("abc" "def" "jhi")
```

"empty?" - функція повертає значення "true", якщо її аргумент – не порожній рядок і не порожній список. "filter" повертає список, в якому залишаються тільки ті елементи, для яких тестова функція повертає істину.

Зворотною за дією є функція "clean":

```
(clean empty? '("abc" "" "def" "jhi" "")) -> ("abc" "def" "jhi")
```

Ще одна проста, але корисна функція – "join" – об'єднує список рядків в один рядок.

```
(join '("abc" "" "def" "jhi") ":") -> "abc::def:jhi"
(join '("abc" 123 "def" "jhi") ":") -> Помилка! - 123 не є рядком
(join (map string '("abc" 123 "def" "jhi")) ":") -> "abc:123:def:jhi"
```

Перший параметр "join" – список рядків, другий – рядок, що розділяє. Використана функція "string" перетворює будь-який тип даних у текстове подання.

І, звичайно, "найвідомішою" є розглянута функція "map". Відзначимо, що вона може опрацьовувати декілька списків.

```
(map (fn (x y) (+ x y)) '(1 2 3 4) '(5 6 7 8)) -> (6 8 10 12)
```

А, враховуючи відому властивість "+", можна навіть простіше:

```
(map + '(1 2 3 4) '(5 6 7 8)) -> (6 8 10 12)
```

Тепер ви знаєте достатньо, щоб повернутися до макросу "ecase" і зрозуміти, як він працює.

Методи програмування

Якщо у вас є навики процедурного програмування, то швидко досягнете успіху в написанні програми на мові LISP (і, тим більше, на newLISP). Однак, отриманий результат не буде перевершувати аналоги на традиційних мовах програмування. LISP – функціональна мова, в ній діють свої закони ефективності і знайдені свої методи комфортного програмування.

Самодостатні функції

"Чисте", академічно правильне функціональне програмування вимагає, щоб використовувані функції не мали "сторонніх ефектів". Тобто, функція може обробляти тільки ті дані, які отримує як параметри, і єдиним результатом її роботи повинно бути значення, що повертається. Функція не повинна використовувати або змінювати які-небудь "глобальні" змінні.

Не зважаючи на те, що в реальній практиці такий стиль не завжди можливий і ефективний, прагнення до нього дозволяє зробити програму прозорішою і зробити логічні помилки нагляднішими.

Функціональний запис

Скорочення використання сторонніх ефектів дозволяє, у свою чергу, збільшити можливості функціонального запису – те, що в процедурній мові зазвичай записується у вигляді декількох виразів, у LISP часто можна скласти в один. При цьому вкладена структура LISP-виразу добре виявляє логічні зв'язки його компонентів. До того ж на вигляд функціональний запис більш схожий на природну мову.

Розроблення знизу-вгору

Звичайно, не варто чекати "прозорості" від функціонального виразу довжиною в дві сторінки. Всім відомо, що дуже великий код корисно ділити на функції. Звичайний для процедурних мов підхід – визначити завдання, виділити підзадачі, і ділити їх, поки не стане можливим написати для них окремі функції. Функції все одно виходять великі і складні, а під кінець розроблення стає зрозуміло, що треба було вирішувати трохи інше завдання... Такий спосіб називається розробленням "зверху-вниз".

У LISP, завдяки його властивостям мета-мови, все робиться навпаки. Для вирішення завдання досліджується наочна область і на основі можливостей LISP створюється нова мова. Потім на цій мові записується рішення необхідної задачі. Якщо "раптом" з'ясовується, що початковий план зазнав істотних змін, то не біда – на вже готовій мові нескладно записати і щось нове. Цей спосіб називається розробленням "знизу-вгору".

Особливо приємним наслідком розроблення "знизу-вгору" є можливість на кожному рівні розв'язування задачі використовувати найбільш відповідну мову, знову ж таки, легко трансльовану в логіку природної мови.

Самодокументування

До всього вище сказаного додамо два зауваження: – частіше використовувати "самодокументальні" імена функцій і змінних. Імена "t", "tmp", "ex" і т.п. хороші в локальних ділянках коду. Проте для глобальних імен на зразок "exit-state" підійде набагато краще. Іноді – одноразову дію краще оформити у вигляді окремої функції.

Не позбавляйте себе можливості писати так:

```
(do-select ((CustomerName CustomerEmail)
            :from Customers
            :where (> CustomerAge 100))
  (send-email CustomerEmail
              :subject "Congratulations!"
              :body (format nil
              "Dear ~A, you won a prize! Call ~A."
              CustomerName company-phone)))
```

Коментарі зайві, чи не так? Цей приклад написаний в синтаксисі Common Lisp. З ним можна ознайомитись на сторінці http://linux.org.ru.

Опрацювання списків

І, нарешті, пам'ятайте, що LISP створений для опрацювання списків. Чим більше можливостей для застосування списків ви побачите у вашому завданні, тим більше можливостей щодо опрацювання ваших даних ви дістанете.

Форматування коду

Існують різні думки про те, як краще форматувати код на LISP і newLISP. Нижче викладений достатньо зручний "канонічний" метод.

Відступи

Звичайний відступ для позначення вкладеності встановлюється рівним двом пропускам.

Відступ для вкладених списків даних – один пропуск (оскільки відкривальні дужки на попередній стрічці можуть розташовуватися одна за одною).

Якщо у функції багато аргументів і перший розташований в одному рядку з ім'ям функції, решту аргументів зручно писати під першим "у стовпчик" (як у попередньому прикладі).

Для спеціальних функцій, таких як "let" або "if", у яких перший параметр має особливе значення, перший параметр зазвичай починають в одному рядку з ім'ям функції, а інші – на інших рядках із стандартним відступом в два пропуски:

```
(if (= a b)
  (println "рівність")
  (println "нерівність"))
```

Закривальні дужки

Закривальні дужки зазвичай не виносять на окремі рядки, як це робиться в мові "Сі" і подібних. Це безглуздо для виділення структури, оскільки відступи і так несуть всю необхідну інформацію.

Для контролю балансу дужок краще використовувати текстовий редактор, що забезпечує можливість автоматичного пошуку парних дужок, – це уміють всі сучасні редактори "програмістів".

Якщо такий редактор відсутній, дужки зручно закривати "всліпу" – очима рахувати відступи коду з найбільш внутрішнього до рівня, який треба завершити, і кожного разу всліпу натискати закривальну дужку на клавіатурі.

Практичний скриптинг

Найбільш загальновживаним завданням скриптингу є, звичайно ж, опрацювання текстових звітів. Безумовним лідером за зручністю тут є AWK – немає мови, на якій можна було б писати лаконічніше. На жаль, це досить проста мова, що швидко здає позиції, коли виникає необхідність у складному опрацюванні даних. Цю область вже традиційно очолює PERL.

Для тих же завдань непогано підходить і newLISP. Ця мова також підтримує Perl-сумісні регулярні вирази PCRE і дозволяє здійснювати аналіз текстових документів. Особливістю застосування newLisp є його орієнтація на опрацювання списків (в той час, як стиль Perl тяжіє до потокового опрацювання). Це означає, що використовуючи newLisp, зазвичай вигідніше не опрацьовувати звіт (рядок – за рядком), а прочитати його весь, розділити на список, що складається з його частин (рядків, слів в рядках) і потім опрацювати, використовуючи всю потужність LISP.

Для прикладу, розглянемо просте завдання. Припустимо, у файлі "report.txt" ви маєте звіт вигляду:

```
+----------------------------+
|  Приклад текстового звіту  |
+----------------------------+
| uid     | balance | volume |
+----------------------------+
| user1   | 100000  | 1000   |
| user2   | 200000  | 1234   |
........
```

і хочете дізнатися суму по стовпчику volume:

```
(apply +
(map (fn (x) (int (x 3) 10))
(filter (fn (x) (and (> (length x) 3) (regex "^[0-9]+$"
(x 3))))
                (map (fn (x) (parse x " *\| *" 0))
                (parse (read-file "report.txt") "\n")))))
```

Нові функції: "read-file" – читає файл, як рядок. "parse" – ділить рядок на список рядків за вказаними роздільниками; при вказівці додаткових опцій, може використовувати регулярні вирази (PCRE). "\n" – символ кінця рядка. "regex" – шукає відповідність регулярному виразу, якщо не знаходить, повертає nil. "int" – перетворить рядок в ціле число.

Щоб розібратися, як це працює, запишемо те ж, але акуратніше:

```
(define (split-report-line str)
  (map (fn (x) (parse x " *\| *" 0))
       str))
(define (line-has-data? lst)
  (and (> (length lst) 3) (regex "^[0-9]+$" (lst 3))))
(apply +
  (map (fn (x) (int (x 3) 10))
       (filter line-has-data?
       (map split-report-line
       (parse (read-file "report.txt") "\n")))))
```

Тепер основний код коментарів практично не потребує. Коротко: читається файл звіту (цілком), розбивається на список рядків, рядки розділяються на елементи, фільтруються рядки, що мають дані для аналізу, з кожного рядка береться четвертий елемент (індекс – 3) і перетворюється в число, список чисел, що утворився в результаті, підсумовується.

Дві допоміжні функції також очевидні: перша розділяє рядок регулярним виразом (і одночасно видаляє всі зайві пропуски), друга перевіряє розділяючий рядок-список на наявність числового значення в четвертому (що цікавить нас) елементі.

Мабуть, на Perl це вдасться записати набагато компактніше. Проте не варто забувати, що основний результат цього скрипту – отримання списку всіх елементів звіту, які потім можна групувати, фільтрувати, співставляти і перераховувати з використанням всього багатства інструментів опрацювання списків.

На завершення – декілька слів про можливості мови newLISP

newLISP створений і активно розвивається стараннями Луца Мюллера (Lutz Mueller). Це інтерпретатор, написаний на чистому Сі, із застосуванням тільки стандартної

бібліотеки libc. Така реалізація зробила newLISP багатоплатформною мовою, що однаково працює під багатьма юніксами і під всіма версіями MS Windows.

В newLISP вбудовані такі можливості, як:

- TCP/IP, можливість роботи мережевим доменом і підтримка http,
- породження процесів, семафори, сигнали, таймери,
- виклик функцій з динамічних бібліотек (.so, .dll і т.ін.),
- регулярні вирази PCRE,
- XML-парсер,
- інтерфейс до TCL/Tk (Windows-дистрибутив включає графічну бібліотеку Tk),
- система уніфікації (unify), що дозволяє реалізувати Пролог-подібну логіку,
- байесівські класифікатори (із застосуванням допоміжної бібліотеки з дистрибутива),
- підтримка Unicode,
- множина математичних функцій.
- newLISP опублікований під ліцензією GNU GPL і його початкові тексти і бінарні файли доступні за адресою http://newlisp.org. На цьому ж сайті ви дістанете доступ до документації (на англійській мові), корисні посилання і підказки.

Серед документації слід особливо виділити надзвичайно корисні "офіційні" документи: "newLISP Manual and Reference" – довідник з мови і "newLISP Design Patterns" – опис прийомів програмування завдань зі всіх основних областей застосування мови.

Додаток Г.
Мова програмування python

Основи

Мову Python можна розглядати як псевдокод. Змінні ніби-то не мають типів – тому їх не потрібно оголошувати. Вони виникають, коли ви присвоюєте їм значення, і зникають, коли ви більше їх не використовуєте. (Типи змінних визначаються автоматично при виконанні програми). Присвоєння здійснюється оператором =, а рівність перевіряється оператором == . Ви можете присвоїти значення декільком змінним одночасно:

```
x, y, z = 1, 2, 3
first, second = second, first
```

(спочатку обчислюються всі вирази в правій частині рівності, зліва направо, потім ці значення по черзі присвоюються змінним, перерахованим у лівій частині)

```
a = b = 123
```

Блоки операторів позначаються відступами, і лише відступами (ніяких "операторних дужок" ніби BEGIN – END або фігурних дужок). Ось деякі звичайні структури, що керують:

```
if  (x > 10) and (x < 20) :
print "ікс хороший"
if 10 < x < 20 :
print "ікс хороший"
for i in [1, 2, 3, 4, 5] :
   print "це - номер циклу: ", i
x = 10
while x > 0 :
   print "x все ще додатний"
```

Перші два приклади еквівалентні.

Змінна циклу (індекс) у прикладі циклу `for` приймає всі можливі значення зі списку (list) (який записується, як в прикладі). Щоб зробити "звичайний" цикл (тобто з лічильником кількості виконань), використовуйте вбудовану функцію `range()`.

