Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна

В. М. КУКЛІН

ПОДАННЯ ЗНАНЬ І ОПЕРАЦІЇ НАД НИМИ

Навчальний посібник

Харків – 2019

УДК 004.889
К 89

Рецензенти:

В. М. Корчинський – доктор технічних наук, професор, завідувач кафедри електронних засобів телекомунікації Дніпровського національного університету імені Олеся Гончара;

С. І. Шматков – доктор технічних наук, професор, завідувач кафедри прикладної та теоретичної системотехніки Харківського національного університету імені В. Н. Каразіна.

Затверджено до друку рішенням Вченої ради
Харківського національного університету імені В. Н. Каразіна
(протокол № 6 від 27 травня 2019 року)

Куклін В. М.

К 89 Подання знань і операції над ними : навчальний посібник / В. М. Куклін. – Харків : ХНУ імені В. Н. Каразіна, 2019. – 164 с.

ISBN 978-966-285-606-4

У книзі викладено значною мірою розвинені в роботах попередників різні способи подання знань в глобальних базах даних комутативних і некомутативних систем продукцій. Розглянуто формалізовані операції для отримання нових рішень, зокрема способи розширення баз даних, що відповідає процедурам навчання. Основна увага приділяється методам формування на основі математики і математичної логіки мов штучного інтелекту, таких як Пролог і Лісп. Обговорюються особливості застосування нечіткої логіки для створення алгоритмів і штучних нейронних мереж. Представлені технології створення планів для роботів з урахуванням конфліктів цілей. Обговорюються проблеми опису і реалізації семантичних мереж і семантичної павутини.

Цікавим є як навчальний посібник для аспірантів і студентів факультетів природничо-наукового профілю та комп'ютерних наук, які вивчають системи штучного інтелекту.

УДК 004.889

ISBN 978-966-285-606-4

ЗМІСТ

ПЕРЕДМОВА

Матеріал, поданий в цій книзі, зібраний з різних джерел є початковим курсом лекцій, практичних занять та набором питань для засвоєння знань студентами. Цей курс читався протягом тривалого часу на факультеті комп'ютерних наук в Харківському національному університеті імені В. Н. Каразіна. Автор книги висловлює свою подяку першому декану ФКН, доценту О. Ф. Целуйко, який визначив напрямок розвитку кафедри штучного інтелекту та програмного забезпечення, декану механіко-математичного факультету професору Г. М. Жолткевичу, який відчутно допоміг у становленні перших навчальних курсів зі штучного інтелекту, професору кафедри В. В. Яновському, який за допомогою А. В. Приймака організував належну наукову роботу на кафедрі, що відповідає цим напрямкам, моїм молодим колегам доценту Є. В. Бєлкіну, О. С. Петренко, І. В. Гущину, О. В. Мішину та заступнику завідувача кафедри доценту О. Є. Спорову, які взяли кожний в свій час на себе обовя'зок читання лекцій і проведення лабораторних робіт з курсів штучного інтелекту. Також автор висловлює свою подяку рецензентам і своїм колегам за увагу і підтримку, особливо знавцеві української версії IT термінології А. М. Горбаню. Слід віддати належне якісній роботі директора університетського видавництва І. М. Дончик та її колег О. В. Пікаловій та В. В. Савінковій, які активно сприяли виданню цієї книги.

ВСТУП[1]

Перш ніж обговорювати подання знань і операції з їх участю, слід визначити походження і характер цих знань, усвідомити, якими знаннями ми можемо оперувати.

> ***Знання неформалізовані*** – не сформульовані у вигляді теорій і алгоритмів.[2] Процес мислення пов'язаний з безперервним порівнянням отриманих від органів чуття даних з гігантськими запасами інформації про навколишній світ, накопиченими протягом життя індивідуума. Досвід пережитих в реальності і в уяві ситуацій разом з відпрацьованими рефлексіями формують негайну адекватну[3] реакцію, всі види якої також зберігаються в інформаційній системі людини. Усвідомлення явищ на рівні рефлексії швидше інтуїтивне, причому саме практика – удачі, а більш за все, невдачі – формує емпіричний підбір відповідних дій.

Друга сигнальна система, що дісталася нам у спадок від предків, які пережили довгий і важкий шлях еволюції, дозволяє формалізувати деякий, правда, вельми незначний обсяг знань про оточення. І дає можливість, оперуючи формальними образами, уявити прогноз поведінки або схему розв'язання різних задач (прийняття рішень), які постають перед людиною.

> ***Знання формалізовані.*** Формалізоване знання, сформульоване у виявлених закономірностях, в кількісних критеріях, в методах опису, в технологіях пошуку рішень і представлене в книгах, в підручниках, на відео- і аудіоносіях ні в яке порівняння не може йти за обсягом інформації з гігантськими обсягами неформалізованого, інтуїтивного-емпіричного знання, що збирається в результаті життєвого досвіду і спілкування в соціумі.

Формалізація даних і знань вимагає великих зусиль. Ускладнюється вона і тим, що частина цих даних і знань людина не бере до уваги, вважаючи їх відомими або очевидними, за замовчуванням. Цікаво, що сама процедура формалізації інтуїтивно зрозумілих знань здатна прояснити і супутні проблеми, і підказати нові рішення.

> ***Емпіричні знання.*** До добре формалізованих, представлених в теоріях, законах і правилах знань примикають знання недостатньо добре формалізовані, емпіричні, в масиві яких тільки належить виявити закономірності. Повністю не формалізовані або не цілком формалізовані знання не можуть претендувати на строгість і страждають невизначеністю, несистемністю і суперечливістю. Однак саме емпіричні, не цілком формалізовані знання і формують те, що прийнято називати «know-how», вони є попереднім етапом процесу усвідомлення проблем. Цей необхідний в еволюції свідомості крок випереджає більш конструктивний етап формалізації знань[4].

---

[1] У вступі використані матеріали книги [1].

[2] Детальніше див. змістовну книгу [2]), де автор першим із західних дослідників розглянув роль неформалізованих форм існування і передачі знань. Знань, які ним були названі неявними (tacit knowlege), прихованими, імпліцитними.

[3] Див., наприклад,. J. Hawkins and S. Blakeslee, On intelligence [3].

[4] Але не треба переоцінювати і якість на перший погляд серйозних теоретичних побудов, бо вони часто засновані на неточних постулатах і переоцінених умоглядних закономірностях

Подібні знання з'являються в результаті осмислення людиною процесів і явищ, накопичуються при спілкуванні і при різноманітній діяльності. Розширення меж усвідомленого світу може відбуватися завдяки виявленій можливості сформулювати у вигляді загальних закономірностей і правил (формалізувати) частину емпіричних (тобто отриманих безпосередньо з досвіду, недостатньо формалізованих, хоча і частково структурованих, представлених в базах даних) знань.

Пошуки відповідей. Як зазначено М. М. Клайном [4], «існує багато типів міркувань, наприклад, міркування по індукції, по аналогії і дедукції». На практиці часто застосовують методи розв'язання, використовуючи вже існуючі дані, тобто індукцію і аналогії, хоча математики вважають, що це ненадійніший метод, що не забезпечує досить бажану ними однозначність виводу. Надійніше, вважають вони, використовувати систему постулатів і аксіом для підтвердження теорем або правила дедуктивної логіки[5], виділені Арістотелем з практики доказів[6].

За аналогією, тобто «якщо зазвичай відомо з практики, що малина червона, то і отримана на замовлення повинна бути червоною».

За індукцією – «якщо батькам вдалося вступити до університету, значить і їх дітям це буде можливо».

Дедукція (правила логіки – силогістичні висновки за Арістотелем): «якщо всі люди смертні, то і Сократ смертний». Арістотель відносив до дедукції закон суперечності (одночасно істинним і хибним висловлювання бути не може) і закон виключення третього (вислів або істинний, або хибний). Застосування до посилок дедукції гарантує отримання висновків, які не поступаються в надійності посилкам[7].

Парадокс Дойча. Поява все більшого обсягу формалізованих знань мала б привести людину в зневіру, якби не зазначений Д. Дойчем парадокс [5], який полягає в тому, що більш пізні наукові теорії охоплюють все більшу кількість явищ і простіше засвоюються вченими-початківцями. Зібране в апробованих теоріях і сформульоване в законах формалізоване знання інтегрує і об'єднує в компактну і зручну для освоєння систему вражаючий досвід цивілізації, дає інструментарій для підтримки і розвитку технологічного і соціального укладу життя планети.

---

[5] Арістотель ввів поняття силогістичного виведення, ввів вимогу, щоб кожен вислів був або істинним, або хибним, ввів в практику логічних побудов розуміння того, що висновок повинен бути настільки ж вірним, як і посилки, з яких його було отримано.

[6] Хоча логічні побудови в міркуваннях попередників вже існували, наприклад, визнаний вчитель Арістотеля Платон використовував метод «доказів від протилежного», де в істинності твердження переконуються при виявленні явної невідповідності й суперечності обраним посилкам.

[7] Емпіричні, отримані з досвіду дані перевіряються багаторазовими повтореннями, і доказ їх правдивості побудовано на міркуваннях за аналогією.

Критерії відповідності вимогам істинності. Це критерій науковості Поппера, принцип верифікації і принцип Оккама.

1. Критерій Поппера – теорія є науковою (такою, що може фальсифікуватись), якщо існує методологічна можливість її спростування, зокрема шляхом постановки експерименту.

2. Принцип верифікації вимагає визначення достовірності, точності та обґрунтованості прогнозів.

3. Принцип Оккама рекомендує не вводити в розгляд нові сутності без крайньої необхідності.

Основними проблемами машинного навчання будуть або формалізація всього масиву необхідних знань для експертних систем і систем логічного висновку, або створення ефективного інтерфейсу для взаємодії з нейронними мережами, здатними освоювати знання в «сирому», неформалізованому вигляді. У першому випадку подібним діям передує трудомістка формалізація всіх освоєних людьми інтуїтивних, неформалізованих і емпіричних знань, що на сьогодні є непідйомним завданням. А в другому випадку подібний інтерфейс для інтерактивної взаємодії людей і пристроїв з нейронними мережами майбутнього вимагає пошуку інших підходів до технологій освоєння знань машинами[8].

Тому так важливо вивчити всі відомі методи представлення знань, способи їх обробки і отримання нових знань.

[8] Якщо окремій людині для освоєння всього масиву знань потрібні десятиліття, то для нейронних мереж майбутнього, які будуть пов’язані з великою кількістю людей, а також з безліччю приладів і пристроїв, можливо, знадобиться істотно менші терміни навчання. Тобто великі терміни набору знань, як у людини, можуть бути певною мірою компенсовані багатоканальною взаємодією зі значною кількістю людей і приладів.

РОЗДІЛ 1
ПРОСТІ ЕКСПЕРТНІ СИСТЕМИ[9]

1.1. Основні уявлення

Природна мова в більшості своїй містить конструкції, які, крім простих логічних посилок, є також ключами для ініціалізації існуючого у кожної людини цілого спектру понять і логічних структур.

Фрази природної мови здатні викликати відлуння пережитих і зафіксованих раніше в пам'яті відчуттів, можуть формувати цілісні чуттєві уявлення. При цьому ми тут не обговорюємо зовсім інтонацію, жести, вирази обличчя мовця. На перший погляд складається враження, що спроби створити аналог природної мови формальними методами приречені на провал. Великі труднощі виникають при формальному кодуванні і декодуванні фраз природної мови.

Формалізація

Формалізація даних і знань вимагає великих зусиль. Ускладнюється вона і тим, що частину цих даних і знань людина не бере до уваги, вважаючи їх відомими або очевидними, за замовчуванням.

Проблеми конструювання інтелектуальної системи:

– Труднощі розуміння запитів на природній мові (але обійти можна, визначивши зрозумілу машині мову запитів [6]).

Треба навчати вмінню задавати питання

Людина теж часто не знає, що запитати, чи не здатна навіть сформулювати питання, якщо її попередньо не підготували до розмови, якщо заздалегідь не окреслили контури обговорюваної проблеми.

– труднощі логічного виводу відповіді, виходячи з наявної в базі інформації;

– для розуміння запиту і виведення висновку може знадобитися інформація, якої немає в базі (цю інформацію експерти могли опустити, вважаючи її загальновідомою).

Консультуючі експертні системи – системи діагностики, системи, здатні формулювати висновки в певній предметній області. Для цього використовують знання експертів. Ці знання часто є неповними, неточними і досить невизначеними (інтуїтивними) а, часом, на перший погляд і нелогічними.

---

[9] Для ознайомлення з принципами створення інтелектуальних систем можна обмежитись розділом 1

Види завдань

Доказ теорем. Цей клас інтелектуальних завдань заснований на дедукції на підставі гіпотез, тут потрібні інтуїтивні навички (що з'являються, на жаль, тільки з досвідом) вибору проміжних лем.

Конструктивні теореми

При доведенні констуктивних теорем видно не тільки шлях розв'язання а й сам розв'язок: Так звані конструктивні теореми корисні тим, що після їх доведення видно шлях розв'язання, сформульовані логічні зв'язки (леми), вказані значення параметрів і змінних. Власне, саме при виконанні цих умов подібні теореми можуть бути названі конструктивними. Їх використання може виявитися корисним на практиці.

Роботика (robotics) − наукова дисципліна про організацію доцільної поведінки робота з сенсорними і виконавчими (ефекторними) механізмами. Розгляд побудовано на відображенні станів (світу) і опису процесів зміни цих станів. При цьому створюються плани цих змін, плани послідовності дій, методи управління планами. Плани мають різні рівні деталізації.

Автоматичне програмування (приклад – компілятори, що отримують специфікацію на вхідній мові і генерують програму на об'єктній мові). У системах штучного інтелекту (Ш І) – це опис на високому рівні деякого твердження (зокрема, точного на формальній мові – обчислення предикатів, – або не точного на природній мові, тоді потрібно передбачити уточнюючі запити), яке завдяки автоматичному програмуванню буде сприйняте машиною. Супутні завдання – *верифікація програми* – доказ того, що система досягла результату. *Налагодження* – стратегія всієї (повної) програми.

Логічні операції – основа систем штучного інтелекту

Введення великого числа логічних операцій, що дозволяють при їх застосуванні машині фактично вибирати шлях вирішення, в якійсь мірі наближає програму до систем штучного інтелекту. Взагалі кажучи, перехід від жорсткої алгоритмічної структури до систем логічного висновку лежить спочатку в заміні множини традиційних математичних і конструкторських операцій з фактами і відносинами на логічні і потім в поступовій відмові від жорсткої послідовності їх виконання.

Пошук оптимальних розкладів (наприклад, задача комівояжера – пошук мінімальної довжини маршруту, не відвідуючи міст більше одного разу або пошук шляху з мінімальними витратами по дугах графа, що містить множину вершин, які не можна відвідувати більше одного разу). До цього класу задач відносять задачу розміщення об'єктів на певних умовах. Прості спроби вирішення (перебором) породжують ***комбінаторний вибух*** – вибухове зростання варіантів рішення, які вичерпують можливості обчислювальної системи. ***Складність задачі з ростом її обсягу*** може рости лінійно, степеневим чином (поліноміальним чином), експоненціально (наприклад, для NP -повних задач) і навіть швидше – вибуховим чином ($\propto 1/(V - V_0)$).

NP **-задачі** – це задачі з т. зв. невизначеними знаннями, які не забезпечені евристичними правилами, спрощуючими схемами, що спонукає до розв'язання методами перебору варіантів.

Неможливість універсальної схеми вирішувача

Не можна вирішувати взагалі, кожна задача вимагає свого підходу, своєї евристики, своїх припущень і пріоритетів щодо вибору рішення, які спираються на власний досвід і досвід чужий – залучений (вивчений і усвідомлений). Звідси наслідок – не можна придумати загальну технологію рішення логічних задач принципово. Можна лише сформулювати процедури та інтегрувати їх в методи з обмеженим застосуванням. А потім подумати, як їх раціонально використовувати в конкретному випадку.

Експертні системи з недостатньо визначеними знаннями

Чотири важливі проблеми, які виникають при проектуванні і створенні ЕС з недостатньо визначеними (неточними) знаннями:

• Як кількісно виразити ступінь визначеності при встановленні істинності (або хибності) деякої частини даних?

• Як висловити ступінь підтримки висновку конкретною посилкою?

• Як використовувати спільно дві (або більше) посилки, які незалежно впливають на висновок?

• Як бути в ситуації, коли потрібно обговорити ланцюжок виведення для підтвердження висновку в умовах невизначеності?

Структури

Глобальна база даних – база даних та знань експертної системи.

Система продукцій – взаємодія 3 елементів інтелектуальної системи – бази даних, множини правил продукції та системи управління. ***Модульність*** – зміни в базах даних, наборах правил і системі управління можуть бути незалежними. Елементами інтелектуальної системи є бази даних (знань), що дозволяють представляти знання в ефективному вигляді, забезпечувати введення, зберігання, видобування цих знань. У низці випадків необхідно для відповіді на запит провести деякі дедуктивні (від загального, тобто від аксіом і правил, до конкретного) операції з фактами з цих баз даних.

Зміна бази знань (фактів і правил продукції) – основа навчання в експертних системах. Це вже активно використовують. Для більшої гнучкості треба б змінювати і саму систему управління – але зазвичай вона задана.

Правило продукції створюється на базі узагальнення обчислювального формалізму; існує попередня умова, якій база даних задовольняє; використання (застосування) правила, взагалі кажучи, змінює базу даних; використання (застосування) правила відбувається на основі прийняття рішення системою управління; самостійно між собою правила не взаємодіють (не «викликають» один одного). ***Термінальна умова*** – умова зупинки.

Декларативні знання – дані, що включають в себе факти, ***процедурні*** – представлені в правилах, ***управляючі*** – представлені в стратегії управління.

Основна проблема Ш І – грунтуючись на задачі, організувати декларативну, процедурну і управляючу компоненти для використання в системах продукції.

Відображення станів, ходів і цільових умов. Основне завдання – знайти найбільш оптимальне представлення. Це і найменша кількість (значущих) станів, і, по можливості, обмеження можливих ходів, і спрощення процедур управління. З цього, власне кажучи, слід починати. Хоча оптимізація відображень – більше мистецтво, ніж технології.

Обчислювальні витрати експертних систем складаються з: 1) витрат на застосування правил і 2) витрат на управління. Низька інформованість про задачу призводить до невеликих витрат на управління, однак до великих витрат на перебір правил.

Ентузіазм чи кваліфікація?

У людини перший випадок – це метод «мало усвідомленого перебору», працює ефективно коли багато часу і здоров'я у виконавця, який стрімко перебирає варіанти вирішення. Другий випадок – висока кваліфікація (на досягнення якої було раніше витрачено багато часу та зусиль), але розв'язок знаходиться швидко.

Висока інформованість про завдання призводить до витратної (і дуже дорогої) системи управління (з великим об'ємом пам'яті і необхідністю громіздких обчислень), але економить на втратах при застосуванні правил, розв'язки знаходяться швидко.

Як ми розпізнаємо образи?

Розпізнавання у людини відбувається на основі спільної дії обчисленних і різних, не цілком певних ознак, які з високою надійністю дають відповідь (причому тільки всі разом). Зазвичай в пам'яті людини і при його спостереженнях є скінченне число N об'єктів, з якими може бути проведено порівняння. Задача формулюється так: як за неповною, неточною інформацією за M ознаками серед N об'єктів виділити один об'єкт. Взагалі кажучи, чи існує критерій на рівень неповноти інформації і цих значень (ознак і об'єктів), що дозволяє з упевненістю вважати, що вибір є однозначним. До речі, китайці, які вперше опинилися серед маси європейців, вважають, що європейці всі на одне лице. Не запам'ятовують, точніше не впізнають раніше баченої людини [3]. Цікавий ефект «de ja vu» – це відлуння напівузнавання, щось в спостережуваному подібне раніше побаченому.

Стратегії управління

Загальні підходи [7] (див. також [8]). Для систем продукції стратегії управління – процес пошуку, де правила використовуються і перевіряються, поки деяка їх послідовність не породжує базу даних, що задовольняє термінальній умові.

Пряма система продукцій – від початкового до цільового стану. Якщо можна виділити їхні стани і цілі, то можна визначити глобальну базу даних для опису цих станів. ***П-правила*** – правила, які застосовуються до описів станів для породження нових станів. Цей підхід корисно застосовувати, якщо цільових станів багато, а початковий (вихідний) стан один.

Зворотна система продукцій. При цьому множина описів цілей завдання може служити глобальною базою даних. ***О-правила*** – правила, які застосовуються до описів цілей для породження підцілей. Зворотний рух від цільового до вихідного дає можливість знайти підцільовий стан (стан, шлях від якого до цільового скорочується на один або кілька кроків). Цей підхід корисно застосовувати, якщо цільовий стан один, а початкових (вихідних) станів багато.

> Пряма система продукцій стартує від початкових даних і отримує якийсь результат. База даних розширюється, і термінальною умовою може бути задане число кроків (незалежно від результату) або її повне заповнення (всі факти з початкового набору увійшли в базу).
>
> Зворотна система продукцій стартує від цілей до початкових даних. Узгодження відповідає можливому результату. Залишається питання про однозначність рішення (стандартне запитання при доведенні теорем).

Двостороння система продукцій. Глобальна база даних містить початковий стан і цільові стани. До початкового (вихідного) застосовують П-правила, до цільових – О-правила.

Комутативні системи продукцій – кожне з множини правил може бути застосоване до зміненої (при застосуванні одного з правил) бази даних; якщо цільова умова задовольняється базою даних, то вона задовольняється зміненою (при застосуванні одного з правил) базою даних; бази даних, отримані після застосування послідовності правил, інваріантні при перестановках в цій послідовності. Переваги цих систем в тому, що можна розглядати лише один з можливих шляхів вирішення для даного набору правил (свобода вибору черговості застосування правил). Правила залишаються застосовними до нових породжених баз даних (що в загальному випадку не так).

Розкладні системи продукцій –початкова база даних розбивається на сегменти, які можна обробляти незалежно. Процедура розбиття − декомпозиція бази даних. Обов'язкова вимога − аналогічна декомпозиція термінальної умови (термінальна умова є кон'юнкцією термінальних умов для кожного сегмента). Наприклад, довга молекула, кожен сегмент (ген) якої може зазнавати змін незалежно від сусідніх сегментів.

Методи

Методи – *безповоротний* (застосування правила без можливості повернення до початкового стану), *пробний* (правило використовується, але резервується можливість повернутися до початкового стану). Два типу

пробних режимів – *з поверненням*, коли визначена точка повернення; *управління з пошуком на графі*, коли запам'ятовуються результати застосування декількох послідовностей правил.

Безповоротний режим підходить лише в тому випадку, якщо точно відомо, що метод вирішення (пошуку рішення), заснований на т. зв. локальній інформації, апробований, надійний і зі 100 % вірогідністю призведе до правильного результату (який, взагалі кажучи, ніхто не знає) – т. зв. ***глобальної інформації (розв'язку).*** Приклад застосування локальної інформації для побудови глобального розв'язку – процес «***підйому на гору***» – рух у бік «найбільшої крутизни та / або висоти», поки не досягається потрібний результат. Для цього потрібно мати *спеціальну функцію за допомогою якої здійснюється стратегія вибору правил.*

Якщо на шляху пошуку розв'язку досягається локальний мінімум замість потрібного глобального, то виникають проблеми. У цьому випадку повинні бути додаткові програми перевірки потрібного результату і при невідповідності – механізм виведення («витрушування») системи з цього локального мінімуму. Як це робить радіаційний фон у генетичній структурі організмів.

Режим з поверненням (backtracking). Процедура повинна завершити роботу, якщо породжувана база даних задовольняє термінальній умові. Список правил, використаних для породження цієї бази даних, друкується в кінці. База даних співпадає з термінальною (остаточною, вірною). У рекурсивної процедури немає зупинки (але її можна передбачити, вводячи обмеження на глибину рекурсії), вона може породжувати все нові бази даних. Впорядкування правил для їх послідовного застосування може бути засноване на різних міркуваннях, які доведеться формалізувати в процедурі упорядкування.

В таких режимах система зазвичай не запам'ятовує невдалі шляхи і множину перетворених (в результаті застосування правил) баз даних, хоча можна змусити її запам'ятати всі кроки останнього шляху і бази даних, що виникають на цих кроках.

Роздуми людини – це практично одномоментне і масштабне зіставлення як фрагментарно структурованих, так і раніше не узгоджених образів: вхідна і проміжна неповна інформація завдяки асоціаціям (часткового впізнавання) піднімає структуровану в блоки-образи множину одиниць інформації (знання про об'єкти і явища). У людини думка − множина ланцюжка асоціативних спогадів, бо всі нові конструкції образів формуються з елементів образів, що запам'ятали раніше. Далі пошук потрібного стереотипу і його фіксація: асоціативність, одночасне врахування множини ієрархічних послідовностей зв'язків між уявленнями (саме такі інформаційні конструкції краще запам'яталися) дозволяє швидше виділити, сформувати певний стереотип (думку), домінуючий образ. Тому навіть неймовірна швидкість при, на жаль, послідовному застосуванні логічних операцій, поки не дозволяють експертним системам досягти продуктивності людського інтелекту.

1.2. Байєсова система логічного висновку

Томас Байєс
Reverend Thomas Bayes

Томас Байєс *Reverend Thomas Bayes* (1702−1761) – англійський математик, представив важливі досягнення, що набули широкого поширення у галузі теорії ймовірностей та математичної статистики. Сформулював т. зв. теорему Байєса (1763), формула його імені дозволяє оцінити ймовірність подій, використовуючи емпіричні дані, загальноприйнята термінологія: «байєсівська оцінка», «баєсова мережа».

При використанні байєсівської системи логічного висновку інформація, що обробляється експертною системою, ***не є абсолютно точною, а має імовірнісний характер.*** Користувач не обов'язково повинен бути впевнений в абсолютній істинності чи хибності свідчення, він може відповідати на запити системи з якоюсь мірою впевненості. У свою чергу, система видає результати консультації у вигляді ймовірностей настання наслідків (див, наприклад, [7, 9]).

Теорія

Аксіоми:

1. Ймовірність будь-якої події А є невід'ємною.
2. Ймовірність всіх подій вибіркового простору дорівнює 1.

Визначення:

DET1. Доповнення до А, що позначається (¬А), містить сукупність всіх подій за винятком А. Оскільки А та ¬А є взаємонезалежні (тобто $A \cup \neg A = W$) то $p(A) + p(\neg A) = p(A \cup \neg A) = p(W) = 1$.

DET2. Якщо $B \in \Omega$ дещо інша подія. Тоді ймовірність того, що станеться А за умови, що сталося В, записується у вигляді $p(A \mid B)$ і називається ***умовною ймовірністю події А при заданої події В.***

DET3. Імовірність того, що обидві події А і В відбудуться $p(A \cap B)$ називається ***сумісною ймовірністю*** подій А і В.

> ***Умовна ймовірність:*** Імовірність настання Аподії A, за умови настання події B, називається умовною ймовірністю A (при даній умові), позначається $P(A/B)$ і обчислюється з самого визначення ймовірності. Для даних двох подій A і B розглянемо такий набір несумісних подій: $A\overline{B}, AB, \overline{A}B, \overline{A}\overline{B}$ (повний набір подій). Загальна кількість рівно можливих випадків n. Якщо подія B вже настала, то рівно можливими наслідки обмежується лише двома

подіями AB, $\overline{A}B$. Нехай кількість цих результатів n_B. З цих результатів події A сприяють лише ті, що пов'язані з подією AB. Кількість відповідних результатів позначимо n_{AB}. Тоді відповідно до класичного визначення ймовірності ймовірність події A за умови настання події B буде дорівнювати $P(A/B) = n_{AB} / n_B$, розділивши чисельник і знаменник на загальну кількість рівно можливих випадків n, враховуючи визначення, отримаємо формулу умовної ймовірності: $P(A/B) = P(AB) / P(B)$.

ВИСНОВКИ:

1.Умовна ймовірність p (A | B) дорівнює відношенню сумісної ймовірності ***p (A∩B)*** до ймовірності події B, за умови, що вона не дорівнює 0.

*p(A|B) * p(B) = p(A∩B)* і аналогічно *p(B|A) * p(A) = p(B∩A)* оскільки *p(A∩B) = p(B∩A),* то отримаємо співвідношення

p(A|B) * p(B)= p(B|A) * p(A) **−−−− теорема Байєса**

2 .Для подій, що не перетинаються $B \cap A$ и $B \cap \neg A$ справедливо $B = (B \cap A) \cup (B \cap \neg A)$, тоді

p(B)=p((B∩A)∪(B∩¬A))=p(B∩A)+ p(B∩¬A) = p(B|A)* p(A) + p(B|¬A)*p(¬A).

P апостеріорна = Py * P / (Py * P + Pn * (1 – P))

Таблиця 1.1

P = апостер.	(Py *	P) /	(Py *	P +	Pn *	(1 – P))
Оцінка ймовірності результату	імовірність ствердної відповіді при наявності результату	Апріорна ймовірність результату	Імовірність ствердної відповіді при наявності результату	Апріорна ймовірність результату	Імовірність позитивної відповіді при відсутності результату	Апріорно ймовірність відсутності результату

Структура

База знань є текстовим файлом (який в подальшому може бути зашифрований), що включає три секції з такою структурою:

– Опис бази знань, ім'я автора, коментар тощо (можна в кілька рядків, загальна довжина яких не повинна перевищувати 10000 символів; дана секція закінчується після першого порожнього рядка).

Свідчення № 0 (будь-який текст (не більше 1000 символів), який закінчується переносом рядка)

Свідчення № 1

Свідчення № 2

...

Свідчення № N (після останнього свідчення додається один порожній рядок, і друга секція закінчується)

Результат № 0, P [, i, Py, Pn]
Результат № 1, P [, i, Py, Pn]
Результат № 2, P [, i, Py, Pn]
...
Результат № M, P [, i, Py, Pn]
———

Сенс перших двох секції цілком зрозумілий. Остання секція вимагає більш докладного розгляду. У ній перераховуються правила виведення: кожне задається в окремому рядку; перерахування закінчується з кінцем файлу.

Правило виводу

На початку опису правила виведення задається результат, вірогідність якого змінюється відповідно до цього правила. Це текст, що включає будь-які символи, крім ком. Після коми вказується завжди апріорна ймовірність даного результату (P), тобто ймовірність результату в разі відсутності додаткової інформації. Після цього через кому йде ряд повторюваних полів з трьох елементів. Перший елемент (i) – це номер відповідного питання (симптому, свідчення). Наступні два елементи (Py = P (E / H) і Pn = P (E / НЕ H)) – відповідно ймовірності отримання відповіді «Так» на це питання, якщо можливий результат вірний і невірний.

Ці дані вказуються для кожного питання, пов'язаного з цим результатом.

(Примітка: P <= 0.00001 вважається рівною нулю, а P> = 0.99999 – одиниці, тому не вказуйте такі значення – результат з подібною апріорною ймовірністю розглядатись не буде.)

Приклад (три питання-симптоми):

Грип, 0.01, 1,0.9,0.01, 2,1, 0.01, 3,0,0.01.

Тут сказано: існує завжди апріорна ймовірність P (H) = 0.01 того, що будь−яка навмання взята людина хворіє на грип. Припустимо, програма ставить запитання 1 (про наявність симптому 1). Тоді ми маємо P (E / H) = 0.9 і P (E / неH) = 0.01, а це означає, що експерти дали такі оцінки: Наявність симптому в 90 % означає, що людина хвора на грип, а в 1 % випадків нічого не значить. Якщо у пацієнта грип, то він в дев'яти випадках з десяти відповість «Так» на це питання, а якщо у нього немає грипу, він відповість «Так» лише в одному випадку зі ста (тобто цей симптом зустрічається досить рідко при інших хворобах (випадки)). Очевидно, відповідь «Так» підтверджує гіпотезу про те, що у нього грип. Відповідь «Ні» дозволяє припустити, що людина на грип не хворіє.

Для другого симптому маємо запис «2,1,0.01». В цьому випадку P (E / H) = 1, тобто якщо у людини грип, то цей симптом обов'язково повинен бути присутнім. Відповідний симптом може мати місце і при відсутності грипу (P (E / неH) = 0.01), але це малоймовірно.

Питання 3 виключає грип при відповіді «Так», тому що P (E / H) = 0. Це може бути питання на кшталт такого: «Чи спостерігається у Вас такий стан протягом більшої частини життя?» − або що-небудь на зразок цього.

Значення P (E / H) і P (E / неH), що підставлено в теорему Байєса, дозволяють обчислити апостеріорну ймовірність результату, тобто ймовірність, скориговану відповідно до відповіді користувача на це питання:

$$P(H / E) = P(E / H) * P(H) / (P(E / H) * P(H) + P(E / неH) * P(неH))$$

Таблиця 1.2

P(H / E) апостер.	(P(E / H) *	P(H)) /	(P(E / H) *	P(H) +	P(E / неH) *	P(неH))
Оцінка ймовірності результату H при позитивній відповіді E	Імовірність ствердної відповіді E при наявності результату H	Апріорна ймовірність результату H	Імовірність ствердної відповіді E при наявності результату H	Апріорна ймовірність результату H	Ймовірність позитивної відповіді E при відсутності результату (Тобто, неH)	Апріорна ймовірність відсутності результату (Тобто, неH)

або

P апостеріорна = Py * P / (Py * P + Pn * (1 – P))

Ймовірність реалізації якоїсь гіпотези H за наявності певних свідчень, що підтверджують E обчислюється на основі апріорної ймовірності цієї гіпотези.

Приклад актуальної проблеми. Подія: людина озирнулась. Результат: закохалась?

Оцінки ступеня довіри (міри істинності, ймовірності)

Теорія Демпстера (Dempster) −Шафера (Shafer). ***Міра істинності*** визначається інтервалом

Довіра <міра істинності <правдоподібність.

Артур Демпстер *Arthur Pentland Dempster* (праворуч),
Глен Шафер *Glenn Shafer*

Артур Демпстер – *Arthur Pentland Dempster* (1929) – математик, (праворуч) професор Гарвардського університету,

Глен Шафер – *Glenn Shafer* професор у багатьох університетах, математик в області статистики і штучного інтелекту. Розробили методи оцінки достовірності подій, зокрема, Dempster−Shafer theory (DST).

Довіра – це сума мас свідчень, що підтримують гіпотезу. ***Правдоподібність*** = 1 − сума мас свідчень, що відкидають гіпотезу. Розглянемо приклад. Можна припустити, що у дівчини є один, якого вона любить, або не любить.

Таблиця 1.3

Гіпотеза	Маса свідчень	Довіра	Правдоподібність
Байдуже. Тобто, друга у неї немає	0	0	0
Вона когось любить	0,2	0,2	0,5
Вона має одного, але його не любить	0,5	0,5	0,8
Універсальне. Вона має друга, але невідомо, любить вона його чи не любить	0,3	1,0	1,0

Цей метод дає можливість, виходячи зі свідчень експертів, оцінити міру істинності кожного твердження, зазначеного в таблиці.

Система Перла (Perl). Це формування так званих мереж Байєса, які дають можливість оцінити ймовірність однієї події, якщо сталася інша подія. Неважко бачити, що система Байєса, розглянута в цьому розділі, – це окремий випадок мережі Байєса. Розглянемо приклад.

Оцінити ймовірність дощу, якщо відомо, що трава мокра. Є поливальна установка (S) і можливий дощ (R) як причини наявності мокрої трави (G).

Очевидно, умовна ймовірність, яку слід визначити, записується у вигляді

$$P(R=T/G=T)=\frac{P(G=T,R=T)}{P(G=T)}=$$

$$=\frac{P(G=T,R=T)}{P(G=T,S=R=T)+P(G=T,S=T,R=F)+P(G=T,S=F,R=T)+P(G=T,S=R=F)}.$$

Якщо відомі всі ці ймовірності, то можна знайти шукану умовну ймовірність.

Вибір основних змінних, параметрів

Факторний аналіз – виділення головних показників, оптимізація опису об’єкту, завдяки об’єднання змінних, що сильно корелюють між собою.

1. Всі показники (змінні) класифікуються за характером взаємних зв’язків, виділяються основні та другорядні, від яких зазвичай надалі відмовляються. Фактор – прихована (комбінована) або явна змінна. Навантаженням називають кореляцію між вихідною змінною і фактором.

2. Приклад з теорії розмірностей: один фактор може об’єднувати в собі кілька змінних, якщо кореляція змінних всередині фактора сильніша, ніж між змінними різних факторів. Формування факторів з набору змінних може відбуватися і з інших причин, взагалі кажучи, не пов’язаним з їх корельованістю. Деякі змінні, що мають високу взаємну кореляцію, або пов’язані розмірністю, можуть бути описані деякою усуспільненою змінною, яку можна також вважати фактором.