```
#Друкує всі значення від 0 до 99 включно:
for value in range (100) :
print value
```

(Рядок програми, що починається з "#", є коментарем й ігнорується інтерпретатором).

Розглянемо як створити деякий елементарний інтерфейс користувача. Щоб він міг вводити дані (на запрошення програми), використовуйте функцію `input`:

```
x = input ("Введи сюди число:")
print "Квадрат цього числа складає ", x * x
```

Функція `input` показує задане запрошення (яке може бути порожнім), дозволяючи користувачеві ввести будь-яке допустиме в Python значення. У поданому прикладі ми чекаємо, що буде введене число. Але якщо буде введено щось інше (скажімо, рядок), то програма завершиться аварійно. Щоб уникнути цього, нам знадобиться якийсь контроль помилок. Ми не будемо заглиблюватися в це. Досить сказати, що, якщо ви хочете, щоб

те, що ввів користувач, було записане в програмі буквально, у вигляді рядка символів, то використовуйте замість input функцію `raw_input`.

Зауваження. Якщо ви хочете, щоб функцією `input` вводився рядок символів, то користувач повинен в явному вигляді вказувати лапки. У Python для рядків може використовуватися пара як одинарних, так і подвійних лапок.

Отже, у нас є структури, що керують, команди введення-виведення, тепер ми хочемо мати якісь цікаві структури даних (тобто типи все ж таки є). Найбільш важливими з них є *списки* (list) і словники (dictionary). Списки записуються в квадратних дужках і, природно, можуть бути вкладеними:

```
name = ["Cleese", "John"]
x = [[1, 2, 3], [y, z], [[ ]] ]
```

Однією з привабливих якостей списків є те, що ви можете звертатися до їх елементів не тільки окремо, але і по групах, використовуючи індексацію (indexing) і перетини (slicing). Індексація, як і в багатьох інших мовах, здійснюється додаванням до імені списку індексу в квадратних дужках (перший елемент має індекс 0).

```
print name [1], name [0]
Друкує: "John Cleese"
name [0] = "Smith"
```

Перетини нагадують індекси, проте ви вказуєте початковий і кінцевий індекс, через ":", отримуючи в результаті підсписок:

```
x = ["сміття", "сміття", "сміття", "яйця", "і", "сміття"]
print x [3:5]
Друкує список: ["яйця", "і"]
```

Зверніть увагу, що верхня межа діапазону індексів не включається. Якщо одна з меж діапазону опущена, то розуміються всі елементи в цьому напрямку. Наприклад, вираз `spysok [:3]` означає "кожен елемент списку spysok з першого і до третього, не включно". (Ви можете заперечити, що індекс 3 насправді означає четвертий елемент, оскільки рахунок йде з 0. Дійсно ...). З іншого боку, вираз `spysok [3:]` означає "всі елементи списку, починаючи з того, що має індекс 3, включно, і до кінця списку, включаючи останній". Дійсно цікавий результат можна отримати, застосовуючи від'ємний індекс: `spysok [-3]` це третій елемент з кінця. Кажучи про індекси, потрібно згадати, що вбудована функція len повертає довжину списку.

Тепер щодо словників. Кажучи просто, вони схожі на списки, тільки елементи в них не впорядковані. Як ви можете робити в них індексацію? Для цього кожен елемент має *ключ* (key), або "ім'я", за яким його можна знайти точно так само, як і в звичайному словнику. Ось декілька прикладів словників:

```
"Alice" : 23452532, "Gennady" : 252336, "Clarice" : 2352525}
person={'ім'я' : "Robin", 'прізв.' : "Hood", 'заняття' : "партизан"}
```

Тепер, щоб дізнатися, чим займається людина, перерахована в словнику person, ми використовуємо вираз вигляду person ['заняття']. Якщо ми хочемо змінити прізвище людини, ми пишемо:

```
person ['прізвище'] = "of Locksley"
```

Просто, чи не так? Як і списки, словники можуть містити в собі інші словники або списки. І, природно, списки можуть містити в собі словники. Це дозволяє вам створювати досить складні структури даних.

Функції

Наступний крок: абстракція. Ми хочемо присвоїти ім'я частині програмного коду і потім викликати його, задаючи параметри. Іншими словами, ми хочемо визначати функції (або процедури). Це легко. Для визначення функції використовується ключове слово `def`, наприклад:

```
def square (x) :
return x*x
print square (2)
Друкує: 4
```

Для тих з вас, хто розуміє, що це таке: всі параметри функцій в Python передаються посиланням (як, наприклад, в Java). Для тих, хто не розуміє: не піклуйтеся про це. Python має такі привабливі особливості, як іменовані аргументи функцій і значення для аргументів за замовчуванням. Якщо ви маєте загальне уявлення про те, як користуватися функціями, то це, в основному, і все, що вам треба знати про їх використання в Python (треба ще знати ключове слово return, яке зупиняє роботу функції і повертає вказане значення як її результат). Важливо, проте, знати, що Python функції є значеннями. Тому, якщо ви маєте функцію, скажімо, square, то ви можете написати:

```
queeble = square
print queeble (2)
Друкує: 4
```

Викликаючи функцію без аргументів, не забувайте писати її ім'я з дужками, тобто `doit ()`, а не `doit`. В останньому випадку буде, як показано вище, повернена сама функція як значення (це буває потрібно для задання методів для об'єктів, про що дивіться нижче).

Об'єкти і все, що їх стосується

Ми вважаємо, що ви знаєте основні властивості об'єктно-орієнтованого програмування (інакше цей розділ для вас буде не дуже корисний, але це неважливо, можна почати програмувати і без об'єктів). У Python ви визначаєте клас, використовуючи ключове слово `class` . Наприклад:

```
class Basket :
#створюємо клас Basket ("кошик") з
#методами add("помістити щось у кошику")
# і print_me ("роздрукувати весь вміст
#кошика").Не забувайте використовувати
#аргумент "self"
def __init__(self, contents=NONE):
self.contents = contents or []
def add (self, element):
self.contents.append (element)
def print_me (self):
result = ""
for element in self.contents:
result = result + " " + `element`
print "Містить:" + result
```


Тут новим є те, що:

- всі методи (функції, задані для об'єкту) мають додатковий аргумент, що розташований на початку списку аргументів, – а саме `self` ("*сам*"); він має спеціальне значення – позначає сам об'єкт класу, що визначається;
- методи викликаються в програмі так: *Об'єкт.Метод (Аргументи);*
- деякі імена методів, такі як `__init__`, зарезервовані і мають спеціальне значення; так `__init__` – це ім'я конструктора класу, тобто функції, яка автоматично викликається кожного разу, коли ви створюєте екземпляр класу (тобто змінну);
- деякі аргументи є необов'язковими і їм дається значення за замовчуванням (як було згадано вище, в розповіді про функції); значення за замовчуванням задається таким чином:

```
def f (x=100):
```

тут функція f може бути викликана з одним параметром або без параметрів; якщо вона викликана без параметрів, то за замовчуванням параметру x присвоюється значення 100;

- *"логіка коротких висновків"* (short cirquit logic); це розумно; див. нижче;
- лапки зі зворотним нахилом перетворять об'єкт в його стрічкове подання (отже якщо element містить число 1, то `element` це рядок "1", тоді як 'element' це рядок зі словом element);
- знак додавання „+" використовується для конкатенації списків, а також і рядків, оскільки вони насправді є списками символів (це означає, що ви можете використовувати для рядків індексацію, перетини і функцію `len` .

У Python ніякі методи класів не є захищеними (приватними і т.ін.). Інкапсуляція в основному є питанням стилю програмування.

Тепер про "логіку коротких висновків" ... Всі значення в Python можуть використовуватися як логічні. Ті, які "порожні" (як 0 [] або "" і `None`) відображають логічне значення "хибно", а більшість інших (напр., [0], 1, "Hello, comrade") відображають логічне значення "істина". Тому значення логічних виразів на зразок a and b ("a І b") обчислюються таким чином: спочатку перевіряється, чи є a істиною. Якщо ні, то просто повертається a. Якщо так, то повертається b (яке задаватиме кінцеве значення виразу). Аналогічно, для значення виразу a or b ("a АБО b") маємо: якщо a істинне, то повертається a. Якщо ні, то повертається b.

Це дозволяє операторам `and` і `or` діяти на зразок булевих операторів "і" та "або". Також це дозволяє коротко записувати умовні вирази. Наприклад, вираз:

```
if a:
print a
else:
print b
```

можна записати коротше:

```
print a or b
```

Фактично, подібне скорочення є щось подібне до ідіом в Python, тому вам треба до цього звикнути. Ми вже використали його в методі `Basket.__init__`. Аргумент contents має значення за замовчуванням `None` (нічого), яке крім іншого означає логічну *"хибність"* . Тому, щоб перевірити, що contents має якесь значення, ми могли б написати:

```
if contents :
self.contents = contents
else:
self.contents = []
```

Тепер ви, звичайно, знаєте, що є кращий спосіб. А чому просто ми не привсвоємо із самого початку значення за замовчуванням [] параметру contents? У цьому випадку Python присвоїв би всім нашим "кошик" (екземплярам об'єкту Basket) одне і те ж значення за замовчуванням, порожній список []. Коли ми стали б заповнювати один список, всі інші містили б ті ж елементи, і вже не мали б значень за замовчуванням. Щоб краще з'ясувати цей момент, прочитайте в документації по Python пояснення різниці між *ідентичністю* (identity) і рівністю (equity). Інший спосіб запису вищенаведеного фрагменту:

```
def __init__ (self, contents=[]) :
self.contents = contents [:]
```

Подумайте, як це працює. Замість того, щоб скрізь використовувати один і той же порожній список, ми за допомогою виразу contents [:] робимо його копію. (Ми просто робимо перетин списку, включаючи в нього всі елементи списку. Цей перетин і є потрібною нам копією). Отже, щоб створити екземпляр об'єкту Basket і використовувати його (тобто викликати деякі його методи), ми можемо написати приблизно наступне:

```
b = Basket (['яблуко', 'помаранча'])
b.add ("цитрина")
b.print_me ()
Друкує: яблуко помаранча цитрина
```

Є й інші потрібні методи, окрім __init__. Одним з них є метод __str__, який задає, як виглядає об'єкт, коли він розглядається як рядок. Ми можемо його використати для нашого "кошика" замість методу print_me:

```
def __str__ (self) :
result = ""
for element in self.contents :
result = result + " " + element
return "Старий дірявий кошик містить:" + result
```

Тепер, якщо ми хочемо роздрукувати вміст "кошика" b, ми пишемо в програмі:

```
print b
```

Створення підкласу здійснюється таким чином:

```
class SpamBasket (Basket) :
    # ...
    (вводимо новий клас "кошик для сміття"
    як підклас класу "кошик").
```

Python допускає множинне наслідування, тому ви можете вказати в круглих дужках відразу декілька суперкласів, розділяючи їх комами. Екземпляр класу створюється так:

```
x = Basket ()
```

Конструктори задаються за допомогою спеціальної функції-члена класу __init__. Припустимо, клас SpamBasket має конструктор __init__ (self, type), де параметр type позначатиме тип вмісту в "кошику для сміття". Тоді ми можемо створити новий "кошик для сміття" таким чином:


```
y = SpamBasket ("яблука")
```

Python використовує не лексичні, як завжди, а динамічні простори імен. Це означає, що якщо у вас є функція типу:

```
def orange_juice () :
return x*2
```

де змінна (в нашому випадку x) не пов'язана з аргументом і не набуває значення всередині функції, то Python використовує те значення, яке вона мала у момент виклику функції. У нашому випадку:

```
x = 3
orange_juice ()
Повертає: 6
x = 1
orange_juice ()
Повертає: 2
```

Звичайно, це та поведінка, яку ви і хочете. Проте, іноді хочеться мати статичний простір імен, тобто зберігати в тілі функції деякі значення, задані ззовні функції, якими вони були у момент її створення. От як це робиться в Python за допомогою значень для аргументів за замовчуванням:

```
x = 4
def apple_juice (x = x) :
return x*2
```

Тут аргументу x задається значення за замовчуванням, яке співпадає зі значенням однойменної змінної x, якою вона була під час визначення функції. Отже, якщо викликати функцію без аргументу, то вийде наступне:

```
x = 3
apple_juice ()
Повертає: 8
x = 1
apple_juice ()
Повертає: 8
```

Отже, значення x не змінюється. Якби це було все, чого ви хочете, то можна було б з тим же успіхом написати:

```
def tomato_juice () :
x = 4
return x*2
```

або навіть:

```
def carrot_juice () :
return 8
```

Проте, ідея полягає в тому, що значення x береться з програмного оточення функції і зберігається таким, яким воно було у момент визначення функції. Для чого це? Розглянемо наступний приклад – функція, яка комбінує дві інші функції. Ми хочемо, щоб функція працювала приблизно так:

```
from math import sin, cos
sincos = compose (sin, cos)
x = sincos (3)
```

де compose – функція, яку ми хочемо створити, а x має значення -0.836021861538, яке рівне значенню виразу sin(cos(3)). Тепер, як ми робитимемо це? Функція compose бере як параметри дві функції, а повертає ще одну функцію, яка, у свою чергу, бере один параметр. Отже, схема рішення може бути такою:

```
def compose (fun1, fun2) :
def inner (x) :
# ...задаємо функцію, яка буде
# значенням, що повертається, для compose
return inner
```

Ми могли б спробувати помістити в тіло функції inner оператора return fun1 (fun2 (x)) і зупинитися на цьому. Однак, так не можна. Така функція поводилася б дуже дивно. Розглянем наступний сценарій:

```
from math import sin, cos
def fun1 (x) :
return x + " comrade!"
def fun2 (x) :
return "Hello "
sincos = compose (sin, cos)#використовуючи неправильну версію
```

Яке тепер значення x? Правильно, "Hello, comrade!". Чому так? Тому, що у момент виклику, функція compose бере значення функцій fun1 і fun2 із свого програмного оточення, а не ті fun1 і fun2, які були присутні в програмі у момент створення функції. Все, що нам потрібно зробити, щоб отримати правильне рішення, це застосувати раніше описаний прийом:

```
def compose (fun1, fun2)
def inner (x, fun1=fun1, fun2=fun2) :
return fun1 (fun2 (x))
return inner
```

Тепер ми тільки можемо сподіватися, що ніхто не викличе результатну функцію з більш ніж одним аргументом, оскільки це може мати катастрофічні наслідки для неї. І, до речі, нам не потрібно вводити ім'я для функції inner – оскільки вона містить тільки один вираз, то ми можемо використовувати анонімну функцію, яка оголошується, використовуючи ключове слово lambda замість імені:

```
def compose (f1, f2) :
return lambda x, f1=f1, f2=f2 : f1 (f2 (x))
```

Більшість корисних функцій і класів поміщені в модулі, які фактично є текстовими файлами, що містять програмний код на мові Python. Ви можете імпортувати їх в свою програму і використовувати. Наприклад, щоб використовувати метод split із стандартного модуля string, ви можете зробити наступне:

```
import string
x = string.split (y)
```

або ...

```
from string import split
x = split (y)
```


Детальніше про модулі стандартної бібліотеки, див. http://www.python.org/doc/lib. Там є дуже багато корисного. Якщо ви хочете, щоб ваша програма могла працювати і як модуль, і як незалежна програма, то ви можете додати в кінці стрічку вигляду:

```
if __name__ == "__main__": go ()
```

Це – спосіб перевірити, чи працює програма як виконуваний скрипт (тобто не будучи імпортована в інший скрипт) – в цьому випадку буде викликана функція go. Зрозуміло, замість неї ви можете вставити після ":" будь-який програмний код. І для тих з вас, хто хоче створювати скрипти для UNIX: щоб скрипт працював як незалежна програма, його першою стрічкою повинен бути наступний коментар, що здійснює керування:

```
#!/usr/bin/env python
```

Нарешті, стисло згадаємо важливе поняття: *виняток* (exception).

Деякі операції, такі як спроба ділення на 0 або читання з неіснуючого файлу, можуть створювати стан помилки, або виняток. Ви навіть можете робити свої власні винятки і *порушувати* (raise) їх у відповідний момент. Якщо винятки ніяк не опрацьовуються в програмі, то вона завершує свою роботу з видаванням повідомлення про помилку. Ви можете уникнути такого завершення, застосувавши оператора try/except . Наприклад:

```
def safe_division (a, b):# безпечне ділення a на b
try :
return a/b
except ZeroDivisionError: # стандартний виняток
#"ділення на 0"
return None
```

ZeroDivisionError – це ім'я стандартного винятку ("Ділення на 0"). У поданому прикладі ви могли б перед діленням вставити перевірку того, чи b не рівне 0. Але у багатьох випадках така стратегія непридатна. І крім того, якби у нас не було секції try в тілі функції safe_division ("безпечне ділення"), так що нам справедливо довелося б назвати її unsafe_division ("небезпечне ділення"), то ми все одно змогли б опрацювати цей виняток таким чином:

```
try:
unsafe_division (a, b)
except ZeroDivisionError:
print "Ділення на 0 у функції unsafe_division"
```

У випадках, коли у вас зазвичай не повинна виникати деяка проблема, але в принципі вона все ж таки може трапитися, використання опрацювання винятків позбавить вас від трудомісткого тестування і т.ін.

Додаткові приклади див. на сторінці *"Tidbit"* Джо Строута (Joe Strout) [262].

Додаток. Приклад програми QuickSort

Розглянемо приклад програми з функцією сортування за алгоритмом "швидкого сортування" Хоара. Оскільки, не знаючи суть методу, зрозуміти програму важко, то наводимо його опис, який можна знайти у відомій книжці Д.Кнута [0] і ін., використовуючи позначення з цього програмного прикладу.

Цей приклад виключно навчальний, оскільки сортування запрограмоване просто за визначенням методу, без додаткових прийомів підвищення ефективності. І, крім того, в Python і так є стандартний метод sort, який використовується так: *Список_имп.sort (Фпорівняння)*, де *Фпорівняння* – необов'язкова призначена для користувача функція

порівняння елементів. Якщо функцію не вказати, то буде використана стандартна функція cmp(), яка дає "природний порядок" елементів. Наприклад:

```
a = [2, 5, 1, 3]
a.sort ()
print a
Друкує: [1, 2, 3, 5]
```

Алгоритм "швидкого сортування" (Quick Sort) Чарльза Хоара (Hoar)

Дано: список list, індекси першого і останнього впорядкованих елементів – start і end. Змінні: bottom – менший індекс порівнюваного елементу ("нижнього"), top – більший індекс порівнюваного елементу ("верхнього"), pivot – значення роздільного елементу, partition – допоміжна функція, що ділить список на два підсписки, в одному з яких всі елементи менші за pivot, а в іншому – більші.

Вибирається довільний елемент списку (у нашому прикладі - останній, хоча це поганий вибір, якщо список вже частково впорядкований). Він називається роздільним елементом, оскільки використовується, щоб розділити список на дві частини, так що в одній частині елементи будуть менші за той, що ділить, а в іншій більші. Послідовно перебираються всі елементи відразу з нижнього і верхнього кінця. Якщо значення нижнього елементу більше pivot, то він пересилається на місце верхнього (значення якого вже збережене). Якщо верхній елемент більший за pivot, то він пересилається на місце нижнього. Коли індекси верхнього і нижнього елементу співпадуть, список виявиться розділеним на дві частини – елементи з меншим індексом, ніж у того, що розділяє (split), будуть по величині не більше за нього, а інші – не менші. Отже, неврегульованість залишиться тільки в межах цих частин. Рекурсивно застосовуючи цей алгоритм до отриманих частин списку, отримуємо у результаті частини, що складаються з одного елементу, тобто впорядкований список. На початку програми присутня текстова константа, яка ніде не використовується. Це не помилка, а необов'язковий, але рекомендований в Python "рядок документування" для ідентифікації модуля (класу, функції). Щоб можна було випробувати програму, виділіть фрагмент цього документа (починаючи з рядка *#!/usr/bin/env python*), що містить її, і збережіть його у вигляді неформатованого текстового файлу з ім'ям на зразок qsort.py (py – тип файлу для програм на Python).

```
#!/usr/bin/env python
# Written by Magnus Lie Hetland
"Everybody's favourite sorting algorithm... :)"
def partition(list, start, end):
pivot = list[end]   # Хай останній елем. буде таким, що
розділяє
bottom = start-1    # Починаємо з частини списку, що ззовні
розділяється
top = end      # Аналогічно
done = 0
while not done:      # Поки не всі елем. розділені ...
while not done:      # Поки ми не знайдемо
    # неврегульований елемент ...
bottom = bottom+1   # ... Рухаємо нижній індекс вгору.
```