Приклад: приведемо рівняння гідродинаміки – рівняння Нав’є-Стокса

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=-\frac{1}{\rho}\frac{\partial P}{\partial x}-\nu\frac{\partial^2 u}{\partial x^2}$$

до безрозмірного вигляду. Використаємо безрозмірні змінні:

$$V = u / u_0, T = t / \tau, \xi = x / l, P = \rho \cdot g \cdot x = \rho \cdot l \cdot g \cdot \xi,$$ та отримаєм

$$\frac{u_0}{\tau}\frac{\partial V}{\partial T} + \frac{u_0^2}{l} V \frac{\partial V}{\partial \xi} = -g \frac{1}{\rho'} \frac{\partial (\rho' \cdot \xi)}{\partial \xi} - \frac{\nu u_0}{l^2} \frac{\partial^2 V}{\partial \xi^2}.$$

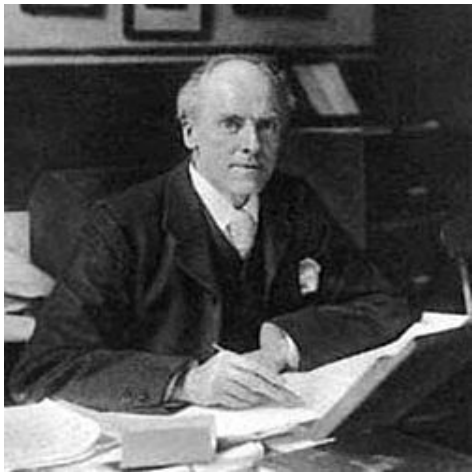
Після множення на $\frac{l}{u_0^2}$, представимо рівняння у вигляді

$$\frac{1}{S}\frac{\partial V}{\partial T} + V \frac{\partial V}{\partial \xi} = -\frac{1}{F} \frac{\partial (\rho' \cdot \xi)}{\rho' \partial \xi} - \frac{1}{R} \frac{\partial^2 V}{\partial \xi^2}.$$

Характерні числа, які визначають параметри течії – фактори:

$S = \frac{u_0 \tau}{l}$ – число Струхала (Strouhal); $F = \frac{u_0^2}{gl}$ – число Фруда (Froud), $R = \frac{u_0 l}{\nu}$ – число Рейнольдса (Reynolds).

3. Фактори формуються в структуру[10]. Для формування цієї структури використовують «метод головних компонент» (Principal component analysis, PCA) – один з основних способів зменшити розмірності даних, втративши найменшу кількість інформації.



К. Пірсон *(Karl Pearson)* 1901 р.

Карл Пірсон *Karl (Carl) Pearson,* (1857−1936) англійський статистик, біолог іфілософ. Засновник математичної статистики, зокрема, теорії кореляції, кореляційного аналізу, професор прикладної математики і механіки Лондонського університету.

Візуалізація даних – подання даних експерименту або результатів теоретичного дослідження. Візуалізація даних є ортогональне проеціювання на площину перших двох (трьох) головних компонент. Такі проекції оптимальні за***мінімізації суми квадратів відстаней від точок даних до проекцій на площину головних компонент***.

[10] Аналітично найбільш уживаний ортогональний метод обертання матриці даних – «варимакс». Метод «варимакс» максимізує розкид квадратів навантажень для кожного фактора, що призводить до збільшення великих і зменшення малих значень кореляторів (косокутне обертання використовують для визначення простої структури другорядних факторів).

1.3. Простий логічний висновок

Використання формальної логіки для створення логічного методу вирішення завдань було розпочато ще Фрідріхом Фреге і його сучасниками в XIX столітті. У наступних розділах докладніше розглянемо логіку предикатів, яка стала основою мов логічного програмування [10−12].

Фрідріх Людвіг Готлоб Фреге
Friedrich Ludwig Gottlob Frege

Внесок в логіку порівнюють із внеском Аристотеля і К. Геделя. Фреге (1879) створив логіку предикатів, ввів квантори і багато іншого, що дозволило з'явитися книзі Principia Mathematica і теоремі про неповноту Геделя

Жак Ербран
Jacques Herbrand

Основні праці в математичній логіці, теорії полів класів, ввів рекурсивні функції, розробив математичний метод резолюції.

Теорема Ербрана про дедукції є результатом дисертації з теорії доказів.

Джон Алан Робінсон
John Alan Robinson

Вніс визначальний внесок в розвиток логічного програмування, доктор університету Прінстона, працював в концерні DuPoint і в університеті Райса, де сформулював практичний метод резолюції в логіці, що визначило розвиток мов штучного інтелекту

Далі зупинимося на принципах логічного висновку, і розглянемо такий **приклад найпростішого логічного висновку.**

База знань (складається з двох правил):
Правило 1: **Якщо** "відпочинок – влітку" і "людина – активна", **то** "їхати в гори",

(відпочинок – влітку $\wedge$ людина – активна $\rightarrow$ їхати в гори)

Правило 2: **Якщо** "любить сонце", **то** "відпочинок влітку",

(любить сонце $\rightarrow$ відпочинок влітку)

Вхідні дані – "людина активна" і "відпочинок влітку"

Отже: Правила – це база знань. База даних містить факти.
Правило складається з фактів (літералів) і зв'язок.
Факти (літерали) в складі правил можуть знаходитися в базі даних частково або всі. Якщо вони всі вже знаходяться в базі даних, то зупинка програми. Ця умова – термінальна.

Перше правило складне, бо там три літерали, друге – просте, тільки два літерали.

Розглянемо дії, зображені в таблиці

Таблиця 1.4

Прямий вивід	Зворотний вивід
1-й прохід. Крок 1. Пробуємо Правило 1 – не працює (не вистачає даних «відпочинок – влітку»). Крок 2. Пробуємо Правило 2 – працює, в базу надходить факт «відпочинок – влітку». 2-й прохід. Крок 3. Пробуємо Правило 1 – працює, активується мета «їхати в гори», яка і виступає як порада, яку дає ЕС.	1-й прохід. Крок 1. Мета – «їхати в гори»: пробуємо Правило 1 – даних "відпочинок – влітку" немає, вони стають новою метою, і шукається правило, де вона в правій частині. Крок 2. Мета "відпочинок – влітку": Правило 2 підтверджує мету і активує її. 2-й прохід. Крок 3. Пробуємо Правило 1, підтверджується мета, яку шукаємо.

Прямий висновок – розглядається спочатку ліва частина (посилка). Якщо посилка серед фактів, то правило спрацьовує і мета (висновок) цього правила, тобто новий факт, додається в базу даних. Якщо в посилці не всі факти серед фактів бази даних, правило не спрацьовує і програма переходить до другого правила. Після використання всіх правил (перший прохід) база даних модифікується (в цьому випадку додаються нові факти). Другий прохід з оновленою базою даних дозволяє задовольнити першому правилу і висновок-факт додається до бази даних. Далі машина може: 1) перевірити всі задіяні правила, при цьому змін в базі даних немає або 2) усі факти в структурі правил виявилися в базі даних або 3) ще як-небудь (?), але якось треба задати термінальну умову, при задоволенні якої програма зупиняється.

Форма навчання ЕС

Заповнення глобальної бази даних новими фактами і реченнями, які є наслідком виконання процедур виведення, є формою навчання інтелектуальної системи. Дійсно, при розв'язанні подібних задач, можна скористатися цими новими фактами і реченнями, що здатне спростити процедуру і скоротити час розв'язання..

Зворотний висновок. Стартують від висновків правил (тобто цілей). Перевіряють першу мету (праву частину першого правила), в лівій частині

два літерали-факти. Один є в базі даних, з ним все в порядку, а іншого – немає. Він активізується, і шукається інше правило бази знань, де він знаходиться в правій частині. Знаходимо таке правило, перевіряємо ліву частину – чи є вона в базі даних. Якщо є, то тепер початкова мета отримує право на життя, її включають в базу даних. При цьому в базу даних можуть включати і всі проміжні цілі цього вдалого рішення. Перевірка всіх задіяних правил дає їх здійснення і змін бази даних вже немає – це може служити термінальною умовою – умовою зупинки програми.

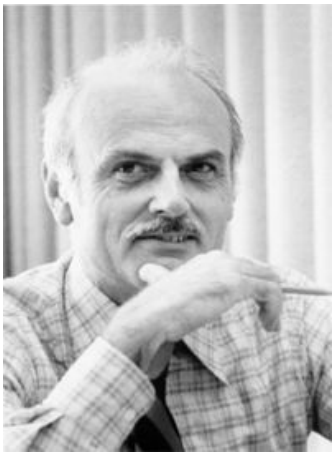
Зворотний сценарій більш конструктивний, бо якщо в прямому реалізований перебір правил, то в другому формується дерево цілей, правила застосовуються вибірково. Перебір може збільшувати час розв'язання, формування дерева скорочує час розв'язання.

До речі, в зворотному виведенні теж можна організувати перебір, аналогічний розглянутому прямому методу, а в прямому – побудувати схему за типом формування дерева. Це може бути домашнім завданням.

1.4. Реляційна база даних

Тут ітиметься про реляційні бази даних, які засновано на однойменних моделях даних. Визначається залежність, відносини між даними. Використання даного підходу і розробка перших моделей було виконано доктором Е. Коддом з IBM (1970).

Едгар Франк «Тед» Кодд – *Edgar Frank Codd* (1923−2003) британський вчений, докторський ступінь в університеті Мічигану, працював в Альмаденському дослідному центрі IBM, де створив реляційну модель даних, заклав основи теорії реляційних баз даних.



Едгар Франк Кодд
Edgar Frank Codd

Дані про сутність зберігаються у відношеннях. Відношення містить кілька кортежів (рядків таблиці). Кортеж складається (заповнюється) з атрибутів. Множину кортежів даного відношення називають тілом відношення. Атрибут – властивість, що характеризує сутність. Набір доменів визначає відношення, а кожен домен – множину можливих значень одного атрибута. Сукупність атрибутів, які однозначно визначають кожен з кортежів відношення, – ключ відношення.

ПРИКЛАД:

Таблиця.1.5

ПІБ	Відділ	Посада	Дата народження
Вовк О. Б.	1	майстер	10 01 70
Заєць В. Г.	2	робітник	20 02 80
Мисливець Д. Є.	3	начальник	30 03 90

Таблиця 1.5 в цілому – це відношення, перший рядок (рядок заголовків) – схема відношення, 2–4 рядки – кортежі, заголовок стовпчика – атрибут, запис в осередку – значення атрибута, домен – набір всіх значень атрибута, ключ – набір атрибутів, що ідентифікує кортеж (даного рядка). ***Зв'язування відношень***. Розглянемо два відношення: ВІДДІЛИ (**код відділу**, назва, число співробітників); СПІВРОБІТНИКИ (**код співробітника**, ПІБ, *код відділу*, зарплата). *Атрибути, виділені жирним шрифтом – внутрішні первинні ключі, а курсивом – зовнішній ключ відношення СПІВРОБІТНИКИ (є первинним ключем відношення ВІДДІЛИ). Ключі можна індексувати – система індексації фактично автономна база даних. Можна отримувати нові відношення (обчислювані) зі старих (збережених). Функціональна залежність одного (складеного) атрибута B від іншого (складеного) атрибута A полягає в тому, що кожному значенню A відповідає в точності одне значення B ($A \to B$). Транзитивна залежність атрибутів A, B, C – якщо $A \to B$, а також $B \to C$ але зворотні залежності, взагалі кажучи, відсутні.*

1. Первинні ключі можуть бути індексовані індексами різного рангу, що дозволяє формувати довільну кількість автономних баз даних.

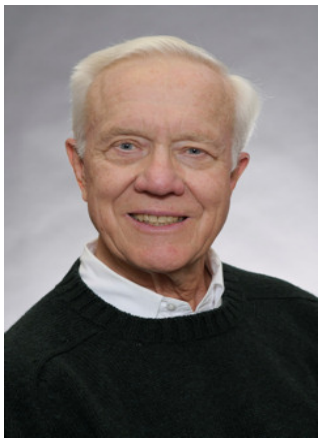
2. Зовнішні ключі дозволяють пов'язувати різні елементи з різних баз даних.

Заповнення глобальної бази даних − ключове завдання для будь-якого інтелекту. По-перше, навіть при великому обсязі «приміщення» для даних і знань переріз входу в нього вельми обмежений. По-друге, слід все так укласти і розкласти, щоб було легко дістатися до потрібної упаковки. По-третє, неминуче доведеться все упаковане багато разів перекладати, оскільки потрібні для певної мети упаковки бажано зібрати в одному місці. Очевидно, що якщо весь час з'являються нові об'єкти, які слід упакувати і розкласти, і якщо вони повинні виявитися в потрібному місці складу, поряд з подібними до них, то можна уявити яка велика робота повинна безперервно виконуватися в цьому складі − глобальній базі даних. Особливо на початковому етапі, коли цей склад наповнюється. Тому процеси заповнення та перебудови − дефрагментації (аналоги фазових переходів другого роду) глобальної бази даних вимагають значних витрат енергії, що, зокрема, затримує зростання молодого організму. Особливо у людського дитяти, яке і в дошкільній стадії (яка подібна до стадії розвитку будь-якого теплокровного), а особливо на стадіях навчання в школі і в університеті засвоює гігантські обсяги інформації. Інтегрування в культуру, соціально-виробничі відносини вимагає усвідомлення обчисленних смислів і освоєння безлічі технологій, що і призводить до затримки фізичного розвитку у багато разів, в порівнянні з іншими ссавцями. Людина третину свого життя зростає, що абсолютно не можна порівняти з відведеним іншим тваринам для свого дорослішання відрізком їх життя.

1.5. Розв'язання задач роботом

Розглянемо задачу зі створення планів в рамках програми автоматичного планувальника STRIPS (Stanford Research Institute Problem Solver). Розв'язання задач роботом – це некомутативна система продукцій [7].

Нільс Нільсон – *Nils J. Nilsson*
Керівник Центру штучного інтелекту професор Стенфордського університету, творець планувальника STRIPS, провів аналіз властивостей резолюції, випустив кілька підручників з штучного інтелекту, визнаних найбільш повними і змістовними (зокрема, книга «Принципи штучного інтелекту»).



Нільс Нільсон
Nils J. Nilsson

ПРИКЛАД: *Гра в кубики. Опис завдання.*
Описують стан світу: предикати CLEAR(x) – не зайнята верхня поверхня кубика x, ON (x,y) – кубік x знаходиться на кубику y, ONTABLE (x)– кубік x знаходиться на столі.
Описують стан руки робота: предикати HANDEMPTY – рука порожня, HOLDING (x) – рука тримає кубик x.
Описують дії – **pickup** *– взяти зі столу,* **putdown** *– поставити на стіл,* **stuck** *– поставити на інший кубик,* **unstuck** *– зняти з іншого кубика.*
Опис даного стану: CLEAR (B); CLEAR (C); ON (C, A); HANDEMPTY; ONTABLE (A); ONTABLE (B) – кубики A і B лежать на столі, кубик C – на кубику A.
Опис цілей: ON (B, C) $\wedge$ ON (A, B) – кубик A знаходиться на кубику B, а кубик B, в свою чергу, на кубику C.

Дії робота. При діях робота П-правила повинні допускати викреслювання виразів, які можуть перестати бути справедливими. П-правила (систем типу STRIPS) явно вказують списки викреслювання. Нагадаємо, що до початкових (вхідних) станів застосовують П-правила, до цільових – О-правила.

Застосування правила – моделювання реальної дії.

Форма П-правила (в STRIPS) складається з (1) формули попередньої умови, подібної умові «якщо» («якщо» = антецедент, «то» = консеквент) в імплікації, має випливати з фактів в описі стану, форма – кон'юнкція літералів, змінні відносяться до квантору існування. **Підстановка відповідності** – набір уніфікаторів, який забезпечує **відповідність** П-правил фактам (таких підстановок відповідності може бути кілька).

(2) Список викреслювання (список літералів можливо з вільними змінними) – друга частина П-правила. Підстановка відповідності застосовується до списку викреслювання і отримані основні окремі випадки (коли всі змінні замінені на константи) викреслюються з опису колишнього стану.

(3) Формула додавання (складається з кон'юнкції літералів) – третя частина П-правила, – подібна частині «то» в імплікації. Підстановка відповідності застосовується до цієї формули додавання і результуючий окремий випадок додається до опису стану.

ПРИКЛАД: Модель дії: взяття кубика зі столу
Передумови − кубик на столі, рука робота не зайнята, зверху на кубику нічого немає. Результат − рука тримає кубик.
pickup (x)
P (precondition − передумова): ONTABLE (x) $\wedge$ HANDEMPTY $\wedge$ CLEAR (x).
D (delete list − список викреслювань): ONTABLE (x), HANDEMPTY, CLEAR (x).
A (add formula − формула додавання): HOLDING (x).
Підстановка відповідності в цьому випадку дає − pickup (x) може бути застосоване лише якщо x = B. Тоді новий опис стану: CLEAR (C), ON (C, A), ONTABLE (A), HOLDIBG (B). У формулу додавання можна включати заперечення викреслювати літералів.

Звичайні умови – всі формули є кон'юнкції літералів, тому формули передумов і доповнень можна представити у вигляді списку літералів.

Проблема фрейму – які ППФ опису стану слід змінювати, а які – ні. Існує стан світу (фон) і моделюються дії (мають локальний характер). При більш точному розгляді треба враховувати зміни фону, тобто існує ієрархія планів (дій), які враховують зв'язки між компонентами світу, що активуються даною простою дією. Існують також труднощі опису та правил (дій) в разі аномальних умов (яких може виявитися занадто багато).

ПРИКЛАД*: опис дій: брати і ставити кубик на стіл, брати і ставити один кубик на інший, знімати поставлений кубик з нижнього. Тут формула передумови і список викреслювання збігаються.*

1. **pickup** *(x)*
P & D: ONTABLE (x), CLEAR (x), HANDEMPTY
A: HOLDING (x)
2. **putdown** *(x)*
P & D: HOLDING (x)
A: ONTABLE (x), CLEAR (x), HANDEMPTY
3. **stuck** *(x, y)*
P & D: HOLDING (x), CLEAR (y)
A: HANDEMPTY, ON (x, y), CLEAR (x)
4. **unstuck**
P&D: HANDEMPTY, CLEAR (x), ON (x, y)
A: HOLDING (x), CLEAR (y)

Внутрішній світ інтелектуальних систем

Всі обговорювані системи накопичення, зберігання і використання знань, як, втім, і людський розум, містять свій світ, де всі операції, образи і дії закладені раніше[11]. Основною проблемою є навіть не створення універсального формату представлення цих знань (що є цілком посильним завданням), а можливість **адекватного** і визнаного всіма однозначного їх подання. Важко домовитися про єдине подання знань не тому, що це неможливо в принципі. А тому, що люди не бачать в цьому необхідності, бо вони можуть обмінюватися даними (нехай навіть в універсальному форматі) без використання їх однозначного уявлення. Асоціативність проявлення інформації, багатоканальне виділення образів і проведення процедур в різних ***інтелектуальних системах*** (пропонують використовувати цей термін для систем логічного висновку, більшою мірою, ніж визначення «штучний інтелект») дуже різна, але має місце всюди, що дозволяє говорити, принаймні, про наявність елементів евристичності. На перший погляд це зовсім не обмежує нас в застосуванні інших схем подібного роду. Хоча це «швидше нужда, ніж чеснота». Але варто мати на увазі, що всі створені і створювані системи формують примітивний тією, чи іншою мірою світ. А методи отримання висновків, рішень і розпізнавань, засновані на дедукції, мають обмежену сферу застосування. Люди і створені ними пристрої розв'язують задачі або відмовляючись від надмірно складних побудов, або конструюючи громіздкі схеми описів, населені новими сутностями і упередженнями, засновані на очікуваних відповідях. Часом створюється враження, що люди взялися за вирішення інтелектуальних проблем, погано усвідомлюючи, що ж вони самі з себе в цьому плані являють.

[11] У деяких людей іноді навіть з народження.

РОЗДІЛ 2
ЕЛЕМЕНТИ ФОРМАЛЬНОЇ ЛОГІКИ

2.1. Силогізм[12]

Силогізм – це вид логічних міркувань, вперше описаний Арістотелем, за якого «... якщо певні речі встановлені, то деякі інші неминуче випливають зі справедливості перших».

Судження = квантор + суб'єкт + зв'язка → предикат

Силогізм = правила отримання суджень на основі чотирьох допустимих форм (логіка класів, але не містить доповнення класів):

всі * S є P
ніякий * з S не є P
деякі ** з S є P
деякі з S не є P
(* Універсальний і ** екзистенціальний квантори $\forall$, $\exists$).

Висловлювання – узагальнення судження, якому можна приписати значення *істинне (правдиве)* або *хибне (неправдиве*).

modus ponendo ponens P → Q, if P− tru, then Q − tru
modus tollendo tollens P → Q, if P− fal, then Q −fal
modus ponendo tollens P ≠ Q, if P− tru,, then Q −fal
modus tollendo ponens P or Q − tru, if P− fal, then Q − tru (диз'юнктивний силогізм)
modus – правило, міра.

2.2. Початки формальної логіки

Булева алгебра[13] − нова форма логіки, де виконувалися: комутативність, асоціативність і дистрибутивність для множин і істиннисних значень, де знак підсумовування + це «або», знак множення * − це «і».

Зв'язки $\wedge$ (і), $\vee$ (або) $\Rightarrow$ (імплікація).

Нове, що було введено в традиційну логіку:

Доповнення – унарна операція з множиною або істінностним значенням.

---

[12] За влучним зауваженням Клея Ширки, силогізми «описують світ набагато простіший, ніж наш з вами», бо для застосування силогізмів «потрібен світ, де мова – це просто набір математичних операцій над словами».

[13] З імплікації і заперечення можна вивести всі з'єднувачі (Ф. Л. Г. Фреге) [10].

Правила де Моргана

$$\sim (x + y) = \sim x * \sim y, \sim (x * y) = \sim x + \sim y.$$

Імплікація (слідування)

Імплікація (якщо, ... то ..., де якщо = антецедент, то = консеквент) if x then y, а для істиннисних значень[14] можна переписати ~ x + y.

Заперечення НЕ.

*) Нагадаємо, що, НЕ F1 $\vee$ F2 має таке саме значення істинності, що і F1 $\Rightarrow$ F2

2.3. Логіка висловлювань

Подальший розвиток формальної логіки, де вводилися нові поняття:

Правильно побудовані формули (ППФ) позначають пропозиціональними змінними p, q, r, у яких нема внутрішніх змінних. Наступні формули теж ППФ: ~p, p&q, pVq, p→q, p←→q. Тут & – «і», V – «або». Замість & часто використовують $\wedge$.

Істинне − одиниця, ***хибне*** – нуль. Тоді & має сенс множення, V − додавання. Імплікація p→q відповідає ~ pVq, еквівалентність p←→q (p & q) V (~ p & ~ q).

Тавтологія = тотожно істинне = AV ~ A;

Непослідовна ППФ = протиріччя = тотожно ***хибне*** = A & ~ A.

Теорія – повна множина атомарних ППФ для даної предметної області, кожна така ППФ – ***аксіома***.

Модель – інтерпретація, згідно з якою кожна аксіома є істинною.

Доказ того, що дана ППФ є наслідком аксіом теорії. Аксіоми вважаються дійсними для всіх окремих інтерпретацій.

Методи логічних побудов [12–15]

1. ***Семантичний метод*** – потрібно показати, що при всіх інтерпретаціях дана ППФ – істинна. Якщо є модель, за якої ППФ буде помилковою, то ППФ – не наслідок теорії.

2. ***Синтаксичний метод*** – шукають повну (якщо можна отримати будь-яку тавтологію, скомбіновану з пропозиціональних позначень) множину правил виводу.

Аксіоми = передумови, результат вживання правил – висновок. Висновок в кінці всіх кроків – дана ППФ. Можна використовувати т. зв. аксіоматичні схеми і одне (два або більше) правило, наприклад, modus ponens.

---

[14] Якщо x хибне, то завжди результат істинне (!).

3. ***Алгоритм*** – дозволяє відповісти на питання, чи є дана послідовність кроків доказом того, що ППФ – наслідок заданих аксіом. Питання, чи є ППФ наслідком аксіом, алгоритмічно розв'язане, тільки якщо ППФ є таким наслідком (обчислення висловлювань – повна множина правил виводу).

Нові поняття:

Формальна мова – об'єктна мова.

Метамова описує всі деталі і особливості застосування об'єктної мови (часто не формалізовані). Аналітично – виділення тавтологій, протиріч, повноти; семантично – встановлення істинносних значень.

Несуперечливість (синтаксична послідовність) теорії – з аксіом не можна вивести протиріччя.

Повнота теорії – кожну справжню ***ППФ*** можна довести на підставі аксіом теорії.

Рекурсивне визначення множини відносин – дотримання відносини в одному випадку спирається на дотримання відносини в попередньому випадку.

Рекурсивний алгоритм – правила виведення і рекурсивні аксіоми.

Рекурсивне переліченння – утворення множини шляхом рекурсивного алгоритму.

Резолюція. Резолюція двох речень утворюється при **примусовій диз'юнкції** двох взаємно протилежних літералів P і ~ P які знаходяться в двох різних реченнях (ППФ).

Процедура: Наприклад, беремо ($R \vee P$) і ($\sim P \vee Q$), візьмемо диз'юнкцію цих висловлювань $R \vee P \vee \sim P \vee Q$, (яка, взагалі кажучи, не суперечить кожному з них) або, що те ж саме $R \vee (P \vee \sim P) \vee Q$, потім спростимо це новий вираз, видаляючи тавтологію ($P \vee \sim P$) і остаточно отримаємо ***резольвенту*** – це $R \vee Q$. Це виявилося нове висловлювання, яке можна занести в глобальну базу даних.

Пояснення: Резолюція двох висловлювань $P \vee Q$ і $\sim P \vee G$ є $Q \vee G$. З іншого боку, ці висловлювання можна уявити як $Q \vee P$ і $P \Rightarrow G$ при цьому P замінюється на G і отримаємо $Q \vee G$.

Приклади: найпростіші резолюції

Таблиця 2.1

Вихідні висловлювання	Резольвенти	Обговорення
P і $\sim P \vee Q$ $(P \Rightarrow Q)$	Q	Модус поненс
$P \vee Q$ і $\sim P \vee Q$	Q	$Q \vee Q$ то Q т.зв. злиття
$P \vee Q$ і $\sim P \vee \sim Q$	$Q \vee \sim Q$, $P \vee \sim P$	Дві тавтології
P і $\sim P$	*NIL*	Порожня множина
$\sim P \vee Q$ $(P \Rightarrow Q)$ $\sim Q \vee R$ $(Q \Rightarrow R)$	$\sim P \vee R$ $(P \Rightarrow R)$	Ланцюжок

Процедура спростування – (доказ від протилежного) − додавання в теорію заперечення ППФ = А, тобто ~ А, що переводить цю теорію в непослідовну, тобто слід довести доказ аж до порожньої множини.

Спрощення −

(1) виключення тавтології,

(2) оцінка окремих літералів (для основних окремих випадків) завдяки т. зв. приєднаної процедури (якщо літерал отримує оцінку Т, то таке речення виключають, а якщо F, то можна речення залишити, але виключити цей літерал),

Чому результат повинен бути порожнєю множиною? Наприклад, у нас є певне питання А. Якщо в базі даних є таке ж значення А, то значить, на питання ми відповідаємо ствердно.

Але можна інакше: візьмемо заперечення питання ~ А і складемо диз'юнкцію з наявними в базі даних А. Отримаємо тавтологію, яка після спрощення (її виключення) дасть порожнє речення. Це інший спосіб відповіді на питання.

Отже, можна замість пошуку в базі даних такого ж А (якщо немає такого А, то можна, використовуючи процедури резолюції, спробувати його створити, а ось якщо це не вдається, якщо не вийде, то значить відповіді немає), можна взяти заперечення питання ~ А і, використовуючи процедури резолюції для цього ~ А і зміст бази даних, отримати пусте висловлювання.

РОЗДІЛ 3
СУТНІСТЬ МОВИ – ОБЧИСЛЕННЯ ПРЕДИКАТІВ

3.1. Основна термінологія

Синтаксис – алфавіт символів і допустимі вирази (в обчисленні предикатів – це правильно побудовані формули – ППФ).

Символи – відносин (предикатні), змінних, констант, функціональні.

Атомні формули – осмислені вирази, складаються з предикатних символів і термів.

ШЛЮБ [батько (ДЖОН), мати (ДЖОН)

Терми – константи, змінні, функції.

Зв'язки $\wedge$ (і), $\vee$ (або) $\Rightarrow$ (імплікація).

Заперечення НЕ.

*) Нагадаємо, що, *НЕ F1* $\vee$ *F2* має те ж значення істинності, що і *F1* $\Rightarrow$ *F2*

Кон'юнкція (складається з кон'юнктів)

ЖИВЕ (ДЖОН, будинок 1) $\wedge$ КОЛІР (будинок 1, ЖОВТИЙ) *Джон живе в жовтому будинку*

Диз'юнкція (складається з диз'юнктів)

ЇСТЬ (ДЖОН, КОВБАСА) $\vee$ ЇСТЬ (ДЖОН, СИР) Джон їсть або ковбасу, або сир

Імплікація (якщо, ... то, якщо = антецедент, то = консеквент)

ВОЛОДІТИ (ДЖОН, машина 1) $\Rightarrow$ КОЛІР (машина 1, ЗЕЛЕНИЙ) *Якщо машина належить Джону, то вона – зелена.*

Таблиця 3.1

X1	*X2*	*X1* $\vee$ *X2*	*НЕ X1* $\vee$ *X2*	*X1* $\wedge$ *X2*	*X1* $\Rightarrow$*X2*	*НЕ X1*
T	***T***	***T***	***T***	***T***	***T***	***F***
F	***T***	***T***	***T***	***F***	***T***	***T***
T	***F***	***T***	***F***	***F***	***F***	***F***
F	***F***	***F***	***T***	***F***	***T***	***T***

Квантори – існування, спільності.

$\forall$ *x* [*СЛОН (x),* $\Rightarrow$ КОЛІР (x, СІРИЙ] *Всі слони сірі*

(з лат. praedicatum – сказане)

Таблиця 3.2

Зв'язки	*Квантори*
$\wedge$ – кон'юнкція – І *	∃ існування
$\vee$ – диз'юнкція – АБО * +	*(*∃ x –*знайдеться такий x)*
⇒ імплікація – ЯКЩО ... ТО **	∀ спільності
–– заперечення – НЕ	*(*∀ *x* – для кожного x *)*
⇔ еквівалентність	

* $\dot{\vee}$ – можлива диз'юнкція в строгому сенсі того, що вона розділяє

** ЯКЩО – антецедент, посилка, ТО – консеквент, висновок.

Таблиця 3.3

Очевидні

T $\vee$ ***T***	***то T***
T $\vee$ ***F***	***то T***
F $\vee$ ***F***	***то F***
T $\wedge$ ***T***	***то T***
T $\wedge$ ***F***	***то F***
F $\wedge$ ***F***	***то F***

T – істинне, F– хибне

Неочевидні

-X1 $\vee$ ***X2***	*еквівалентно*	***X1*** ⇒***X2***
–***T*** $\vee$ ***T***	***то T***	***T*⇒*T***
–***F*** $\vee$ ***T***	***то T***	***F*⇒*T*** *
–***T*** $\vee$ ***F***	***то F***	***T*⇒*F***
–***F*** $\vee$ ***F***	***то T***	***T*⇒*T***

** Хибне наступне твердження: «З істинного твердження випливає хибне твердження»*

Предикати першого порядку – не допускається квантифікації по предикатних і функціональних символах.

Правила виведення 1) утворення A2 з A1 з A1 ⇒A2 (модус поненс);
2) утворення Ф (A) з (∀ x) Ф (x) (спеціалізація).

Уніфікація – пошук підстановок термів на місце змінних. Алфавітний випадок – заміна однієї змінної на іншу.

Основний окремий випадок – підстановка замість змінних – констант. Позначення – підстановка s: замість змінної t підставлена постійна a. Порівняння уніфікованих літералів часто називають «порівнянням із зразком».

Задоволеність – якщо кожна ППФ з множини має значення Т за однієї і теж інтерпретації, то ця інтерпретація задовольняє цій множині ППФ.

Опис стану – опис конкретної ситуації («опис світу»). Будь-яка кінцева кон'юнкція формул описує сімейство (що конкретизується завдяки додавання нових формул, тобто уточнення) станів, кожен стан може розглядатися як інтерпретація.

Інтерпретація – 1. Співставлення висловлюванням і предикатам значення істинність або хибність. 2. Нав'язує відповідність між елементами мови, відносинами, елементами і функціями в зазначеній галузі міркування (визначає семантику мови обчислення предикатів), тоді можна приписати деяким значенням формул значення Т або F.

Система дедукції = доведення теорем – показує, що дана ППФ (цільова ППФ) є теоремою, виведеною з множини формул (опису стану).

Незадовольнима множина ППФ – множина ППФ, яка не може задовольнятися за будь-якої інтерпретації.

Здійснюваність – якщо для всіх можливих інтерпретацій ППФ має значення Т (істина).

Тавтологія – виконана ППФ.

Неповна розв'язність (напірозв'язність) обчислення предикатів – при застосуванні кванторів не завжди ППФ виконуються. Немає універсального методу встановлення факту виконаності квантифікованих виразів. Але існує процедура встановлення виконаності, якщо заздалегідь відомо, що ППФ здійсненна.

Логічне слідування – якщо дана ППФ випливає з множини М інших ППФ в кожній з інтерпретацій, які задовольняють і М, і даної ППФ.

Логічність системи правил виведення (sound) – якщо кожна теорема, що виводиться з множини ППФ, логічно випливає з цієї множини.

Повнота системи правил виведення – всі ППФ, які логічною мірою випливають з будь-якої множини ППФ, є також теоремами, виведеними з цієї множини.

Необхідність розширення бази знань

Прогнозування, як форма розв'язання задач у всіх інтелектуальних системах, включаючи природний розум, спирається лише на попередньо створений внутрішній світ образів, подій і уявлень. Інтелектуальні системи замкнені в цьому світі і не здатні вийти з нього в своїх рішеннях і прогнозах. Єдиним способом вирватися зі створеного внутрішнього світу може бути тільки отримання додаткової інформації про світ зовнішній. Тобто можливість розвитку пов'язана зі створенням ефективних систем моніторингу зовнішнього оточення. Для розвитку штучних інтелектуальних систем потрібно помітно більшу увагу приділяти інтерактивним структурам, які забезпечують зв'язок між базою знань і даних (тобто глобальною базою даних) з одного боку, і зовнішнім оточенням, експертним середовищем, з іншого боку.

Речення (clause) – ППФ, що складається з диз'юнкції літералів: A, B, C.

Резолюції процес – застосовується до двох речень (ППФ) і породжує виведене речення в обчисленні предикатів.

Резолюція двох речень утворюється примусовою диз'юнкцією двох взаємно протилежних літералів P і ~ P, які знаходяться в двох різних реченнях.

Нагадаємо: Резолюція двох речень $P \vee Q$ і $\sim P \vee G$ є $Q \vee G$. З іншого боку, ці речення можна уявити як $Q \vee P$ і $P \Rightarrow G$, при цьому P замінюється на G і отримаємо $Q \vee G$.

Загальний випадок резолюцій – якщо в складі двох речень зі змінними виділяться можливі при деякій підстановці (наприклад, x = y) додаткові літерали (тобто, P (y) і ~ P (x) перетворюються в P (x) і ~ P (x)), то тільки в рамках цієї підстановки можна застосувати правило резолюції і отримати резольвенту.

3.2. Доведення теорем і методи отримання розв'язків

Застосування обчислення предикатів – (1) всі вирази бази даних перетворюються в речення в нормальній формі – ППФ; (2) система управління на основі застосування резолюцій (причому береться заперечення цільової ППФ і доводиться протиріччя, цільова ППФ – теорема, яку потрібно довести); (3) система продукцій є комутативність, що дозволяє застосовувати безповоротний режим управління (де порядок застосування не має значення, а повтори не є небезпечними, проблема лише в скороченні часу розв'язання).

Стратегії вибору застосовуваних речень – формування дерева вибору, – допоміжного графа; пошуку в ширину (повного перебору) і т. ін.

Спрощення –

(1) виключення тавтологій,

(2) оцінка окремих літералів (для основних окремих випадків) за рахунок т. зв. приєднаних процедур (якщо літерал отримує оцінку T, то таке речення виключають, а якщо F, то можна речення залишити, але виключити з нього цей літерал),

(3) виявлення включення одного з речень до складу іншого при певній підстановці змінних (тобто перше речення до підстановки є більш загальним видом відповідної частини другого), при цьому говорять про те, що перше речення підсумує друге (тоді цю частину другого речення можна виключити, бо виключена частина не впливає на незадовольнимість решти).

Конструктивні докази – пошук і визначення значень (окремі випадки) змінних в формулах при доказі теорем в обчисленні предикатів. Тобто, недостатньо довести існування розв'язку в області визначення змінних, потрібно знайти конкретні значення цих змінних (сам розв'язок).

ПРОЦЕС ВИЛУЧЕННЯ З ФОРМАЛЬНОГО РОЗВ'ЯЗКУ (ТУТ ЦЕ СПРОСТУВАННЯ НА ОСНОВІ РЕЗОЛЮЦІЇ) ЗНАЧЕННЯ ЗМІННОЇ, ЩО НАЛЕЖИТЬ ДО КВАНТОРУ ІСНУВАННЯ

Приклад [7]. Завдання: якщо Жора ходить в ті ж самі місця, куди ходить Коля, а Коля в школі, то де ж Жора?

S:. $\forall(x)[B(KOLIA,x) \Rightarrow B(GORA,x)]$; $B(KOLIA, SCHOOL)$.

На основне питання завдання «Де Жора?» можна відповісти, якщо існує розв'язок, інакше кажучи, $(\exists x)B(GORA, x))$ є наслідком ППФ початкової бази знань (***аксіом***). Тут $(\exists x)B(GORA, x))$ – цільова ППФ. Заперечення цільової ППФ: $(\forall x)[\sim B(GORA, x)]$

Тож з'ясуймо, чи є розв'язок.

Важливо зазначити, що навіть без постановки питання можна виявити, що з самих аксіом можна отримати відповідь. Дійсно, з $\sim B(KOLIA, y) \vee B(GORA, y)$ і $B(KOLIA, SCHOOL)$ отримаємо $B(GORA, SCHOOL)$ при відповідній уніфікації.

Тут важливо вказати, що, взагалі кажучи, перша аксіома – це правило. Друга – це факт. Тобто є факт, є правило і визначена мета (ціль) $(\exists x)B(GORA, x))$. Важливо зауважити, що в експертних системах комутативного типу на основі теорії предикатів факти, цілі та правила записуються в подібному вигляді і в процедурі виведення вони часом не відрізняються. Інше, слід зазначити, що правила і факти (тобто вміст глобальної бази даних), взагалі кажучи, можуть самі породжувати нові факти і речення, наприклад факт $B(KOLIA, SCHOOL)$ і правило $\sim B(KOLIA, y) \vee B(GORA, y)$ породжують при $y = SCHOOL$ новий факт, який вже містить відповідь на ще не поставлене запитання. Тобто, навіть не маючи або не знаючи цільову ППФ (не поставивши питання) $(\exists x)B(GORA, x))$ прямим методом дедукції отримуємо деяку потенційну відповідь, яку ми могли б задати, а могли б і не задавати, програма все одно цю відповідь здатна виявити сама. Тобто, інтелектуальні системи самостійно здатні розширювати свою глобальну базу даних, займаючись на дозвіллі (адже питання не задані) самонавчанням.

Але якщо процедура повинна бути заснована на роботі саме з цільовими ППФ (тобто поставлено питання $(\exists x)B(GORA, x))$, то розглянемо проведену нижче схему. **Спростування на основі резолюції:**

Таблиця 3.4

	$\sim B(KOLIA, y) \vee B(GORA, y)$ аксіома 1	$B(KOLIA, SCHOOL)$ аксіома 2
~ B(GORA, x) спростування ППФ, що є ціллю	$\sim B(KOLIA, x)$	*NIL*

Ще раз. Чому тут результат повинен бути порожньою множиною? Наприклад, у нас є певне питання A (x). Якщо в базі даних є подібне речення (літерал) A (a), то значить, на питання ми відповідаємо ствердно, тільки якщо x = a.

Але можна інакше: візьмемо заперечення питання ~ A (x) і складемо диз'юнкцію з наявними в базі даних A (a). Отримаємо при x = a тавтологію, яка після спрощення (її виключення) дасть порожнє речення. Це інший спосіб відповіді на питання.

Але часто саме A (a) в базі немає, але його можна отримати, використовуючи резолюцію щодо до речень з бази даних. Але іноді простіше взяти заперечення питання ~ A (x) і спробувати з масиву речень бази даних отримати порожнє речення при деяких уніфікаціях.

Представлене рішення є доказ теореми і якщо не бачити процедури, за якої проводиться уніфікація змінних (тут це x = a), то це приклад неконструктивної теореми, бо вона говорить про існування розв'язку, але не надає шлях (технологію) розв'язання і його не демонструє. Правда, непрямі відповіді на ці питання можна знайти в роздруківці розв'язання, при розгляді проведених уніфікацій.

Розв'язок існує, але де все ж таки Жора? Додаємо до кожного речення, що виникає із заперечення цільової ППФ його власне заперечення (отримаємо тавтології), і, виконавши ті ж самі резолюції, отримаємо в кореневій вершині відповідь.

Спростування на основі резолюції:

Таблиця 3.5

	$\sim B(KOLIA, y) \vee B(GORA, y)$	$B(KOLIA, SCHOOL)$
$\sim$ B(GORA, x) $\vee$ B(GORA, x) початкова тавтологія	$\sim B(KOLIA, x) \vee B(GORA, x)$	$B(GORA, SCHOOL)$

Отже – це перетворення дерева спростування (з порожньою множиною в кореневій вершині) в дерево доказу з деяким твердженням в кореневій вершині, яке може служити відповіддю.

3.3. Практичні методи логічного висновку

Стратегії управління для методу резолюції. У цьому розділі розглянуто випадки використання глобальної бази даних [7], що представлена лише в формі ППФ, де виключена імплікація (твердження). При цьому всі речення належать до одного класу, неможливо відокремити правила і факти. Це, взагалі кажучи, має свої переваги – все одно, з якого речення починати процес виведення, – та й недоліки, – ефективність системи виявляється нижчою, губляться деякі евристичні складові, властиві імплікації.

Розглянемо стратегії пошуку розв'язку на прикладі наступної задачі:

Дано

1. Хто читає – грамотний $(\forall x)$[Ч (x) $\Rightarrow$Г(x)]
2. Дельфіни неграмотні $(\forall x)$[Д (x) $\Rightarrow\sim$ Г(x)]
3. Деякі дельфіни мають інтелект $(\exists x)$ [Д (x) $\wedge$ И(x)]

Треба довести,

4. Деякі з тих, хто володіє інтелектом, не можуть читати $(\exists x)$ [И (x) $\wedge$ $\sim$ Ч(x)]

Нехай x = A. Замість *Д (A)* $\wedge$ *И(A)* можна записати Д (А) і І (А), Заперечення цільової ППФ це – $\neg$ *И(z)* $\vee$ *Ч(z)* – тоді все зводиться до доведення теореми, і відшукання порожньої множини на певній глибині пошуку

В теорії предикатів, як уже зазначалося, має місце подібність між твердженнями (фактами), метазнаннями (правилами) і цілями в системах доведення теорем. Факти і правила завжди переводяться в форму речень − ППФ. У системах спростування на основі резолюції цільові ППФ замінялися їх запереченнями і перетворювалися в форму речень, тобто в ту ж форму, що і факти. Тобто заперечення мети можна було уявити як факти і правила.

Стратегія пошуку в ширину. Спочатку обчислюють всі резольвенти першого рівня (верхній рядок речень – базова множина, з якої формуються резольвенти першого рівня), потім резольвенти другого рівня (які формуються з резольвент першого рівня) і т. ін. Стратегія повна і тому неефективна.

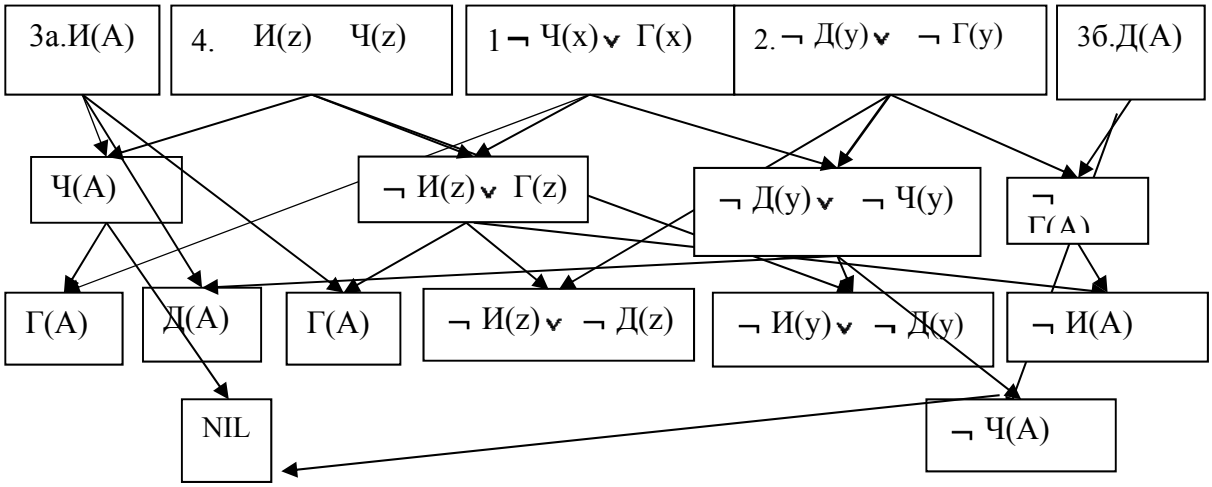


Рис. 3.1. До стратегії пошуку в ширину

Стратегія опорної множини. Опорна множина – це речення, отримані запереченням цільової ППФ чи їхніх нащадків. Опорна множина створює менше речень порівнянно з необмеженою резолюцією в ширину (див. вище). Це дозволяє послабити комбінаторний вибух. Однак така стратегія може збільшити глибину пошуку порожньої множини. На рис. 3.1 вище (що ілюструє необмежену резолюцію в ширину) кількість речень на другому рівні помітно більша, ніж на другому рівні рис. 3.2 нижче (відповідає стратегії опорного речення). Але якщо порожня множина виникає на верхньому рис. 3.1 на третьому рівні, то на подібному рівні нижнього рисунка його ще немає.

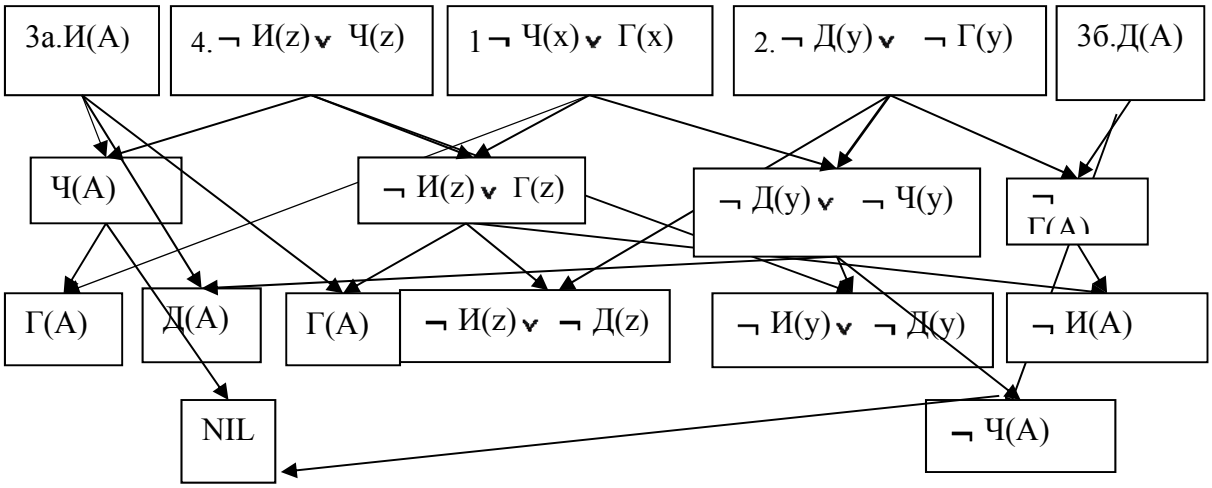


Рис. 3.2. До стратегії опорного речення

Стратегія переваги одночленним виразам. В цьому випадку для батьківських вершин краще вибирати одночлени. Але бажано стартувати із заперечення цільової ППФ.

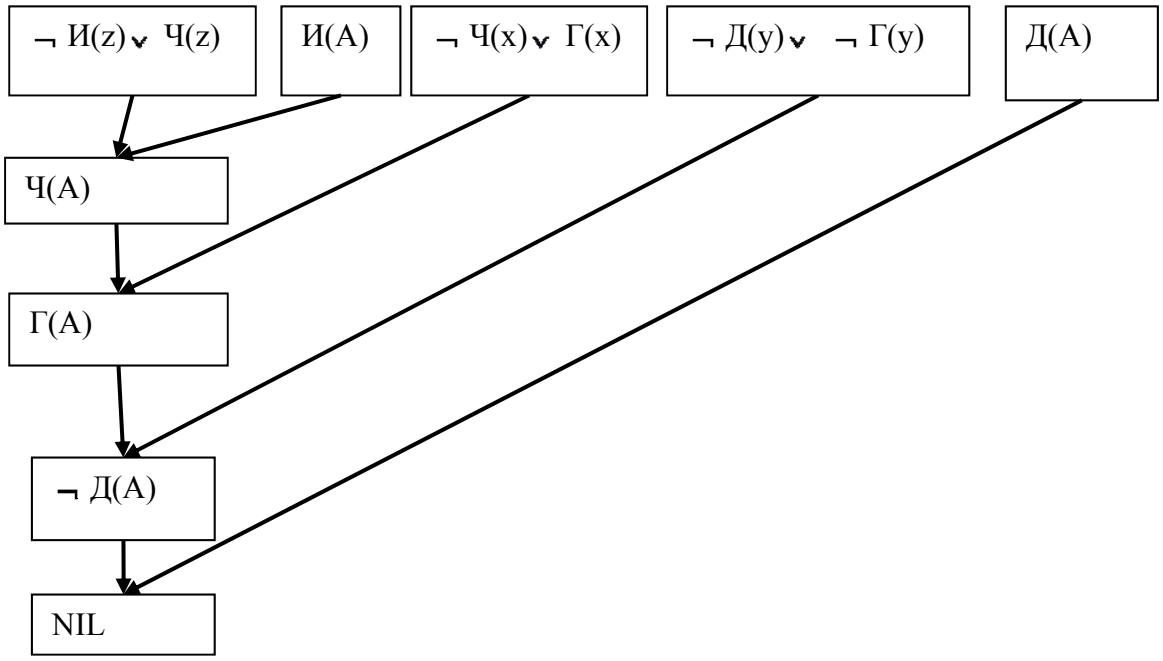


Рис. 3.3. До стратегії переваги одночленним виразам

Лінійна по входу стратегія. Лінійне по входу спростування – спростування, коли одне батьківське речення в резольвенті належить базовій множині.

Ця стратегія не відрізняється повнотою, і, зокрема, не дає розв’язку, якщо в базовому наборі немає однолітерального речення (для породження порожнього речення в цій стратегії потрібно, щоб було однолітеральне речення з базового набору і однолітеральне, як наслідок розв’язання).

Якщо виключити пунктирну криву на наступному малюнку, це твердження стане очевидним.

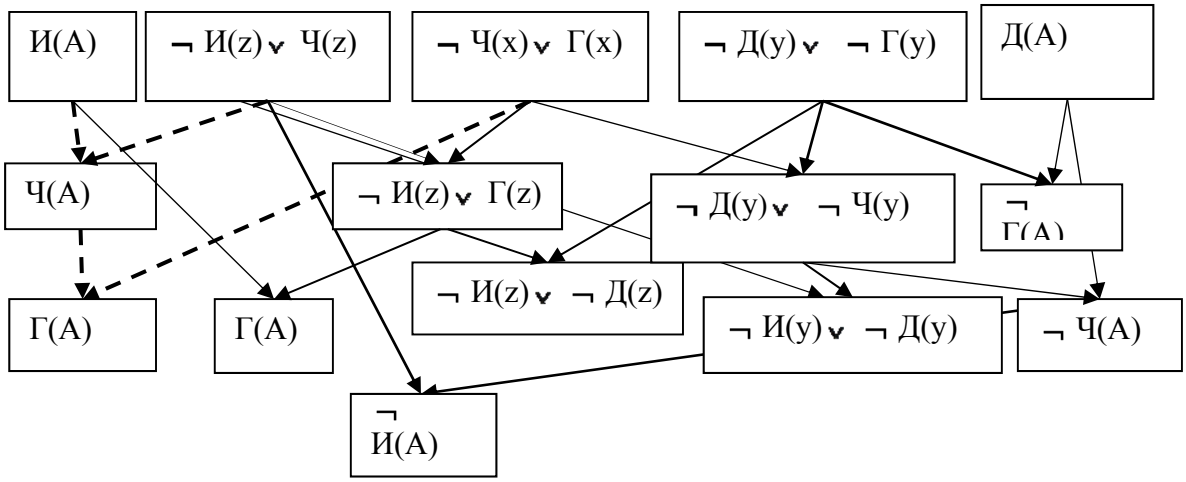


Рис. 3.4. Лінійна по входу стратегія

Стратегія з урахуванням попередніх вершин. Тут потрібно, щоб одне з батьківських речень для резольвенти було або з базової множини (це те ж саме, що і у попередній, лінійної по входу стратегії), або було одним з нащадків іншого батьківського речення (що розширює можливості стратегії і дозволяє домогтися повноти). Саме пунктирна крива зв'язку на рис. 3.5 в цьому випадку дозволяє отримати порожнє речення і довести теорему.

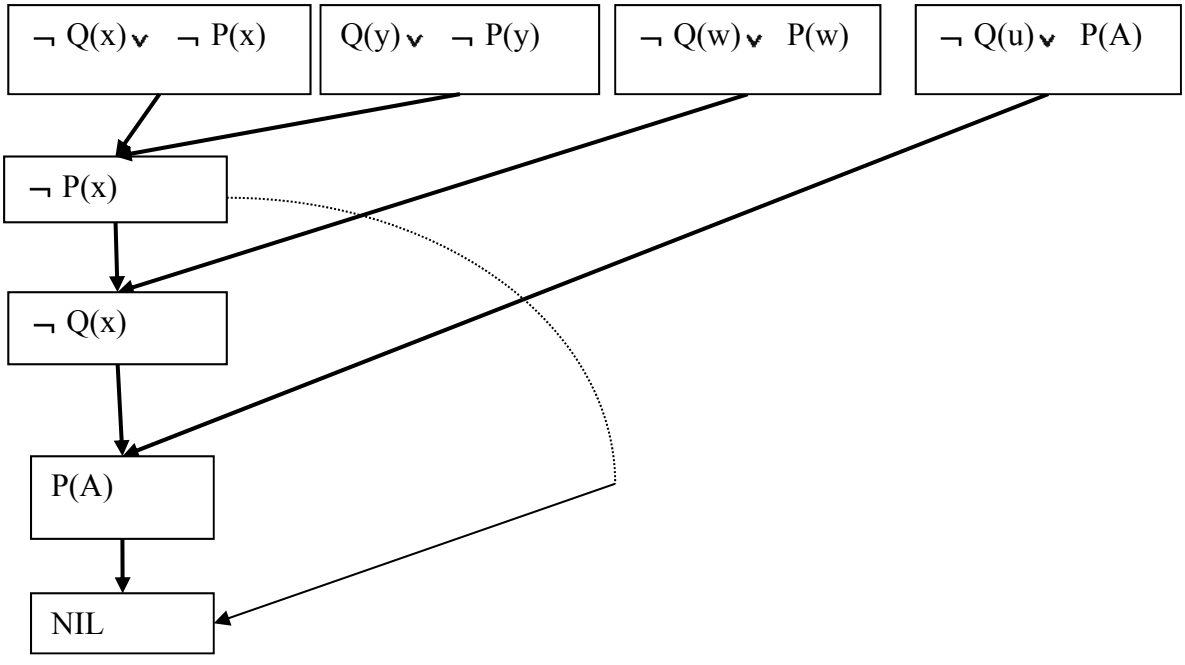


Рис. 3.5. До стратегії з урахуванням попередніх вершин

Стратегія спрощення завдяки виключенню тавтологій. У цьому випадку з базового набору і при подальших операціях можуть бути відкинуті речення, що містять тавтології.

Приклад: повністю виключаються з розгляду такі речення

$$P(x) \vee \sim P(x),$$
$$Q(x) \vee P(y) \vee \sim P(y).$$

Стратегія спрощення завдяки оцінці літерала в реченні. Оцінка деяких літералів може бути проведена незалежно завдяки т. зв. приєднаним процедурам.

Приклад: якщо $Q(x) \vee P(y) \vee E(x = 3, y = 5)$, де $E(x = 3, y = 5)$ де $E(x = 3, y = 5)$ істинне при $x = y$ і хибне в іншому випадку, то можна ввести процедуру (приєднану до предикатного символу E) EQUALS (x, y), і якщо при проведенні процедури для $x = 3$, $y = 5$ отримаємо «хибне», то і літералу $E(x = 3, y = 5)$ теж слід присвоїти значення «хибне». Тільки при цьому замість $Q(x) \vee P(y) \vee E(x = 3, y = 5)$ можна записати $Q(x) \vee P(y)$.

Виняток при підсумовуванні. При розв'язаннях допустимо виключати речення, які при певній підстановці є частиною іншого речення (або збігаються з ним). При цьому підстановка повинна бути врахована в подальшому. Кажуть, що речення, які при даній підстановці є частиною іншого речення, підсумовують це інше речення.

Приклад: речення $P(x) \vee \sim Q(A)$ підсумовує $R(x) \vee P(y) \vee \sim Q(z)$ і може бути відкинуте при $x = y$ і $z = A$.

РОЗДІЛ 4
ГРАФИ

Теорія графів – це розділ дискретної математики, одна з гілок дискретної топології, теорія мереж, наука про топологічні форми, мережеві моделі представлення будь-якого процесу або системи. Основним об'єктом дослідження цієї теорії є графи. Графом називають геометричну схему, що є системою ліній, які пов'язують задані точки.

Родоначальником теорії графів вважається Л. Ейлер. У 1736 році в одному зі своїх листів він формулює і пропонує розв'язок задачі про сім мостів, що стала однією з класичних задач теорії графів. Як наука теорія графів сформувалася після появи першої монографії на цю тему Д. Кьоніга «Теорія скінченних і нескінченних графів» (1936) [16].

Портрет, створений Я. Є. Хандманном

Леонард Ейлер *Leonhard Euler* (1707−1783) – автор фундаментальних робіт з математики, матфізіки, механіки, оптики, суднобудування і теорії музики.
Академік Берлінської, Туринської, Лісабонської і Паризької (іноземний член) наук.

Денеш Кьоніг *Kőnig Dénes* (1884−1944) – докторський ступень (1907). Професор Будапештського університету. Під впливом Г. Мінь-ківського (Hermann Minkowski) написав першу монографію з теорії графів «Теорія скінченних і нескінченних графів» (1936).

4.1. Мережеві представлення

Для структурування (і локалізації структури бази даних або збору інформації по одному об'єкту) можна використовувати графічне зображення в формі ***концептуальних графів.*** Цей граф являє собою логічну формулу. Імена та аргументи предикатів представлені двома типами вузлів. Дуги з'єднують імена предикатів з їх аргументами.

Концептуальний граф. Логіка предикатів – тут інтерпретація в термінах області міркувань (експертизи). Логічні формули – фрази метамови. Аргументи предикатів – атрибути, події, стани. Імена предикатів вказують спосіб зв'язку (правила граматики, правила з'єднання, процедури) між поняттями.

> Соріт (від грецьк. Σωρός − «купа») − ланцюг силогізмів, в яких висновок є однією з посилок наступного за ним.
>
> наприклад:
> Англійці − мужній народ.
> Мужній народ вільний.
> Вільний народ щасливий.
> Отже, англійці щасливі.

Семантичні мережі складаються з множини концептуальних графів. Це уявлення дозволяє візуалізувати множина відносин між концептуальними графами і складовими окремих графів.

4.2. Пошук на графі

Пошук на графі – формування різних застосувань набору правил із загального їх числа й породжених цими застосуваннями баз даних утворюють ***дерево пошуку***. У корені дерева – опис вихідної конфігурації. Правила – дуги, які ведуть до вершин-наступників (нащадків).

Тут вершини відповідають базі даних, а дуги – правилам. Дуги спрямовані від батьків до наступників (***спрямований граф***). ***Дерево*** – окремий випадок графа, кожна вершина якого має не більше одного з батьків. Якщо дві вершини одночасно наступники один одного, то пара дуг замінюється ***ребром***. Вершина, яка не має батьків, – ***кореневa***, а та, що не має наступників, – ***кінцева***. Кореневій вершині відповідає нульова глибина, глибина будь-якої іншої вершини дорівнює глибині попередника плюс одиниця. Послідовність вершин, кожна наступна з яких є попередником попередньої, називається ***шляхом***, з довжиною, рівною кількості вершин в послідовності.

Для одного шляху можна ввести поняття ***нащадка*** і ***предка***, а також визначити ***досяжність*** однієї вершини з іншої. Кожній дузі можна визначити значення різних характеристик, наприклад, ***вартості***. Це дозволяє в системах управління розглядати, наприклад, ***сумарну вартість*** шляху і вимагати її оптимізації. Множину вершин, що представляють бази даних

термінальної умови, називають цільовою, і кожну вершину з цієї множини – цільовою. Граф може бути заданий явно або неявно (якщо породжується самою стратегією управління, тобто задається цією стратегією явно). Можна ввести ***оператор побудови наступників*** – застосування цього оператора до окремої вершини виявляє всіх наступників (процес застосування називають ***розкриттям*** цієї вершини).

4.3. Стратегія управління

Стратегію управління з пошуком на графі можна також розглядати як процес виявлення частини ***неявного графа***, що містить цільову вершину. Дерево (швидше кущ) зростає до тих пір, поки одна з його гілок не задовольнить термінальну умову, причому, взагалі кажучи, така стратегія досить неефективна. Таким чином, на відміну від попереднього випадку (тобто режиму з поверненням) система пам'ятає всі попередні шляхи, окремі кроки кожного шляху і сформовані в результаті цих кроків модифіковані бази даних.

Перевага такої ситуації в тому, що система може стартувати не з початкової позиції, а з тієї модифікованої бази даних, яка виявилася б найбільш близькою до результату задачі, якщо в процедурі пошуку існують критерії близькості проміжного стану до результуючого стану. Корисно, з іншого боку, знайти умови (взагалі кажучи, евристичні), які все ж обмежували б кількість гілок, робили дерево (кущ) більш вузьким.

4.4. Розвиток системи

Неінформовані процедури – процедури, які не мають евристичної інформації для упорядкування вершин і оцінки характеристик дуг при процесах формування неявного графа. Для неінформованих процедур (як правило, вони рідко використовуються в системах штучного інтелекту) існують методи: пошук в глибину з введенням обмежень на глибину проходить кожен раз до граничної глибини, потім повертається в найвищу точку розгалуження і знову на саму глибину тощо (зазвичай замість цього використовують режим з поверненням – він менш ресурсномісткий, і пошук в ширину проходить шарами на кожному рівні глибини).

Евристичні методи – методи евристичного пошуку – використання додаткової інформації для скорочення пошуку. Засновані на мінімізації комбінації вартості шляху до мети (сумі величин, що характеризують елементи-дуги на шляху до мети) і вартості (обсягу обчислень) пошуку цього шляху. Для цього вводяться т. зв. ***оціночні функції***, на підставі яких деякі вершини дерева пошуку викидаються з розгляду.

Двонаправлений пошук – два фронти формування пошуку від початкової і від цільової вершин в разі неінформованих процедур зустрічаються швидше і формують найбільш короткий шлях. У разі евристичних проце-

дур відбувається спотворення фронтів – вони можуть зустрітися далеко на периферії або в разі обмежень на глибину пошуку взагалі не зустрітися.

Поетапні пошуки – пошуки «з обраної плаваючої точки». Використовують в разі обмежених ресурсів або в разі великих графів для випадків евристичного пошуку. Інформація першого етапу запам'ятовується, і старт другого відбувається з найбільш прийнятної (з позицій евристики, оптимального значення оціночної функції) досягнутої на першому етапі вершини.

Характеристики якості роботи – спрямованість пошуку P = L / T, де L – довжина шляху до мети, T – загальне число породжених вершин. ***Показник ефективності розгалуження*** – середнє число наступників окремої вершини дерева. Оцінити можна з рівняння $B + B^2 + ... + B^L = T$ або, підсумовуючи, отримаємо неявний вид $T = B(B^L - 1)/(B - 1)$.

4.5. Версія обчислення предикатів

Подання знань у вигляді графа. Графи типу І / АБО. Використання графів в обчисленні предикатів і організації логічного висновку часто дозволяє спростити і скоротити процедури. Але слід пам'ятати, що подання у формі графів часто не дозволяє перейменовувати змінні, як це було можливим раніше.

У цьому розділі ми відмовимося від переводу знань у формі імплікації у речення, які виключають імплікації. Тоді всі ППФ, поділяються на правила і факти. ППФ в формі імплікацій – це загальні знання про предметну область, і вони використовуються як такі, що породжують правила. Факти, які не мають імплікативної форми, – це специфічні знання, що належать до даного конкретного випадку.

Попереднє перетворення: Форма (безімплікативна) ППФ, при необхідності після сколемівского перетворення, переводиться в префіксну форму, змінні в межах дії кванторів перейменовуються, квантори спільності опускаються [7].

Приклад попереднього (що передує) перетворення

Розглянемо вираз, $(\exists u)(\forall v)\{Q(v,u) \wedge \sim[[R(v) \vee P(v)] \wedge S(u,v)]\}$ який неважко перетворити спочатку в $Q(v, A) \wedge [[\sim R(v) \wedge \sim P(v)] \vee \sim S(A, v)]$, а потім, щоб змінні відрізнялися, в форму І / АБО (тобто з включенням кон'юнкцій і диз'юнкцій).

$$Q(w, A) \wedge [[\sim R(v) \wedge \sim P(v)] \vee \sim S(A, v)]$$

Схема побудови графа І / АБО (часто такий граф відносять до гіперграфів) в розглянутій вище правильній формі досить проста, кон'юнкція представляється у вигляді традиційних зв'язків, а диз'юнкція у вигляді к-зв'язок, тобто об'єднаних зв'язок. Зазначимо, що таке уявлення гіперграфу

зазвичай характерно для *прямої продукції*. Взагалі кажучи, для зворотної продукції, як показано нижче, краще змінити форму представлення графа.

Приклад побудови графа І / АБО

Важливо вказати, що тут зустрічаються кон'юнкція і диз'юнкція. Побудуємо граф І / АБО

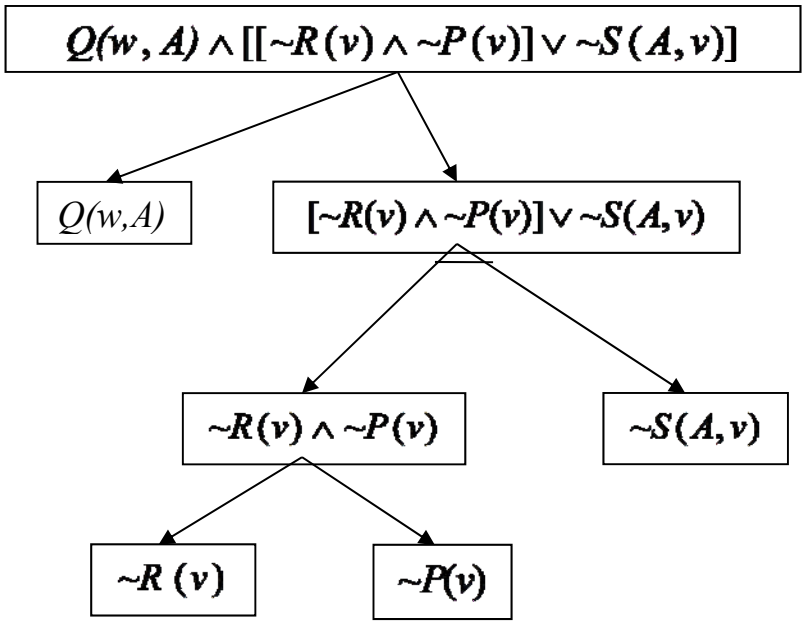


Рис. 4.1. Гіперграф тверджень І / АБО

Диз'юнктивно пов'язані речення $\sim R(v) \wedge \sim P(v)$ і $\sim S(A, v)$ в описі графа містять так звану к-зв'язку (тут к = 1). Множина речень, в які ця ППФ може бути перетворена, є множиною літералів (зрозуміло, кон'юнктивно пов'язаних між собою) на кінцевих вершинах графа, тобто

$$Q(w, A),$$
$$\sim R(v) \vee \sim S(A, v),$$
$$\sim P(v) \vee \sim S(A, v),$$

Зазначимо, що подання знань у вигляді графа є менш загальним, ніж подання у вигляді цих трьох речень. Через існуючу в останньому випадку довільність змінної *v* в різних реченнях. Тобто останній вираз, наприклад, можна (в останньому випадку комутативної системи знань) записати інакше $\sim P(w) \vee \sim S(D, w)$.

Приклади застосування правил і отримання розв'язків на графі.

Тип доведення: тут буде використовуватися безпосереднє доведення (дедукція, заснована на правилах), а не система спростування, як раніше в розділах 2, 3. Пряма система доказів – застосовуються П-правила для

досягнення умови зупинки, що включає цільову ППФ. Зворотна система – використання О-правил до досягнення умови зупинки, яке включає початкові умови-факти.

Пряма система продукцій

Подивимося, як можна після застосування до даного гіперграфу правила, наприклад, такого

$$\sim S(x, y) \Rightarrow R(x) \wedge Q(y),$$

отримати розв'язок. Підставимо правило у граф тверджень. При цьому слід застосувати відповідні уніфікатори для змінних.

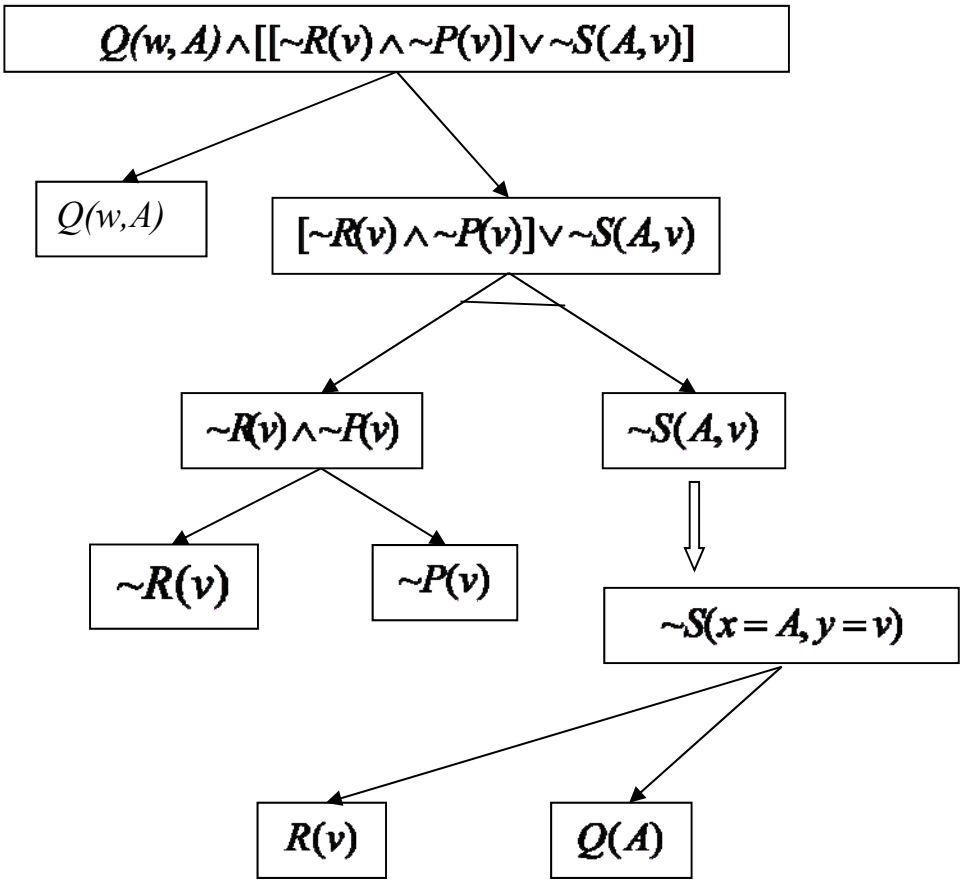


Рис. 4.2. Приєднання правила до графу фактів (пряма система продукцій)

Відповіддю, тобто після застосування до графа фактів правила S, буде набір диз'юнкцій кореневих виразів $\sim R(v) \vee Q(A)$, $\sim P(v) \vee Q(A)$, $\sim R(v) \vee R(v)$, $R(v) \vee \sim P(y)$.

Завдання:

1. Який результат з отриманих вище потрібно виключити?
2. Покажіть, що станеться при застосуванні правила $\sim S(x, y) \Rightarrow \sim R(x) \wedge Q(y)$.

Використання цільової ППФ для зупинки. Якщо припустити, що цільова ППФ є $\sim P(v) \vee Q(A)$, то при застосуванні правила одним з розв'язків може бути саме це речення, що і зупинить програму.

При застосуванні все нових правил або даних для заданого набору фактів в прямій системі продукцій генеруються нові речення, тобто нові знання. Тобто пряма система продукцій найбільш пристосована для подальшого заповнення глобальної бази даних. Іншими словами, вона пристосована для самонавчання.

Зворотна система продукцій

Тут будемо використовувати графи І / АБО, де к-зв'язки потрібні для позначення речень, що являють собою кон'юнкції літералів, наприклад $L_1 \wedge L_2$ [15].

1. Так як правила мають вигляд $W \Rightarrow L$, *то при такому опису все одно зводяться до* $W \Rightarrow L_1$ *і* $W \Rightarrow L_2$, *що полегшує аналіз.*

2. Можна приводити правила до їх заперечення з виключенням імплікації $W \Rightarrow L_1$ *тобто заміни її на* $\sim W \vee L_1$, *і потім переходу до заперечення, що дає* $W \wedge \sim L_1$.

У цій системі продукцій раціонально використовувати гіперграфи іншого виду, де ***кон'юнкція описана к-зв'язками, а диз'юнкція не використовує зв'язок.***

ПРИКЛАД: Доведемо мету $P(\mathrm{z}) \wedge Q(x)$, фактами є $R(A)$ і $Q(A)$, а правила такі: П1: $R(y) \Rightarrow P(y)$ і П2: $S(z) \Rightarrow P(B)$.

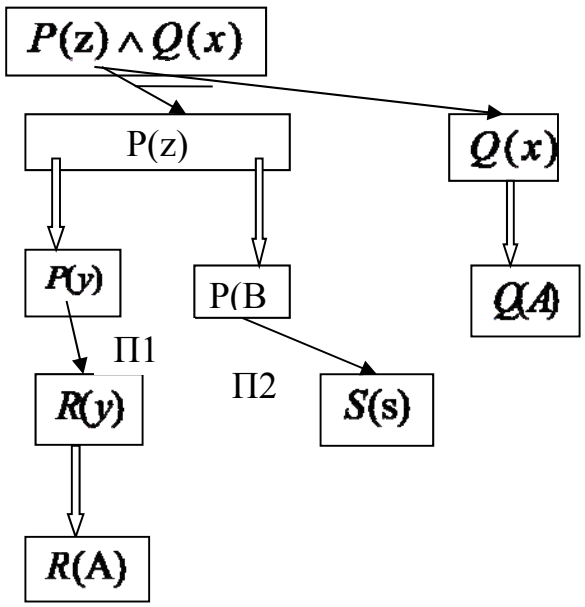


Рис. 4.3. Виключення гілки графа при зворотній системі продукцій.