```
if bottom == top:   # Якщо він співпаде з верхнім ...
done = 1 # ... то все зроблено.
break
if list[bottom]> pivot:  # Чи не є нижній елем.
   #неврегульованим?
list[top]= list[bottom]# Тоді переміщуємо його
# на місце верхнього ...
break    # ... і починаємо пошук зверху.
while not done:      # Поки не знайдемо неврегульований елем.
top = top-1    # ... рухаємо верхній індекс вниз.
if top == bottom:   # Якщо він співпав з нижним ...
done = 1 # ... то все зроблено.
break
if list[top]< pivot:      # Чи є верхній
# елем. неврегульованим?
list[bottom]= list[top]# Якщо так, то переміщаємо
# його на місце нижнього ...
break    # ...і починаємо пошук знизу.
list[top]= pivot # Вміщується роздільний елемент на місце.
return top     # І повертаємо його індекс
# (точку розділення підсписків).
def quicksort(list,start,end):
if start < end:     # Якщо є хоч би
   # два елементи ...
split = partition(list, start, end) # ...то розділяємо на
# два підсписки і ...
quicksort(list, start, split-1)     # сортуємо обидві поло-
вини.
quicksort(list, split+1, end)
else:
return
if __name__=="__main__":# Якщо цей скрипт виконується як
# незалежна програма:
import sys
list = sys.argv[1:] # Отримуємо арґументи командного рядка
start = 0
end = len(list)-1
quicksort(list,start,end) #Сортуємо весь список арґументів
import string
print string.join(list)  # Роздруковуємо сортований список.
sys.exit (0)  # Вихід з програми.
```

Література

1. Аверкин А. Н. Нечеткие множества в моделях управления и искусственного интеллекта / А.Н.Аверкин, И.З.Батыршин, А.Ф.Блишун. – М.: Наука, 1986. – 312с.
2. Айвазян С.А. Прикладная статистика: Основы моделирования и первичная обработка даннях / С.А.Айвазян, И.С.Енюков, Л.Д.Мешалкин. – М.: Финансы и статистика, 1983. – 471с.
3. Айзенк Г. Коэффициент интеллекта / Г. Айзенк. – К.: Гранд, 1994. – 192с.
4. Алахвердов В. М. Когнитивные стили в контурах процесса познания. Когнитивные стили / В.М. Алахвердов ; Под ред. В. Колги. – Таллинн, 1986. – С. 12–23.
5. Александров Е.А. Основы теории эвристических решений / Е.А.Александров. – М.: Сов. радио, 1975. – 256 с.
6. Алексеева И.А. Психосемантическая реконструкция картины мира подростков, употреблявших токсические вещества / И.А.Алексеева, А.В.Воинов, А.Р.Сейсян, А.М.Эткинд // Сб. Конструктивная психология – новое направление развития психологической науки. Под ред. А.Г. Копытова. Красноярский гос. ун-т., 1989. – С. 36-41.
7. Алексеевская М. А. Диагностические игры в медицинских задачах / М.А.Алексеевская, А.В.Недоступ // Вопросы кибернетики : Задачи медицинской диагностики и прогнозирования с точки зрения врача – 1998. – № 112. – С. 128-139.
8. Андриенко Г.Л. Игровые процедуры сопоставления в инженерии знаний / Г.Л.Андриенко, Н.В.Андриенко // Сборник трудов III конференции по искусственному интелекту. – Тверь, 1992. – С. 93-96.
9. Анисимов А.В. Система обработки текстов на естественном языке / А.В.Анисимов, А.А.Марченко // Научно-теоретический журнал „Искуственный интелект", ІПШІ „Наука і освіта". – 2002. – Вип. 4 – С. 157-163.
10. Апресян Ю. Д. Экспериментальное исследование семантики русского языка / Ю.Д.Апресян. – М.: Наука, 1977. – 251с.
11. Аткинсон Р. Человеческая память и процесс обучения / Р.Аткинсон. – М.: Прогресс, 1980.
12. Бабак В.Ф. Аспекты проектирования информационных систем / В.Ф.Бабак, И.Н.Рыженко // Тезисы конференции посвященной 200-ю со дня рождения Пушкина, Бішкек – КРСУ июль 1999. – С. 34-35.
13. Бахтин М. М. Вопросы литературы и эстетики: Исследования разных лет. / М.М.Бахтин. – М.: Художественная литература, 1975. – 502с.
14. Белнап Н. Логика вопросов и ответов / Н.Белнап, Т.Стил. – М.: Прогресс, 1981. – 288с.
15. Берков В. Ф. Вопрос как форма мысли / В.Ф.Берков. – Минск: Изд-во БГУ, 1972. – 136с.
16. Берн Э. Игры, в которые играют люди. Люди, которые играют в игры / Э.Берн. – М.: Прогресс, 1988. – 155с.

17. Бойкачев К. К. «Сценарий» – инструмент визуальной разработки компьютерных программ / К.К.Бойкачев, И.Г.Конева, И.З.Новик // Компьютерные технологии в высшем образовании: Сб. статей. – М.: Изд-во МГУ, 1994. – С. 167-178.
18. Болотова Л.С. Неформальные модели представления знаний в системах искусственного интеллекта / Л.С.Болотова, А.А.Смольянинов. – Московский институт радиотехники, электроники и автоматики (ТУ) – М., 1999. – 100с.
19. Борисов А. Н. Приобретение знаний для интеллектуальных систем / А.Н.Борисов, А.В.Алексеев. – М.: Радио и связь, 1989. – 304с.
20. Борисова Н. В. Деловая игра "Документ": Метод. пособие / Н.В.Борисова, А.А.Соловьёва. – Гос. ком. СССР по науке и технике. ИПКИР. – М., 1983. – 55 с.
21. Боронников А.Б. Построение информационных систем на основе технологии XML Веб-сервисов / А.Б.Боронников, С.В.Семенов // Программные продукты и системы, №4. – 2004. – С.61-62.
22. Братко И. Программирование на языке ПРОЛОГ для искусственного интеллекта / И.Братко. – М.: Мир, 1990. – 560с.
23. Брунер Дж. Исследование развития познавательной деятельности / Дж.Брунер // Пер. с англ. М.: Педагогика, 1971. – 413с.
24. Бублик Б.Н. Основы теории управления / Б.Н.Бублик, Н.Ф.Кириченко. – К.: Вища шк., 1975. – 328с.
25. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Гради Буч. – М.: Бином, 1998. – 560с.
26. Величковский Б.М. Современная когнитивная психология / Б.М.Величковский. – М.: МГУ, 1982. – 336с.
27. Величковский Б. М. Психологические проблемы изучения интеллекта / Б.М.Величковский, М.С.Капица // Интеллектуальные процессы и их моделирование. – М.: Наука, 1987. – С. 120-141.
28. Верес О.М. Алгоритм укладання розкладу навчальних зайнять у ВНЗ / О.М.Верес // Інформаційні системи та мережі. Вісник Національного Університету "Львівська політехніка". – Львів. – 1998. – № 330. – С. 40-51.
29. Вертгеймер М. Продуктивное мышление / М.Вертгеймер. – М.: Прогресс, 1987. – 336с.
30. Винер Н. Кибернетика или управление и связь в животном и машине / Н.Винер. – М.: Сов. радио., 1958. – 344с.
31. Виноград Т. Программа, понимающая естественный язык / Т.Виноград. – М.: Мир, 1976. – 296с.
32. Винцюк Т.К. Анализ, распознование и интерпретация речевых сигналов / Т.К.Винцюк. – К.: Наук. думка, 1987. – 264с.
33. Вовк О.Б. Метрологічні основи інтелектуальних систем / О.Б.Вовк, Р.Б.Кравець, Я.О.Івахів, В.В.Пасічник // Матеріали Міжнар. наук. конф. „The VI International Workshop for Candidadates for a Doctor's Degree 2004". – 2004. – Т.1 : Archiwum Konferencji, Vol.19. – P.44-46.

34. Волков А. М. Классификация способов извлечения опыта экспертов / А.М.Волков, В.С.Ломнев // Известия АН СССР. Техническая кибернетика. – 1989. – № 5. – С. 34-45.

35. Вольфенгаген В. Э. Система представления знаний с использованием семантических сетей / В.Э.Вольфенгаген, О.В.Воскресенская, Ю.Г.Горбанев // Вопросы кибернетики: Интеллектуальные банки данных. – М.: АН СССР, 1979. – С.49-69.

36. Гаврилова Т. А. Представление знаний в экспертной диагностической системе АВТАНТЕСТ / Т.А.Гаврилова // Изв. АН СССР. Техническая кібернетика. – 1984. – № 5. – С.165–173.

37. Гаврилова Т. А. От поля знаний к базе знаний через формализацию / Т.А.Гаврилова // Журнал «Представление знаний в экспертных системах». – Л.: ЛИМАН, 1989. – С.16-24.

38. Гаврилова Т. А. Объектно-структурная методология концептуального анализа знаний и технология автоматизированного проектирования баз знаний / Т.А.Гаврилова // Труды Междунар. конф. «Знания – диалог–решение 95». – 1995. – т. 1. – С.9.

39. Гаврилова Т. А. О концептуальном анализе знаний при разработке экспертных систем / Т.А.Гаврилова, М.Р.Красовская // Доклад на Всесоюзной научно-практической конференции «Гибридные интеллектуальные системы». – Ростов, н/Д, 1991. – С. 110-113.

40. Гаврилова Т. А. Экспертные системы для оценки качества деятельности летного состава / Т.А.Гаврилова, С.П.Минкова, Г.С.Карапетян // Тез. докладов научно-практической школы-семинара «Программное обеспечение и индустриальная технология интеллектуализации разработки и применения ЭВМ». – Ростов, н/Д, 1988. – ВНИИПС. – С. 23-25.

41. Гаврилова Т. А. МИКРОЛЮШЕР: экспертная система интерпретации данных / Т.А.Гаврилова, А.И.Тишкин, А.Ю.Золотарев // Доклад на школе-семинаре «Проблемы применения ЭС в народном хозяйстве». Кишинев, 1989. – С.17-23.

42. Гаврилова Т. А. Базы знаний интеллектуальных систем / Т.А.Гаврилова, В.Ф.Хорошевский. – СПб: Питер, 2001. – 384с.

43. Гаврилова Т. А. Извлечение и структурирование знаний для экспертных систем / Т.А.Гаврилова, К.Р. Червинская. – М.: Радио и связь, 1992. – 200с.

44. Гаврилова Т. А. Формирование поля знаний на примере психодиагностики / Т.А.Гаврилова, К.Р.Червинская, А.М. Яшин // Техническая кибернетика. – 1988. – № 5. – С.72-85.

45. Гаек П. Автоматическое образование гипотез: математические основы общей теории / П.Гаек, Т.Гавранек. – М.: Наука, 1983. – 264с.

46. Гаращенко Ф.Г. Аналіз та оцінка параметричних систем / Ф.Г.Гаращенко, Л.А.Панталієнко. – К.: ІСДО, 1995. – 140с.

47. Гельфанд И. И. Структурная организация данных в задачах медицинской диагностики и прогнозирования / И.И.Гельфанд, Б.И.Розенфельд, М.А.Шифрин // Вопросы кибернетики. Задачи медицинской диагностики и прогнозирования с точки зрения врача. – М.: АН СССР. 1998 – С.5-64.

48. Георгиев В. О. Модели представления знаний предметных областей диалоговых систем / В.О.Георгиев // Изв. АН СССР. Техническая Кибернетика. – 1991 – № 5.– С. 3-23.

49. Гиг Дж. Прикладная общая теория систем / Дж.Гиг // В 2-х кн. – М.: Мир, 1981. – 773с.

50. Гинкул Г. П. Игровой подход к приобретению знаний и его реализация в системе КАПРИЗ / Г.П.Гинкул // Проблемы применения экспертных систем в народном хозяйстве: тез. докл. респ. школы-семинара. – Кишинев, 1989. – С.71-74.

51. Глибовець М.М. Штучний інтелект / М.М.Глибовець, О.В.Олецький. – К.: КМ Академія, 2002. – 366с.

52. Глушков В. М. Введение в кибернетику / В.М.Глушков. – К.: Издательство АН УССР, 1964. – 324с.

53. Гордон Д. Терапевтические метафоры / Д.Гордон. – Личная рукопись М. М. Кагана. – 1987. – 234с.

54. Горелов И. Н. Разговор с компьютером / И.Н.Горелов. – М.: Наука, 1987. – 255с.

55. Городецкий В.И. Свойства моделей координации поведения агентов и модель планирования на основе аукциона / В.И. Городецкий // DIAMAS' 97, СПб., 1997. – С. 34-39.

56. Даревич Р.Р. Метод автоматичного визначення інформаційної ваги понять в онтології бази знань / Р.Р.Даревич, Д.Г.Досин, В.В. Литвин // Відбір та обробка інформації. – 2005. – Вип. 22(98). – С.105-111.

57. Даревич Р.Р. Оцінка подібності текстових документів на основі визначення інформаційної ваги елементів бази знань / Р.Р.Даревич, Д.Г.Досин, В.В. Литвин, З.Т. Назарчук – Исскуственный интеллект. – Донецк. – № 3. – 2006. – С. 500-509.

58. Джексон П. Введение в экспертные системы / П.Джексон. – М.-С-П.-К.: Изд. дом "Вильямс ", 2001. – 616с.

59. Дрейфус Х. Чего не могут вычислительные машины: Критика искусственного разума / Х.Дрейфус. – М.: Прогресс, 1978. – 333с.

60. Дэйвисон М. Многомерное шкалирование. Методы наглядного представления данях / М.Дэйвисон . – М.: Финансы и статистика, 1988. – 254с.

61. Дюбуа Д. Теория возможностей. Приложения к представлению знаний в информатике / Д.Дюбуа, А.Прад. – М.: Радио и связь, 1990. – 288 с.

62. Дюк В.А. Компьютерная психодиагностика / В.А.Дюк. – СПб.: Братство, 1994. – 364с.

63. Дюран Б. Кластерный анализ / Б.Дюран, П.Оделл. – М.: Статистика, 1977. – 128с.

64. Епифанов М. Е. Индуктивное обобщение в ассоциативных сетях / М.Е.Епифанов // Известия АН СССР. Техническая кібернетика. – 1984. – № 5. – С.132-146.

65. Ефимов Е.И. Решатели интеллектуальных задач / Е.И.Ефимов. – М.: Наука, 1982. – 316 с.

66. Жинкин Н. И. Речь как проводник информации / Н.И.Жинкин. – М.: Наука, 1982. – 250с.

67. Зенкин А. А. Основы когнитивной компьютерной графики / А.А.Зенкин. – М.: Наука, 1991. – 191с.

68. Иберла К. Факторный анализ / К.Иберла. – М.: Статистика, 1980. – 398с.

69. Иванов П. И. Влияние некоторых индивидуально-психологических особенностей на процесс обобщения / П.И. Иванов // Автореферат дис. канд. психол. наук. М., 1986.

70. Ивахненко Г. И. Системы эвристической самоорганизации в технической кибернетике / Г.И.Ивахненко. – Киев: Техніка, 1971. – 102с.

71. Искусственный интеллект: Справочник: в 3-х т. – М.: Радио и связь, 1990.

72. Івашків А.М. Класифікація інтелектуальних інформаційних систем / А.М.Івашків, В.В.Литвин // тези доповідей сьомої всеукраїнської наукової конференції "Сучасні проблеми прикладної математики та інформатики". – Львів, 2000. – С. 49-50.

73. Івашків А.М. Проблема класифікації інтелектуальних інформаційних систем / А.М.Івашків, В.В.Литвин // Інформаційні системи та мережі. Вісник Національного Університету "Львівська політехніка", №406. – Львів, 2000. – С. 112-117.

74. Каган М. С. Мир общения: проблема межсубъектных отношений. / М.С.Каган М.: Политиздат, 1988. – 319с

75. Капица П. Л. Приглашение к спору / П.Л.Капица // Журнал „Юность", 1967. – № 1. – С. 79-82.

76. Кириченко Н.Ф. Множественные проблемы анализа и синтеза систем управления / Н.Ф.Кириченко, В.Т.Матвиенко // Кибернетика и системн. Анализ. – 1996. – №3. – С.68-77.

77. Клайн П. Справочное руководство по конструированию тестов / П.Клайн. – К.: ПАН Лтд, 1994. – 286с.

78. Кнут Д. Искусство программирования / Д.Кнут: в 3-х т., 3-е изд. – М.: "Вильямс", 2000.

79. Комаров В. Ф. Управленческие имитационные игры / В.Ф.Комаров. – Новосибирск: Наука, 1989. – 256с.

80. Коов М. И. Интеграция концептуальных и экспертных знаний в САПР / М.И.Коов, М.Б.Мацкин, Э.Х.Тыугу // Известия АН СССР. Техническая кибернетика. – 1988. – № 5. – С. 108-118.

81. Криницкий Н.А. Алгоритмы и роботы / Н.А.Криницкий. – М.: Сов. Радио, 1983. – 164с.

82. Круглов В.В. Нечеткая логика и искусственные нейронные сети / В.В.Круглов, М.И.Дли, Р.Ю.Голунов. – Санкт-Петербург, 2006. – 221с.

83. Кузичева З. А. Языки науки, языки логики, естественные языки / З.А.Кузичева // Логика научного познания. Актуальные проблемы. М.: Наука, 1987. – С. 57–73.

84. Кузнецов В. Е. Представление в ЭВМ неформальных процедур / В.Е.Кузнецов. – М.: Наука, 1989. – 160с.

85. Кук Н. М., Макдональд Дж. Формальная методология приобретения и представления экспертных знаний / Н.М.Кук, Дж.Макдональд // ТИИЭР. – 1986. – Т. 74. – № 10. – С.145-155.

86. Кулюткин Ю. И. Индивидуальные различия в мыслительной деятельности взрослых учащихся / Ю.И.Кулюткин, Г.С.Сухобская. – М.: Педагогика, 1971. – 111с.

87. Лазарева Т. К. Деловые имитационные игры в экспертных системах / Т.К.Лазарева, Н.Д.Пашинин // Деловые игры и их программное обеспечение: тез. докл. – Пущино, 1987. – С. 63-64.

88. Лещев В.А. Обзор основных стандартов W3C / В.А.Лещев, И.А.Конюхов, А.С.Семенов // Программные продукты и системы. – №4. – 2004. – С.62-64.

89. Лимантов Ф. С. О природе вопроса / Ф.С.Лимантов // Вопрос. Мнение. Человек. Уч. записки ЛГПИ им. Герцена. – 1971. – Т. 497. – С. 4-20.

90. Литвин В.В. Двоетапний метод пошуку розв'язку в метаінтелектуальній інформаційній системі / В.В. Литвин // тези доповідей сьомої всеукраїнської наукової конференції "Сучасні проблеми прикладної математики та інформатики". – Львів, 2000. – С. 59-60.

91. Литвин В.В. Модель функціонування інтелектуальної інформаційної системи / В.В. Литвин // XV відкрита науково-технічна конференція молодих науковців і спеціалістів Фізико-механічного інституту ім. Г.В.Карпенка НАН України. Тези доповідей. – Львів, 2000. – С. 122-123.

92. Литвин В.В. Об'єктно-орієнтований підхід до проектування баз даних / В.В. Литвин // Задачі та методи прикладної математики. Вісник Львівського Національного Університету ім. І.Франка, Випуск 50. – Львів, 1998. – С. 158-159.

93. Литвин В.В. Основні методи фільтрування інформації та їх використання при побудові інтелектуальних інформаційних систем / В.В.Литвин // Інформаційні системи та мережі. Вісник Національного Університету "Львівська політехніка", №438. – Львів, 2001. – С. 94-99.

94. Литвин В.В. Побудова інтелектуальних інформаційних систем на основі об'єктно-орієнтованої моделі представлення знань / В.В.Литвин // Комп'ютерна інженерія та інформаційні технології. Вісник Національного Університету "Львівська політехніка", №433. – Львів, 2001. – С. 94-102.

95. Литвин В.В. Реалізація множинного наслідування в об'єктно-орієнтованих базах даних / В.В.Литвин // Інформаційні системи та мережі. Вісник Національного Університету "Львівська політехніка", №383. – Львів, 1999. – С. 140-144.

96. Литвин В.В. Формальна модель інтелектуальної інформаційної системи / В.В.Литвин // Комп'ютерна інженерія та інформаційні технології. Вісник Національного Університету "Львівська політехніка", № 386. – Львів, 1999. – С. 104-108.

97. Литвин В.В. Основні принципи та функціональне наповнення інформаційної системи "Школа" для автоматизації управління навчальним процесом / В.В.Литвин, Р.Б.Бакаїм, О.Й.Процовський, В.М.Садовий, Н.Б.Шаховська, Р.В.Шаховський // Інформаційні системи та мережі. Вісник Національного Університету "Львівська політехніка", № 383. – Львів, 1999. – С. 145-149.

98. Литвин В.В. Застосування утилітарного підходу до реалізації природномовного інтерфейсу інтелектуальних інформаційних систем / В.В.Литвин, В.В.Григор'єв // Інформаційні системи та мережі. Вісник Національного Університету "Львівська

політехніка", №438. – Львів, 2001. – С. 99-103.

99. Литвин В.В. Інтелектуальні інформаційні системи з кількома інтерпретаторами / В.В.Литвин, Р.Б.Кравець // Інформаційні системи та мережі. Вісник Національного Університету "Львівська політехніка", №438. – Львів, 2001. – С. 104-108.

100. Литвин В.В. Інтелектуальна інформаційна система планування навчального процесу / В.В.Литвин, О.Ю.Назаров, С.В.Чумаченко // Інформаційні системи та мережі. Вісник Національного Університету "Львівська політехніка", №406. – Львів 2000. – С. 179-184.

101. Литвин В.В. Застосування методів логічного програмування для автоматизації процедур прийняття рішень у діяльності служби працевлаштування / В.В.Литвин, Ю.В.Нікольський // Інформаційні системи та мережі. Вісник Національного Університету "Львівська політехніка", № 330. – Львів, 1998. – С. 153-163.

102. Литвин В.В. Моделювання компонент інтелектуальної інформаційної системи / В.В.Литвин, Ю.В.Нікольський // труды международной научно-практической конференции KDS-2001 "Знание – диалог – решение". – Санкт-Петербург, июнь, 2001. – т. II. – С. 433-438.

103. Литвин В.В., Паров'як І.П., Пелещишин А.М., Процовський О.Й., Садовий В.М. Принципи реалізації інтегрованих комп'ютерних технологій для періодичних видань / В.В.Литвин, І.П.Паров'як, А.М.Пелещишин, О.Й.Процовський, В.М.Садовий // Інформаційні системи та мережі. Вісник Національного Університету "Львівська Політехніка", №383. – Львів, 1999. – С.149-154.

104. Литвин В.В. Проблема координації інтелектуальних інформаційних систем / В.В.Литвин, Н.Б.Шаховська // тези доповідей III Міжнародної науково-практичної конференції "Системний аналіз та інформаційні технології" "Сучасні технології в інформаційному суспільстві". – Київ, 2001. – С. 77-81.

105. Литвинов В. Методы построения имитационных систем / В.Литвинов, Т.Марьянович – К.: Наук. думка, 1991. – 115с.

106. Логика рассуждений и ее моделирование / Под ред. Д.А.Поспелова. – М.: Науч. совет по комплекс. пробл. "Кибернетика" АН СССР, 1983. – 180 с.

107. Логический подход к искусственному интеллекту: от классической логики к логическому программированию / А.Тейз, П.Грибомон, Ж Луи и др. – М.: Мир, 1990. – 432с.

108. Лорьер Ж.-Л. Системы искусственного интеллекта / Ж.-Л.Лорьер. – М.: Мир, 1991. – 568 с.

109. Лунева О. В. Психология делового общения / О.В.Лунева, Е.А.Хорошилова. – М.: ВКШ при ЦК ВЛКСМ, 1987. – 213с.

110. Любарский Ю.Я. Интеллектуальные информационные системы / Ю.Я.Любарский. – М.: Наука, 1990. – 232 с.

111. Мальковский М. Г. Диалог с системой искусственного интеллекта / М. Г.Мальковский. – М.: МГУ, 1985. – 216с.

112. Малюта Т.А. Интелектуализация реляционных баз данных путем введения в них возможностей работы с неполной и неточной информацией / Т.А.Малюта, В.В. Пасічник // Conference on intelligent management systems / Bulgarian academy of science. – Varna, 1989. – P.36–103.

113. Мартынов В. В. Универсальный семантический код / В. В.Мартынов. – Минск: Наука и техника, 1977. – 191с.

114. Маслов С.Ю. Теория дедуктивных систем и ее применение / С.Ю.Маслов – М.: Радио и связь, 1986. – 135 с.

115. Материалисты древней Греции // Собрание текстов Гераклита, Демокрита и Эпикура. – М.: Политиздат, 1955. – 240с.

116. Мельчук И. А. Опыт теории лингвистических моделей «Смысл-Текст». Семантика, синтаксис. / И.А.Мельчук. – М.: Наука, 1974. – 315с.

117. Месарович М. Общая теория систем: Математические основы / М.Месарович, Я.Такахара. – М.: Мир, 1978. – 310с.

118. Минский М. Фреймы для представления знаний / М.Минский. – М.: Энергия, 1979. – 151с.

119. Миркес Е.М. Нейрокомпьютер. Проект стандарту / Е.М.Миркес. – Наука, Новосибирск, 1998. – 337с.

120. Мицич П. П. Как проводить деловые беседы / П. П.Мицич. – М.: Экономика, 1987. – 208с.

121. Мичи Д., Джонстон Р. Компьютер-творец / Д.Мичи, Р.Джонстон. – М.: Мир, 1987. – 255с.

122. Моисеев Н. Н. Математические задачи системного анализа / Н.Н.Моисеев. – М.: Наука, 1981. – 488с.

123. Молокова О. С. Методология анализа предметных знаний / О.С.Молокова // Новости искусственного интеллекта. – 1992. – № 3. – С. 11-60.

124. Моргоев В. К. Метод структурирования и извлечения экспертных знаний: имитация консультацій / В. К.Моргоев // Человеко-машинные процедуры принятия решений. – М.: ВНИИСИ, 1988. – С. 44-57.

125. Немов Р. С. Социально-психологический анализ эффективной деятельности коллектива / Р.С.Немов. – М.: Педагогика, 1984. – 201с.

126. Нечеткие множества в моделях управления и искусственного интеллекта / А.Н.Аверкин, И.З.Батыршин, А.Ф.Блишун и др. – М.: Наука, 1986. – 312с.

127. Нильсон Н. Принципы искусственного интеллекта / Н.Нильсон. – М: Мир, 1985. – 373с.

128. Нікольський Ю.В. Дискретна математика / Ю.В.Нікольський, В.В.Пасічник, Ю.М.Щербина – Львів: Магнолія Плюс, 2007. – 608 с.