[15] Зрозуміло, що перейменовувати змінні можна лише в різних реченнях, які являють собою кон'юнкції.

РОЗВ'ЯЗАННЯ: $P(z) \wedge Q(x)$ насправді об'єднана мета, причому кожен літерал також є метою і ці дві підцілі $P(z)$ і $Q(x)$ обидві повинні бути неодмінно досягнуті[16]. Правила тут простіше застосовувати у вигляді імплікації. Тобто правила застосовуються в зворотному порядку, наприклад, до літералу $P(z)$ який є першою підціллю застосовуємо розвернене навпаки правило $P(y) \Leftarrow R(y)$ за умови $z = y$, тоді нова підціль $R(z)$ і при $z = A$ досягається, оскільки в базі даних був факт $R(A)$. На правій гілці графа підціль $Q(x)$ аналогічно досягається при підстановці $x = A$, так як є факт $Q(A)$. Цікаво, що гілка графа побудована на правилі П2 формуючи нову мету $P(B)$, ніяк не досягається, але оскільки на спільне рішення це не впливає – ця гілка просто відкидається. Іншими словами, для вирішення цього правила П2 не потрібно.

Лівий частковий граф – підстановка A = y = z.
Середній частковий граф – підстановка B / z.
Правий частковий граф – підстановка A / x.

Відкидання неузгоджених часткових графів. Стратегія управління зворотною системою продукцій заснована на виявленні узгодженого графа. Неузгоджені часткові графи при цьому можуть бути відкинуті, що зменшує при цьому обсяг перебору варіантів. Вибір структури графа, де кон'юнкція описана зв'язками, а диз'юнкція – без них, дозволяє наочно ігнорувати деякі гілки графа, які не є узгодженими. Розв'язок включає ліву і праву гілки графа, а середню, що не узгоджується, – видалимо.

4.6. Основи мови ПРОЛОГ

ПРОЛОГ – мова, успадкована від математики»
(mathematically derived languages)

А. Колмерауэр (*Alain Colmerauer)*
Захистив дисертацію в університеті Гренобля, працював професором в університеті Монреаля (де розробив т. зв. Q-system), потім в університеті Марселя. Творець мови ПРОЛОГ [17].

[16] Тому тут є к-зв'язка. Вона ніби пов'язує обидві підцілі в одну спільну мету. Це в якійсь мірі пояснює чому обрана саме така форма подання гіперграфу.

Технології опису (теорії, мови), засновані на математиці або на інших логічно несуперечливих формальних методах, дозволяють при правильній постановці задачі і коректному застосуванні процедур домогтися несуперечливого і визначеного результату. Причому можна іноді *навіть не дбати про інтерпретацію проміжних викладок і уявлень*.

ЗВ'ЯЗОК З ТЕОРІЄЮ ПРЕДИКАТІВ

Прологівскі речення бувають трьох типів: факти, правила і питання.

• ***Факти*** містять твердження, які є завжди безумовною істиною.

• ***Правила*** містять твердження, істинність яких залежить від деяких умов.

• За допомогою ***питань*** користувач може запитувати систему про те, які твердження є істинними [18−20].

Речення Прологу складаються з *голови* і *тіла*. ***Тіло*** – це список цілей, розділених комами. ***Факти*** – це речення, що мають пусте тіло. ***Питання*** мають тільки тіло. Правила мають ***голову*** і (непорожнє) тіло.

	голова	**тіло**
Факти	батько (том, боб).	
	батько (боб, пат).	
Правила		
`пр1.`	предок (X, Z) :− батько (X, Z).	
`пр2.`	предок (X, Z) :−	
	батько (X, Y),	
	предок (Y, Z).	
Питання		
`(ціль, мета)`	? − предок (том, пат).	

ПОДІБНІСТЬ

1. Всі факти – в кінці крапка. Факт – власні імена та назви! Змінні – імена загальні[17]!

2. Диз'юнкція – крапка з комою.

3. Кон'юнкція – кома, має пріоритет перед диз'юнкцією.

4. Зіставлення в Пролозі відповідає деякій дії в логіці, яка називається *уніфікацією*.

ВІДМІННОСТІ І ОСОБЛИВОСТІ

1. **Правила використовують імплікацію** (раніше в теорії предикатів від імплікації відходили, тут вона залишається).

2. **Резолюція.** У ПРОЛОЗІ застосовують зворотну продукцію. Створення нового речення з мети (цілі) G (A) і правила P (y) ⇒ G (y). Тут голова правила G (y), а тіло правила (умовна частина) P(y). При збігу мети (цілі) G(A) – тіла і голови правила G (y = A) обидві прибираються

[17] В людських мовах це назви груп об'єктів, що мають однакові характеристики. Тут це просто невизначені величини.

і залишається тіло правила P(y = A) з заміною змінної (конкретизація). Це тепер нова мета. Розглянуте вище еквівалентно процедурі

$$G(x)|_{x\to y} \vee -G(y) \vee P(y) \Leftrightarrow P(y)$$

3. **Квантори** – представлені неявно.

4. **Комментарі** / * Це коментар * / % Це теж коментар

5. **Як задають питання?**

а) Просте запитання:

? − батько (боб, пат).

б) «Питання – відповідь»:

? − Дід (альфонс, юля)

− −> так (це фактично доведення теореми)

Питання

? − книга (Гюго, X, Y). (Це фактично доведення теореми, але модифіковане)

− −> X = знедолені, Y = вид (пош., 1984)

− −> X = Ернані, Y = вид (Ґалімар., 1974)

− −> немає (це вирішення проблеми зупинки подібних операцій)

Питання (інакше поставлене – чи є книги Гюго?)

? − книга (Гюго, _____, _____).

− −> так

Питання (для отримання списку авторів)і

? − книга (A, _____, _____).

− −> A = грем

− −> A = конділяк

− −> немає

6. **Зворотна дедукція.** Замість того, щоб починати з простих (заданих) фактів, наведених в програмі, система починає з цілей і, застосовуючи правила, підміняє поточні цілі новими, до тих пір, поки ці нові цілі не виявляться простими (заданими) фактами. Тобто це зворотна дедукція, що дозволяє в умовах невпевненості в однозначності розв'язку, забезпечити виконання саме зазначеної задачі.

7. **Не все програмується.** Процес, в результаті якого ПРОЛОГ-система встановлює, чи задовольняє об'єкт запиту, часто досить складний і включає в себе логічний висновок, дослідження різних варіантів і, можливо, повернення.

ВСЕ ЦЕ РОБИТЬСЯ АВТОМАТИЧНО САМОЮ ПРОЛОГ-СИСТЕМОЮ І ЗДЕБІЛЬШОГО ПРИХОВАНО ВІД КОРИСТУВАЧА.

8. **Щодо істинності цілей.** Кома між цілями позначає кон'юнкцію цілей: вони всі повинні бути істинними. Однак в ПРОЛОЗІ можлива і диз'юнкція цілей: повинна бути істинною, принаймні одна з цілей. Диз'юнкція позначається крапкою з комою. Наприклад:

P: – Q; R.

читається так: P – істинне, якщо істинне Q або істинне R.

Сенс такого речення той же, що і сенс наступної пари речень:

P: – Q.

P: – R.

9. **Порядок виконання операцій:**

P: – Q, R. % P – істинне, якщо Q і R істинні. З Q і R випливає P.

Щоб розв'язати задачу P, спочатку розв'яжіть підзадачу Q, а потім – підзадачу R. Щоб досягти P, спочатку досягніть Q, а потім R. Таким чином, різниця між "декларативним" і "процедурним" прочитаннями полягає в тому, що останнє визначає не тільки логічні зв'язки між головою речення і цілями в його тілі, але ще і ***порядок***, в якому ці цілі обробляються.

10. **Обчислення цільових тверджень ПРОЛОГу**. Кожного разу, коли рекурсивний виклик процедури «обчислити» призводить до неуспіху, процес обчислень повертається до ПЕРЕГЛЯДУ і триває з того речення C, яке використовувалося останнім. Оскільки застосування речення C не призвело до успішного завершення, ПРОЛОГ-система повинна для продовження обчислень спробувати альтернативне речення.

Насправді система анулює результати частини обчислень, що призвели до неуспіху, і здійснює повернення в ту точку (речення C), в якій ця неуспішна гілка починалася. Коли процедура здійснює повернення в деяку точку, всі конкретизації змінних, зроблені після цієї точки, анулюються.

Такий порядок забезпечує систематичну перевірку ПРОЛОГ-системою всіх можливих альтернативних шляхів обчислення доти, доки не буде знайдено шлях, що веде до успіху, або ж доти, доки не виявиться, що всі шляхи приводять до неуспіху.

ЗВ'ЯЗОК З ГРАФАМИ І / АБО

ПРИКЛАД ЯК ПРОЛОГ РОЗВ'ЯЗУЄ[18]

Факти

```
батько (том, боб).
батько (боб, пат).
```

Правила

```
ПР1.  предок (X, Z) :- батько (X, Z).
ПР2.  предок (X, Z) :-
                батько (X, Y),
                предок (Y, Z).
```

Питання (ціль)

```
? − предок (том, пат).
```

Система спробує досягти цієї мети. Для того, щоб це зробити, вона намагається знайти таке речення в програмі, з якого негайно випливає згадана мета (ціль). Очевидно, єдиними реченнями є ПР1 і ПР2.

[18] Будемо використовувати ілюстрації з сторінки: Программирование на языке Пролог для искусственного интеллекта [Електронний ресурс] – Режим доступу: http://wm-help.net/lib/b/book/60837740/9

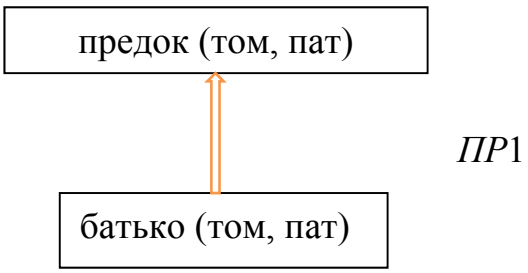


Рис. 4.4. Перший крок обчислень (зворотна продукція)

Спочатку система пробує речення, що стоїть в програмі першим:

предок (X, Z): – батько (X, Z)

Оскільки мета (ціль) – предок (том, пат), значення змінних

X = том, Z = пат.

Тоді вихідна мета (ціль) предок (том, пат) замінюється новою метою:

батько (том, пат)

АЛЕ!

У програмі немає фактів і правила, голови яких можна було б порівняти з метою батько (том, пат), тому така мета виявляється неуспішною.

ПОВЕРНЕННЯ

до вихідної мети, щоб спробувати другий варіант виведення мети верхнього рівня

предок (тому, пат).

Правило ПР2: предок (X, Z): − батько (X, Y), предок (Y, Z).

Як і раніше, змінним X і Z приписуються значення

X = том, Z = пат.

У цей момент змінній Y ще не приписано ніякого значення. Верхня мета предок (том, пат) замінюється двома цілями:

батько (том, Y),
предок (Y, пат).

Цей крок обчислень показаний нижче (праворуч) з урахуванням попередньої невдачі (зліва).

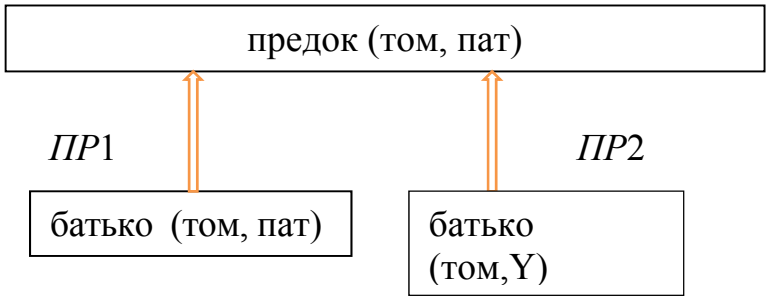


Рис. 4.5. Другий крок обчислень (зворотна продукція)

Тепер, маючи перед собою дві цілі, система намагається досягти їх в тому порядку, в якому вони записані. Досягти першої з них легко, оскільки вона відповідає факту батько (том, боб) з програми. Процес встановлення відповідності – зіставлення (уніфікація) викликає

Y = боб.

Тим самим досягається перша ціль

батько (том, боб),

а решта перетворюється (через уніфікації) в

предок (боб, пат)

Для досягнення цієї цілі знову застосовується правило ПР1. Відмітимо, що це (друге) застосування правила ніяк не пов'язане з його першим застосуванням. Тому система ***використовує нову множину змінних правила щоразу, як воно застосовується***.

Щоб вказати це, ми перейменуємо змінні правила ПР1 для його нового застосування в такий спосіб:

предок (X ‘, Z’): − батько (X ‘, Z’).

Голова цього правила повинна відповідати нашій поточній цілі предок (боб, пат). Тому поточна ціль замінюється на батько (боб, пат), тому ПР1.

предок (X ‘, Z’): − батько (X ‘, Z’).

Так як голова правила має вигляд предок (боб, пат), ціль – тіло правила підходить другому факту

батько (боб, пат)

Всі відповіді узгоджені[19]. X ‘= боб, Z’ = пат

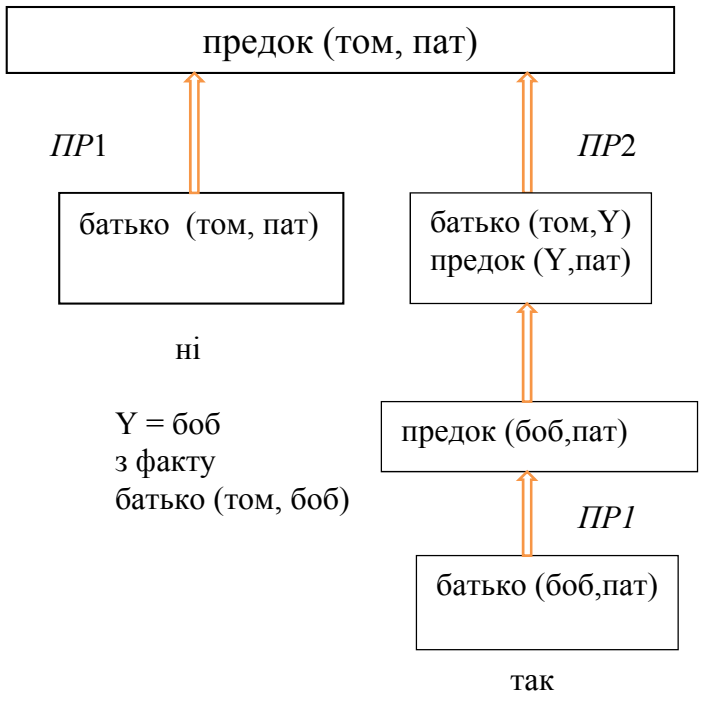


Рис.4.6. Граф завдання (зворотна продукція)

ВИСНОВКИ:

1. Мова ПРОЛОГ – мова, що успадкована від математики (mathematically derived languages).

2. Для розв'язування задач використовується ТЕОРІЯ ПРЕДИКАТІВ. Виведення нових речень засноване на методі РЕЗОЛЮЦІЇ.

[19] Нагадаємо, зворотна дедукція фактично часто полягає в узгодженні лише деяких гілок, що з'єднують цілі і факти.

3. В цю мову вбудована ***прихована від користувача*** програма зворотної дедукції з формуванням на кожному етапі досягнення нових цілей за допомогою резолюції.

4. У мову також ***вбудований прихований від користувача механізм повернення***: якщо виклик процедури «вирахувати» призводить до неуспіху, процес обчислень повертається до перегляду і триває з того речення, яке використовувалося останнім. Для продовження обчислень використовується альтернативне речення.

5. Фактично програма користувача складається з фактів, правил і питань.

Слід звернути увагу на аналогії в розв'язанні оберненої задачі на гіперграфах І / АБО і способом розв'язання задач в техніці мови ПРОЛОГ. Видалення неузгоджених гілок графа, побудованого для зворотної продукції (див.рис. 4.3), внаслідок неузгодженості, відповідає виключенню деяких фактів при розв'язанні задач ПРОЛОГ-системою.

Креативність як наслідок асоціативних зв'язків

У нейтронних мережах кожний розв'язок формує стійкий, принаймні деякий час граф розв'язань, що активно підключає частину елементів нейронної мережі з певними значеннями зв'язків і локальних потенціалів. Крім того, в оточенні такого більш-менш локального графа ініціюються нехай слабше сусідні нейрони. Те ж саме відбувається і при великій кількості розв'язків, які створюють мережу таких графів-зв'язків і потенціалів, накладених частково один на одного. Це накладення, по суті, пов'язує ці розв'язки, формуючи асоціативні зв'язки між близькими і можливо більш далекими розв'язками, що визначається ступенем перекриття їх графів. Зрозуміло, що асоціативні зв'язки здатні підсилити креативність системи.

У системах логічного висновку, взагалі кажучи, при вирішенні цілої низки завдань також може (за певних умов і певної архітектурі) формуватися множина графів, що є примхливими логічними зв'язками фактів і правил. На перший погляд важко очікувати, що ця система дозволить створити асоціативні зв'язки між віддаленими розв'язками. Принаймні це може послабити вимоги до однозначності розв'язку, що не подобається математикам. Але якщо розглядати ситуацію мозкового штурму, пошуку відповідей, то такий зв'язок був би корисним, посилюючи асоціації між розв'язками. Якщо в нейронної мережі зв'язки виникають в самій її структурі, то в експертній системі такі зв'язки можна було б забезпечити завдяки спільному використанню фактів і правил з глобальної бази даних для вирішення низки завдань. Тому не треба боятися ставити інтелектуальній системі велике число задач, можливо, це якраз дозволить посилити її креативні можливості. Як у людей, змушуючи людину розв'язувати багато задач, можна побачити, як росте її інтелект, який оцінюють зазвичай здатністю до виявлення несподіваних розв'язків, що і називають креативністю.

РОЗДІЛ 5
СИСТЕМА ФРЕЙМІВ

Фрейм – поняття, що означає в загальному вигляді смислову рамку, яка використовується для розуміння набору об'єктів і дій.

Система фреймів – як модель подання знань, добре відображає концептуальну основу організації пам'яті людини, а також характеризується гнучкістю і наочністю.

Фрейм (англ. frame – каркас або рамка) запропонований М. Мінським в 70-ті рр. як структура знань для сприйняття просторових сцен. Зараз під фреймом розуміють абстрактний образ або ситуацію.

Основний внесок у створення теорії фреймів зробив М. Мінський [21], до речі, однокурсник творця першого персептрона Ф. Розенблатта.

Ма́рвін Лі Мінський *Marvin Lee Minsky* (1927−2016) дисертації в Гарвардському і Прінстонському університетах, професор Массачусетського технологічного інституту

5.1. Структура фрейму

Ім'я фрейму: (ім'я 1-го слоту: значення 1-го слоту), (ім'я 2-го слоту: значення 2-го слоту), ... (ім'я N-го слоту: значення N-го слоту).

Подання у вигляді таблиці 5.1 нижче, яка доповнена двома стовпцями.

Таблиця 5.1

Ім'я слоту	Значення слоту	Спосіб отримання значення	Приєднана процедура

Додаткові стовпці призначені для опису способу отримання слотом його значення і можливого приєднання до того чи іншого слоту спеціальних процедур, що допускається в теорії фреймів. В якості значення слоту може виступати ім'я іншого фрейму; так утворюють мережі фреймів.

Фрейми-зразки, або прототипи, зберігаються в базі знань.

Фрейми-екземпляри створюються для відображення реальних ситуацій на основі даних, що надходять.

Процедура порівняння (зіставлення). *Слоти, задані за замовчуванням можуть коригуватися (міняти значення) і не виключаються з процесу порівняння. Якщо немає відповідності, то переходять до іншого фрейму. Пов'язані (в кожному слоті є задана умова, яка повинна виконуватися при встановленні відповідності між значеннями) фрейми – вони самі є відносинами, – формують фреймову систему. Такі системи вже використовують для вирішення завдань формування звітів і розрахунків, а також для обробки зображень ...*

Модель фрейму є досить універсальною, оскільки дозволяє відобразити все різноманіття знань про світ через:

- фрейми-структури, для позначення об'єктів і понять (позика, застава, вексель);
- фрейми-ролі (менеджер, касир, клієнт);
- фрейми-сценарії (банкрутство, збори акціонерів, святкування іменин);
- фрейми-ситуації (тривога, аварія, робочий режим пристрою) та ін.

Найважливішою особливістю теорії фреймів є запозичене з теорії семантичних мереж ***успадкування властивостей***. І у фреймах, і в семантичних мережах успадкування відбувається за АКО-зв'язків (A−Kind−Of = це). Слот АКО вказує на фрейм більш високого рівня ієрархії, звідки неявно успадковуються, тобто переносяться, значення аналогічних слотів.

Спеціальні мови представлення знань в мережах фреймів FRL (Frame Representation Language).

Широко відомі такі фреймо-орієнтовані експертні системи, як ANALYST, МОДІС.

Теорія фреймів поки більше відповідає теорії постановки завдань, ніж результативній теорії.

5.2. Опис знань за допомогою фреймів

1. ***Взаємопроникнення*** структур даних і структур процедур.

2. Закони виведення через це ***позбавляються формальної строгості*** правил виводу (подібних логіці предикатів).

3. Досягнута зате велика ***ефективність обчислень*** (завдяки ослабленню вимог повноти та ін.).

Об'єктне уявлення можна отримати з логічного або мережевого уявлень. З логічних формул, що містять одні й ті ж конкретизації збирають фрейми. Це можна зробити зібравши, наприклад, всі бінарні предикати, які відносяться (належать) до даного об'єкту. Це реалізовано в сімействі об'єктно-орієнтованих мов.

Зчіпка (unit) – множина всіх фактів про даний концепт. Нагадаємо, концепт (даного типу) можна визначити «множиною його можливих застосувань».

Фрейми (slot-and-filler notation) – зчіпки, де всі фрази виражені бінарними предикатами.

Слот – пара (атрибут, значення) фрейму.

Явний фрейм (case–frame) – фрейм, в якому представлені конкретні значення аргументів і явні імена формул.

Функціональний фрейм – використовуємо функціональні уявлення:

ПРИКЛАД:

Фраза: «Вася посилає цукерку Маші», яка записується в формі фрейму

Таблиця 5.2

Посылка_10		
елем	:	*(елем_з посилок)*
відправник	:	*(Вася_1)*
отримувач	:	(Маша_10)
об'єкт	:	*(Цукерка_10)*

Тут елем – це множина (посилок).

Квантифікація змінних:

Речення «Вася посилає цукерку кожній жінці» описується бінарними предикатами:

$\forall$ x$\exists$ y$\exists$ z [Відправник (z, Вася_1) $\wedge$ Отримувач (z, x) Об'єкт (z, y) $\wedge$ Елем (z, посилки)

Конкр (y, цукерка) $\wedge$ конкр (x, жінка)

Тут Елем – це множина (посилок).

Функціональний фрейм в області дії квантора спільності набирає вигляду

Таблиця 5.3

z (x)		
елем	:	*(елем_з посилок)*
відправник	:	*(Вася_1)*
отримувач	:	*x*
об'єкт	:	*y(x)*

Уніфікація двох об'єктних уявлень (matching operation) – існують підстановки для змінних, що роблять логічні формули двох об'єктних уявлень ідентичними.

Для використання об'єктного уявлення у якості БД системи запитів (query system), для побудови мови запитів (query language). Йдеться про те, щоб *логічна формула «об'єкта-цілі» була уніфікована з одним із співмножників «об'єкта-факту»* (нагадаємо, останній представлений у вигляді кон'юнкції гіпотез і аксіом).

ПРИКЛАД: Підстановка {(z, Посилка_10), (x, Маша_10), (y, Цукерка_10)} уніфікує наведені вище функціональні фрейми.

Атрибут – конкретизує деякі дані або функціонально пов'язує ці дані (наступного фрейму) з іншими результатами попередніх обчислень (попереднього фрейму). У другому випадку – це ***функціональний атрибут***.

Інформація про об'єкти

Інформація про об'єкти (інваріантна репрезентація) з розширенням і ускладненням внутрішнього світу інтелектуальних систем буде ставати все більш широкою і адекватною, але поки ми далекі від можливостей людського розуму. Тому, знову-таки поки, всі образи, якими ми оперуємо в інтелектуальних системах, досить схематичні і дуже далекі від реальних. Але це можна виправити, і ми швидко будемо нарощувати потужності, обсяги доступної пам'яті, розвивати системи розпізнавання і т. ін.

РОЗДІЛ 6
МАТЕМАТИКА ЧЕРЧА І ФУНКЦІОНАЛЬНА МОВА ЛІСП

6.1. Лямбда-обчислення

До Алонзо Черча (Alonzo Church) і Стівена Кліні (Stephen Cole Kleene), які спільно з Аленом Тьюрингом (Alan Mathison Turing) *створили новий розділ математичної логіки – теорію обчислюваності*, практикували таку процедуру створення складних функцій: в якості основи береться деякий «мінімальний» набір базових функцій і з них будують функції вищого порядку.

Подібним чином Курт Гедель[20] створив теорію рекурсивних[21] функцій. А. Черч **відмовився від базових функцій** і ввів перетворення, за допомогою яких можна отримувати з одних функцій інші [22].

Алонзо Черч
Alonzo Church

Отримав ступінь бакалавра мистецтв в Прінстонському університеті в 1924 році, і докторський ступінь (Ph.D.) в 1927 році під керівництвом Освальда Веблена за роботу «Alternatives to Zermelo's Assumption». Два роки він був стипендіатом (National Research Fellow), рік провів в Гарварді, потім − в Гетінгені і Амстердамі. З 1929 року асистент-професор математики в альма-матер, з 1939 року доцент, з 1947 року професор математики, з 1961 року професор математики і філософії.

1. В ***класичному лямбда-обчисленні*** крім функцій і їх застосувань до інших функцій нічого немає.

2. В ***розширеному лямбда-обчисленні***, яке є *основою функціональних мов програмування*, крім безіменних функцій, заданих лямбда-виразами, використовують зовнішні константи: цілі числа, символи та логічні значення, константу, що позначає порожній список NIL. А також позначення

[20] Теорема Геделя: для мови висловлювань S, кожному з яких функцією P співставлене значення «істинне» або «хибне» (тобто перетворення в булеву множину B, або «обчислюваність»). Функція P є алгоритмом зі скінченним числом кроків (тобто дедуктика). Теорема: не всяка функція над множиною висловлювань обчислювана. (Див. Спрощене пояснення: https://geektimes.ru/post/284486/).

[21] **Рекурсивно задана функція** визначає своє значення через звернення до самої себе з іншими аргументами. Часто для цього використовується рекурентна формула – формула, що виражає кожен член послідовності через n попередніх членів.

функцій: арифметичні операції додавання, множення та інші, операції порівняння величин «більше», «менше», «рівні» та інші, операції над логічними значеннями «і», «або» та ін.

ФОРМАЛІЗМ ЛЯМБДА-ОБЧИСЛЕННЯ

Отже, функція е в нормальній формі, – це нормальна форма лямбда виразу *λx.e.* У формулі *λx.e* входження змінної *x* в вираз *e* будуть ***пов'язаними***. У виразі *λf.f x* змінна *x* – ***вільна***, а змінна *f* – пов'язана, причому оскільки змінна x – вільна, то її зміст в цьому виразі не визначений.

ПЕРЕТВОРЕННЯ ВИСЛОВІВ-*РЕДУКЦІЇ*

α-перетворення: заміна в виразі *λx.e* імені змінної *x* на будь-яке інше (яке не використовується в цьому виразі) ім'я *з одночасною заміною* всіх входжень цієї змінної в вираз *e.*

δ-редукція: вираз *+1 4* може бути перетворений в вираз *5*, а вираз *OR TRUE FALSE* – в вираз *TRUE* (тут *OR* – функція логічного «або»).

β-редукція: відповідає застосуванню функції, представленої лямбда-виразом, до аргументу,

ПРИКЛАД: ((λx. + X x) 3) в результаті застосування β-редукції буде перетворено в *(+ 3 3).*

ПРИКЛАД: *(λx.x x) (λx.x x)* перетворюється знову в *(λx.x x) (λx.x x)* – зациклюється! У виразі $(\lambda x.xx)(\lambda x.xx)$, замінюючи німі змінні в першому доданку, отримаємо те ж саме $(\lambda z.zz)(\lambda x.xx) \Rightarrow (\lambda x.xx)(\lambda x.xx)$. Тобто це не обчислюваний вираз.

η-перетворення: вираз *λx.E x* еквівалентний функції *E*.

ВИЗНАЧЕННЯ

Якщо до виразу можна застосувати одну з редукцій, то він називається таким, що редуціюється або ***редексом*** (від *redex – reducible expression*). Якщо жодного редекса в виразі немає, вираз знаходиться в ***нормальній формі***. Процес «налагодження» зводиться до перетворення вихідного виразу до його нормальної форми.

ВИЗНАЧЕННЯ

Редекс найбільш ***зовнішній***, якщо він не міститься ні в якому іншому з редексів в розглянутому виразі, редекс найбільш ***внутрішній***, якщо всередині нього не міститься редексів.

ПРИКЛАД: у виразі (λx.λy.y) ((λx.xx) (λx.xx)) найбільш внутрішнім є редекс (λx.xx) (λx.xx) що зациклюється, а редекс, що являє собою весь вираз, є найбільш зовнішнім.

МЕТОДИ

Канонічний порядок редукцій: нехай перетворення відбуваються таким чином, що редукція завжди застосовується до найбільш лівого з ***най-***

більш внутрішніх редексів – АПР (аплікативний порядок редукцій). Тобто перш за все «обчислюється» значення аргументу виклику, а потім вже відбувається підстановка цього значення в тіло, що викликається. Таким чином, цей порядок редукцій відповідає ***енергійному способу обчислення*** значення функцій в функціональному програмуванні.

ПРИМІТКА: Зазначимо, що канонічний порядок відповідає Ocaml і LISP.

ПРИКЛАД: У виразі (λx.λy.y) ((λx.xx) (λx.xx)) найбільш внутрішнім є «зациклюваний» редекс (λx.xx) (λx.xx), тому «канонічний» порядок обчислень не зможе привести вираз до нормальної форми.

Нормальний порядок редукцій (НПР) – застосування редукції до найбільш лівого з ***найбільш зовнішніх*** редексів – підстановка аргументу в тіло функції відбувається до того, як перетворення будуть проводитися над самими аргументами. Це схоже на процес ***лінивих обчислень в функціональних мовах*** програмування (де обчислення аргументу здійснюється лише один раз – при першому зверненні до аргументу; надалі відбувається звернення до вже обчисленого значення). Тут процес перетворення виразу, який служить аргументом застосування функції, відбуватиметься стільки раз, скільки разів аргумент з'являється у визначенні цієї функції.

ПРИКЛАД: (λx.λy.y) ((λx.x x) (λx.x x)) при застосуванні НПР використовуємо β-редукцію, вважаючи весь вираз редексом. Дійсно, вираз є застосуванням функції (λx.λy.y) до деякого аргументу. Однак аргумент x не використовується в тілі функції – λy.y, так що незалежно від того, чи є цей аргумент, результатом такої β-редукції буде вираз λy.y. При застосуванні АПР – зациклюємось.

Можна не перейменовувати змінні у виразі, якщо будемо застосовувати НПР, залишаючи редекси всередині лямбда-виразів, при цьому β- і δ-редукції не дадуть нормальної форми, але, отримаємо її еквівалент – слабку заголовну нормальну форму (СЗНФ).

ПРИКЛАД: якщо e в нормальній формі, це нормальна форма лямбда виразу λx.e, та ***якщо e у довільній формі – це СЗНФ форма*** *лямбда вираз у λx.e.*

Тоді процес «обчислення» відповідає приведенню його до СЗНФ за допомогою β- і δ-редукцій, що застосовуються в нормальному порядку *без використання α-перетворень* для «безпечного» перейменування змінних, що відповідає найпростішій мові функціонального програмування.

*ПРИКЛАД: приведемо до СЗНФ (λx. * x x) (+2 3). Якщо використовуємо НПР, то спочатку виконується β-редукція і (+2 3) підставляється замість x в тіло лямбда-виразу, (* (+2 3) (+2 3)). Після цього вираз (+2 3) доведеться обчислити два рази завдяки δ-редукції і отримаємо 25.*

6.2. Перехід до мови ЛІСП

Останнім часом з групи мов ЛІСП більший практичний інтерес викликав Коммон ЛІСП, який відтіснив мову Інтерлісп. Коммон ЛІСП поряд з прологом був довгий час кращею мовою штучного інтелекту. До речі, чистий ЛІСП включав набір функцій, що дозволяв створювати і видозмінювати списки, які складаються з елементів-підсписків. Додаючи новий елемент (підсписок) в початок списку або вилучаючи головну частину списку або його залишок, та використовуючи оператор умовного переходу, на ньому можна було програмувати.

1. У сучасному ЛІСПі процедури можуть стати даними, підставляються в вирази як аргументи.

2. Спочатку думали, що ЛІСП − мова тільки для рекурсивних схем, на практиці ж він виявився досить гнучким.

3. ЛІСП дозволяє сформулювати і запам'ятовувати довільні речення (ідіоми), які корисні для систем штучного інтелекту.

4. На ньому можна писати і програми, відмінні від Ш І, наприклад, на ньому написано AUTOCAD.

Джерела[22]:

1. McCarthy J. LISP 1.5 Programmer's Manual / J. McCarthy et al. – Cambridge, Mass. MIT Press, 1962.

2. Weissman C. LISP 1.5 Primer / C. Weissman – Belmont, Calif. Dickenson Publishing Company, 1967.

3. Steele G. L. Common Lisp – the language / G. L. Steele – Massachusetts: Digital Press, 1986.

4. Хювенен Э. Мир ЛИСПа Т.1: Введение в язык ЛИСП и функциональное программирование / Э. Хювенен, И. Сеппянен. – М. МИР, 1990, 458 с.

ДІАЛЕКТ ЛІСПА – **СКІМ** (див. ВСТУП В МОВУ SCHEME[23])

Як і в математиці Черча

`lambda (x) (* x x));` створити функцію, яка обчислює квадрат числа або те ж саме[24]

`(Define (square x) (* x x)).`

Можна висловити це ж складніше:

`define square (lambda (x) (* x x)));` створити функцію, яка обчислює квадрат числа і назвати її `square`.

---

[22] Для використання Common List можливі реалізації двох систем. Простіша для Windows, https://drive.google.com/open?id=1Jhmu4jz_alFGYTNR_WBmocG_95DXac_B. складніша як для Windows, так і для Linux, і для Mac OS : https://common-lisp.net/project/lispbox/

[23] Иевлев С. И. «Ваш новий язык – Scheme». Опублікований в двох частинах в журналі «Потенциал».

[24] Для завдання нових функцій в Ліспі використовується спеціальна форма defun, define і ще кілька варіантів, наприклад (defun <ім'я функції> <параметри> <тіло функції>).

(+3 5) Сума трьох і п’яти.
(* 5 6 7) Перемноження п’яти, шести і семи.
(Купити булочна батон) Купи в булочній батон.
`(* Width height) (3 5)` width, height − змінні, а 3 і 5 – їх поточні значення.

ОТЖЕ,

змінна величина задається такою конструкцією (define ім’я <початкове значення>)

```
(Define width 3)
(Define height 7)
```
`(* 2 (+ width height))` розрахунок периметру

Таблиця 6.1

Як в математиці	Як в мові СКІМ
`(+ (* a a) (* b b))`	`(+ (square a) (square b))`

тобто `(define (square x) (* x x))`

іншими словами

(`Define` (назва параметр параметр ...) тіло_функції)

тепер

```
(Define a 3)
(Define b 4)
(Define (square x) (* x x))
(+ (square a) (square b))
```

«Коли пишете програму на ЛІСПі, ви описуєте не алгоритм, а спочатку створюєте мову, а потім на ній формулюєте вихідну задачу. А точніше – ви «підганяєте» дану вам мову Scheme доти, доки вона не стане збігатися з мовою, на якій задача формулюється легко».

ПРИКЛАД: СТВОРЕННЯ ФУНКЦІЇ

Модуль числа:

Перший варіант

```
(Define (abs x)
  (If (positive? X)
       x
      (- x)))
```

Тобто «функція abs повертає свій аргумент, якщо він позитивний, інакше – мінус х». (If умова <дію, якщо умова виконується> <дію в іншому випадку>).

Другий варіант

```
(Define (abs x)
  ((if (positive? x) + -) x))
```

Тобто «... якщо аргумент позитивний, то плюс, інакше мінус x». Тут в результаті виконання виразу if повертається функція + або −, яка потім застосовується до аргументу x.

СПИСКИ

Подання списків *list*

(Define lst1 (list 1 2 3)); список з трьох чисел

(Define lst2 (list "hello" "my" "world")); список з рядків

(Define lst3 (list "hello" 1 "world" 3)); список з рядків і чисел

Функція *map* повертає список, в якому кожен елемент є результат застосування <функція> до елементу вихідного списку.

define (inc x) (+ x 1)); збільшує число на одиницю

(Map inc (list 1 1 1)); повертає список з двійок

(Map square (list 1 2 3)); повертає список з квадратів елементів, тобто 1, 4 і 9

ПРИКЛАД: взяти три числа списком і збільшити їх на одиницю

```
(Define abc (list 1 1 1))
(Map (lambda (x) (+ x 1)) abc)
```

А можна навіть і список не вводити як додаткову змінну *abc*:

```
(Map (lambda (x) (+ x 1))
       (List 1 + 1 1))
```

Як написати функцію, яка послідовно буде переміщатися по заданому списку і виводити кожен елемент цього списку? Для цього нам треба тільки знати наступні інструкції:

(Null? <Список>) –перевіряє, а чи є ще такі елементи в списку,

(Car <список>) – повертає перший елемент списку,

(Cdr <список>) – повертає список з усіх елементів, крім першого, а якщо більше нічого не залишилося, то порожній список.