129. Норенков И. П. Введение в автоматизированное проектирование технических устройств и систем / И.П.Норенков. – М.: Высшая школа, 1986. – 304с.

130. Ноэль Э. Массовые опросы / Э.Ноэль. – М.: Прогресс, 1978. – 108с.

131. Обозов И. И. Психологическая культура взаимных отношений / И.И.Обозов. – М: Знание, 1986. – 88с.

132. Орехов А. И. Формирование приемов эффективного решения творческих задач / А.И.Орехов // Автореферат дис. канд. психол. наук. М., 1985.

133. Осипов Г. С. Метод формирования и структурирования модели знаний для одного типа предметных областей / Г.С.Осипов // Известия АН СССР. Техническая кибернетика. – 1988. – № 2. – С.3-12.

134. Осипов Г. С. Приобретение знаний интеллектуальными системами / Г.С.Осип – М.: Наука, 1997. – 345с.

135. Осипов Г.С. Построение моделей предметных областей. Неоднородные семан ческие сети / Г.С.Осипов // Изв. АН СССР. Техническая Кибернетика. – 1990. – 5. – С. 32-45.

136. Пасичник В.В. Анализ структуры экспертных систем / В.В.Пасичник, Н.М.Проданюк // Вестн. Львов. Политехн. Ин-та. – 1986. – №219. – Технические средства автоматизации измерений и управления научными исследованиями. – С.7-10.

137. Пасичник В.В. Информационная система для автоматизированного анализа экспертных систем / В.В.Пасичник, Н.М.Проданюк // Тез. докл. Всесоюз. Семинара „Актуальные проблемы развития перспективных информационных технологий", 27-28 окт. 1987 г. – М., 1987. – С.25-28.

138. Пасічник В.В. Організація баз даних та знань / В.В.Пасічник, В.А.Резніченко. – Київ: ВНУ „ПИТЕР", 2006. – 460с.

139. Петренко В. Ф. Введение в экспериментальную психосемантику: исследование форм репрезентации в обыденном сознании / В.Ф.Петренко. – М.: МГУ, 1983. – 175с.

140. Петренко В. Ф. Психосемантика сознания / В.Ф.Петренко. – М.: Издательство МГУ, 1988. – 207с.

141. Погосян Г. А. Метод интервью и достоверность социологической информации / Г.А.Погосян. – Ереван, АН Арм. ССР, 1985. – 142с.

142. Понтрягин Л.С. Математическая теория оптмальных процес сов / Л.С.Понтрягин, В.Г.Болтянский, Р.В.Гамкрелидзе. – М.: Наука, 1983. – 392с.

143. Попов Э. В. Экспертные системы / Э. В. Попов. – М.: Наука, 1987. – 285с.

144. Попов Э. В. Общение с ЭВМ на естественном языке / Э.В.Попов. – М.: Наука, 1982. – 360с.

145. Попов Э. В. Экспертные системы 90-х гг. Классификация, состояние, проблемы, тенденции / Э. В.Попов // Новости искусственного интеллекта. – 1991. – № 2. – С. 84-101.

146. Попов Э. В. Статические и динамические экспертные системы / Э.В.Попов, И.Б.Фоминых, Е.Б.Кисель. – М.: Финансы и статистика, 1996. – 321с.

147. Поспелов Г.С. Искусственный интеллект – основа новой информационной технологи / Г.С.Поспелов. – М.: Наука, 1988. – 280 с.

148. Поспелов Г.С. Искусственный интеллект – прикладные системы / Г.С.Поспелов, Д.А.Поспелов. – М.: Знание, 1985. – 48 с.

149. Поспелов Д. А. Искусственный интеллект: фантазия или наука / Д.А.Поспелов. – М.: Радио и связь, 1986. – 224с.

150. Поспелов Д. А. Ситуационное управление: Теория и практика / Д.А.Поспелов. – М.: Наука, 1986. – 288с.

151. Поспелов Д. А. Моделирование рассуждений. Опыт анализа мыслительных актов / Д.А.Поспелов. – М.: Радио и связь, 1989. – 184с.

152. Поспелов Д. А. Фантазия или наука: На пути к искусственному интеллекту / Д.А.Поспелов. – М.: Наука, 1982. – 224с.

153. Похилько В. И. Система КЕПУ / В.И.Похилько, Н.Н.Страхов. – М.: МГУ, 1990. – 35с.

154. Представление знаний в человеко-машинных и робототехнических системах. Фундаментальные исследования в области представления знаний. – М.: ВЦ АН СССР ВИНИТИ, 1984. – 262 с.

155. Рассел С. Искусственный интеллект / С.Рассел, П.Норвиг. – М.,С.-П.,К.: Вильямс, 2006. – 1408с.

156. Ребельский И. В. Азбука умственного труда / И.В.Ребельский // 1989. – ЭКО. – №7. – С. 43-150.

157. Розенталь М. Краткий философский словарь / Розенталь М., Юдин П.. М.: Политиздат, 1951. – 450с.

158. Сагатовский В. И.. Социальное проектирование / В.И.Сагатовский // Прикладная этика и управление нравственным воспитанием. – Томск, 1980. – С. 84-86.

159. Сергеев В. М. Когнитивные модели в исследовании мышления: структур и онтология знания / В.М.Сергеев // Интеллектуальные процессы и их моделирование. – М.: Наука, 1987. – С. 179-195.

160. Сергеев К. А. Логический анализ форм научного поиска / К.А.Сергеев, А.Н.Соколов. – Л.: Наука, 1986. – 120с.

161. Сиротко-Сибирский С. А. Смысловое содержание текста и его отражение в ключевых словах / С.А.Сиротко-Сибирский // Автореферат дис. канд. филол. наук. Л., 1968.

162. Словарь по кибернетике /под ред. В.С.Михалевича. – К.: Гл.ред. УСЭ, 1989. – 751с.

163. Совпель И.В. Система автоматического извлечения знаний из текста и ее приложения / И.В.Совпель // Научно-теоретический журнал „Искуственный интелект", ІПШІ „Наука і освіта" 2004. – Вип. 3 – С.668-677.

164. Соколов А. Н. Внутренняя речь и мышление / А.Н.Соколов. – М.: Просвещение, 1968. – 256с.

165. Соколов А. Н. Проблемы научной дискуссии / А.Н.Соколов. Л.: Наука, 1980. – 157с.

166. Стогний А.А. Анализ ЭС по сферам применения / А.А.Стогний, И.И.Брона, В.В.Пасичник //- Сб. тр. ВАН МТА SZTAKI. – Будапешт, 1988. – №214. – С.39-79.

167. Стогний А.А. Экспертные системы – эффективный инструмент современных информационных технологий / А.А.Стогний, И.И.Брона, В.В.Пасичник // Вестн. АН УССР. – К., 1989. – №5. – С.17-26.

168. Терехина А.Ю. Представление структуры знаний методами многомерного шкалирования / А.Ю.Терехина. – М.: ВИНИТИ, 1988. – 97с.

169. Тиори Т., Фрай Дж. Проектирование структур баз данных / Т.Тиори, Дж.Фрай : В 2-х кн. – М.: Мир, 1985. – 288с.

170. Уинстон П. Исскуственный интеллект / П.Уинстон. – М.: Мир, 1980. – 519с.

171. Уотерман Д. Руководство по экспертным системам / Д. Уотерман. – М., Мир, 1989. – 388 с.

172. Уэно Х. Представление и использование знаний / Х.Уэно, М.Исидзука. – М.: Мир, 1989. – 220с.

173. Фаин В. С. Машинное понимание естественного языка в рамках концепции реагирования / В.С. Фаин // Интеллектуальные процессы и их моделирование. – М.: Наука, 1987. – С. 375-391.

174. Фаулер М. UML в кратком изложени / М.Фаулер, К.Скотт. – М.: Мир, 1999. – 340с.

175. Фляків Д.В. Розробка технології застосування штучних нейронних мереж у прикладних інформаційних системах / Д.В.Фляків // Дисс.. канд.техн.наук, Алтайський Державний університет, Барнаул, 2000.

176. Фон Гумбольдт В. Избранные труды по языкознанию / В.Фон Гумбольдт М.: Прогресс, 1984. – 400с.

177. Франселла Ф. Новый метод исследования личности: руководство по репертуарным личностным методикам / Ф.Франселла, Д.Баннистер // М.: Прогресс, 1987. – 62с.

178. Хант Э. Искусственный интеллект / Э.Хант. – М.: Мир, 1978. – 588с.

179. Хейес-Рот Ф. Построение экспертных систем / Ф.Хейес-Рот, Д.Уотерман, Д.Ленат. – М.: Мир, 1987. – 430с.

180. Частиков А.П. Разработка экспертных систем. Среда CLIPS / А.П.Частиков, Т.А.Гаврилова, Д.Л.Белов. – СПб.: БХВ-Петербург, 2003. – 608с.

181. Шенк Р. Обработка концептуальной информации / Р.Шенк. – М.: Энергия, 1980. – 361с.

182. Шенк Р. К интеграции семантики и прагматики / Р.Шенк, Л.Бирнбаум, Дж.Мей // Новое в зарубежной лингвистике. Компьютерная лингвистика. – Вып. 14. – М.: Прогресс, 1989.

183. Шенк Р. Познать механизмы мышления / Р.Шенк, Л.Хантер // Реальность и прогнозы искусственного интеллекта. – М.: Мир., 1987. – С.15-26.

184. Шеннон К. Математическая теория святи / К.Шеннон, У.Уивер. – М.: ИЛ, 1963. – 345с.

185. Шепотов Е. Г. Методы активизации мышления / Е.Г.Шепотов, Б.В.Шмаков, П.Д.Крикун. – Челябинск: Челябинский политехнический институт. – 1985. – 84 с.

186. Штерн І.Б. Вибрані топіки та лексикон сучасної лінгвістики: Енцикл. Словник / І.Б.Штерн. – К.: АртЕк, 1998. – 336с.

187. Шумилина Т. В. Интервью в журналистике / Т. В.Шумилина. – М.: МГУ, 1973. – 136с.

188. Экспертные системы. Принципы работы и примеры. / Под редакцией Р.Форсайта. – М.: Радио и связь, 1987. – 231с.

189. Эндрю А. Искусственный интеллект / А.Эндрю. – М.: Мир, 1985. – 264с.

190. Яцишин Ю.В. Складність обчислення рекурсивних функцій / Ю.В. Яцишин // Доп. АН УССР. Сер. А. –1987. – № 2. – С. 17–20.

191. Яцишин Ю. В. Некоторые свойства абстрактной меры сложности вичислений / Ю.В. Яцишин // Асимптотические методы в исследовании стохастических моделей. – К.: Ин-т математики АН УССР, 1987. – С. 130–139.

192. Яцишин Ю.В. Нижняя оценка регистровой сложности вычисления термов / Ю.В. Яцишин // Дискретная математика. – 1990. – № 4. – Т. 2. – С. 60–63.

193. Яцишин Ю.В. Побудова інтелектуальних систем із застосування технологій data mining / Ю.В. Яцишин // ІІІ Міжнародна науково практична конференція: Тез. допов. – Ірпінь, 2002. –266 с.

194. Яцишин Ю. В. Автоматизація роботи в органах державної податкової служби: Підручник / За заг. ред. В. М. Росоловського та С. П. Ріппи. – Ірпінь: ДПСУ, 2002. – 401 с.

195. Яцишин Ю.В. Інформаційні системи у податковому контролі й аудиті: Навч. посіб. / Ю.В. Яцишин. – Ірпінь: НАДПСУ, 2004. – 70 с.

196. Яцишин Ю.В. Аналітичні системи підтримки прийняття рішень для формування та супроводження державних цільових програм пріоритетних напрямів розвитку України / Ю.В. Яцишин // Проблеми інформатизації та управління: Збірник наукових праць. – Випуск 11. – К.: НАУ, 2004. – С. 84–88.

197. Яцишин Ю.В. Наукове видання «Fuzzy Technology: математичне й програмне забезпечення цільових програм у стратегічному менеджменті» / Ю.В.Яцишин, В.П.Бочарніков, С.В.Свешніков. – К.:Ніка-Центр, 2005. – 264с.

198. A. Strey A Unified Model for the Simulation of Artificial and Biology-Oriented Neural Networks / A. Strey // In Proc. of the International Workshop on Artificial Neural Networks. – 2, 1999. – pp. 1-10

199. A. Strey, EpsiloNN - A Tool for the Abstract Specification and Parallel Simulation of Neural Networks // Systems Analysis - Modelling - Simulation (SAMS), Gordon&Breach. – vol.34, n.4, 1999

200. Adeli H. Knowledge Engineering / H.Adeli. – McGraw-HillPublishing Company, N. Y, 1994. – 506p.

201. Advances in knowledge discovery and data mining / Fayyad U.M., Piatetsky-Shapiro G., Smyth P., Uthurusamy R. AAAI/MIT. – Press, 1996.

202. Assadi H. Knowledge aquisition from texts: using an automatic clustering method based on nounmodifier relationship. Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97), Madrid, Spain. 1997.

203. Aussenac-Gilles N., Natta N. Making the Method Solving Explicit with MACAO: the SIZYPHUS case-study in Sisyphus'92: Models of problem solving. – Ed. by M.Linster, GMD. – 1992. – P.213-218.

204. Batchilo L.S., Sovpel I.V., Tsourikov V.M. Computer based summarisation of natural language documents: US Patent Appl. №20030130837 – December, 2000. – P. 151-157.

205. Belgrave M. The Unified Agent Architecture: A White Paper, 1996. – P.5-9.

206. Benjamins V. R., Fensel D., et. alCommunity is Knowledge! // In KA2, In: Proc. KAW'98, Banff, Canada, 1998. – P.43-48.

207. Bertalanffy L. An Outline of General Systems Theory // British Journal of the Philosofy of Science. – 1950. – Vol. 1. – P.134-164.

208. Bisson G. Learning in Folwith a similarity measure. Tenth NationalConference on Artificial Intelligence (AAAI'92). – AAAI Press, San Jose, CA. – 1992. – pp. 82–87.

209. Boose J. H. ETS: a PCP – based program for building knowledge-based systems // Proc. WESTEX-86: IEEE West. Conf. Knowledge-Based Eng. And Expert Syst., Anaheim, Calif. June 24-26 1986. – Washington: D. C.

210. Boose J. H. Knowledge Acquisition Tools, Methods, and Mediating Representations // In Motoda H., Mizogochi R., Boose J., Gaines B. (Eds.) Knowledge Acquisition for Knowledge-Based Systems. IOS Press, Ohinsha Ltd., Tokyo, 1990.

211. Boose J. H., Bradshaw J. H., Shema D. B. Transforming repertory grids to shell-based knowledge bases using AQUINAS, a knowledge acquisition workbench // Proceedings of the AAAI-88 Integration of Knowledge Acquisition and Performance Systems Workshop. St. Paul.

212. Boose J.H. A Knowledge Acquisition Program for Expert Systems Based on Personal Construct Psychology // Int. Journal of Man-Machine Studies. – 1985. – Vol. 23. – P. 495-525.

213. Borghoff U., Pareschi R. Information Technology for Knowledge Management. – Springer-Verlag, Bin, 1998. – 237p.

214. Bosak J., 1997. XML, Java, and the future of the Web // Sun Microsystems. Режим доступу: http://sunsite.unc. edu/pub/sun-info/standards/xml/why/xmlapps.htm. – 16.10.2007 р. – Назва з титул. екрану.

215. Boulding K. L, 1956. General Systems Theory // The Skeleton of Science. Management Science. No. 2. P.197-208.

216. Braetman J. A., Magnini B., Rinaldi F. The Generalized Italian, German, English Upper Model // ECAI'94, Amsterdam, 1994. – P.56-63.

217. Breuker J. A., Wielinga B.J.,HaywardS.A., 1986. Structuring of knowledge-based systems development // ESPRIT"85: Status Report of Cont. Work. North-Holland. – P. 771-784.

218. Brooks, R. A. Intelligence without Representation // Artificial Intelligence, 1991. – 47. – P. 139-159.

219. Cane C., Sarson T. Structured System Analisis // Englewood Cliffs: Prentice-Hall, 1979. – 562p.

220. Conklin J. Hypertext: An Introduction and Survey // Computer, 1987. – Vol. 20, N.9. – P. 17-41.

221. Cook N. M. Computer Programming Expertise: Vanation of Cognitive Structure: vanations of cognitive structures in FT durso, chair, human expertise // Papers in Honor of W.Chaise, Meeting of Southern Psychological Association, 1985. – Austin: TX. – P. 73-89.

222. Cooperative ontologies programme (CO-ODE). Режим доступу: www.co-ode.org/ resources/ tutorials/ProtegeOWLTutorial.pdf. 2004. – 16.10.2007 р. – Назва з титул. екрану.

223. Courtois P. On Time and Space Decomposition of Complex Structures // Communinications of the ICM. – 1985. – Vol. 28. – N.6. – P.596-610.

224. Craven M., et al., Learning to extract symbolic knowledge from the world wide web // Artificial Intelligence. – N 118. – 2000. – P.69–113.

225. CYCorp. http://www.cyc.com. 2006. – 16.10.2007 р. – Назва з титул. екрану.

226. Darevych R.R. Modelling of the intelligent text recognition agents based on dynamic ontology / R.R.Darevych, D.G.Dosyn, V.V.Lytvyn // „Інтернет – Освіта – Наука - 2004", четверта міжнародна конференція. Збірник матеріалів конференції. – Том 2. – Вінниця: УНІВЕРСУМ- Вінниця, 2004. – С. 577-579.

227. Darevych R.R. The method of automatic defining of informative weight of concepts in a knowledge base ontology / R.R.Darevych, D.G.Dosyn, V.V.Lytvyn // Vidbir ta obrobka informaciji. 2005. – Vol. 22(98). – P. 146-148.

228. Davis R. TEIRESIAS: Applications of meta-level knowledge // Knowledge-based systems in Artificial Intelligence. N.Y.: McGraw-Hill, 1982. – P.346-348.

229. Diderich J., Ruhman I., May M. KRITON: a knowledge acquisition tool for expert systems // Int. Journal of Man-Machine Studies, 1987. – Vol. 26. – N.1. – P. 9-40.

230. E. Fiesler and H.J. Caulfield, "Neural network formalization", Computer Standards and Interfaces, 1994 – vol. 16(3), – pp. 231-239.

231. Eisenstadt M., Domingue J., Rajan T., Motta E. Visual Knowledge Engineering // IEEE Transactions on Software Engineering, 1990. – Vol. 16. – N.10. – P.1164-1177.

232. European Conference on Artificial Intelligence. Режим доступу: http://www.eccai.org/ ecai.shtml. – 16.10.2007 р. – Назва з титул. екрану.

233. Extensible Markup Language (XML) 1.0. W3C Recommendation. Режим доступу: www.w3.org/TR/1998/REC-xml-19980210. – 16.10.2007 р. – Назва з титул. екрану.

234. Farquhar A., Fikes R., Rice J. The Ontolingua Server: A Tool for Collaborative Ontology Construction // Knowledge Systems Laboratory, 1996. – KSL-96-26, September, 1996. – P. 346-353.

225. Faure D., Nedellec C., Rouveirol C. Acquisition of semantic knowledge using machine learning methods: the system ASIUM. Technical Report number ICS-TR-88-16. Laboratoire de Recherche en Informatique, Inference and Learning Group, Universite Paris, Sud. 19. – P. 79-84.

236. Fensel, D., Decker S., Erdmann M., Studer R. Ontobroker: How to enable intelligent access to the WWW // In AAAI-98 Workshop on AI and Information Integration. Madison, WI, 1998. – P. 56-64.

237. Fernandez-Lopez M. Overview of methodologies for building ontologies. In., Proceedings of IJCAI-99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends, in conjunction with the Sixteenth International Joint Conference on Artificial Intelligence, August, Stockholm, Sweden, 1999. – P. 231-237.

238. Fisher K. M., Faletti J., Patlerson H., Thornton R., Lipson J. & Spring, C., 1990. Computer assisted concept mapping // Journal of College Science Teaching. – No. 19 (6). – P. 347-352.

239. Furna G. W., Zacks J. Multitrees: Enriching and Reusing Hierarchical Structure // Human Factors in Computing Systems. Conference Proceedings. Boston, Ms, 1994. – P. 330-334

240. G. Dorffner, H. Wiklicky, and E. Prem, Formal neural network specification and its implications on standardization // Technical Report OFAI TR-93-24, Austrian Research. – P. 57-64.

241. G. Kock and N.B. Serbedzija, "Artificial neural networks: from compact descriptions to C++", In M. Marinaro and P.G. Morasso, editors, Proc. of the International Conference on Artificial Neural Networks, Springer-Verlag, 1994. – pp. 1372-1375

242. G. Kock and N.B. Serbedzija, "Simulation of Artificial Neural Networks", Systems Analysis - Modelling - Simulation (SAMS). – vol. 27(1). – 1996. – pp. 15-59

243. Gaines B. R., Shaw M. L. G. KITTEN: Knowledge initiation and transfer tools for experts and novices // International Journal of Man-Machine Studies. – 1987. – Vol. 27. – N 3. – P. 251-280.

244. Gammack J. G., Young R. M. Psychological Techniques for Eliciting Expert Knowledge // Research and Development in Expert Systems. Cambridge: University Press, 1985. – P. 68-76.

245. George F. Luger, William A. Stubblefield Artificial Intelligence. Structures and Strategies for Complex Problem Solving. – Addison Wesley Longman, Inc, 1999. – 826p.

246. Goldsmith T. E, Johnson P. J., & Acton W. H. Assessing structural knowledge //Journal of Educational Psychology, 83. – 1991. – P. 88-96.

247. Goldsmith T. E., Schvaneveldt R. W. ACES: Air combat Expert Simulation // Memo. In Comput. And Cogn. Sci., MCCS-85-34, Comput. Res. Lab. New Mexico: State Univ. – 1985. – P. 81-87.

248. Gruber T. R. A translation approach to portable ontologies // Knowledge Acquisition. – 1993. – N 5 (2). – P. 199-220.

249. Gruninger M., Fox M. Methodology for the Design and Evaluation of Ontologies // Proceedings of IJCAI-95 Workshop on Basic Ontological Issues in Knowledge Sharing. – 1995. – P. 231-238.

250. Guarino N. Some Ontological Principles for Designing Upper Level Lexical Resources. Proc. of the First International Conference on Lexical Resources and Evaluation, Granada, Spain, 28-30 May 1998. – P. 79-86.

251. Guarino N., 1996. Ontologies: What Are They, and Where's The Research? // A panel held at KR'96, the Fifth International Conference on Principles of Knowledge Representation and Reasoning, November 5, 1996, Cambridge, Massachusetts. – P. 77-82.

252. Guarino N., Giaretta P.. Ontologies and Knowledge Bases. Towards a Terminological Clarification // In: Towards Very Large Knowledge Bases. – 1995 – N.J.I. Mars (ed.), IOS Press, Amsterdam. – P. 11-19.

253. H. Hopp and L. Prechelt, "CuPit-2: A portable parallel programming language for artificial neural networks // In A. Sydow, editor, Proc. 15th IMACS World Congress on Scientific Computation, Modelling, and Applied Mathematics, Wissenschaft & Technik Verlag. – P. 167-178.

254. Han J., Kamber M. Data Mining: concepts and techniques. – London.: Academic Press, 2001. – 307p.

255. Han J., Kamber M. Data mining: methods and technique. – Morgan Kaufman, 2000. – 298p.

256. Hecht-Nilsen, R. Neurocomputing. – Addison-Wesley, 1990. – 366p.

257. Heflin J., Hendler J., Luke S. Reading Between the Lines: Using SHOE to Discover Implicit Knowledge from the Web // In AAAI-98 Workshop on AI and Information Integration. – P. 121-126.

258. Hinkelmann K. and Kieninger Th. Task-oriented web-search refinement , and information filtering. DFKI GmbH. – 1997. – 237p.

259. http://www.cs.umbc.edu/kqml/. – 16.10.2007 р. – Назва з титул. екрану.

260. http://www.intuit.ru/department/human/isrob/popup.lit.html#91. – 16.10.2007 р. – Назва з титул. екрану.

261. http://www.strout.net/python/tidbits.html. – 16.10.2007 р. – Назва з титул. екрану.

262. Hwang, C.H. Incompletely and imprecisely speaking: Using dynamic ontologies for representing and retrieving information. In Proceedings of the 6th International Workshop on Knowledge Representation meets Databases, Linköping, Sweden, July 29-30. – 1999. – P. 209-215.

263. Jonassen D.H. Changes in knowledge structures from building semantic net versus production rule representation of subject content. Journal of Computer Based Instruction. – 1993. – 20 (4). – P. 99-109.

264. Jones A. The Object Model: a Conceptual Tool for Structuring Software. Operating Systems. N. Y.: 1992. – Springer-Verlag. – P.8.

265. Kahn G., Nowlan S., McDermott J. MORE: An Intelligent Knowledge Acquisition Tool // The proceedings of the Ninth Joint Conference on Artificial Intelligence. – 1985. – August. Los Angeles, CA. – P.581-584.

266. Kelly G.A. The Psychology of Personal Constructs. – N. Y.: Norton, 1955. – 493p.

267. Khoroshevsky V. F., Knowledge V.S. Data Spaces: How an Applied Semiotics to Work on Web // In: Proceedings 3rd Workshop on Applied Semiotics, National Conference with International Participation (CAI'98), Pushchino, Russia, 1998. – P. 57-59.

268. Kifer M., Lausen G., Wu J. Logical Foundations of Object-Oriented and Frame-Based Languages // Journal of the ACM, 1995. – P. 67-73.

269. Kozma R. B. Constructing knowledge with learning tool. In P. Kommers, D. Jonassen, A T. Mayes (Eds.), Cognitive tools for learning. – Berlin: Springer-Verlag, 1992. – P. 23-32

270. Kruskal J. B. Nonmetric multidimensional scaling: a numerical method, Psychometrika, 1964. – 29. – P. 115-129.