ПРИКЛАД:

(Car (list 1 2 3)); поверне 1

(Cdr (list 1 2 3)); поверне список з 2 і 3, тобто (list 2 3).

Введемо функцію print-list.

```
(Print-list (list 1 2 3))
1
2
3
```

ПРИКЛАД:
Якщо список не порожній, то:
1) виводимо голову списку.
2) викликаємо `print-list` для хвоста списку

```
(Define (print-list lst)
(If (not (null? lst))
    (Begin (display (car lst))
           (Newline)
           (print-list (cdr lst)))))
```

6.3. Робота зі списками

(*Common Lisp* –діалект мови *ЛІСП*, стандартизованого *ANSI*)

Квотування (блокування). Для запобігання обчислень аргументи повинні бути «заквотованими» – перший аргумент – список звичайним блокуванням: ‘(4 1 6 7) – апостроф перед списком запобігає його обчисленню, а другий аргумент – функціональним блокуванням #’ ім’я_функції (# ‘працює як традиційне блокування, але повинно бути вказаним для функцій (докладніше це питання розглянемо пізніше)).

ПРИКЛАД: сортування списку чисел за зростанням:

```
(Sort '(4 1 6 7) #' (lambda (x y) (<x y)))
==> (1 4 6 7)
```

і за зменшенням

```
(Sort '(4 1 6 7) #' (lambda (x y) (> x y)))
==> (7 6 4 1)
```

`sort` сортує список, зазначений в якості першого аргументу за правилом, заданим другим аргументом. Другий аргумент повинен бути функцією, що приймає два параметри і повертає узагальнене логічне значення – «істинне», якщо аргументи розташовані в правильному порядку і «хибне» – якщо в неправильному. Функціональне блокування # `'append` можна записати і таким чином: (`function append`)

ПРИКЛАДИ ФУНКЦІЙ

1. Базові функції

`CDR` повертає список, крім першого елемента, а `CAR` повертає перший елемент.

```
Cdr (car '((a b c) d)))
        (b c)
```

ЗВ’ЯЗУЮЧА ФУНКЦІЯ (setq m ‘k)

k

Наприклад, (setq 2+ (make−incr 2)) повідомляє, що об’єкт функції 2+ це те ж саме, що `make-incr 2` функція, яка збільшує своє значення на 2.

НЕЯВНИЙ PROGN `(progn (setq x 2) (setq y (* 3 2)))`

```
6
```

Обчислюється послідовність форм, а в якості значення береться значення останньої форми.
LAST видаляє зі списку всі елементи, крім останнього.
REVERSE, яка відразу змінює порядок елементів на зворотний.

2. Базові предикати

Предикат в ЛІСПі − це функція, яка визначає, чи має аргумент відповідати певним властивостям, і повертає значення T або NIL)

```
(Eq 'cat' cat)
     t
```

Предикат `EQ` порівнює два символи і повертає T, якщо вони однакові, і повертає NIL в іншому випадку.

3. Умовні речення (вирази)

(IF <умова> <то форма> <інакше форма>)

```
(If (atom x) 'atom' not - atom)
```

4. Логічні функції

`OR` використовується для виділення першого без порожнього елемента в списку.

```
(Or t nil)
     t
```

Якщо `AND` зустрічає аргумент, значення якого NIL, вона повертає NIL, не продовжуючи обчислення інших. Якщо NIL аргументів не зустрілося, то повертається значення останнього аргументу.

```
(And 'a' b)
     b
```

5. Функції перевірки

`COND` – Відбирається перший підвираз, чия форма умови обчислюється в не−NIL. Всі інші підвирази ігноруються.
`CHECK` – тестуюча функція, яка порівнює два можливих розв'язки – двох−елементні списки v a і повертає кількість збігів в них.

Технологія роботи зі списками, яка розроблена на основі математики Черча (і яка розвивалася вже за своєю логікою), дозволяє побудувати множину (простір) розв'язків і за допомогою функцій порівняння виділити шуканий. Створити набір функцій, які оцінюють рівень збігів шуканої (або декларованої апріорі) цілі з базою фактів і правил. У цьому є деяка аналогія з системами, що використовують технологію логічного висновку, де теж можна шукати мету (ціль) (пряма дедукція) або виявляти відповідність мети (цілі) фактам і правилам, тобто доводити теорему (зворотна дедукція). Особливістю функціональних мов є можливість модифікувати опис виведення, згідно з уявленнями розробників. Але отримана програма також може відповідати на питання, які тут відіграють роль мети (цілі), подібно до мов логічного висновку. Саме тому ЛІСП і його аналоги – функціональні мови, – не дарма вважаються мовами штучного інтелекту.

РОЗДІЛ 7
НЕЧІТКА ЛОГІКА[25]

7.1. Елементи нечітких множин

Недоліком експертних систем логічного програмування було те, що вони оперували двома значеннями «істинне» і «хибне», тобто окремі факти, правила, цілі – всі мали так звану істиннісну оцінку за двобальною шкалою. У житті так не буває, тому весь час шукали можливість перейти до більш наближеної до реальності структури розмитих понять. Тому так вельми до речі була запропонована людству професором Каліфорнійського університету Лотфі Заде теорія нечітких множин та нечіткої логіки (1965). Нечітка логіка застосовується для задач слабо формалізованих і задач з ненадійними умовами і з невизначеними виразами; розташовується між формальним і природним описами; описує – апроксимує будь-яку математичну модель (теорема FAT – Fuzzy Approximation Theorem, B. Kosko, 1993); але в дійсності вихідний набір даних часто не тільки неповний, а й злегка суперечливий, введені експертами змінні входу і виходу можуть виявитися такими, що не цілком адекватно описують реальність; тому з необхідністю від систем вимагають адаптивності, тобто вони повинні допускати корекцію знань і параметрів в процесі вирішення завдань [23, 24].

Характеристична функція = функція належності елемента до множини, – приймає значення від нуля до одиниці або до іншої величини – верхньої межі (в чітких множинах тільки 0 і 1), тобто $\mu_A(x)$ – це характеристична функція, що приймає значення на множині M .

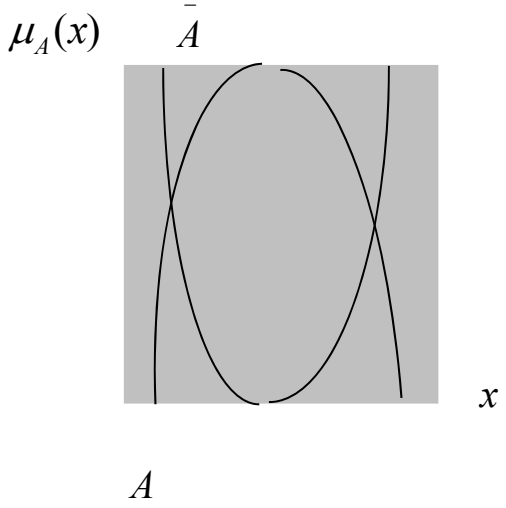
Нечіткі (fuzzy) множини (L. Zadeh, 1965) – множини елементів, що описуються характеристичними функціями.

Лотфи А. Заде *L. Zadeh*

Барт Коско *B. Kosko*

[25] The first time, it's a kludge! The second, a trick. Later, it's a well-established technique! – Спочатку – це **плутанина**, потім хитрість, а після - всім відомий метод. *Mike Broido, «Intermetrics»*

Нехай *E* – універсальна множина елементів x, а *G* – деяка властивість. *A={μ_A (x)/x}* – це підмножина *E*, де μ_A *(x)* – характеристична функція, що набуває значення на множині *M*, які визначають наявність (частку, ступінь) властивості *G* у елемента x.
Якщо *M* = {0,1} – це чітка множина, то *M* = [0,1] – нечітка (нормалізована) множина.



$$A \cap \bar{A} \neq \varnothing$$

$$A \cup \bar{A} \neq E$$

Рис. 7.1. Подання характеристичних функцій нечіткої множини і його доповнення.

Висота нечіткої множини – значення верхньої межі *M* тобто $\sup_{x \in A} \mu_A(x)$ – висота нечіткої множини (якщо висота = 1 – ця множина нормальна, а якщо менше – субнормальна).

Нормалізація нечіткої множини – процедура виду $\mu(x) = \mu_A(x) / \sup_{x \in A} \mu_A(x)$.

Унімодальність елемента $x_0 - \mu_A(x_0)=1$.

Носій нечіткої множини A – його підмножина, для якої $\mu_A(x)>0$.

Точки переходу множини *A* – елементи, для яких справедливо $\mu_A(x) =1/2$.

Таблиця 16

Логічні операції
1. Включення – $\forall x \in E$, $\{\mu_A(x) \le \mu_B(x)\}$, то A міститься в B (B домінує A) або $A \subset B$
2. Рівність – $\forall x \in E \quad \mu_A(x) = \mu_B(x)$, то $A = B$.
3. Доповнення – *M = [0,1]*, *A* и *B* на M, $\forall x \in E$, $\mu_A(x) = 1 - \mu_B(x)$, то $B = \bar{A}$ або $A = \bar{B}$

4. Перетин – $\forall x \in E, \{ \mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)) \}$ – – найбільша підмножина, що міститься і в А і в В одночасно або
$A \cap B$
5. Об'єднання – $\forall x \in E$, – $\{ \mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)) \}$ – найбільша підмножина, що міститься як в А, так і в В або
$A \cup B$

6. Різниця – це А–В = $A \cap \bar{B}$ $\{ \mu_{A \cap \bar{B}}(x) = \mu_{A-B} = \min(\mu_A(x), 1 - \mu_B(x)) \}$,
$A - B = A \cap \bar{B}$

7. Диз'юнктивна сума $A \oplus B = (A - B) \cup (B - A) = (A \cap \bar{B}) \cup (B \cap \bar{A})$
або $\{ \mu_{A \oplus B}(x) = \max\{[\min(\mu_A(x), 1 - \mu_B(x))], \min[(1 - \mu_A(x), \mu_B(x))]\} \}$, де
$A \oplus B = (A - B) \cup (B - A) = (A \cap \bar{B}) \cup (B \cap \bar{A})$

Нечіткі змінні – $<\alpha$, X, A>, де α – найменування нечіткої змінної, X – область визначення змінної (універсальна множина), A – нечітка множина з функцією належності $\mu_A(x)$.

Лінгвістична змінна $<\alpha$, T, X, G, M>, де – α найменування лінгвістичної змінної, X – область визначення змінної (універсальна множина), *T – (базовий) терм – множина (множина значень = найменувань)* лінгвістичної змінної, *G – синтаксична процедура,* що дозволяє генерувати нові терми (значення), причому $T \cup G(T)$ – розширений терм – множина лінгвістичної змінної, де $G(T)$ – згенерована множина термів; *M – семантична процедура*, що перетворює нові значення термів в елементи нечітких множин.

Таблиця 7.2

Алгебраїчні операції

1. Алгебраїчне множення – $\forall x \in E$, $\mu_{A \cdot B}(x) = \mu_A(x) \cdot \mu_B(x)$
$A \cdot B$.
2. Алгебраїчна сума – $\forall x \in E$, $\mu_{A+B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)$
$A + B$,
для яких виконуються властивості комутативності (положення множників неважливе);
асоціативності (порядок двох послідовних однакових операцій неважливий);
$A \cdot E = A$, $A + E = E$,
множення на нуль і додавання нуля – звичайні, справедливі формули де Моргана:,
: $\overline{A \cdot B} = \bar{A} + \bar{B}$, $\overline{A + B} = \bar{A} \cdot \bar{B}$, а також $A \cdot \bar{A} = \varnothing$, $A + \bar{A} = E$.

Не справедливі ідемпотентність (тобто, $A \cdot A = A$, $A + A = A$, які для логічних, навпаки, справедливі) і дистрибутивність (порядок двох різних операцій). Справедливі дистрибутивність однією з алгебраїчних і однією з логічних ($\cup, \cap$) операцій.

Можна підносити до ступеня, якщо ступінь більше одиниці – це ущільнення = концентрація, а менше одиниці – розтягнення.

3. Множення на число – p *(*при $p \sup\limits_{x \in A} \mu_A(x) \le 1$*)*. Множина pA – це множина для якої $\mu_{pA}(x) = p\mu_A$.

4. Опукла комбінація n множин {A} – визначається функцією належності $\mu(x) = \sum\limits_{i=1}^{n} \omega_i \mu_{Ai}(x)$.

5. Пряме (декартове) множення множин – множення n множин {A}, заданих на n універсальних множинах {E}, визначаються функцією належності $\mu = \min\{\mu_{A1}(x_1), \mu_{A2}(x_2), ..., \mu_{An}(x_n),\}$ на множині $E_1 \times E_1 \times ... \times E_n$.

6. Оператор збільшення нечіткості – $H(A,K) = \bigcup\limits_{x \in E} \mu_A(x) K(x)$ де *K(x)* – підмножина універсальної множини E (їх сукупність – ядро оператора збільшення нечіткості H).

7. Чітка множина α рівня – множина α рівня нечіткої множини A універсальної множини E – чітка підмножина $A_\alpha = \{x / \mu_A(x) \ge \alpha\}$ універсальної множини E, де $\alpha \le 1$, де тільки ті x, яким у нечіткої множині A відповідали $\mu_A(x) \ge \alpha$.

Для операцій над нечіткими числами часом використовують позначення

$$\vee_x = \max_x, \wedge_x = \min_x$$

хоча допустимі й інші уявлення.

Нечіткі числа – нечітка множина A на множині дійсних чисел R з функцією приналежності,. $\mu_A(x) \in [0,1]$, $x \in R$.

Нормальні числа A – якщо $\max\limits_x \mu_A(x) = 1$.

Опуклі числа A – якщо для будь-яких $x \le y \le z$ $\mu_A(x) \ge \mu_A(y) \cap \mu_A(z)$.

Множина α рівня нечіткого числа A – це $A_\alpha = \{x / \mu_A(x) \ge \alpha\}$.

Підмножина $S_A = \{x / \mu_A(x) \ge 0\} \subset R$ – ***носій нечіткого числа A***.

Нечіткий нуль – опукле нечітке число A для якого справедливо $\mu_A(0) = \sup\limits_x \{\mu_A(x)\}$.

Позитивне нечітке число A якщо при $\forall x \in S_A, x > 0$.

Нечітке n-арне відношення – нечітка підмножина R на прямому добутку універсальних множин E $= E_1 \times E_1 \times ... \times E_n$ приймає значення на множині належності M.

Приклад: Для випадку n = 2, нечітке відношення визначається функцією R: (X, Y) на [0,1], для якого кожному набору (x, y) з (X, Y) ставитися у відповідність $\mu_R(x, y)$. Позначається нечітке відношення на $X \times Y$ у вигляді: $x \in X, y \in Y : xRy$. У разі X = Y, використовують позначення нечіткого відношення на множині X $R : X \times X \to [0,1]$.

Композиція (згортка) = (max-min композиція, max-min згортка) – $R_1 \bullet R_2$ тобто $\mu_{R1 \bullet R2}(x, z) = \bigvee_y [\mu_{R1}(x, y) \wedge \mu_{R2}(y, z)]$, якщо задані два нечітких відношення $R_1 : X \times Y \to [0,1]$ $R_2 : Y \times Z \to [0,1]$..

Логіка Zadeh. Було введено поняття «розпливчастих» множин – як множин, що мають різні градації ступеня приналежності до даного класу. Тут можна говорити про випадковість лише частково, хіба що лише з позицій невизначеності відносного рівня приналежності до даного класу. Вводилися різні методи розрахунків, наприклад виду $\alpha + b \Rightarrow \max(a, b)$ або $\alpha \cdot b \Rightarrow \min(a, b)$. Розпливчаста множина A в X вводилася як сукупність пар, де $x \in X$,тобто $\mu_A(x)$ – функція, яка відображає X в *M*, де *M* – простір належності $M = [0,1]$. Задача оцінки (визначення) $\mu_A(x)$ на множині пар $\{x, \mu_A(x)\}$ – ***задача розпізнавання образів.*** Оператор розмиття (в оригіналі fuzzifier) розмиває кордони множини, створює не цілком певні образи. Замість «x менше 1 і більше 2» використовуємо «x приблизно знаходиться між 1 і 2».

Перетин множин може бути визначено в жорсткому значенні, наприклад, $\mu_{A \cap B} = \min\{\mu_A(x), \mu_B(x)\}$, тобто $\mu_A(x)$ або $\mu_B(x)$, залежно від того що виявилося менше. Можливий перетин в більш м'якому значенні $\mu_{A \cap B} = \mu_A(x) \cdot \mu_B(x)$, де залишається взаємна участь множин, $\mu_A(x)$ і $\mu_B(x)$. У структурі прийняття рішень зазвичай розглядається множина альтернатив і множина обмежень. Для альтернатив вводиться функція переваги (наприклад, виграш чи програш при її виборі). Зазвичай намагаються створювати симетрію по відношенню до альтернатив і обмежень, що полегшує прийняття рішень. Розпливчасте рішення можна вважати розпливчастою множиною в просторі альтернатив, як результат перетину альтернатив і обмежень. Повне рішення може бути представлене у вигляді диз'юнкції різних отриманих після аналізу характеристичних функцій. Способи переходу до чіткості – це зазвичай зважене середнє всіх. $\mu_A(x)$.

ПРИКЛАД: Розглянемо наступну просту задачу. Нехай мета (ціль) – це G, а обмеження – це *C*. Тоді розв'язком буде *D*, тобто $D = G \wedge C$. Або в термінах характеристичних функцій $\mu_D = \mu_G \wedge \mu_C$. Залежно від інтерпретації кон'юнкції – жорсткою (min) або м'якою, результат буде відрізнятися. Наприклад, для жорсткої інтерпретації цілі і обмеження

вступають в конфлікт і немає однієї альтернативи, що задовольняє всім.

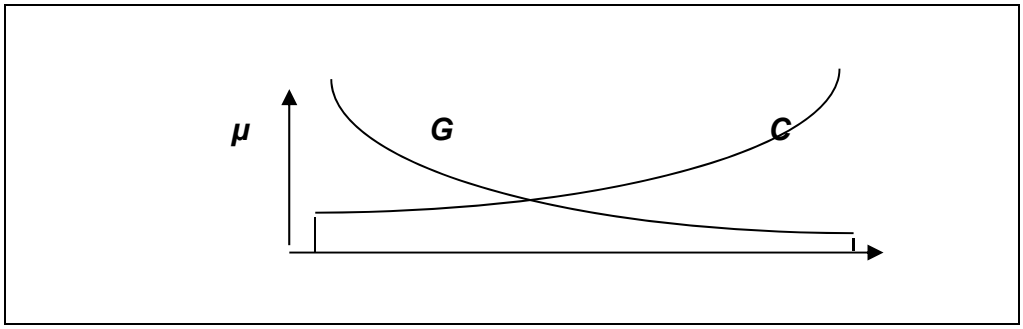


Рис. 7.2. Розв'язком є тільки нижні частини двох кривих[26].

7.2. Формальні схеми нечіткого логічного висновку

Розглянемо чисто формально такі приклади.

Алгоритм Мамдані (Mamdani) –

1) за допомогою експертів вводиться нечіткість у формі нечітких множин = функцій належності A1 (x), A2 (x), B1 (y), B2 (y), C1 (z), C2 (z);

2) за допомогою експертів вводяться правила:

П1: якщо x є A1 і y є B1, то z є C1;

П2: якщо x є A2 і y є B2, то z є C2.

Інформація подається у вигляді речень, що містять кон'юнкцію і імплікації. Відповідь відповідає диз'юнкції отриманих результатів. Тобто, треба побудувати нечітку відповідь у вигляді відповідного значення характеристичної функції

$$\mu_{\Sigma}(z) = A_1(x_0) \wedge B_1(y_0) \wedge C_1(z) \vee A_2(x_0) \wedge B_2(y_0) \wedge C_2(z)$$

а потім перейти до чіткості. Розглянемо всі операції послідовно:

Отже, перша половина правил може бути представлена формально як

$$A_1(x_0) \wedge B_1(y_0) = \alpha_1 \quad \text{і} \quad A_2(x_0) \wedge B_2(y_0) = \alpha_2$$

друга як

$$\alpha_1 \wedge C_1(z)$$
$$\alpha_2 \wedge C_2(z)$$

де (x_0, y_0) – конкретні вхідні змінні;

3) так як $\wedge = \min$, то знаходять рівні «відсікання» $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$ и $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$.

---

[26] Навпаки, якщо G і C – це комбінація двох рішень, то можна використовувати їх диз'юнкцію (max) і інтегральний розв'язок буде $D = G \cup C$ або $\mu_D = \mu_G \cup \mu_C$. Це верхні частини двох кривих на рис. 7.2.

4) тепер знаходять функції належності C

$$\alpha_1 \wedge C_1(z)$$
$$\alpha_2 \wedge C_2(z)$$

5) композиція (операція $\vee = \max$)

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

6). приведення до чіткості

Центроїдний метод: $z_0 = \dfrac{\int z\mu_\Sigma(z)dz}{\int \mu_\Sigma(z)dz}$ – центр ваги = чітке значення

Алгоритм Цукамото (Tsukamoto) –

1) за допомогою експертів вводиться нечіткість у формі нечітких множин = функції приналежності $A_1(x)$, $A_2(x)$, $B_1(y)$, $B_2(y)$, але тут $C_1(z)$, $C_2(z)$ – це чіткі, визначені функції, що дозволяють знайти чіткі значення z по кожному з правил П1 і П2;

2) знаходять рівні «відсікання» $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$ і

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

3) з рівнянь $\alpha_1 = C_1(z_1)$ і $\alpha_2 = C_2(z_2)$ знаходять чіткі значення (відповіді щодо застосування окремих правил) z_1, z_2;

4) визначається відповідь, наприклад, як зважене середнє.

Таким чином, початкові знання представлені у вигляді речень з використанням кон'юнкції (їх використовують на етапі отримання рівнів відсікання) і імплікації. Причому імплікації можуть не перетворюватися і використовуватися безпосередньо (на цьому етапі визначаються значення характеристичних функцій у нечітких множин С). Далі, оскільки всі речення (для набору речень, де використано сполучення) пов'язані диз'юнкцією, її і застосовують для отримання остаточної відповіді в нечіткій формі. Перехід до чіткої форми може бути різним, зокрема таким, як запропонований в наведених традиційних прикладах (алгоритмах).

Треба зробити одне зауваження. Перехід до чіткої форми некоректно визначено, якщо нема виражених екстремумів інтегральної (що є результатом обчислень) функції приналежності. Це означає, що рішення недостатньо визначено і потребує уточнення в структурі алгоритму.

Проблеми розширення можливостей інтелектуальних систем

Створення систем, здатних впоратися з варіативними і некоректними завданнями відзначається великою складністю. Для їх вирішення потрібні великі обсяги інформації, навіть погано структурованої, що є клудж (*жарг.* щось, що не повинне працювати, але чомусь працює) – плутанина образів і зв'язків між ними. Слід тільки домагатися при наповненні такою інформа-

цією глобальних баз знань максимальної сумлінності. Крім того, мабуть, слід послабити обмеження на рівень несумісності змістовної інформації та правил. Корисно остерігатися інформаційного шуму, який з розвитком мережевої структури, прискорення і збільшення обсягів інформаційних потоків стане великою проблемою для інформаційних технологій та інтелектуальних систем особливо, не виключаючи і самої людини. Завдяки розширенню і ускладнення систем зворотних зв'язків треба домагатися створення в штучному інтелектуальному середовищі розвитку механізмів, аналогічних до здібностей людини проводити уявний експеримент. Така внутрішня уява, коли система використовує власні інформаційні ресурси як початкові дані, має активно розвивати її внутрішній світ, створюючи нове знання, синтезучи його і вишукуючи неточності і нестиковки. Не варто боятися того, що система часом буде «замислюватися». Крім того, виявлені нестиковки здатні дати мотив для роздумів людині, такому незграбному творцеві інтелектуальних систем. Вищеозначене підштовхує нас до використання нейронної мережі.

7.3. Нейронна мережа в поданні нечіткої логіки

ДЕЩО ПРО ТРАДИЦІЙНІ НЕЙРОМЕРЕЖІ

Спочатку У. Маккалох і У. Пітс (W. McCulloch, W. Pitts) створили першу модель нейрона (1943), потім в 1960–1970 рр. з'явилися ще сирі моделі нейронних мереж, але тільки в 1982 р Дж. Хопфілд (J. Hopfield) розробив клас нейронних мереж і методи їх навчання.

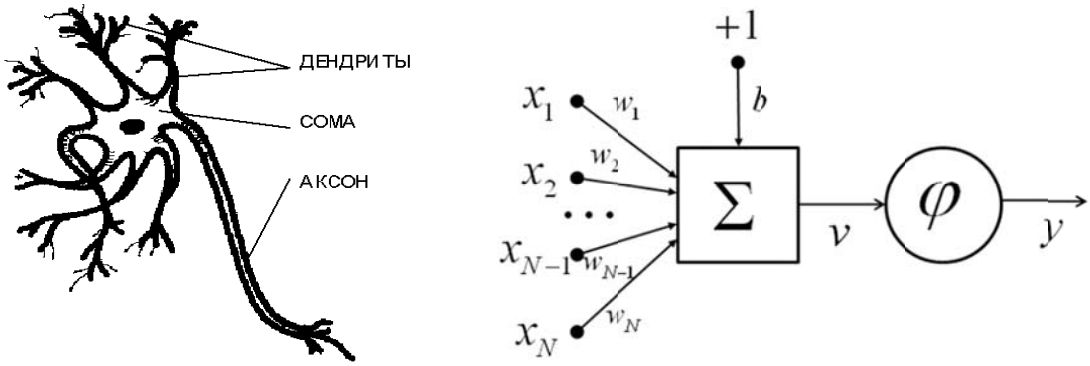


Рис. 7.3. Природний і штучний нейрони

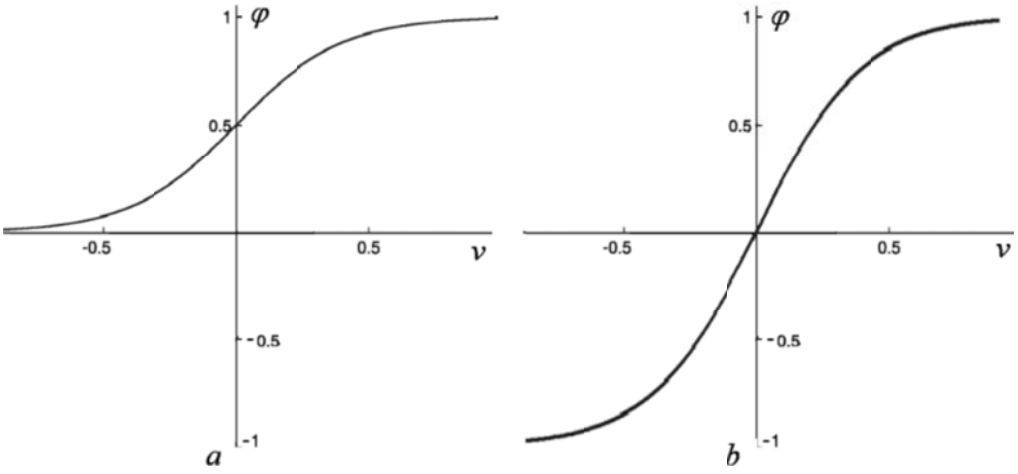


Рис.7.4. Функції реакції нейрона на зовнішнє подразнення

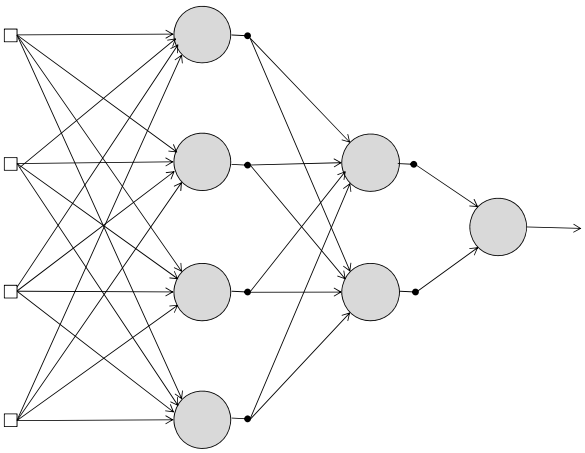


Рис. 7.5. Схема простої штучної нейронної мережі

Почалося все з персептрона Ф. Розенблата (Frank Rosenblatt) МАРК1, який вже в 1960 році, трохи плутаючись, впізнавав-таки деякі літери алфавіту. Як водиться в науці, через десяток років колишній однокурсник Ф. Розенблата М. Мінський переконливо розкритикував можливості персептрона, що призвело до затримки розвитку нейрокомп'ютерів ще на десяток років. Мовляв, не так вже він і добре може розпізнавати образи. Потім люди все-таки вирішили використовувати персептрон за призначенням, бо нічого іншого їм не спало за цей час на думку.

Нейронна мережа – це павутина з нейронів, яку для того, щоб вона робила щось осмислене, потрібно попередньо навчити. Дати їй кілька завдань з відповідями і підігнати параметри мережі так, щоб вона ці відповіді продемонструвала, причому всі з цього навчального комплекту. Бажано, щоб завдань було побільше, тоді є шанс, що нова задача буде розв'язана так, як потрібно.

Девід Румельхард і Джефрі Хінтон (1986) запропонували найбільш поширений нині метод навчання мережі – метод зворотного поширення помилки: оцінивши ступінь відхилення від потрібної відповіді вони рекомендували пройтися в зворотному порядку по пластах мережі і поправити налаштування.

І до цього дня всі нейронні мережі, нейрокомп'ютери, які налічують мільйони бібліотек, навчаються за цією схемою, хоча існують мережі, які навчання не потребують.

Доведена Ф. Розенблатом і колегами теорема збіжності персептрона, якого навчають за методом корекції помилки стверджує що для будь-яких початкових вагових коефіцієнтів, і послідовності процедур цей метод завжди приведе до досягнення розв'язку за скінченний проміжок часу. Показано було також, що при занадто великому обсязі навчання, нейронів може не вистачити, потрібно їх число збільшувати. М. Мінський уточнив обсяги пам'яті для вагових коефіцієнтів і вказав на проблеми з занадто великим обсягом пам'яті для великих задач і показав, що час навчання і розв'язання можуть виявитися занадто великими, що викликало позамежний песимізм у розробників. Потім усвідомили, що ***для розпізнавання образу потрібна далеко не повна інформація, досить лише її частки***. Крім того, М. Мінському належить вислів про властивості мережі: «якщо нейромережа випадково пов'язана, це не позбавляє її від заздалегідь усталеної думки про те, як грати, просто вам вона

буде невідома[27]». Тобто, те, що відбувається в мережі, навряд чи для її творця є зрозумілим..

Франк Розенблат

Марвін Мінський

Ця ідея отримала розвиток – стали шукати більш м'які форми активації нейронів, ускладнення зв'язків нейронів різних шарів (наприклад, архітектура ResNet і подібні до неї). Але навчання має свої проблеми – потрібна велика кількість правильно розв'язаних задач – прикладів (для цього навіть створювали колективи розробників таких прикладів), при великій кількості шарів вплив коригуючих поправок послаблюється, крім того, потрібно проводити багато операцій з налаштування – фактично в режимі ітерацій. Виявилося, наприклад, корисним попередньо готувати мережу для обробки даних певного типу – метод попереднього навчання мережі Д. Хінтона (2006). Перспективною процедурою є використання паралельних обчислень при налаштуванні мережі з технологіями CUDA на графічних картах (GPU – graphics processing unit). Уже для стільникового зв'язку створені спеціальні процесори – NPU (neural processing unit), так звані прискорювачі штучного інтелекту (AI accelerator) для збільшення продуктивності паралельних обчислень аналогічно до GPU при помітній економії енергії.

Якщо після навчання тести показують, що помилки зменшуються, то значить пощастило, навчили. А якщо помилки на тестових прикладах ростуть, значить мережа просто запам'ятала все, що їй повідомили під час навчання, а реально нічому не навчилася, це політкоректно назвали перенавчанням (overfitting). Перенавчання великих мереж очікуване, тому що збільшення вільних управляючих параметрів еквівалентне збільшенню ступенів свободи[28]. Недостатнє число навчальних завдань дозволяє забез-

[27] Всі наші методи отримання розв'язків не єдино вірні. У листі до Бесселя від 21 листопада 1811 р. Гаусс зробив висновок, що в усій математиці немає нічого справжнього. «Не слід забувати про те, що ці функції, подібно до всіх математичних конструкцій, є всього лише нашими твореннями і що в той момент, коли втрачає сенс визначення, з якого ми почали розробку їх теорії, слід питати себе, не "що таке ці функції", а яке допущення зручніше прийняти, щоб введене нами поняття функції зберегло сенс».

[28] Вимога єдиності рішення, яку нав'язали математики, в подальшій практиці створення штучного інтелекту для вирішення необчислюваних задач, які не потребують єдиності

печити правильність необхідних відповідей тільки цих задач. Однак інші подібні завдання можуть бути розв'язані неправильно. Крім того, не можна забезпечити пряму відповідність початкових умов кінцевим результатам. Різні початкові умови можуть давати один і той же або близький результат. Боротися з перенавчанням складно, але деякі підходи виявилися вдалими. Наприклад, часто застосовують ослаблення впливу сусідніх нейронів один на одного (для кількісних критеріїв такого ослаблення використовують відомі в математиці методи регуляризації Тихонова), використовують відключення під час навчання великих ділянок мережі – оригінальна методика навчання Д. Хінтона Dropout (2012). Це тимчасово зменшує складність мережі, створюючи умови подавлення перенавчання, а потім при підключенні раніше виключених ділянок, їх адаптація до нового стану поліпшується. Крім того, людський мозок знайшов же спосіб боротися з перенавчанням (нехай навіть не так успішно, як хотіли б математики), відключаючи цілі відділи кори головного мозку, що не потрібні для вирішення простого завдання.

Але якщо все це не допомагає, тоді мережу вважали нездібною і відмовлялись від неї. Хоча є й інша причина появи несподіваних відповідей. Коли мережа досить потужна, вона здатна вийти з нав'язаного їй програмою навчання класу рішень і запропонувати свої, дещо інші, які експериментатори не припускали. Для них мережа порушує сформовану картину світу. Така кмітливість мережі теж не віталася, і від цієї надмірно розумною мережі теж відмовлялися.

Хоча більш раціональним виходом може бути перманентне збільшення кількості і якості (спільності, різноманітності) навчальних завдань для збереження стійкості картини світу. В книзі автора «Про користь роздумів», було висловлено гіпотезу «необхідного розвитку». Будь-яка інтелектуальна система (зокрема цивілізація), що еволюціонує, весь час збільшує свої інтелектуальні ресурси (тобто підключаються всі нові і нові блоки інтелектуальної мережі) зростає її складність. І колишній обсяг навчальних завдань і відповідно запропонованих рішень незабаром виявляється недостатнім. В тому сенсі, що при цьому виникають проблеми з формуванням похідних рішень, ростуть неузгодженості, які сприймаються як помилки. Тобто порушується стійкість картини світу. Єдиним виходом тут може бути збільшення обсягу відомих задач, що забезпечують ефективне навчання глобальної мережі. Це збільшення може бути підтримане саме наукою, виходом із замкнутого кола колишніх понять і уявлень.

рішення, на базі нейронних мереж величезного обсягу виявилося зайвим. Справа в тому, що сама людина іноді вирішуючи проблему виявляє безліч можливих шляхів її вирішення і її анітрохи не турбує відсутність єдиності рішення в так званих необчислюваних завданнях (де, крім усього також нема формалізованого методу вирішення і відсутня верифікація результату), він вибирає з них таке, яке йому здалося прийнятним.

ГІПОТЕЗА «НЕОБХІДНОГО РОЗВИТКУ»

Інтелектуальна система, що розширюється, вимагає збільшення базових знань для запобігання руйнування картини світу

МЕРЕЖІ НА ОСНОВІ НЕЧІТКОЇ ЛОГІКИ

Об'єднання ближчої до реальності природної мови і сформованих людиною понять системи нечіткої логіки з штучними нейронними мережами було вперше виконане Ж–С. Р. Чангом з Тайванського університету [25].

Ж–С. Р. Чанг *J.–S. Roger Jang*

Нечіткі нейронні мережі побудовані за таким принципом: висновки робляться на основі апарату нечіткої логіки, а функції належності підлаштовуються з використанням алгоритмів навчання нейронних мереж.

> Роль нечіткої логіки виявляється настільки великою, що дозволяє повернутися до одвічної проблеми усвідомлення пошуку рішень нейронними мережами загалом і людським розумом зокрема. Справа в тому, що запропонована давніми вченими система логічних і математичних методів містить далекі для нейронних мереж способи опису і процедури розв'язань. Нейронна мережа шукає відповіді, використовуючи інші принципи і підходи, засновані на порівняннях «більше-менше». Дійсно, ступаючи на проїжджу частину, людина ніколи не вирішує в розумі завдання зустрічі, спираючись на математичний апарат, а оцінює можливість наїзду автомобілем що наближається на основі порівняння спостережуваної ситуації з попереднім досвідом. Природа створила унікальний інструмент – людський мозок, який є нейронною мережею з сукупністю допоміжних систем, що забезпечують його ефективне функціонування. Він діє приблизно так, як описано нечіткою логікою, де кожен предмет, явище або дія має обчисленні сторони та грані. Оцінка цих елементів, що становлять предмет, явище або дію може бути представлена деякими характеристичними функціями, що визначають ступінь належності до певного поняття. Схоже, що пошук відповіді в нейронній мережі відбувається подібно до того, як описав у своїй математичній теорії Лотфі А. Заде. Безсумнівно, що людство неодмінно усвідомить важливість такого опису і спробує зрозуміти, як саме влаштована система прийняття рішень природним і штучним інтелектом в формі нейронних мереж. І тоді, освоївши мову нейронних мереж, прийнявши на озброєння методи його функціонування, люди знайдуть можливість спілкуватися один з одним, не використовуючи збіднені емоціями і почуттями сучасні мови, що не переводять інтуїтивні образи і здогади в формальні символьні описи. Залишаючи останнім тільки прямі розрахунки і обчислення, де інтуїція і здогади вже виконали свою роль.

Наприклад, в тришарових мережах перший шар вводить нечіткість (fuzzification) на основі функцій належності (характеристичних функцій), другий застосовує нечіткі правила, третій приводить до чіткості (defuz-

zification). Можна використовувати меншу кількість шарів. Варто звернути увагу, що сигмоїдні функції нейронів цілком здатні виконувати роль операторів нечіткої логіки. Нагадаємо, що при сигналах на входи нейронів формується сумарна зважена величина, яка зазвичай обробляється активаційною (сигмоїдною) функцією.

Важливо також звернути увагу на неперебoрну неоднозначність відповіді в формі результуючої характеристичної функції, отриманої в процесі застосування нечіткої логіки. Тобто, взагалі кажучи, з позицій чіткої логіки, відсутня однозначність виводу. Бо перехід до чіткості строго не визначений і вибір конкретного значення характеристичної функції при переході до чіткості є більше мистецтвом, ніж методикою. Як і в нейронних мережах, які замкнуті на великі аудиторії користувачів і агентів (великі мережі, що створені останнім часом), так і в природному інтелекті, більшість завдань вирішується методами, подібними методами нечіткої логіки, що не забезпечує однозначність відповідей. Вибір відповіді часто здійснюється на нестрогих міркуваннях (або перевагах), на оптимальному (нехай навіть і нестрогому) узгодженні обраного рішення з великим набором інших раніше отриманих в тій чи іншій мірі подібних рішень, що знаходяться в глобальній базі даних (пам'яті).

Приклад [7]: найпростіша нечітка нейронна мережа, яка використовує два правила:

П1: якщо x є A1, то $y = z_1$ і

П2: якщо x є A2, то y = z2.

Введення нечіткості: нечіткі поняття тут представлені сигмоїдними функціями належності $A_1(x) = \{1 + \exp[-b(x-a)]\}^{-1}$ і $A_2(x) = \{1 + \exp[b(x-a)]\}^{-1}$, до речі, сума яких в даному випадку дорівнює одиниці (хоча це не настільки важливо).

Апарат нечітких множин: ступені істинності правил визначаються $\alpha_1 = A_1$ і $\alpha_2 = A_2$.

Перехід до чіткості: результат на виході визначається зваженим середнім: $o = \dfrac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2}$.

В процесі навчання підлаштовувати тут краще параметри *a* і *b,* виходячи з виду функції помилки *E (a, b, z1, z2),* тобто шляхом присвоєння $a := a - \eta \cdot \partial E(a,b)/\partial a$ та $b := b - \eta \cdot \partial E(a,b)/\partial b$.

Норма і конорма. Визначимо такі операції $\min(\mu_A, \mu_B)$; $\mu_A \cdot \mu_B$; . $\max(0, \mu_A + \mu_B - 1)$. Вони належать до так званої трикутної норми (t-норма), якщо всі μ_A (x) − знаходяться в інтервалі x [0,1].

Властивості норми:

1. $T(0,0) = 0$; $T(\mu_A, 1) = T(1, \mu_A) = \mu_A$; – обмеженість;
2. монотонність;
3. $T(\mu_A, \mu_B) = T(\mu_B, \mu_A)$; – комутативність
4. $T(\mu_A, T(\mu_B, \mu_C)) = T(T(\mu_A, \mu_B), \mu_C)$; – асоціативність.

Для операцій $\max(\mu_A, \mu_B)$; $\mu_A + \mu_B - \mu_A \cdot \mu_B$; $\min(1, \mu_A + \mu_B)$; корисно ввести поняття трикутної конорми (t-конорма).

Властивості конорми:

1. $S(1,1) = 1$; $S(\mu_A, 0) = S(0, \mu_A) = \mu_A$; – обмеженість;
2. монотонність;
3. $S(\mu_A, \mu_B) = S(\mu_B, \mu_A)$; – комутативність
4. $S(\mu_A, T(\mu_B, \mu_C)) = S(T(\mu_A, \mu_B), \mu_C)$; – асоціативність.

Нечіткі нейрони. Нехай x – вхідні сигнали, а w – синаптичні ваги нейрону.

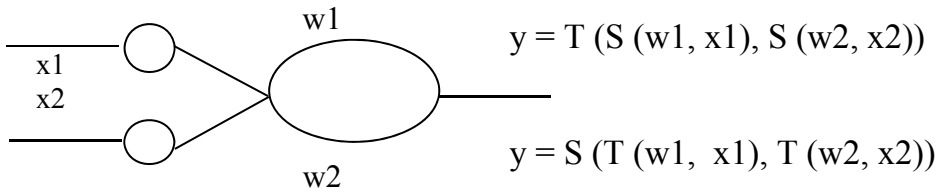


Рис. 7.6. Нечіткі нейрони

Визначимо нейрон «І» як y = T (p1, p2) = T (S (w1, x1), S (w2, x2)),
p1 = S (w1, x1).
Визначимо нейрон «АБО» як y = S (p1, p2) = S (T (w1, x1), T (w2, x2)),
p1 = T (w1, x1).

Роботу нейрона І можна проілюструвати наступним чином. Спочатку вибираються сигнали великої кількості агентів, які перевищують поріг (у кожного агента він може бути різним) їх сприйняття нейроном. Цей поріг заданий синаптичною вагою. З тих сигналів, які перевищили поріг і з фону інших синаптичних ваг обирається мінімальне значення, яке і є реакцією нейрона[29]. Нейрон АБО – це, навпаки, вибір максимального значення з безлічі сигналів, що не перевищують допустимий поріг (визначається для кожного агента синаптичною вагою) і фону інших синаптичних ваг[30].

[29] Це подібно до того як владна структура вибирає свою мінімальну реакцію на почуті вимоги активних людей з натовпу.

[30] Тут це схоже на те як влада вибирає найбільш ефективне рішення з усіх пропозицій, які не порушують деякі обмеження, наприклад, відкидають надмірно радикальні і неприйнятні пропозиції.

Взагалі кажучи, можна простежити зв'язок між принципами нечіткої логіки і роботою нейтронних мереж. Багато алгоритмів нечіткої логіки побудовані на кон'юнкції вхідних даних, яка набуває форму вибору мінімального сигналу (жорсткий зв'язок, жорсткий відбір) або змішування сигналів (множення характеристичних функцій – м'який зв'язок, м'яка взаємодія, участь), потім композиція результатів набуває форму диз'юнкції (вибір або максимального сигналу або складання сигналів). У реальної нейронної мережі слабкі вхідні сигнали посилюються, що не можна сказати про сигнали сильні. Подальші операції подібні до композиції, синтезу сигналів.

Подання нейронної мережі з нечіткими нейронами (реалізує алгоритм Tsukamoto).

П1: якщо x1 є L1, якщо x2 є L2, якщо x3 є L3, то y є G,

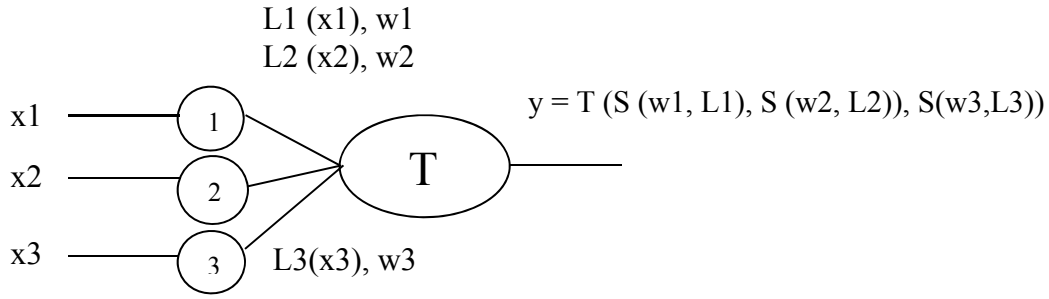
П2: якщо x1 є H1, якщо x2 є H2, якщо x3 є L3, то y є P,

П3: якщо x1 є H1, якщо x2 є H2, якщо x3 є H3, то y є R.

Визначимо рівні відсікання як (другий шар мережі)

$$\alpha_1 = L_1 \wedge L_2 \wedge L_3,$$
$$\alpha_2 = H_1 \wedge H_2 \wedge L_3,$$
$$\alpha_3 = H_1 \wedge H_2 \wedge H_3.$$

Тут рівні відсікання α – це є результат дії нейронів «І», наприклад, мінімум всіх змінних L або H. Наприклад, дія нейрона «І» для отримання $y \Rightarrow \alpha_1 = L_1(x1) \wedge L_2(x2) \wedge L_3(x3)$ з урахуванням синаптичних ваг даного нейрона на кожному каналі введення можна представити у вигляді



До речі, наявність синаптичних ваг дозволяє розширити можливості алгоритму, ***фактично перетворюючи його в нейронну мережу***. Нейронна мережа з урахуванням наведеного вище пояснення набуває вигляду:

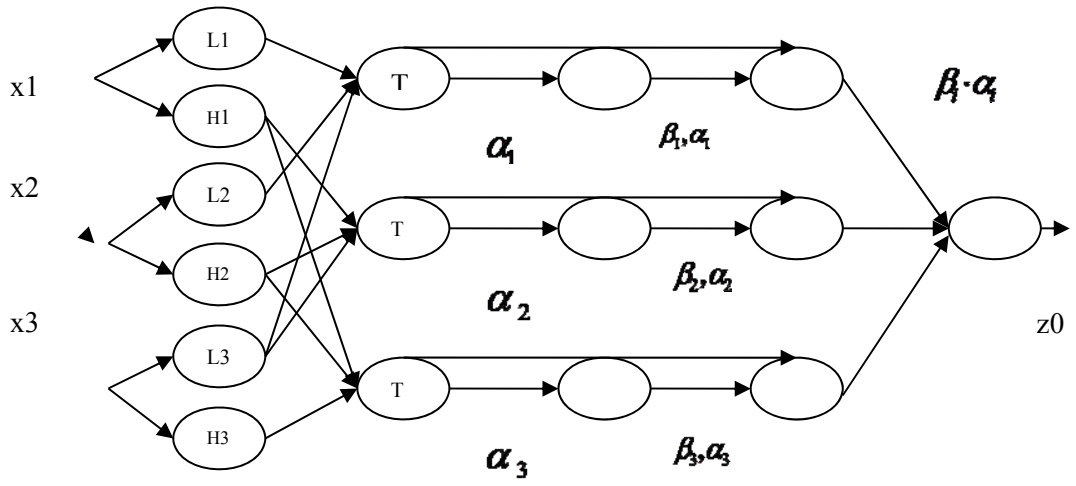


Рис. 7.7. Нейронна мережа на основі нечіткої логіки

Визначимо також відображення (четвертий шар мережі)

$$z_1 = G^{-1}(\alpha_1)$$
$$z_2 = P^{-1}(\alpha_2)$$
$$z_3 = R^{-1}(\alpha_3)$$

де, наприклад, G^{-1} – є функція зворотна функції G.

А також приведення до чіткості

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3}{\alpha_1 + \alpha_2 + \alpha_3},$$

для чого корисно визначити величини виду $\beta_i = \dfrac{\alpha_i}{\alpha_1 + \alpha_2 + \alpha_3}$ (третій шар мережі).

До речі, той факт, що в основі такої мережі лежить деякий цілком певний алгоритм може викликати сумніви в застосовності для таких систем поняття «нейронна мережа». Навіть наявність нечітких нейронів не дає підстави вважати їх повноцінними нейронами, хоча тут є помітна їх схожість з нейронами природними і штучними. Але треба не забувати, що варіанти нейронних мереж (спеціалізованих для різних застосувань), зібрані в величезних бібліотеках, подібні в принципі мережам, побудованим на алгоритмах нечіткої логіки. Бо вибір архітектури спеціалізованої штучної нейронної мережі подібний вибору алгоритму мережі на нечіткій логіці, а деяке свавілля відповідно архітектури та алгоритму, формують вже нейрони, що настроюються. Об'єднує природні і штучні нейронні мережі, і мережі на основі нечіткої логіки розмиття істинності і хибності в формі функцій належності. А також можливість настройки нечітких

нейронів, тобто можливість навчання мережі, нехай і в певних межах, пов'язаних з вибором базового алгоритму.

Введені в опис так звані нечіткі нейрони дозволили зрозуміти характер роботи таких нейронних систем з нечіткою логікою, і з'явилася можливість переглядати процес вирішення. Ці системи дозволяють, правда на силу, але все-таки розуміти, що відбувається зі знанням в процесі розв'язання такою нейронною мережею задач. Таким чином відбулося ***історичне об'єднання штучних нейронних мереж і експертних систем прийняття рішень на основі логіки***. Цей процес ще не завершений, але його темп вселяє впевненість в його успішності.

Стандарти застосування

ANFIS (adaptive network-based fuzzy inference system) – 1. на вході – значення переводяться в функції належності, 2. Т – норма, 3. нормалізація значення, 4. формується нечіткий висновок, 5. дефазифікація. Налаштування за допомогою алгоритму зворотного поширення помилки.

NNFLC (neural–network–based fuzzy logic control system = controller) – об'єднання нечіткого контролера і нейронної мережі.

Проблема: правильна побудова 1. функцій належності і 2. нечітких правил.

ОТЖЕ

Розв'язання задач в математичній логіці засноване на формуванні понять (перший рівень – це літерали в теорії предикатів) і створенні правил, за якими вони взаємодіють (другий рівень – речення в теорії предикатів). Правила взаємодії речень сформовані на основі формалізму резолюції. Це еквівалентно твердженням, що розв'язання задач засноване на дедукції. ***Пряма дедукція від фактів до висновків*** *має труднощі з вибором шляхів розв'язання і не знайшла застосування в мовах логічного програмування. Однак вона корисна для самонавчання, для отримання нового знання з раніше поданого.*

Зворотна дедукція, (фактично – це доведення теорем) – від питань до фактів, *навпаки отримала розвиток в мовах логічного програмування (наприклад, ПРОЛОГ). Подальший розвиток логічних систем – це семантична павутина, формування предикатів високих порядків і далеких, і ближніх зв'язків між реченнями. У мовах логічного програмування, заснованих на логіці предикатів першого порядку (і в перспективі на семантичних мережах, оформлених подібним чином), алгоритм вирішення (резолюція) побудований на узгодженні гілок (речень) явно або неявно створюваного машиною графа розв'язання (тобто це доведення теорем). Кваліфікація користувача може бути досить невисокою, що дозволяє застосовувати побудовані на цих мовах експертні системи повсюдно.*

Програміст може побудувати логічну систему зв'язків, орієнтуючись на тип задачі між заданими в початкових умовах поняттями і дозволити машині шукати умови узгодження початкових даних і обраної логічної схеми. По суті, розв'язком є ці підібрані машиною умови. На основі функціональних мов (типу ЛІСП і його модифікації) з розвиненим логічним формалізмом, це цілком можна робити. Однак вимоги до кваліфікації програміста значно вищі, ніж в разі застосування мов логічного програмування. Оскільки роль програміста при використанні функціональних мов більш відповідальна, сама діяльність має явно творчий характер, інтелектуалам цей підхід імпонує більше.

Що ж собою являли традиційні нейронні мережі? Тут на вхід мережі подається запит – сигнал, і мережа реагує на нього на виході, розподіляючи сигнали в n-вимірному просторі значень. У найпростішому випадку – це одновимірний простір значень. У двовимірному – це значення, розподілені на площині двох вихідних параметрів.

Подібні (близькі за змістом) розв'язки локалізуються в якомусь одному місці простору виведення. Для того щоб розділяти розв'язки різного типу (класу, виду), потрібно, щоб області локалізації розв'язків різних класів не перекривалися. Саме тому перші нейронні мережі називалися персептронами, вони були орієнтовані на впізнавання об'єктів. Таким чином в персептроні створюється перший рівень – система понять.

З ускладненням нейронної мережі формується другий рівень – зв'язок між створеними в ньому поняттями, тобто речення-правила, чому теж доведеться мережу навчати.

***

Подальший розвиток нейронної мережі – це вже формування семантичної павутини – прямих (сильних) та асоціативних (слабких) зв'язків між реченнями – правилами. Природний інтелект (людський розум) – приклад подальшого розвитку і ускладнення нейронної мережі, де відбувається посилення можливостей системи зв'язків – «перколяція образів», – що дозволяє формувати динамічні картини уявлень, те, що прийнято називати уявою.

Чи можна сподіватися на появу штучного інтелекту, який можна порівняти з можливостями природного?

Коли ми хочемо створити штучну (саме) інтелектуальну систему і маємо намір заповнити її базу даних і знань, ось тут-то і виникають проблеми. По-перше, у людини маса знань, які вона вважає відомими (за замовчуванням), машині все це треба «розжувати» і пояснити. По-друге, заповнення бази даних машини повинні робити експерти, а їх робота – високооплачувана. По-третє, час, який витрачається на заповнення баз даних, перевірку цього заповнення досить значний. Вже від цього опускаються руки. Тому, поки не знайдуть ефективний машинний спосіб автоматичного заповнення баз даних, справа швидко не піде, хіба що у військових.

До речі, як у нейронних мереж. Там виявився вельми корисний машинний режим навчання нейронних мереж, зокрема, мереж Теувово Кохонена. Бентежить також явна невідповідність швидкості реакції природного нейрона і нейрона штучної мережі, тобто швидкості окремих операцій обчислювальних систем. Проте швидкість прийняття рішення людиною часто істотно перевершує швидкість досягнення результату у сучасних машин. Особливо в умовах одночасного розв'язання багатьох задач. У чому ж справа? Справа швидше за все в тому, що мозок людини – це мультипроцесорна система, причому кількість процесорів – окремих нейронів і нейронних вузлів-скупчень, – обчислюється сотнями мільйонів. Одночасне і незалежне підключення цього гігантського числа процесорів до зовнішніх рецепторів і датчиків прискорює прийняття рішення в таке ж число раз. Тому всі зусилля дослідників корисно застосувати в галузі розробки мультипроцесорних (швидше навіть мега процесорних) систем нових поколінь.

Про перспективи подальшого розвитку глобальної інтелектуальної системи

Природа створила нейронну мережу кори головного мозку, яка дозволяє вищим живим організмам вирішувати завдання виживання в цьому світі. Зростання обсягу кори головного мозку, тобто накопичення кількісних змін, дозволило з'явитися змінам якісним. Вважається, що зростання зв'язків в нейронній мережі кори, що розширюється, призвело до ефекту перколяції, тобто різкого збільшення взаємодії між нейронами в різних областях мозку і настільки ж різкого збільшення швидкості передачі інформації. Зауважимо, що еволюційний розвиток кори головного мозку супроводжувався і розвитком систем, що підтримують її функціонування, а також систем, що забезпечують зв'язок із зовнішнім світом і з механізмами впливу на нього.

Обвальне зростання кількості браузерів, зростання кількості зв'язків між ними призводить до формування нейронної мережі, яку можна порівняти з нейронною мережею кори головного мозку людини. Паралельно розвиваються системи підтримки і забезпечення функціонування Інтернету, ускладнюється мережева структура. Зрозуміло, що настане момент, коли ступінь складності, обсяги інформації глобальної мережі досягнуть рівня, що дозволяє незалежно від нашого бажання дати початок процесу формування в ній другої сигнальної системи[31]. Основною проблемою людства при розвитку такої глобальної планетарної інтелектуальної системи буде проблема взаємодії з нею цивілізації.

І тут потрібно освоєння мови, на якій з цієї планетарною інтелектуальною системою, що взагалі кажучи не залежить від людей, можна буде спілкуватися. Такою мовою може бути мова, розроблена на основі нечіткої логіки. Або доведеться навчити інтелектуальну систему глобальної мережі мові людства, але це вбачаємо більш ніж проблематичним, бо може просто не відповідати її бажанням.

[31] См. Куклин В. М. Взгляд на будущее планетарной цивилизации / В. М. Куклин // Universitates: Наука и просвещение. – 2003. – № 4 (16). – С. 18–22. Куклин В. М. О пользе размышлений / В. М. Куклин – Харьков: ХНУ имени В. Н. Каразина, 2008. – 212 с.

РОЗДІЛ 8
ПЛАНИ ДЛЯ РОБОТА

8.1. Процедури складання програми дій робота в системі STRIPS

Вивчення цього питання раціонально починати з розгляду конкретних прикладів [7].

Розглянемо досить просту ситуацію, коли кубик С стоїть на кубику А *ON (C, A)*, а поверхня кубика В вільна *CLEAR (B)*, причому кубики А і В стоять на столі *ONTABLE(A); ONTABLE (B)*. Рука робота вільна *HANDEMPTY*. Таким чином, база даних така:

HANDEMPTY, CLEAR (B), CLEAR (C), ONTABLE (B), ONTABLE (A), ON (C, A).

П-правило. Нагадаємо, що таке П-правило, яке вперше було застосовано у розділі 1.

unstuck (x, y)

P & D HANDEMPTY, CLEAR (x), ON (x, y) P = prediction, D = delete list

HOLDING (x), CLEAR (y) A = add formula

Це правило означає, що треба «взяти х, який стоїть на у (або на позиції у)». Перший рядок – це передумови застосування правила (тобто умови, які є необхідними для його виконання): HANDEMPTY, CLEAR (x), ON (x, y). Другий рядок – це наслідки застосування цього правила: HOLDING (x), CLEAR (y).

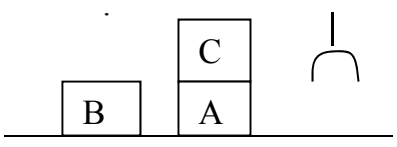
Технологія виконання завдання (система STRIPS) за допомогою П-правил

Розглянемо ситуацію, коли кубик С стоїть на кубику А, а поверхня кубика В вільна, причому кубики А і В стоять на столі. Рука вільна.

HANDEMPTY, CLEAR (B), CLEAR (C), ONTABLE (B), ONTABLE (A), ON (C, A)

Мета для робота поставити кубик С на кубик В, а кубик А – зверху на кубик С:

$$ON\ (C, B) \wedge ON\ (A, C)$$



Побудуємо ***послідовність вибору стратегії (план):***

1. Ціль явно складова, тому вона розбивається відразу ж на підцілі (пов’язані зрозуміло). Перша підціль ON (C, B), а друга підціль ON (A, C).

2. Вибір з чого починати, взагалі кажучи, випадковий, але, припустимо, почали з першої підцілі.

3. Перевірка передумов. Перевіряються відповідність правил з набору, даної підцілі. Правило застосовується (активується), якщо його можна виконати. Для з'ясування можливості виконати перевіряються передумови (виконання) правила. Наприклад, передумовою (виконання) правила **unstuck** (x, y) – «взяти x, який стоїть на y (або на позиції y)»:

$$\text{HANDEMPTY} \wedge \text{CLEAR}(x) \wedge \text{ON}(x, y)$$

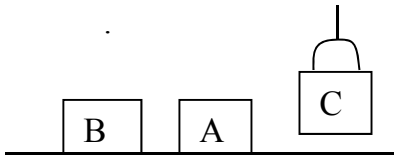
4. Пошук уніфікації. База даних дозволяє виконання правила **unstuck** (C, A) при наступній уніфікації x = C, y = A. Рука робота вільна, поверхня кубика чиста, він стоїть на кубику A :

$$\text{HANDEMPTY} \wedge \text{CLEAR}(C) \wedge \text{ON}(C, A)$$

5. Наслідки застосування правила **unstuck** (x, y) призводять до того, що рука після цього зайнята: HOLDING (x), а поверхня, на якій стояв x – очищена: CLEAR (y). Ці зміни повинні бути неодмінно узгоджені з виконанням правила.

6. Одразу зазначимо, що оператори HOLDING (x) і HANDEMPTY конкурують (суперечать один одному) і той, хто останній, виключає попередній. Також, оператори ON (C, A) і CLEAR (A) не можуть існувати одночасно, залишається тільки останній в реченні.

7. Виникає новий стан (нова база даних). Нові елементи HOLDING (C), CLEAR (A), і старі CLEAR (B), ONTABLE (B), ONTABLE (A). Тут же зазначимо, що ті з елементів бази даних, які не брали участі в операціях та змінні яких не згадували в процедурі застосування правила, залишаються без зміни.



8. Предумови + уніфікація. Тепер потрібно шукати нове правило, передумови якого не суперечать новій базі даних і визначаються першою підціллю. Наприклад, для правила **stuck (x, y)** – «x, який тримає рука, поставити на y (або на позицію y)» потрібно, щоб те, що потрібно поставити на y було саме x, а поверхня y була вільна. Іншими словами, в загальному вигляді передумова правила **stuck** (x, y)

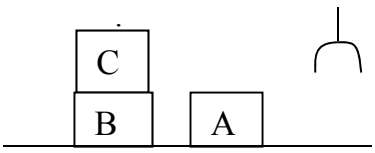
$$\text{HOLDING}(x)\ \text{CLEAR}(y)$$

Ця передумова має бути обов'язково пов'язана із застосуванням попереднього правила.

9. У разі уніфікації x = C, y = B, передумови виконуються і можна застосовувати правило **stuck** (C, B).

10. Наслідки застосування правила: через взаємну невідповідність HOLDING (x) і HANDEMPTY, залишається тільки HANDEMPTY, а CLEAR (B) замінюється на ON (C, B) відповідно до цього правила.

11. Виникає знову новий стан (нова база даних). Нові елементи HANDEMPTY, ON (C, B), підтверджується CLEAR (C), і залишаються колишні CLEAR (A), ONTABLE (B), ONTABLE (A).



12. Важливо, що в новій базі даних з'являється перша підціль ON (C, B). Факт її появи (локальна термінальна умова) зупиняє всі процедури для досягнення першої підцілі.

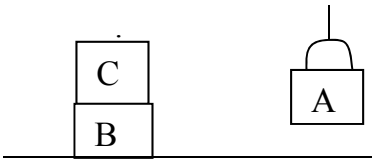
13. Тепер розглядається друга підціль ON (A, C). Для її реалізації потрібно мати вільну поверхню кубика C і вільну поверхню кубика A. Тобто, CLEAR (A) і CLEAR (C). Але кубик A стоїть на столі.

14. Теперь починається пошук нового правила, тобто узгодження передумов правил з новою базою даних. Зрозуміло, що підходить нове правило **pickup** (x) – «взяти x зі столу», причому з уніфікацією x = A.

15. Використовуємо **pickup** (A). Далі: узгодження правил і передумов з оновленою базою даних.

16. Наслідки застосування правила: через взаємну невідповідність HOLDING (x) і HANDEMPTY залишається тільки HOLDING (A), а CLEAR (A), ONTABLE (A) згідно з цим правилом видаляються.

17. Отримуємо нову базу даних. Нові елементи HOLDING (A), і залишаються ON (C, B), CLEAR (C), ONTABLE (B).



18. Тепер шукається правило, передумова якого відповідає другій підцілі і останній базі даних. Згідно з другою підціллю, потрібно поставити кубик A на кубик C, для цього потрібно тримати кубик A в руці і мати вільну поверхню на кубику C. Є таке правило – **stuck** (x, y) – «x, який тримає рука, поставити на y (або на позицію y)», раніше ним вже користувалися. Крім того, з бази даних випливає, що передумови цього правила при уніфікації x = A, y = C теж виконані.

19. Застосовуємо **stuck** (A, C).

20. Нова база даних: Нові елементи HANDEMPTY, ON (A, C), і залишаються ON (C, B), ONTABLE (B). Тут після дії правила з'явилися нові елементи і видалено CLEAR (C), що відповідає результатам його застосування. Поява в базі даних відразу двох підцілей, які неявно пов'язані кон'юнкцією, відповідає повній термінальній умові. Тобто мета досягнута.

Таким чином, побудований, так званий, стік цілей (послідовність застосування правил):

{ **unstauk** (C,A), **stuck**(C,B), **pickup**(A), **stuck** (A,C)}

Форма запису процедур зазвичай така

Таблиця 8.1

HANDEMPTY ∧ CLEAR(C) ∧ ON (C, A) ∧ CLEAR (B)
unstuck (C, A)
HOLDING (C) CLEAR (A) CLEAR (B) HOLDING (C) ∧ CLEAR (B) ∧ CLEAR (A)
stuck(C, B)
ON (C, B) ON (A, C) ON (C, B) ∧ ON (A, C) і, мабуть, HANDEMPTY

* Перед першим правилом наведено всі передумови його застосування (окремо і разом). Записуються тільки основні літерали, ті, які не використовуються, буває, не пишуть.

** Під застосованим правилом виписаний результат його застосування і передумови застосування наступного правила.

*** Внизу виписуються всі підцілі і мета повністю. В одну групу виділяються компоненти мети і складові мети.

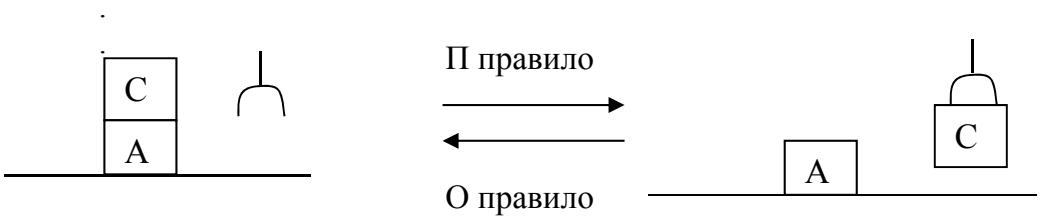
Взагалі кажучи, можна помітити, що викликаються всі правила, передумови яких відповідають базі даних і першій підцілі. Спробуйте самі скласти повний стік цілей в схемі розв'язання, яка представлена в цьому розділі.

Система STRIPS поводиться нехитро. Отримавши завдання, вона розв'язує його частину (підціль), а якщо виникають труднощі з розв'язанням наступної частини завдання (наприклад, іншої підцілі), вона, анітрохи не вагаючись, здатна зруйнувати попередній розв'язок і досягти цієї вже другої підцілі. А потім, як ні в чому не бувало, повернутися до першої підцілі. Це нагадує поведінку людини, яка відразу береться за справу, не обдумуючи свої дії, а діючи по ситуації. При цьому вона здатна, якщо не виходить за один раз, починати роботу знову і знову, поки не вийде. Нижче розглянемо систему RSTRIPS поведінка якої інша і нагадує поведінку людини, яка перш ніж діяти, розробляє детальний план своєї поведінки. Причому якщо щось в плані не виходить, він цей план коригує, переробляє і домагається його (плану) раціональності і несуперечності. Щоб потім, в процесі вже його виконання, не заганяти себе в кут, не створювати конфліктних ситуацій

8.2. Конструювання програми дій робота за допомогою О-правил

Прямий метод перебору правил, розглянутий вище, прийнятний, якщо їх небагато, якщо ж база даних і правил велика, то можна поступити інакше – ввести т. зв. О-правила.

О-правило. Можна ввести поняття О-правила, Якщо, в рамках того ж прикладу, задана мета CLEAR (A) і слід знайти П-правило, застосування якого призвело до цієї мети. Неважко бачити, що це П-правило може бути **unstuck** (x, A). Таким чином, О правило – це правило, зворотне **unstuck** (x, A).



Процедура пошуку О-правила. Якщо задана мета CLEAR (A), перевіряються всі П-правила, які містять в списку додатків CLEAR (y), наприклад, **unstuck** (x, y). Його передумови HANDEMPTY CLEAR (x) ON (x, y). При уніфікації y = A, передумови матимуть вигляд HANDEMPTY CLEAR (x) ON (x, A), де x – не визначено. Якщо є ситуація, коли на кубику A стоїть якесь x, підходить О-правило, зворотне **unstuck** (x, A).

Регресія. Процедура, що дозволяє провести пошук О-правил, називається регресією. Регресія перевіряє, чи відповідають поточному стану бази даних умови використання даного правила.

Приклад регресії

Розглянемо, що таке регресія на попередньому прикладі. Нехай кубик C стоїть на кубику A, а кубики A і B стоять на столі, при цьому рука робота вільна: HANDEMPTY, CLEAR (B), CLEAR (C), ONTABLE (B), ONTABLE (A), ON (C, A) – це початкова база даних.

Розглянемо правило

unstuck *(x, y)*

P & D HANDEMPTY, CLEAR (x), ON (x, y)

A HOLDING (x), CLEAR (y)

Регресія для HANDEMPTY (узгоджена з базою даних) на правилі **unstuck** *(x, y), – це F (хибне). Тобто, якщо дописати HANDEMPTY знизу, наприклад, як передумову для задоволення наступного правила, то воно не буде погоджене з правилом* **unstuck** *(x, y), точніше з наслідками його застосування на базі даних.*

Регресія ONTABLE (C) на правилі **unstuck** *(x, y) із урахуванням бази даних – це F (хибне). Тобто і в частині початкової бази даних, що не була пов'язана з застосуванням правила, і в наслідках дії правила це не реалізується.*

Регресія для CLEAR (A) на правилі **unstuck** *(x, y) із урахуванням бази даних – це T (істинне). Оскільки кубик C знімається з кубика A.*

Регресія для CLEAR (B) на правилі **unstuck** *(x, y) із урахуванням бази даних – це CLEAR (B). Оскільки кубик B не чіпають, він стоїть так, як стояв.*

Механізм формування програми дій робота з використанням регресії. О-правила використовуються системою для створення макета (неперевіреного списку, структури) програми дій робота, який потім уточнюється і перевіряється.

1. Ціль розбивається на підцілі (пов'язані кон'юнкцією).

2. До першої підцілі послідовно застосовуються відповідні О-правила (використовуючи істинність регресії кожної конкретної мети на даному правилі) поки система не прийде у відповідність з початковою базою даних.

3. Потім включається процедура перевірок і узгодження.

4. Потім все повторюється для наступної підцілі ...

Перші два пункти процедури, виписаної вище, представлені нижче в таблиці

Таблиця 8.2

HANDEMPTY ∧ CLEAR (B) ∧ **CLEAR (C)** ∧ ∧ ONTABLE (B) ∧ ONTABLE (A) ∧ **ON (C, A)**
CLEAR (B) ∧ ∧ **HANDEMPTY** ∧ **CLEAR (C)** ∧ **ON (C, A)**
unstuck (C, A)
CLEAR (A) ∧ ∧ **HOLDING (C)** ∧ CLEAR (B)
stuck(C, B)
ON (C, B) ∧ HANDEMPTY
ON (C, B) ON (A, C) ON (C, B) ∧ ON (A, C)

У верхньому рядку таблиці наведена початкова база даних, в нижньому рядку – підцілі і повна мета завдання. Тут виділено підціль ON **(**C, B**)**[32]. Підціль ON (C, B) – це частина результату застосування правила, в даному випадку підібраного за цим критерієм **stuck** (C, B), тобто ON (C, B) ∧ HANDEMPTY, а передумова його виконання HOLDING (C) ∧ CLEAR (B). Регресія ON (C, B) на правилі **stuck** (C, B) дає T, тобто таке правило підходить HOLDING (C) – це результат дії іншого правила, наприклад, **unstuck** (C, A). Регресія HOLDING (C) на правилі unstack (C, A) дає теж T, а регресія ще однієї необхідної частини передумови CLEAR (B) на цьому ж правилі дає CLEAR (B)[33]. Тобто, і в цьому випадку застосування такого правила виправдано.

[32] Вибір цієї підцілі може бути випадковим або певною додатковою уточнюючою умовою (яка випливає з повної мети) типу ONTABLE (B), тобто кубик B повинен стояти на столі, тому всі переміщення починаються з нього, значить вибирається та підціль, яка має на увазі позицію кубика B внизу, тобто ON (C, B).

[33] Тут регресія HOLDING (C) підтверджує список передумов і нічого записувати не треба, а регресія CLEAR (B) і дорівнює CLEAR (B), тому вимагає записи перед цим правилом. Однак, якщо цей літерал записаний в тій частині початкової бази даних, яка не змінилася

8.3. Технологія розв'язання задачі в системі RSTRIPS

При виборі другої підцілі, вирішення з видаленням елементів бази даних призведе до очевидних труднощів, заводячи систему в глухий кут. Саме такі проблеми і привели до модернізації системи STRIPS. Такою модернізованою системою стала RSTRIPS.

Розглянемо основні особливості цієї системи.

Маркер і збереження передісторії. Форма запису процедур перед застосуванням першого правила трохи змінюється

Таблиця 8.3

HANDEMPTY ∧ CLEAR(C) ∧ ON (C, A)
unstuck (C, A)
*HOLDING (C)_____ CLEAR (B) HOLDING (C) ∧ CLEAR (B)
stuck (C, B)
ON (C, B) ON (A, C) ON (C, B) ∧ ON (A, C)

* Під умовою HOLDING (C) розміщують маркер _____.
Маркер показує, що тільки що виконано HOLDING (C).
** Всі правила вище маркера були застосовані.
*** Умови нижче маркера перевіряються прямо зараз.
**** Маркер, входячи в зону дії правила, залишає вгорі досягнуті цілі, а внизу цілі, які належить досягти. Важливо, що досягнута підціль * HOLDING (C), позначена зірочкою. Система забороняє її викреслювати або робити помилковою, тобто застосовує захист цієї мети. Якщо застосування наступного правила корелює із захищеною підціллю, то захист знімається, а якщо не корелює (тобто зазначена зірочкою підціль не залежить від застосування правила), то захист залишається.

Раніше в STRIPS застосовувалася процедура викреслювання, тобто заміна попередньої бази даних іншою БД, оновленою в результаті застосування правил. Тут, в системі RSTRIPS, немає необхідності викреслювання, бо послідовність застосування операцій визначається положенням маркера. Крім того, система «пам'ятає» всі попередні операції і колишні бази даних.