271. Kuehn O., Abecker A. Corporate Memories for Knowledge Management in Industrial Practice: Prospects and Challenges, 1998. – 189p.

272. L. Smith, Using a framework to specify a network of temporal neurons, Technical Report, University of Stirling, 1996. – 289p.

273. Lai K. W. Acquiring expertise and cognitive skills in the process of constructing an expert system: A preliminary study // Paper presented at the annual meeting of the American Educational Research Association, San Francisco. CA, 1989. – P. 256-261.

274. Lamping L, Rao R., Pirolli P. A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies // In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems. – 1995. – P. 108-117.

275. Lenat D.B., Guha R.V., Building Large Knowledge Based Systems. Representation and Inference in the CYC Project. Addison-Wesley, Reading, MA. – 1990. – P. 34-41.

276. Lenat, D. B. CYC: A Large-Scale Investment in Knowledge Infrastructure // Communications of the ACM, 1995. – 38. – № 11. – P. 67-70.

277. Lesser V., Horning B., Klassner F., Raja A., Wagner T., Zhang S. A Next Generation Information Gathering Agent // Umass Computer Science Technical Report 1998-72, May, 1998. V P. 45-50.

278. Lippert R.C. An expert system shell to teach problem solving. Teach Trends. – 1988. – 33(2). – P. 22-26.

279. M. A. Atencia, G. Joya and F. Sandoval, "A formal model for definition and simulation of generic neural networks", Neural Processing Letters, Kluwer Academic Publishers. – vol. 11. – 2000. – pp. 87-105.

280. Macintosh A. Knowledge asset management // Airing. – 1997. – N.20. – P.3-10.

281. Maedche A., Staab S. Ontology learning for the semantic web // IEEE Intelligent Systems 16 (2). – 2001. – P. 72–79.

282. Maedche, A. & Staab, S. Discovering conceptual relations from text. In, ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence, IOS Press, Amsterdam, 2000. – P. 35-41.

283. Maikevich N. V., Khoroshevsky V. F. Intelligent Processing of Web Resources: Ontology-Based Approach and Multiagent Support// Ift: Proceedings of 1st International Workshop of Central and Eastern Europe on Multi-agent Systems (CEEMAS'99). – 1999. – 1-4 June. – St.-Petersburg, Russia. – P.15-19.

284. Mehrnoush Shamsfard, Ahmad Abdollahzadeh Barforoush: Learning ontologies from natural language texts. Int. J. Hum.-Comput. Stud. 60(1): 2004. – pp.17-63

285. Minsky M.A. Theory of memory // Perspectives in Cognitive Science, 1981. – Norwod. – N. S., Ablex. – P. 23-27.

286. MonahanJ.S., Lockhead G.R.. Identification of integral stimuli //Journal off Experimental Psychology. – 1977. – General 106. – P. 94-110.

287. Musen, M. A. Automated support for building and extending expert models // Machine Learning, 4. – 1989. – pp.347-376.

288. Natalya F. Noy and Deborah L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology'. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001. – P. 87-90.

289. Nielsen J. HyperText & HyperMedia. Academ. Press Inc, 1990. – 461p.

290. Nierenburg S., Beale S., Mahesh K., Onyshkevych B., Raskin V., Viegas E., Wilks Y., Zajac R. Lexicons in the mikrokosmos project. Proceedings of AISB Workshop on Multilinguality in the Lexicon. – Brighton, UK, 1996. – pp. 26–33.

291. Nonaka I. and Takeuchi LThe Knowledge-Creating Company. N. Y.: Oxford: Oxford University Press, 1995. – 309p.

292. Olson J. R., Reuter H. H. Extracting expertise from experts: methods for knowledge acquisition // Expert systems. – 1987. – Vol. 4. – N 3. – P. 31-37.

293. Piatetsky-Shapiro G. and Frawley W., eds. Knowledge Discovery in Databases. AAAI/ MIT Press, 1991. – 211p.

294. Quinlan J. Induction of decision trees // Machine Learning, 1. – 1986. – P.81-106.

295. R.R. Leighton, The Aspirin/MIGRANES neural network software // Users manual, MITRE Corp., 1992. – 143p.

296. Rabbits L., Wright G. Card tricks for small players // Expert Systems User. – 1987. – Vol. 22. – N 11. – P. 22-27.

297. Reich J.R. Ontological Design Patterns for the Integration of Molecular Biological Information // In Proceedings of the German Conference on Bioinformatics GCB'99. Hannover, Germany, 1999. – pp.156-166

298. Reisman S. Developing Multimedia Applications // IEEE Computer Graphics & Applications. – 1991. – P. 58-66.

299. Resource Description Framework (RDF). Режим доступу: http://www.w3.org/RDF/. – 16.10.2007 р. – Назва з титул. екрану.

300. Ricoeur P. La metaphore vive. – Paris, Editions du Seuil, 1975. – 321p.

301. Sarle W. Frequent asked question on neural network. Режим доступу: ftp.sas.com/pub/neural/FAQ.html. – 16.10.2007 р. – Назва з титул. екрану.

302. Schoucksmith G. Assessment through interviewers. – Oxford, 1978. – 289p.

303. Schreiber G., Breuker J., Bredeweg B., Wielinga B. Modeling in KBS Development // Proceedings of the Second European Knowledge Acquisition Workshop. – 1988. (EKAW-88). – Bonn. – P.1-15.

304. Semantic Web Community Portal. http://www.semanticweb.org/. – 16.10.2007 р. – Назва з титул. екрану.

305. Sheng-Tun Li, Huang-Chih Hsieh, I-Wei Sun: An Ontology-based Knowledge Management System for the Metal Industry // WWW-2003 (Alternate Paper Tracks). Budapest, Hungary, 2003. – P. 276-284.

306. Shortliffe E. Computer based medical consultations: MYCIN. – N. Y., American Elsevier, 1976. – 278p.

307. Stuart J. Russel, Peter Norvig Artificial Intelligence // A modern approach. Williams Publishing House. – 2006. – P. 779-784.

308. Tesauro, G. Temporal difference learning and td-gammon // Communications of the ICM. –1995. – 38(3). – P.58–68.

309. The Data Mining Group, PMML 1.1 Neural Network. Режим доступу: www.dmg.org/html/neuralnetwork.html. – 16.10.2007 р. – Назва з титул. екрану.

310. Torgerson W. S. Theory and methods of scaling. N. Y.: Wiley, 1958. – 321p.

311. TOVE, 1999. TOVE Manual. - Department of Industrial Engineering, University of Toronto. Режим доступу: http://www.ie/utoronto.ca/EIL/toye. – 16.10.2007 р. – Назва з титул. екрану.

312. Tschaitschian B., Abecker A. and SchmalhoferA. Putting Knowledge Into Action: Information Tuning With KARAT // In 10th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW-97). – 1997. – P. 34-39.

313. Turing A. Computing machinery and intelligence. Mind. – 59. – P. 433-460.

314. Uschold M. Creating, integrating and maintaining local and global ontologies. In. Proceedings of the First Workshop on Ontology Learning (OL-2000) in conjunction with the 14th European Conference on AI (ECAI 2000). August, Berlin, Germany, 2000. – P. 12-18.

315. Uschold M., Gruninger M. Ontologies: principles, methods, and applications // Knowledge Engineering Review. – 11(2). – 1996. – P.93-155.

316. Van der Vet P.E., Mars N.J., Speel P.H. The Plinius Ontology of Ceramic Materials // ECA94, Amsterdam, 1994. – P.12-19.

317. Voinov A., Gavrilova T. Knowledge Acquisition through Elicitation of Latent Cognitive Structures: Metaphor – Based Approach // Proc. of East-West International Conference on Artificial Intelligence: From Theory to Practice EWAIC'93. Moscow. – 1993. – P. 113-119.

318. Wagner A. Enriching a lexical semantic net with selectional preferences by means of statistical corpus analysis // Proceedings of ECAI'2000 Workshop on Ontology Learning (OL'2000). – Berlin. – Germany. – 2000. – P. 234-246.

319. Wang Hui-jin, Hu Hua, Li Qing. A dynamic knowledge base based search engine. Journal of Zhejiang University Science. – 6A(7). – 2005. – pp. 683-688.

320. World Wide Web Consortium. Режим доступу: http://www.w3.org/2004/OWL. 2004. – 16.10.2007 р. – Назва з титул. екрану.

321. Y. Miyata, A User's Guide to PlaNet Version 5.6 // A Tool for Constructing, Running, and Looking in to a PDP Network, Computer Science Department. – University of Colorado, Boulder. – 1991. – P. 452-458.

322. Zelle M., Mooney R.J. Learning semantic grammars with constructive inductive logic programming // Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93). – Washington. – 1993. – pp. 817–822.

Предметний покажчик

Список скорочень

DAI – Distributed Artificial Intelligence
DTD – document type definition
KIF – Knowledge Interchange Format
KM – Knowledge Management
OMIS – Organizational Memory Information Systems
АІС – автоматизована інформаційна система
АНС – автоматизовані навчальні системи
БГ – багатовимірне градуювання
БД – база даних
БЗ – база знань
ГТ – гіпертекст
ЕДС – експертні діагностичні системи
ЕС – експертна система
ЕСО – експертні системи-оболонки
ІК – індуктивна компонента
ІО – інформаційний образ
КІМ – корпоративна інформаційна мережа
МАС – мультиагентні системи
ММ – мультимедіа
МП – машинний переклад
МПЗ – мова подання знань
МПО – модель предметної області
ОД – об'єкт діагностування
ООП – об'єктно-орієнтований підхід
ОСА – об'єктно-структурний аналіз
ОСП – об'єктно-структурний підхід
ОСР – об'єктно-структурне розроблення
ПЗ – програмні засоби
ПО – предметна область
СБР – система булевих рівнянь
СКБД – системи керування базами даних
СОЗ – система одержання знань
СШІ – системи штучного інтелекту
УПК – універсальний предметний код
ЦП – цифровий пристрій
ШІ – штучний інтелект

НАВЧАЛЬНЕ ВИДАННЯ

Литвин Василь Володимирович
Пасічник Володимир Володимирович
Яцишин Юрій Володимирович

Інтелектуальні системи

Підручник

Керівник видавничого проекту: *В.М. Піча*
Комп'ютерна верстка: *Кочетов І.*
Формат 70×100 1/16. Папір офсетний.
Підписано до друку з оргінал-макета 4.11.2008
Умовн. друк. арк. 25. Зам. № 64.
Гарнітура Таймс Нью Роман. Тираж 1000 прим.

НБ ПНУС
748425

Видавництво ПП "Новий Світ-2000"
а/с 2623, м. Львів-60, 79060, Україна,
E-mail: novyisvit2000@lviv.farlep.net.

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців і розповсюджувачів видавничої продукції: серія ДК № 59 від 25.05.2000 р., видане Державним комітетом інформаційної політики, телебачення та радіомовлення України

Віддруковано в друкарні видавництва ПП "Новий Світ – 2000"
Львів, вул. В. Великого, 4

СИСТЕМНИЙ АНАЛІЗ ОБ`ЄКТІВ ТА ПРОЦЕСІВ КОМП`ЮТЕРИЗАЦІЇ
Навчальний посібник
Катренко А.В.
2008. – 424 с. Тв. обкл.

Рекомендовано Міністерством освіти і науки
ISBN 966-7827-21-6

У навчальному посібнику викладено основні поняття та методологію системного аналізу. Зокрема такі методи системного аналізу, як метод аналізу ієрархій, метод дерева цілей, комбінаторно-морфологічні методи, методи функціонально-вартісного аналізу та аналізу процесів функціонування систем. Розглянуто зв'язок системного аналізу з моделюванням, застосування методологій системного аналізу у процесі створення інформаційних систем, методи отримання інформації від експертів та інші, що широко використовуються у практиці системного аналізу. Подано лекційний матеріал, наведено приклади застосування методів системного аналізу. З метою закріплення матеріалу до кожної теми подано перелік питань, а також завдання для самостійного виконання та тести. Розрахований для бакалаврів, спеціалістів та магістрів. Ним можуть скористатися аспіранти та викладачі комп'ютерних наук ВНЗ, практичні працівники і спеціалісти.

СТИСЛИЙ ЗМІСТ