перед застосуванням правила, то такий запис, взагалі кажучи, не є обов'язковим, бо мається на увазі за замовчуванням.

Система повинна перевірити два типи умов. Перша умова (крок назад) – чи відповідають передумови[34] підібраного системою нового правила попередньому, вже виконаному правилу[35]. За необхідності розглядають і другу умову (крок вперед) – нове правило, що вибране, не повинно порушувати ранніх захищених підцілей.

Процедура, що дозволяє провести перевірку виконання першої з умов, – це регресія. Проведення регресії з кроком назад дозволяє перейти до застосування нового правила.

Щодо кроку вперед, то ***якщо збереження досягнутої раніше захищеної мети виявляється неможливим, то при подальшому розвитку плану (програми), скасовується не тільки захист цієї мети разом з правилом, яке її породило. Скасовується також і подальше – нове правило (як показано нижче,) яке використовує цю досягнуту мету як передумову.***

Подібне скасування попереднього правила відбувається і pfнеможливості застосування будь-якого подальшого правила, якщо воно порушує захист вже досягнутої підцілі, явно декларованої як невід'ємна частина загальної мети в умовах завдання. Тоді доведеться повернутися назад, видаляючи обидва правила – попереднє і наступне.

Пряма процедура створення програми в системі RSTRIPS. Тепер розглянемо повністю процедуру розв'язання задачі. Мета (ціль) ON (C, B) $\wedge$ ON (A, C) розбивається на підцілі: перша підціль – ON (C, B), друга підціль ON (A, C). Перша підціль ON (C, B) з бази даних HANDEMPTY, CLEAR (B), CLEAR (C), ONTABLE (B), ONTABLE (A), ON (C, A) знаходить передумови правила **unstuck** (x, y), то є HANDEMPTY, CLEAR (C), ON (C, A). При уніфікації x = C, y = A. Це правило стає на чолі списку. Маркер після результату дії правила переміщається нижче результату HOLDING (C) – цей літерал позначається зірочкою, що означає його захист.

Система шукає нове правило, передумова якого включає HOLDING (C) і літерали зі змінною B, тобто CLEAR (B) і ONTABLE (B). Таким чином, виконані передумови правила **stuck** (x, y) при уніфікації x = C, y = B.

Однак для застосування цього правила, слід перш знайти регресію CLEAR (B) на попередньому правилі **unstuck** (C, A). Регресія дорівнює CLEAR (B), тобто явно не «хибне», що дозволяє застосування наступного правила **stuck** (C, B).

Одночасне виконання всіх передумов HOLDING (C) CLEAR (B), зміщує маркер нижче і включає правило **stuck** (C, B). Маркер після застосування правила переміщається і зупиняється нижче за результат застосу-

---

[34] Йдеться також про передумови, що не є наслідками застосування попереднього – тільки-но виконаного, – правила, а взяті з ранньої (або навіть початкової, в тій її частині, яка не змінилася) бази даних.

[35] Якщо ця умова порушена, система буде шукати правила, дія яких введе потрібну передумову.

вання цього правила ON (C, B). Захист з HOLDING (C) знімається. Тепер підціль – ON (C, B) досягнута і захищена – перед нею ставлять зірочку.

Таблиця 8.4

HANDEMPTY ∧ CLEAR(C) ∧ ON(C,A)
unstuck(C,A)
HOLDING(C) CLEAR (B) HOLDING(C) ∧ CLEAR (B)
stuck(C,B)
*ON(C,B) HANDEMPTY ∧ CLEAR (A) ∧ ONTABLE (A)
pickup(A)
*HOLDING(A) CLEAR (C) HOLDING(A) ∧ CLEAR (C)
stuck(A,C)
ON(A,C) ON(C,B) ∧ ON(A,C)

Тепер перша підціль досягнута. Система вибирає другу підціль ON (A, C). Шукаються правила, передумови яких відповідають базі даних і містять змінні A і C. Одне з них **pickup** (x) при уніфікації x = A має всі необхідні і достатні передумови

HANDEMPTY ∧ CLEAR (A) ∧ ONTABLE (A)

В результаті його дії маркер рухається далі і опускається нижче результату застосування HOLDING (A), на яких ставлять захист. *Але захист з ON (C, B) не знімається, бо правило pickup (A) не корелює (немає загальних змінних) з ON (C, B).*

І це важливо, бо підціль ON (C, B) ***не може бути порушена в разі застосування будь-якого подальшого правила. Бо це половина всієї мети завдання*** ON (C, B) ∧ ON (A, C).

Система шукає правило, для передумов якого має місце захищене HOLDING (A) і ті компоненти бази знань, які є в наявності. Це може бути CLEAR (C), і тоді загальне передумова HOLDING(A) ∧ CLEAR (C). Воно відповідає правилу **stuck** (x, y) при уніфікації x = A, y = C. Система його виконує і маркер опускається нижче результату застосування правила **stuck** (A, C), тобто ON (A, C).

Одночасне виконання всіх умов зміщує маркер нижче ON (C, B) ∧ ON (A, C).

Оскільки подальшого продовження немає − всі цілі досягнуті, − формування програми закінчено.

8.4. Вирішення проблеми взаємодії цілей

Розглянемо той же початковий стан, коли кубик С стоїть на кубику А, а поверхня кубика В вільна, причому кубики А і В стоять на столі. Рука вільна.

HANDEMPTY, CLEAR (B), CLEAR (C), ONTABLE (B), ONTABLE (A), ON (C, A).

Нова мета (ціль) для робота поставити кубик А на кубик В, а кубик В зверху на кубик С:

ON (A, B) ∧ ON (B, C)

Аналогічно до попереднього легко побудувати стек цілей[36]

{**unstuck** (C, A), **putdown** (C), **pickup** (A), **stuck** (A, B)},

при цьому підціль ON (A, B) досягнута і захищена, аналогічно тому, як це було зроблено в попередньому розділі. Маркер опустився нижче літералу ONTABLE (B), також захищеного.

Таблиця 8.5

1	*ONTABLE (A)* ∧ *ONTABLE (B)* ∧ *CLEAR (B)* ∧ ∧ **HANDEMPTY** ∧ **CLEAR (C)** ∧ **ON (C, A)**
2	**unstuck** *(C, A)*
3	*ONTABLE (A)* ∧ *ONTABLE (B)* ∧ *CLEAR(B)* ∧ *CLEAR(A)* ∧ ∧ **HOLDING (C)**
4	**putdown** (C)
5	*ONTABLE (B)* ∧ *CLEAR(B)* ∧ *ONTABLE (C)* ∧ *CLEAR(C)* ∧ ∧ **ONTABLE (A)** ∧ **CLEAR(A)** ∧ **HANDEMPTY**
6	**pickup** (A)
7	*ONTABLE (B)* ∧ *ONTABLE (C)* ∧ *CLEAR(C)* ∧ ∧ **HOLDING (A)** ∧ CLEAR (B)
8	**stuck** (A,B)
9	*ONTABLE (C)* ∧ *CLEAR (C)* ∧ *CLEAR (A)* ∧ ∧ ***ON (A, B)*** ∧ **ONTABLE (B)* ∧ ∧ **HANDEMPTY** ∧ CLEAR(z) ∧ ON(z, B)
10	**unstuck** (z, B)
11	*ONTABLE (C)* ∧ *CLEAR(C)* ∧ **ONTABLE (B)* ∧ ***CLEAR (B)*** ∧ **HOLDING (z)**

[36] Жирним шрифтом виділені основні літерали, які входять в результат і в передумови правил. Причому курсивом ті з них, які не є передумовою виконання подальшого правила.

Продовження таблиці 8.5

12	**putdown** (z)
13	*ONTABLE (C)* ∧ *CLEAR(C)* ∧ *CLEAR (z)* ∧ *ONTABLE (z)* ∧ ∧ ONTABLE (B) ∧ CLEAR (B) ∧ **HANDEMPTY**
14	**pickup** (B)
15	*ONTABLE (C)* ∧ *CLEAR (z)* ∧ *ONTABLE (z)* ∧ ∧ **HOLDING (B)** ∧ CLEAR (C)
16	**stuck** (B, C)
17	*ONTABLE (C)* ∧ *CLEAR (z)* ∧ *ONTABLE (z)* ∧ ***CLEAR (B)*** ∧ ∧ **ON (B, C)** ON(A, B) ∧ ON(B, C)

Аж до верхніх рядків 9 розділу програми система діє вже відомим з попереднього розгляду чином. Оскільки перша мета досягнута, система починає добиватися другої підцілі ON (B, C), для чого їй потрібно звільнити поверхню кубика B. Але при цьому порушується захищена підціль *ON (A, B). Таким чином, друга підціль ON (B, C) – це підціль, що порушує захищеність. Система зіткнулася з явищем ***взаємодії (конфлікту) цілей***.

Це означає, що для продовження роботи, необхідно зруйнувати частину колишніх досягнень. Саме тому раціонально спочатку створити план роботи, узгодити його, очистити від непотрібних дій, скоротити, оптимізувати, а тільки потім виконувати. Тому часто створення програми для робота називають створенням планів.

Повертаючись до STRIPS

Якщо використовувати уніфікацію z = A, то подальші дії системи цілком логічні і це повністю відповідає принципам дії системи **STRIPS**. Зрозуміло, що маркер і захист в цьому випадку відсутні. Можна побачити, що відмовляючись від досягнутої першої цілі, система **STRIPS** перемикається на виконання другої. Домігшись реалізації в розділі 17 другої цілі система **STRIPS** може повернутися до виконання першої цілі, причому її вона вже досягне без особливих зусиль. Читачеві пропонується самостійно виконати цю частину завдання.

Перш ніж обговорити особливості поведінки системи RSTRIPS при взаємодії цілей, звернемо увагу на низку важливих деталей. Строки 8 і 10 визначають виконання двох операцій прямої і зворотної, тобто **stuck** (A, B) і **unstuck** (z, B), якщо використовувати уніфікацію z = A. Відповідно при цій уніфікації база даних до застосування правила **stuck** (A, B) (строка 7) і база даних, отримана після застосування правила **unstuck** (A, B) збігаються. Взагалі кажучи, ці операції можна виключити, прибравши зі списку строки 8–11. Але це можливо, тільки при цій

уніфікації. Точно так, база даних до застосування правила **pickup** (A) (розділ 5) збігається при тій же уніфікації z = A з базою даних, отриманої після застосування зворотного правила **putdown** (z) (строка 13). І в цьому випадку результат не зміниться, якщо виключити зі списку строки 6–13. Взагалі кажучи, при даній уніфікації можна різними програмними способами забезпечити таку процедуру видалення комбінацій прямих і зворотних операцій (за необхідності переконуючись в незмінності бази даних після подібних винятків), якщо система до них приходить.

Система **RSTRIPS** цю операцію виконує наступним чином. Відволікаючись від важливих процедурних питань, переконуємося в тому, що система в 9 строкі стикається з проблемою вирішення другої підцілі. Вона перевіряє можливість захисту підцілі * ON (A, B) на нове правило **unstuck** (z, B).

Зазначимо, підціль ON (A, B) це не просто проміжний результат, який, захищається тимчасово, поки не знайдеться правило,що підходить. ***Як вже зазначалося вище, підціль ON (A, B) не може бути порушена в разі застосування будь-якого подальшого правила. Бо це половина (частина) всієї мети завдання ON (A, B) ∧ ON (B, C).***

Неможливість збереження захищеної підцілі ON (A, B) при спробі досягти другої підцілі ON (B, C) при цьому прибирає (скасовує) не тільки нове правило, а й попереднє **stuck** (A, B), видаляючи зі списку строки 8–11. Захист разом з маркером переноситься в зворотному напрямку, тобто вгору.

> **STRIPS** в цьому випадку, не звертаючи увагу на вже досягнуту мету ON (A, B), «нічтоже сумняшеся», зруйнувало б її, намагаючись досягти наступної ON (B, C). Вся справа в тому, що пам'ять про проведені операції в STRIPS стирається, і система все починає нібиспочатку.

Тепер захист прикладено до літерала HOLDING (A). Результат (7) застосування правила (6) **pickup** (A) виявляється зворотним результату (13) застосування правила (12) **putdown** (z = A), тому обидва ці правила з їх результатами можна виключити, видаляючи зі списку розділи 6–7 та 12–13. Бо в результаті їх спільної дії нічого не змінюється.

Формально ж регресія на правилі **putdown** (z), змушує позбутися цього правила і попереднього теж.

Таким чином, список скорочується і руху маркера вниз нічого не перешкоджає:

Таблиця 8.6

1	*ONTABLE (A)* ∧ *ONTABLE (B)* ∧ *CLEAR (B)* ∧ **HANDEMPTY** ∧ **CLEAR (C)** ∧ **ON (C, A)**
2	**unstuck** *(C, A)*
3	*ONTABLE (A)* ∧ *ONTABLE (B)* ∧ *CLEAR (B)* ∧ *CLEAR (A)* **HOLDING (C)**
4	**putdown** (C)
5	*ONTABLE (B)* ∧ *CLEAR (B)* ∧ **ONTABLE (C)* ∧ **CLEAR (C)* ∧ _____ **ONTABLE (A)** ∧ **CLEAR (A)** ∧ **HANDEMPTY**
6(14)	**pickup** (B)
7(15)	*ONTABLE (C)* ∧ *CLEAR (z)* ∧ *ONTABLE (z)* **HOLDING (B)** ∧ CLEAR (C)
8(16)	**stuck** (B, C)
9(17)	*ONTABLE (C)* ∧ *CLEAR (z)* ∧ *ONTABLE (z)* ∧ ***CLEAR (B)*** **ON (B, C)** ON (A, B) ON (A, B) ∧ ON (B, C)

Тепер з новими силами програма буде намагатися досягти підцілі ON (A, B).
І на її шляху вже не виникне конфліктних ситуацій.
В цьому ми рекомендуємо розібратися читачам.

Що краще: відразу робити, а потім переробляти, або спочатку подумати ...

Перша система, як ми бачимо, вирішує завдання, і якщо щось не так, просто переробляє. Дійсно, STRIPS спочатку досягає першої підцілі, потім переходить до досягнення другої підцілі. Але при цьому порушує першу підціль, щоб потім, після досягнення другої підцілі, повернутися до вирішення першої. Друга система – RSTRIPS, – спочатку створює програму дій (план), потім її коригує, перш ніж починати виконувати дії.

8.5. Представлення програми у вигляді графа. Декомпозиції графа. Система DCOMP

Ця система створює «тимчасове рішення у вигляді графу типу І / АБО. Потім цей граф коригується для зняття проблем із взаємодіями. Розглянемо вже відому ситуацію, коли кубик С стоїть на кубику А, а поверхня кубика В вільна, причому кубики А і В стоять на столі. Рука вільна.

HANDEMPTY, CLEAR (B), CLEAR (C), ONTABLE (B), ONTABLE (A), ON (C, A).

Мета (ціль) для робота поставити кубик С на кубик В, а кубик А зверху на кубик С:

ON (C, B) ON (A, C)

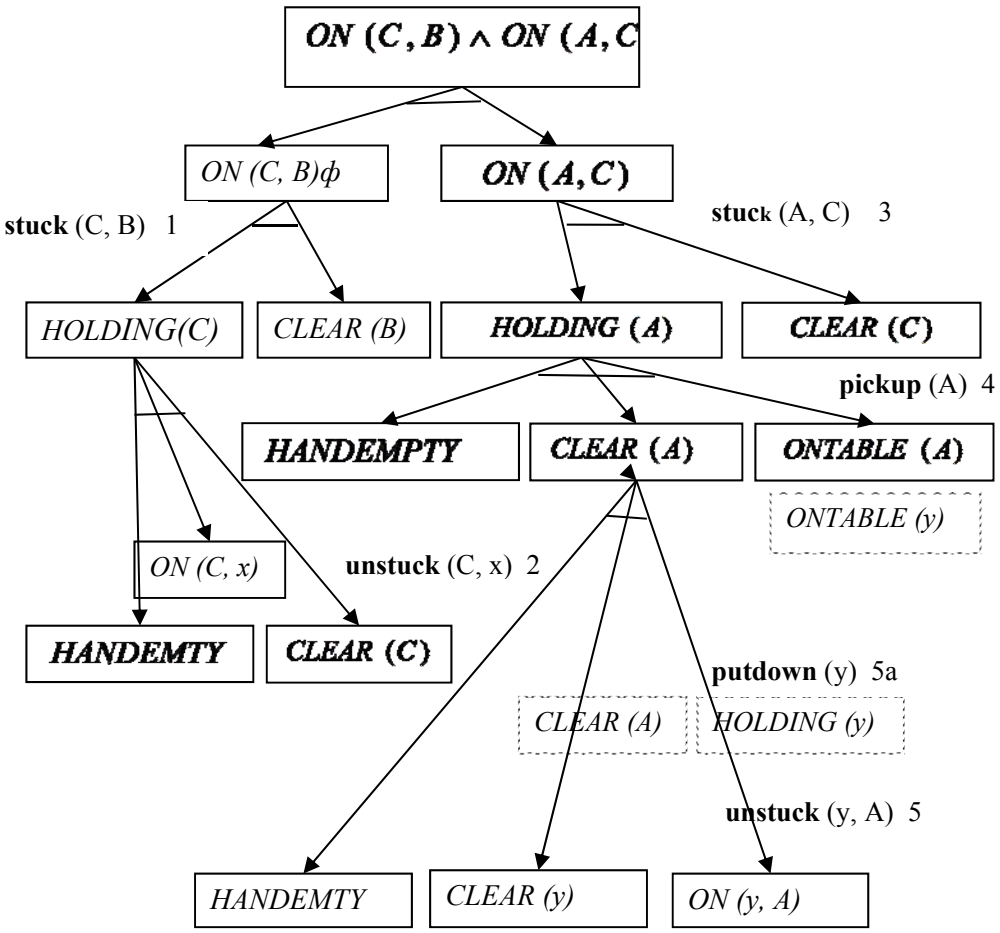


Рис. 8.1. Побудова графа «тимчасового» розв'язання. Пунктиром вставлена «латочка»

О-правила застосовують до цільових вершин, а кінцеві вершини відповідають початковим умовам. Отримане «тимчасове» рішення потребує корекції.

Латочки. Перш за все, звернемо увагу, що в структурі такого тимчасового розв'язання можливі порушення. Частина графа, яка формується з правої підцілі, може ігнорувати операції (дії), які не мають відношення до часткової програми розв'язання. Таким чином, наслідком дії 5 unstuck (y, A), є процедура очищення поверхні кубика A від стороннього предмета, який не цікавить вирішувальну систему.

Але тоді виникає протиріччя, яке визначається процедурою регресії, котра полягає в тому, що HANDEMPTY дає «хибне» при регресії на правилі 5 unstuck (y, A) яке в якості П-правила діє від низу до верху. Нагадаємо, що регресія перевіряє, чи відповідає HANDEMPTY на другій сходинці знизу рисунка наслідків застосування правила unstuck (y, A) до нижньої сходинки правої частини рисунка. Раніше рука була вільна, після застосування правила вона повинна бути зайнята, а цього немає.

Вирішенням проблеми є введення додаткового правила 5a putdown (y), яке звільняє руку робота і знімає її. При цьому регресія HANDEMPTY на

правилах виявляється істинною. Таким чином, система DCOMP здатна всюди у «тимчасових» планах накласти «латочку».

Вибір послідовності дій. Іншою важливою особливістю системи DCOMP є здатність вибору послідовності дій. Для цього в системі передбачено створення двох множин П-правил, що відповідають літералам C_{ij} використовуваним для передумов правил. Визначимо C_{ij} як і-тий літерал передумови *j*-того правила. Одна множина «руйнівників» (для *i*-того літерала, який входить в передумову *j*-того правила) і інша – «творців». ***Руйнівник*** здатний порушити передумови цього (j-того) правила. ***Творці*** додають C_{ij} і не є ні попередниками *j*-того правила, ні самим *j*-тим правилом.

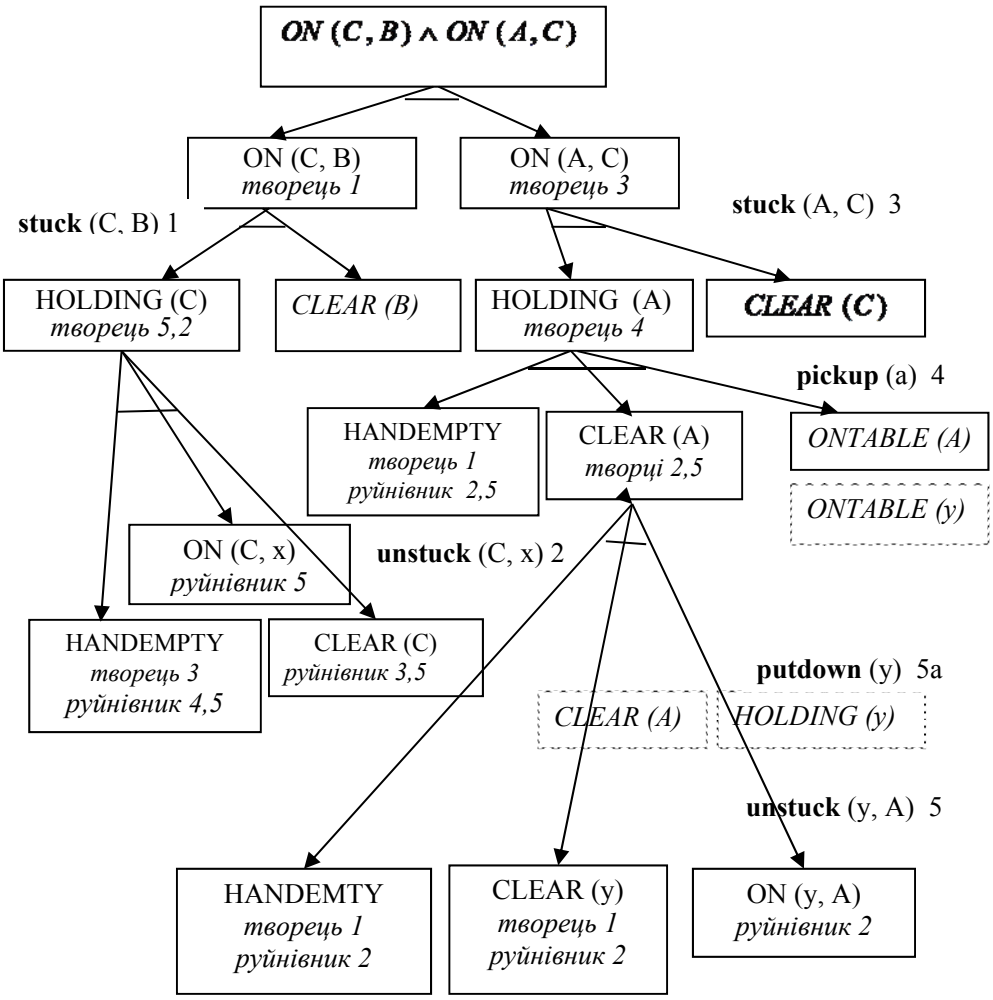


Рис. 8.2. Формування множин правил «творців» і «руйнівників» для декомпозиції графа.

Слід звернути увагу на важливу особливість. Якщо правило 2 (яке є руйнівником для літерала CLEAR (C), котрий є передумовою для правила 2) передуватиме в програмі правилу 3, то правило 3 НЕ буде викреслювати цей літерал (умову).

Тобто на цій основі можна створити метод формування послідовності застосування правил.

Якщо вставити правило 2 (при x = A) перед правилом 4, то CLEAR (A) виконується, і правило 5 взагалі можна виключити (мета 5 правила – очищення поверхні кубика A). Але виконувати правило 4 після правила 2 не можна, регресія показує помилкові значення для літералу HANDEMPTY на правилі 2. Тому потрібно скористатися правилом – творцем HANDEMPTY, вставляючи його у вигляді «латочки» між правилом 2 і правилом 4. Ця «латочка» – правило 1 **stuck** (C, B). Тепер всі проблеми вирішені. Послідовність дій: виконання поспіль правил 2, 1 ,4 , 3.

РОЗДІЛ 9
СЕМАНТИЧНІ МЕРЕЖІ

Ще Ч. Пірс запропонував прообраз семантичної мережі – екзистенціальний граф (1909). Далі цей підхід у формі графів і семантичних відношень був використаний психологами і фізіологами. Але комп'ютерні семантичні мережі розроблені були Р. Річенс (1956) для цілей машинного перекладу. Сенс семантичних мереж полягав у розвитку і різноманітності зв'язків між елементами графів. Нижче обговоримо деякі особливості опису семантичних мереж.

Однак семантичні мережі містять, крім властивих графам логічних зв'язків між поняттями, щось важливіше. Семантична мережа будується від загального до конкретного, тобто існують ієрархічні конструкції, що дозволяють переходити від більш загальних понять до понять конкретних. Тому структура семантичних мереж завжди буде відрізнятися від традиційного графа наявністю в ній узагальнень, що дозволяють при розв'язанні конкретних задач повернутися до них і знаходити зв'язки між, на перший погляд різними поняттями. І роль успадкування властивостей, зокрема, тут стає важливим інструментом пошуку відповідей.

9.1. Зв'язок семантичних мереж з обчисленням предикатів

Концептуальні графи та семантичні мережі можуть бути також мережевими версіями обчислення предикатів. Графи містять (тут) прямокутники для представлення аргументів і кола для імен предикатів. Сполучені вхідними та вихідними стрілками. Найбільш простим є концептуальний граф бінарного предиката.

Рис. 9.1. Slot-assertion

Взагалі кажучи, багато з предикатів можна представити як добуток (кон'юнкцію) декількох бінарних предикатів. При цьому використовують такі домовленості: кожна функція колишнього предиката стає іменем бінарного предиката, перший аргумент якого – ім'я початкового предиката,

а другий є значенням аргументу цієї функції, ця пара називається слотом. Інакше, слот складається з свого імені (ім'я цієї функції) і ключа (значення відповідного аргументу):

slot, slot – name, slot – value.

Отже, предикат виду

функція _ j (предикатне_ ім'я, значення _ j)

називають – ***slot – assertion notation***.

За домовленістю всі стрілки n-арного предиката спрямовані до кола (тобто до імені предиката), а остання n-на стрілка – від кола до останнього n-ного аргументу.

Концепти в семантичних мережах можуть визначати властивості аналітичні (властивості типу) і синтетичні (властивості множин).

Деякі визначення

Ієрархія типів – ***надтипи*** і ***підтипи*** (надтип> підтип) - надтип – узагальнюючий образ підтипу. Універсальний надтип U = надтип всіх типів і абсурдний тип A = підтип всіх типів. Для двох різних міток (типів) існує найбільший спільний підтип і найменший спільний надтип. Вони завжди знаходяться між U і A.

Денотат типу B – множина всіх сутностей, які є конкретизацією деякого концепту типу B.

Функція «тип» – відображає множину концептів на множині «міток (ярликів) типу».

Прототип – конкретизація типу (властивості справжні в типовому і необов'язкові в реалізації).

Вузли-індивіди або вузли-посилання (метод Сови):

зазвичай використовують бінарні предикати, тоді вузол-коло, який вказує ім'я предиката, є сполучним вузлом, а функція, яку він представляє називається ***концептуальним відношенням***, інші вузли, що відповідають прямокутникам – аргументам називають вузлами-концептами.

Приклади:

Перетворення унарного предиката в бінарний:

Ім'я _ сукупності (ім'я _ індивідуума)

перетвориться в

Конкр (ім'я _ індивідуума, ім'я _ сукупності).

Перетворення 3-арного предиката в бінарний:

фраза: «Вася посилає цукерку Маші», яка записується в формі предиката

Посилка (Вася _ 1, Маша _ 10, цукерка _10) і перетворюється в три бінарних предиката

1. Відправник (Посилка, Вася _1);
2. Отримувач (Посилка, Маша _10);
3. Об'єкт (Посилка, цукерка _10).

До речі, уявлення [книга: x] означає просто об'єкт типу книга (тому можна замінити на [книга]), а [книга: Книга_2] – цілком конкретну книгу.

Правила кон'юнкції – якщо вузол-концепт c1 в G1 і вузол-концепт c2 в G2 ідентичні, то вузол-концепт c2 видаляють, а всі його зв'язки замикають на вузол-концепт c1.

Правило спрощення – якщо є два ідентичних зв'язуючих вузли, з'єднаних одними і тими ж вузлами-концептами, то один зв'язуючий вузол можна видалити.

Дія кванторів. Концептуальний граф ділять на ієрархічну множину зон дії окремих кванторів.

Канононічні графи – семантично коректні – такі, які не містять в собі нереальних або неможливих ситуацій.

Схеми – подальше обмеження після канонічності – специфічні знання про область міркувань (експертизи), представлено все правдоподібне. Концепт (даного типу) можна визначити «множиною його можливих застосувань», можна ввести поняття кластера або набору схем, кожна з яких дає спосіб застосування даного концепту. Набір всіх можливостей застосування типу називається схематичним кластером, який для типу B є множина $\{\lambda x_1 F_1, \lambda x_2 F_2 \ldots, \lambda x_n F_n\}$ λ – виразів, причому кожен $x_к$ належить до типу B і кожен вираз $\lambda x_к F_к$ – схема для типу B. Зауважимо, що λxF – ***λ вираз, що представляє схему***, – F – логічна формула (або граф, що її представляє), формальний параметр x представляє концепт того типу, який потребує визначення, а тіло F вкаже на один зі способів застосування цього типу. Схеми відповідають ***сузір'ям***, ***фреймам (кадрам), сценаріям (розпорядженням).***

Успадковані властивості – властивості, які неявно, але з певністю випливають із вже поданої інформації.

Мережі Петрі – уявлення процесів в формі графів – орієнтовані графи, вершини яких є «позиції» (представлені колами) або «переходи» (поперечні відрізки). Позиції містять «жетони», що дозволяють подальше переміщення по мережі (жетон при цьому передається по ланцюжку, тобто в попередній позиції жетона вже не виявляється). Мережа включає в себе систему управління у вигляді жетонів і закладених в переходах і позиціях правил.

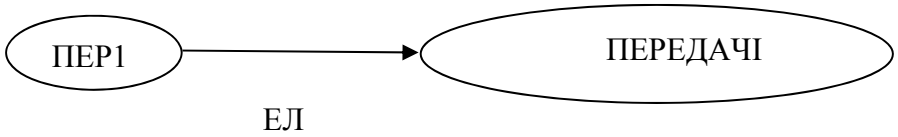
Введемо предикати ПМ – «перша множина є підмножиною другої», ЕЛ – «щось є елементом даної множини», Рід занять – РЗ, адреса – АДР. Можна використовувати комірку «сам», де об'єкт сам по собі елемент деякої множини, наприклад, сам: (елемент множина КУПІВЛІ). Предикат РІВ – це «рівність»

Сколемівські константи і функції, псевдозначення. Зазвичай намагаються спростити всі форми запису. Речення «Хтось дав Маші книгу» містить предикат, наприклад, РІВ [дає (ПЕР3), х]. Можна використовувати сколемівську змінну РІВ [дає (ПЕР3), S]. Цей хтось може бути людиною, тоді потрібно це зазначити:

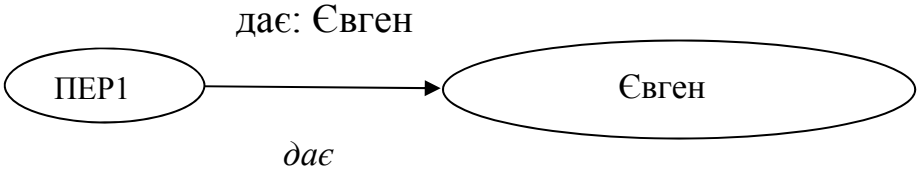
$$(\exists x)\ \{\text{РІВ [дає (ПЕР3), х]} \wedge \text{ЕЛ (х, ЛЮДИ)}\},$$

Семантичні (смислові) зв'язки. Якщо акцентувати увагу на організації зв'язків, то слід додати ***функції доповнення***, – до, перетин, об'єднання ...

Вираз ЕЛ (ПЕР1, ПЕРЕДАЧІ) можна представити як



Для виразу РІВ [дає (ПЕР1), Євген] або в спрощеному вигляді для блоку ПЕР1



Для сколемівського представлення функцій sk (x)
g (x) сам: (елемент множини ПЕРЕДАЧІ)

дає: Євген
одержувач: х
об'єкт: sk (x)

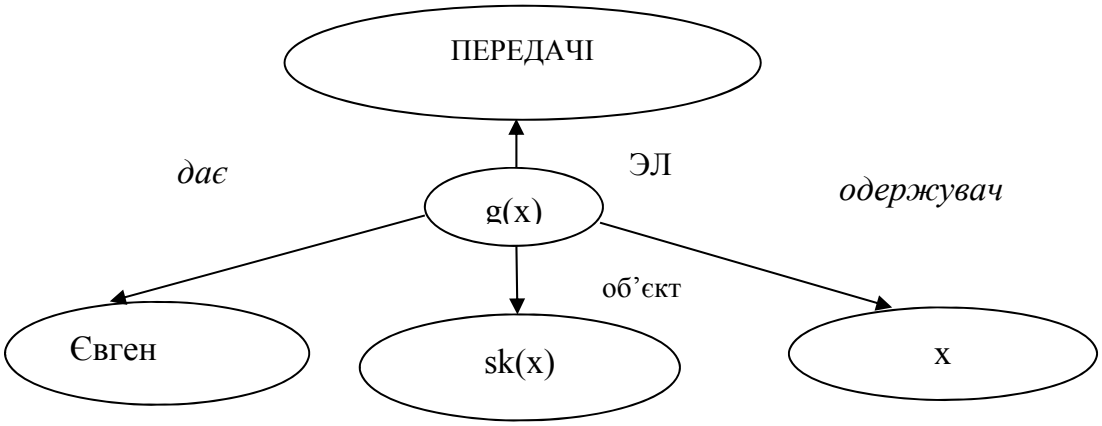


Рис. 9.2. Мережа з вершинами, які представляють сколемовські функції

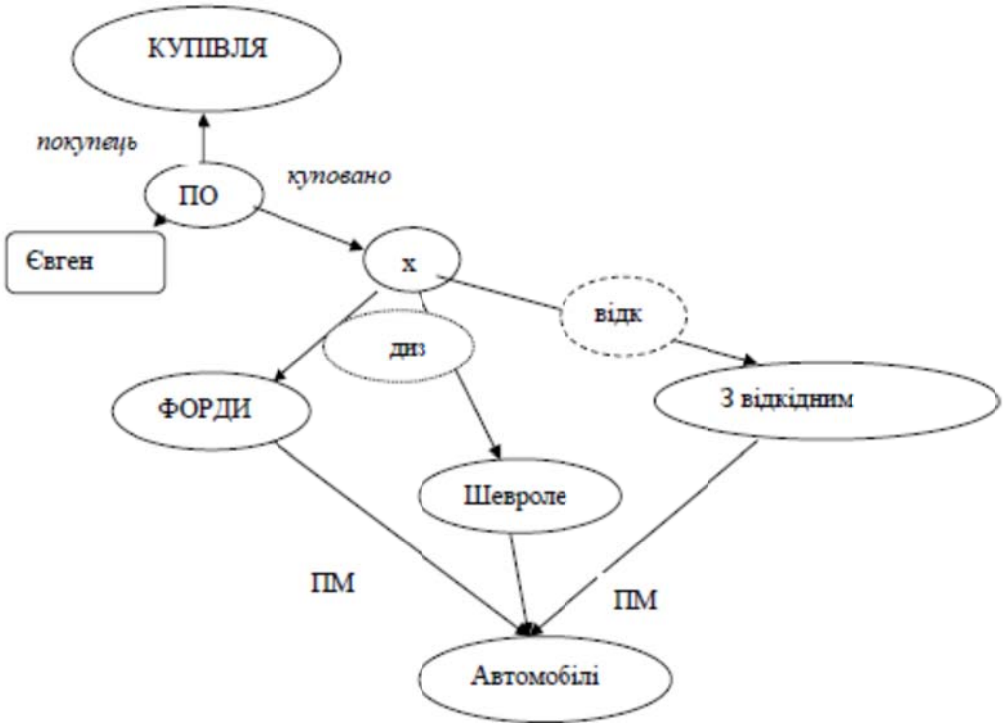


Рис. 9.3. Семантична мережа з логічними зв'язками

Ця мережа може бути представлена у вигляді складного речення:
РАВ [придбано (ПОК1), Х] ∧ЕЛ (Х, З_ВІДКІДНИМ _ВЕРХОМ) ∧
∧ [ЕЛ (Х, Форди) ∨((Х, Шевроле)]
∧ПМ (Форди, Автомобілі) ∧ ПМ (Шевроле, Автомобілі)
∧ПМ (З_ВІДКІДНИМ, Автомобілі).

9.2. Уніфікація vs відповідність

Уніфікація – досить сильна вимога, відповідність – більш слабка властивість і, головне, несиметрична.

Таблиця 9.1

<table>
<tr><td colspan="2">БР1
сам: (елемент-множини Шлюби)
чоловік: Євген
жінка: Маша
або
ЕЛ (БР1, Шлюби) ∧РІВ [чоловік (БР1), Євген] ∧РІВ [жінка (БР1), Маша]</td></tr>
<tr><td>БР1
сам: (елемент-множини Шлюби)
чоловік: Євген</td><td>БР1
сам: (елемент-множини Шлюби)
чоловік: Євген
жінка: Маша
тривалість: 10</td></tr>
</table>

Верхній блок знаходиться відповідно до нижнього правого блоку, але не до нижнього лівого.

Розглянемо ці ситуації в поданні семантичних мереж

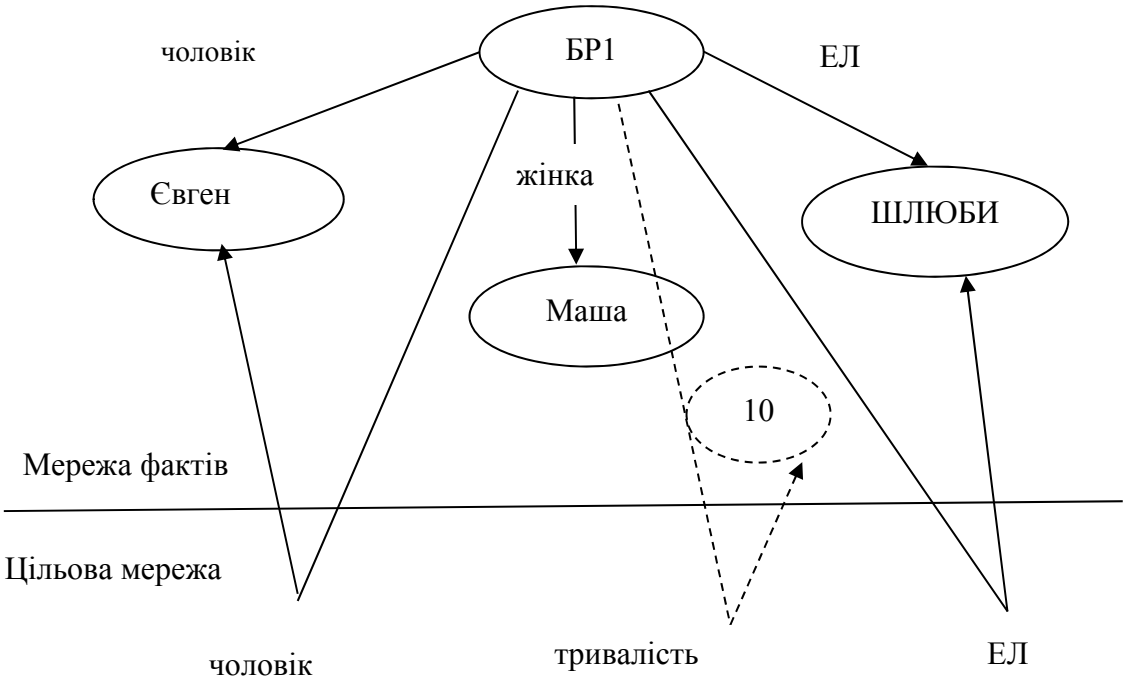


Рис. 9.4. Приклад відповідності цільової мережі і мережі фактів, при виключенні пунктирних елементів та невідповідності при включенні пунктирних елементів.

Якщо виключити пунктирну вставку, то маємо дві мережі: цільова знаходиться у відповідно до мережі фактів, але при додаванні пунктирних деталей, ця відповідність порушується.

При альтернативних відображеннях висловлювань семантичні конструкції можуть взагалі не відповідати одна одній. Висловлювання «Євген одружений на Маші» «У Євгена дружина Маша» можна представити таким чином:

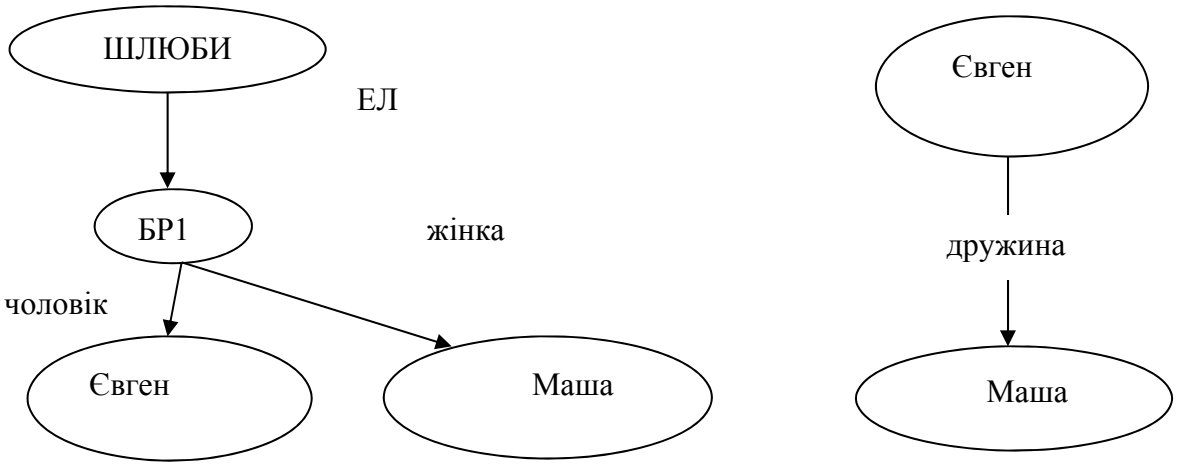


Рис. 9.5. Невідповідні альтернативні відображення

Приклад запиту.
Питання: «Боря дав Маші ручку?» Це цільовий блок:
х
сам: (елемент-множини ПЕРЕДАЧІ)
дає: Боря
отримує: Маша
об'єкт: ручка
Блоки фактів наступні:
ПЕР1
сам: (елемент-множини ПЕРЕДАЧІ)
дає: Євген
отримує: Маша
об'єкт: книга
Пер2
сам: (елемент-множини ПЕРЕДАЧІ)
дає: Боря
отримує: який одержує (ПЕР1)
об'єкт: ручка
Цільовий блок відповідає блоку Пер2, але при правильній уніфікації.

9.3. Дедуктивні операції над структурованими об'єктами

Подання імплікації. Розглянемо правило «Всі студенти ФКН навчаються на випускному курсі»

(а) ЕЛ (х, ФКН Студенти) ⇒РІВ [курс (х), Випускний]

Блоки імплікації будемо представляти блоком описів, де х – видова змінна, що належить до квантору спільності. Ім'я множини, що належить до квантору спільності будемо записувати після змінної х, відокремлюючи її вертикальною рискою:

(б)
х | ФКН_Студенти
спеціалізація: КН
курс: Випускний

Представлення у мережі

(в)

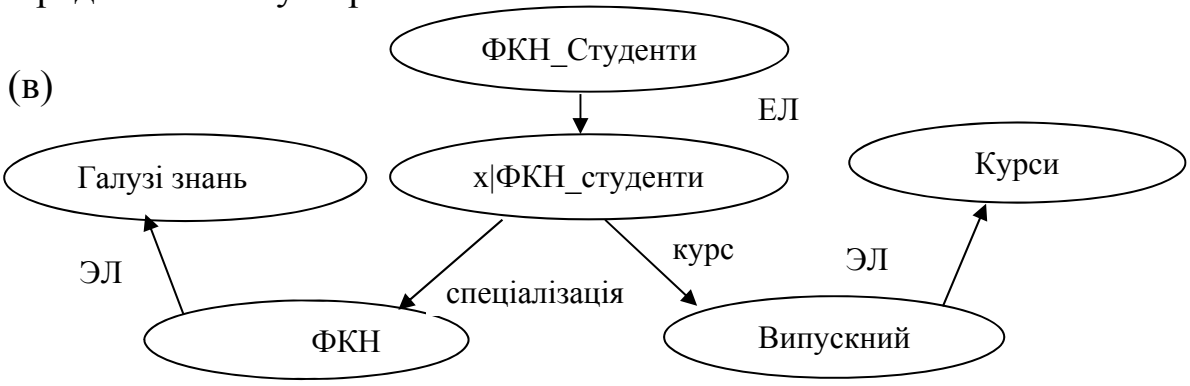


Рис. 9.6. Використання блоку опису в мережі

тепер

Факт: ЕЛ (Євген, ФКН_Студенти)

Правило: ЕЛ (х, ФКН_Студенти) ⇒РІВ [курс (х), Випускний]

Мета: РІВ [курс (Євген), Випускний]

блок фактів

Євген

сам: (елемент множини ФКН_Студенти)

мета:

Євген

Курс: Випускний

1. Блок описів х | ФКН_Студенти відповідає блоку факту Євген, якщо уніфікувати змінну х = Євген, тоді

2. Додаючи до блоку описів функції *спеціалізація:* ФКН і *курс*: Випускний, переконуємося: блок для факту відповідає цільовому блоку.

Абстракція елемента множини. За необхідності, якщо введені елементи множини не здатні нести сенс його абстрактного елементу, то такий об'єкт може бути введений в семантичну мережу.

Приклад: в семантичної мережі в множині «автомобілі» виділені конкретний автомобіль і блок описів, що представляє кожен (будь-який) автомобіль, проте нам знадобиться автомобіль, який був винайдений в 1892 році. Тоді доведеться ввести ще об'єкт = «абстрактний автомобіль», для побудови нового семантичного зв'язку про винахід 1892 року. Бо всі попередні об'єкти не здатні такий зв'язок організувати.

Приклад: Поняття – об'єкти «Юрист», «Програміст» – це теж приклади абстрактних об'єктів.

Спадкування властивостей. При дедукції доводиться переходити до об'єктів, які є множиною, що включає в себе цей об'єкт. При цьому об'єкт успадковує властивості цих узагальнюючих множин, що дозволяє продовжувати процедури.

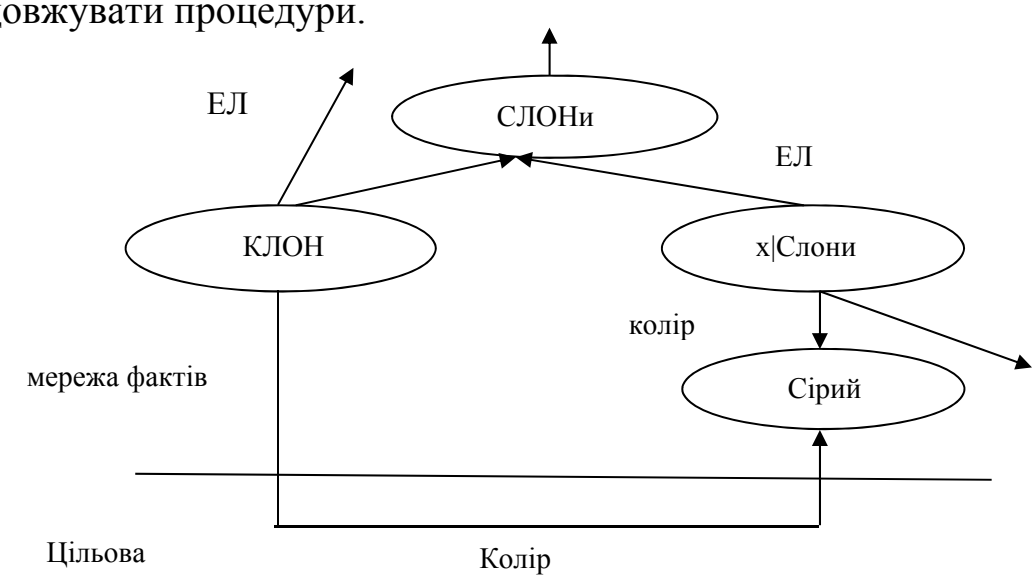


Рис. 9.7. Приклад доказу завдяки успадкуванню властивостей об'єктом КЛОН

Програма автоматично ***успадковує*** всі необхідні властивості множин і супермножин, в які об'єкт входить.

З вершини КЛОН в вершину Сірий немає прямої дуги «колір», то слід підніматися до узагальнення = множини типу СЛОНИ і через імплікації в формі блоку описів приходимо до дузі «колір» і далі до цільової вершини.

Змінні цільові вершини – це зазвичай константні вершини, причому константи сколемівські, тобто, такі, які слід визначити.

Константні вершини – це реальні певні об'єкти.

Стратегія розв'язання задач наступна:

1. *Пошук константних вершин в мережі фактів*

(а) здійснюється пошук константних вершин в мережі фактів, які знаходяться в ланцюзі, що з'єднує змінну цільову вершину або

(б) піднімаючись в множину, звідки виділено змінний цільовий об'єкт (змінна цільова вершина, що пов'язана з множиною, яка дозволяє операцію успадкування), і потім опускаючись по інших напрямках, шукають константні вершини.

2. *Пошук відповідності мережі цільової та мережі фактів.* У першому випадку (а) потрібно встановити відповідність між змінною цільової вершиною і вершинами мережі фактів (можна дозволити програмі пошуку відповідності використовувати опис при пошуку відповідності).

Приклад.

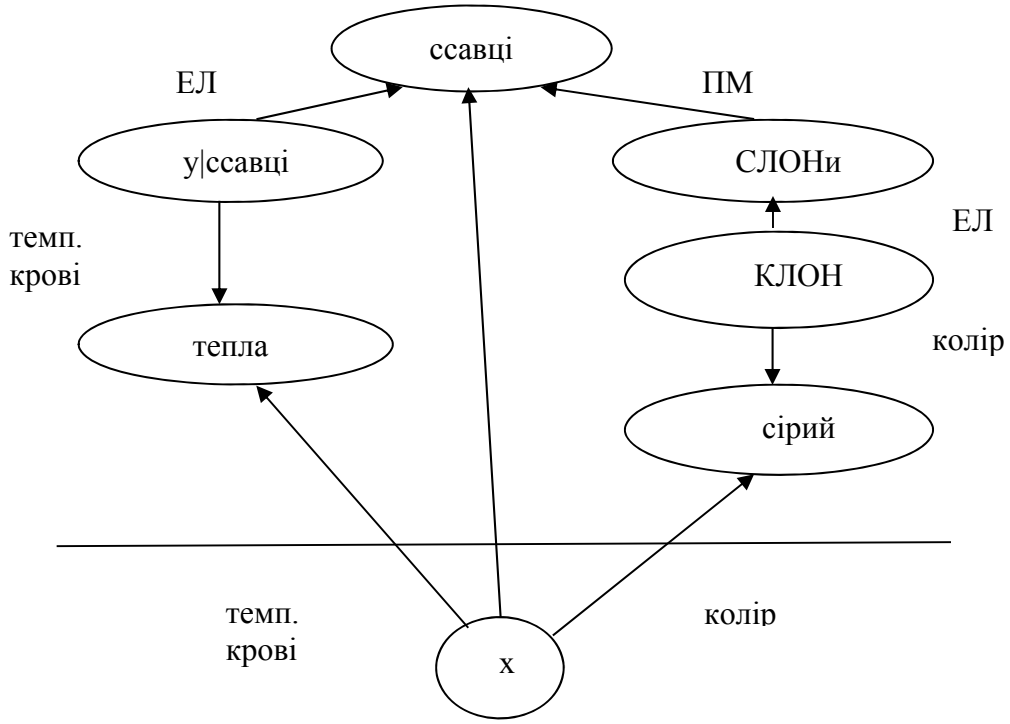


Рис. 9.8. Пошук відповідності мережі цільової зі змінною цільовою вершиною і мережі фактів з явною константною вершиною

Потрібно встановити об'єкт з множини «ссавці», такий, що він сірий і що кров у нього тепла. Розглянемо мережу фактів (вище горизонтальної лінії) і цільову мережу (нижче горизонтальної лінії). Константна вершина тільки одна – КЛОН. Потрібно знайти

– дугу «температура крові» в вершину «тепла» з вершини КЛОН і

– дугу «колір» в вершину «сірий» з вершини КЛОН;

– дугу ЕЛ з КЛОН в вершину «ссавці».

Якщо ми це зробимо, то цільова мережа відповідає мережі фактів при підстановці x = КЛОН.

Приєднані процедури. Це програми, які викликаються занеобхідності.

Уявімо цільовий блок

z

множник1: 3
множник2: 6
добуток: x

для отримання значення добутку введемо блок опису множення

x | МНОЖ
множник 1: (елемент множини «натуральні числа»)
множник 2: (елемент множини «натуральні числа»)
добуток: ПЕРЕМНОЖИТИ [множник1 (x), множник2 (x)]

Тут перемножити [множник 1 (x), множник 2 (x)] – це приєднана процедура, (програма множення). При підстановці (уніфікації): {ПЕРЕМНОЖИТИ (3, 6) / x} отримаємо відповідь.

Блочні правила. Блочне правило містить два списки блоків – антецедент (АНТЕ) і консеквент (КОНСЕ). Застосовуються ці правила при імплікації.

Якщо всі блоки в АНТЕ (цільові) зіставлені з блоками-фактами, то після застосування блочного правила, блоки КОНСЕ можна також приєднувати до блоків-фактів.

* Якщо блоки АНТЕ вважати цільовими, то змінні належать до квантору існування.

** Якщо блоки, що додаються, вже є серед блоків-фактів, то при приєднанні додають тільки властивості, які фігурують в КОНСЕ.

Приклад:

П1: ГОЛОВА (x, y) $\Rightarrow$ ПРАЦЮЄ _В (x, y)

або

{ЕЛ (x, ВІДДІЛИ) $\wedge$ РІВ [ГОЛОВА (x), y]} $\Rightarrow$ РІВ [ПРАЦЮЄ _В (y), x]

або

П1:

АНТЕ: x
сам: (елемент-множини ВІДДІЛИ)
голова: y
КОНСЕ: y
працює_в x

П2: [ПРАЦЮЄ_В (x, y) ∧ГОЛОВА (x, z)] ⇒БОС (y, z)
або
{РІВ [ПРАЦЮЄ_В (y), x] ∧РІВ (ГОЛОВА (x, z)]} [РІВ (БОС(y, z)]
або
П2
АНТЕ: y
працює_в x
x
голова: z
КОНСЕ: y
бос: z

Тотожність двох співвідношень:
РІВ [y, дружина (x)] ≡(∃z){ЕЛ (z, ШЛЮБИ) ∧РІВ [x, чоловік (z)] ∧РІВ [y, жінка (z)]}
* Взагалі кажучи, замість W1 ≡W2 потрібно було б записати [W1⇒W2]∧ [W2⇒W1].

Сколемівське перетворення:

РІВ [y, дружина (x)] ⇒ {ЕЛ (шл (x, y), ШЛЮБИ) ∧
∧РІВ [x, чоловік (шл (x, y))] ∧
∧РІВ [y, жінка (шл (x, y))]}

Блочне правило:
П_ШЛ
АНТЕ: x
дружина: y
КОНСЕ: шл (x, y)
сам: (елемент-множини ШЛЮБИ)
чоловік: x
жінка: y

Процедура застосування – *встановити відповідність між блоком АНТЕ і деяким блоком-фактом, а потім створити константний блок, відповідний до блоку КОНСЕ.*

Мережеві правила

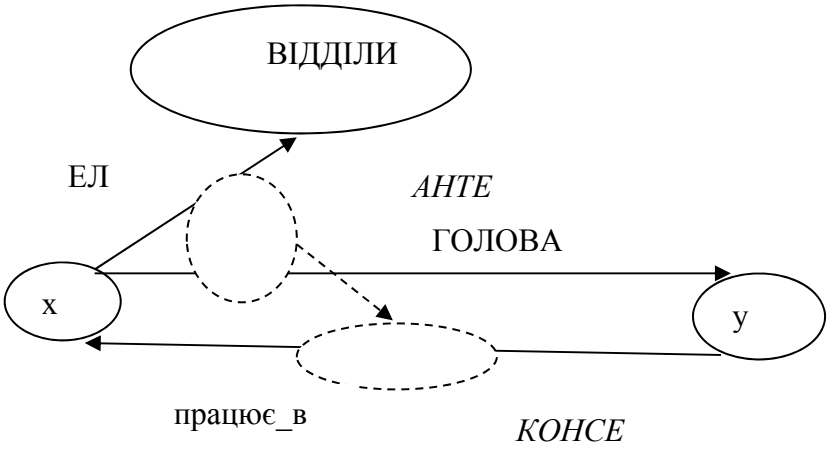


Рис. 9.9. Імплікація: {ЕЛ (x, ВІДДІЛИ) ∧ РІВ [ГОЛОВА(x), y]} ⇒ РІВ [ПРАЦЮЄ_В(y), x]

Пряме правило: структура АНТЕ, як цільова. Показуємо, що вона відповідає мережі фактів. Після цього до мережі фактів приєднують структуру КОНСЕ, що і є застосуванням імплікації.

Неточні описи. Якщо КЛОН є слоном-альбіносом, то виникає колізія. У детальному описі може бути наведена інформація про те, що він білий. А в мережі є вихід в ієрархічно вищу вершину СЛОНИ, а потім вже до вершини «сірі». Як буде діяти машина? З одного боку, на детальному рівні ясно, що цей слон білий, а через ієрархічно вищу вершину отримуємо – «сірий». Тому повинна бути створена схема розв'язання, коли ***лише за відсутності детальної (ієрархічно нижчої) інформації потрібно підніматися вгору до більш високої і загальної ієрархічної вершини, успадковувати її характеристики і рухатися по мережі в пошуках відповіді***. Можна ввести т. зв. неточні описи, тобто вершини – описи, які мають нижчий рівень приоритетів[37] щодо черговості їх виконання. А можна слідувати наведеним вище правилам.

Додавання рекомендацій. Розглянемо цю ситуацію на наведеному вище прикладі двох правил П1 і П2 (лівий стовпець).

Таблиця 9.2

ПРАВИЛА	ФАКТи	Процедури вирішення та результати
П1 АНТЕ: x сам: (ел. множ. ВІДДІЛИ) голова: y	ВПК сам: (ел. множ. ВІДДІЛИ) голова: Євген	Згадали про Відділи Рекомендація 2 Є співвідношення
П1 КОНСЕ: y працює_в x		Євген працює_в: ВПК

[37] Часто використовують замість терміна «неточна» – default, що може мати сенс «за відсутності точної» або «за замовчуванням».

Продовження таблиці 9.2

П2 АНТЕ: y працює_в x x голова: z	ВАСЯ сам: (ел_мн. СЛУЖБОВЦІ) процює_в: ВПК	
П2 КОНСЕ: y бос: z	**Питання** ВАСЯ бос: u	Згадали про «бос» і про П2. Питання визначить y=Вася. 1. виникає відповідність у верхній частині П2 АНТЕ, тоді x = ВПК, нижня частина відповідає 2 факту і z = Євген.

Рекомендація 1:

u | СЛУЖБОВЦІ
бос: (елемент_множини СЛУЖБОВЦІ)
<при заповненні П2>
працює_в (елемент множини ВІДДІЛИ)

Рекомендація 2:

r | ВІДДІЛИ
голова (елемент_множини СЛУЖБОВЦІ)
<при заповненій П1>

У Рекомендації 1 позначення при заповненні П2 вказує, що П2 повинне застосовуватися в зворотному напрямку щоразу, коли в блоці є якесь значення комірки «бос», за умови, що прямої відповідності з блоком фактів встановити не можна.

У рекомендації 2 позначення при заповненні П1, говорить про те, що П1 має застосовуватися, коли у деякого блоку фактів комірка «сам» містить елемент множини ВІДДІЛИ і комірці «глава» присвоєно якесь значення.

Приклад:

ВАСЯ
сам: (елемент_множини СЛУЖБОВЦІ)
працює_в: ВПК
ВПК
сам: (елемент множини ВІДДІЛИ)
голова: Євген

1 варіант: Нехай пред'явили другий блок, то опис (рекомендація2) r | ВІДДІЛИ вкаже на те, що П1 застосовується в прямому напрямку (елемент множини ВІДДІЛИ конкретний – це ВПК і відбувається уточнення значення змінної x, яка належала до квантору існування), що породить

Євген
працює_в: ВПК

2 варіант: питання «Хто є босом Васі?» Тобто це блок виду

ВАСЯ

бос: u

Прямої відповідності з блоком-фактом не встановити. Але згідно з Рекомендацією 1, оскільки згадали комірку «бос», рекомендується застосувати П2 в зворотному напрямку. Звідки

ВАСЯ

працює_в: x

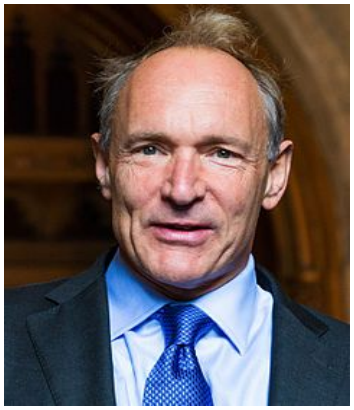
x

глава: u

зіставити з блоком-фактом ВАСЯ можна при x = ВПК, тоді другий блок можна зіставити із блоком-фактом ВПК при підстановці u = Євген, що і було потрібно з'ясувати.

9.4. Семантічна павутина

Semantic Web – це Інтернет, де інформація попередньо обробляється машинами. Традиційний Інтернет заснований на HTML-сторінках, інформація витягується за допомогою браузера самим користувачем. Семантична ж павутина використовує можливості семантичної мережі, обробляючи інформацію програмою-клієнтом і надаючи користувачу результат логічної обробки. Термін «семантична павутина» був вперше введений Т. Дж. Бернер сом–Лі (травень 2001 року, журнал «Scientific American») і був визначений ним як «наступний крок у розвитку Всесвітньої павутини»[38].



Тімоті Джон Бернерс–Лі
Sir Timothy John «Tim» Berners–Lee

Тімоті Джон Бернерс–Лі *(Sir Timothy John «Tim» Berners–Lee)* – винахідник, спільно з колегами, URL, URI, HTTP, HTML, творець спільно з Робертом Кайо Всесвітньої павутини (Веб). Чинний глава Консорціуму Всесвітньої павутини при Лабораторії інформатики (Laboratory for Computer Science, LCS, інші назви консорціуму: World Wide Web Consortium, W3C, W3) MIT. Автор концепції семантичної павутини і обчисленних інших розробок.

[38] Проблеми вбачають в неможливості використання реклами в цьому варіанті пошукових систем, які самі відвідують сайти і відбирають інформацію для передачі користувачеві. Рекламодавці не впевнені в тому, що безпосередньо користувач побачить рекламу, тому фінансувати цей проект нікому.

Зараз в Інтернеті автоматична обробка даних заснована на лексичному і частотному аналізі для зручності користувача.

У семантичній павутині використовується стандарт RDF[39], що описує семантичні мережі, де вузли і дуги мають ідентифікатори ресурсів (адреси) URI. Причому в семантичній павутині всі сутності – предмети, явища і абстракції, мають свої URI. Виклики цих адрес можна робити як зазвичай, використовуючи протокол HTTP. Для логічних висновків машинами пропонують використовувати розробки RDF SCHEMA[40], OWL[41]. Техніка опису спирається на дескриптивну логіку[42].

Отже, основними мовами для Семантичного Інтернету залишаються Розширена Мова Розмітки XML (eXtensible Markup Language) і Засоби Описів Ресурсів RDF (Resource Description Framework) а також мова мережевих онтологій OWL (Web Ontology Language).

У павутині повинні використовуватися подібні традиційним веб-сервіси, що використовують програмну логіку. Поки ефективність використання спеціалізованих веб-сервісів невелика через труднощі створення сервісно-орієнтованої архітектури і вдаються поки тільки вузькоспеціалізовані запити на вибір сервісів.

Семантичний Web об'єднує різні види інформації в єдину структуру, де кожному смисловому елементу даних буде відповідати спеціальний синтаксичний блок (тег). Сподіваються, що теги стануть складати єдину ієрархічну структуру. В рамках проекту «Семантичний ВЕБ» повинні розроблятися мови для вираження доступної для машинної обробки інформації. Інша сторона Семантичного Web пов'язана з напрямками, близькими до області штучного інтелекту, і названа онтологічним підходом. Цей підхід включає в себе засоби анотації документів, якими могли б скористатися комп'ютерні програми – Web-сервіси та агенти при обробці складних запитів користувачів. У цьому підході використовують теоретичні уявлення (описи) моделей предметних областей. Такі моделі предметних областей в термінології Семантичного Web називаються онтологіями.[43]

---

[39] Спосіб опису даних в форматі суб'єкт–відношення–об'єкт, де використовуються тільки ідентифікатори ресурсів. Консорціумом W3 визначена схема XML-документів, що містять RDF-описи.

[40] Мова описує набір атрибутів (відносин), для визначення нових типів RDF-даних. Стандартний синтаксичний аналізатор мови XML в змозі перевірити довільний XML-документ на відповідність його структури так званій схемі документа, описаній в XML Schema.

[41] В основі мови OWL (Web Ontology Language) – представлення «об'єкт-властивість». Мова розширює можливості опису нових типів (зокрема, додає перерахування). OWL придатна для опису веб-сторінок і будь-яких об'єктів дійсності. Кожному елементу опису в цій мові і властивостям, що зв'язують об'єкти, присвоєно URI (див. Semantic Web [Електрониий ресурс]. – Режим доступу**:** http://sherdim.ru/pts/semantic_web/REC-owl-features-20040210_ru.html).

[42] Або інакше - термінологічні системи, логіки концептів.

[43] Ланде Д. Семантичний веб: від ідеї – до технології. [Електрониий ресурс]/ Д. Ланде. – Режим доступу http://poiskbook.kiev.ua/sw.html

Логічні рівні

Найнижчий рівень – це Universal Resource Identifier (URI), уніфікований ідентифікатор, що визначає спосіб запису адреси довільного ресурсу. Семантичний Web, називаючи всяке поняття просто за допомогою URI-ідентифікатора[44], дає можливість кожному користуватися потрібними поняттями. Наступний рівень – мова XML як базова форма розмітки і засоби, призначені для визначення і опису класів XML−документів (DTD, XML-схеми). Додатковий рівень орієнтований на роботу з цифровим підписом, для визначення ступеня достовірності даних. Рівень RDF дозволяє виконувати пошук необхідних понять в Семантичному Web. На базі XML розгортаються засоби опису ресурсів RDF і RDF-схеми, щоб узгодити XML−дані в мережі і створювати каталоги і словники понять. Мова мережевих онтологій OWL дозволяє організувати більш повну автоматичну обробку мережевого контенту, ніж та, яку підтримують XML і RDF, надаючи додаткову семантичну підтримку поряд з формальною семантикою. Онтологія визначає семантику конкретної предметної області і сприяє встановленню зв'язків між значеннями її елементів. Онтології використовують для підвищення точності пошуку в Internet – пошукова система буде видавати тільки такі сайти, де згадується з точністю відшуковуване поняття, а не позиції, де зустрілося задане ключове слово.

Локальні завдання.

1. Створення бази онтологій.

2. Формування агентів (автономних програм), що мають розширений доступ до Web-контенту та до автоматизованих сервісів (включаючи інших агентів).

3. Створення глобального і універсального механізму пошуку і виявлення автоматизованих Web-сервісів.

Прогрес:

1. Проект Microsoft взаємодіючих мережевих ресурсів .NET дозволяє проводити автоматизований обмін мережевими ресурсами між окремими програмами, базами даних, користувачами, грунтуючись на XML.

2. Європейська «Мережа знань», Knowledge Web орієнтована на потреби інформаційних технологій в промисловості, науці та освіті.

3. Система mSpace в School of Electronics & Computer Science (ECS) Університету Саутгемптона збирає дані з різних джерел і організовує інформацію за категоріями.

[44] URI-ідентифікаторів є також URL-адреси, але URI-ідентифікатор задаючи або посилаючись на деякий ресурс, не обов'язково вказує на його місцезнаходження в Internet.

ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Навіщо потрібна операція приведення до кон'юнктивної нормальної форми будь-якого виразу в логіці предикатів?

2. Поясніть Закони Моргана.

3. Як перейти до нечіткості в описі чітких подій?

4. У чому сенс розділення змінних для теорії предикатів?

5. Якщо дерево спростування в теорії предикатів дає в кореневій вершині порожню множину, де шукати відповідь?

6. Поясніть принцип Байєсової системи логічного висновку.

7. Наведіть структуру П-правила в STRIPS для побудови схеми управління роботом.

8. Доведіть, що в разі нечіткого логічного висновку $A \cap \bar{A} \neq \varnothing$, де $\bar{A}$ це – додаток А.

9. Вкажіть підстави вважати реляційні бази даних системами штучного інтелекту.

10. Наведіть структуру правила виведення Байєсової системи.

11. Доведіть, що в разі нечіткого логічного висновку $A \cup \bar{A} \neq E$, $\bar{A}$ це – додаток А, а Е – універсальна множина.

12. Поясніть сенс кванторів спільності та існування, визначте області їх дії. Який з них має відношення до сколемівських функцій?

13. Сформулюйте загальні принципи нечіткого логічного висновку.

14. Як вводиться сколемівська функція?

15. Що таке «тіло» в виразах мови Пролог?

16. Поясніть як слід побудувати алгоритм дії робота в структурі STRIPS.

17. У чому ви бачите історичне об'єднання штучних нейронних мереж і експертних систем прийняття рішень на основі логіки?

18. Поясніть, в якому вигляді може бути записана диз'юнкція і кон'юнкція в нечіткій логіці.

19. Застосуйте метод резолюції до наступної бази знань $\sim P \vee \sim Q \vee S$; P; $\sim S$; Q.

20. Що приховано від користувача в мові Пролог?

21. Як приводять результати нечіткого виведення до чіткості?

22. Перетворіть потрійний предикат в три бінарних: Посилка (Вася _ 1, Маша _ 2, лист_1000).

23. Наведіть практичну форму структури П-правила в STRIPS для побудови схеми управління роботом.

24. Поясніть зв'язок між описом фактів у формі графів типу І / АБО і описом в формі предикатів.

25. Поясніть в чому полягає алгоритм Sugeno?

26. У які види речень мови Пролог входить «голова»?

27. Що таке уніфікація і як вона застосовується в логіці предикатів і при використанні системи STRIPS?

28. Як переходять до чіткості в алгоритмі Tsukamoto?

29. Як виключити символи імплікації в теорії предикатів?

30. Що таке локальні і загальні термінальні умови?

31. Алгоритм нечіткого виведення Mamdani. Принципи побудови.

32. Імплікація, заперечення, диз'юнкція і кон'юнкція. Що з них використовується в мові Пролог?

33. Поясніть різницю між уніфікацією в теорії предикатів і відповідністю в Пролог.

34. У чому різниця між фреймами і реляційними базами даних?

35. Як розв'язує задачі система RSTRIPS? Поясніть, як рухається маркер, як перевіряється список і що знаходиться під захистом.

36. Чи можна використовувати диз'юнктивну нормальну форму?

37. Чому кажуть, що в ПРОЛОЗі не все програмується?

38. Що визначає теорема Байєса?

39. Поясніть відмінності поведінки системи STRIPS і системи RSTRIPS при взаємодії цілей.

40. Для чого вводиться характеристична функція в нечіткій логіці?

41. Як використовується імплікація в теорії предикатів і в мові ПРОЛОГ?

42. Наведіть приклади комутативних і некомутативних систем подання знань.

43. Нейронна мережа може бути описана як система нечіткої логіки? Покажіть на прикладі.

44. Що таке спростування на основі резолюції? Поясніть, як це робиться.

45. Що таке метод резолюції в логіці предикатів?

46. Про що теорія Демпстера (Dempster)-Шафера (Shafer)?

47. Поясніть як можна пов'язати два фрейми (блоки). Наведіть приклади.

48. В мові ПРОЛОГ мета G (A) і правило $P(y) \Rightarrow G(y)$. Тут голова правила G (y), а тіло правила (умовна частина) P (y). При уніфикации y = A об'єднання цілі та правила дає P(A) – нову ціль. Якій процедурі при резолюції це еквівалентно в теорії предикатів?

ЛІТЕРАТУРА

1. Куклин В. М. Разбуженный мир : Эссе [Электронный ресурс] / В. М. Куклин. – Харьков : ХНУ имени В. Н. Каразина, 2013–2014. – 216 с. – Режим доступа : http://ekhnuir.univer.kharkov.ua/handle/123456789/12484
2. Polanyi M. Personal knowledge: Towards a Post-Critical Philosophy / M. Polanyi. – London, 1958. (в рус. пер. Поланьи М. Личностное знание: На пути к посткритической философии / М. Поланьи. – Москва : Прогресс, 1985. – 345 с.).
3. Hawkins J. On Intelligence / J. Hawkins with S. Blakeslee. – New York : Times Books, 2004.
4. Клайн М. М. Математика. Утрата определенности / М. М. Клайн ; перев. Ю. Данилова. – Москва : РИМИС. 2007, – 640 с.
5. David Deutsch. The Fabric of Reality / David Deutsch. – The Penguin Press, 1997. – 390 p.
6. Payr P. Speaking Minds: Interviews with Twenty Eminent Cognitive Scientists / P. Payr, S. Payr. – Princeton, N. J.: Princeton University Press, 1995.
7. Нильсон Н. Принципы искусственного интеллекта / Н. Нильсон ; пер. с англ. – Москва : Радио и связь, 1985. – 376 с.
8. Введение в методы программных решений. учебное пособие / Е. В. Белкин, А. В. Гахов, А. М. Горбань, В. М. Куклин и др. – Харьков : ХНУ имени В. Н. Каразина, 2010. – 308 с.
9. Лорен Ж.–Л. Системы искусственного интеллекта / Ж.–Л. Лорен. – Москва : Мир, 1991.
10. Фреже Г. Избранные работы / Готлоб Фреге ; сост. В. В. Анашвили, А. Л. Никифоров ; пер. с нем. В. В. Анашвили. – Москва : Доминтеллектуал, 1997. – 159 с.
11. Herbrand J. Recherches sur la théorie de la demonstration / J. Herbrand // Travaux de la société des Sciences et des Lettres de Varsovie, Class III, Sciences Mathématiques et Physiques, 1930. – Vol. 33.
12. Robinson J. A. A Machine–Oriented Logic Based on the Resolution Principle / J. A. Robinson // Communications of the ACM, 1965. – Vol. 12. – № 1. – P. 23–41.
13. Ковальский Р. Логика в решении проблем / Р. Ковальский. – Москва : Наука, 1990. – 280 с.
14. Представление и использование знаний / под ред. Х. Уэно, М. Исидзука. – Москва : Мир, 1989. – 220 с.

15. Логический подход к искусственному интеллекту: от классической логики к логическому программированию ; пер. с франц. / А. Тейз, П. Грибомон, Ж. Луи и др. – Москва : Мир., 1990. – 432 с.
16. Kőnig D. Theorie der endlichen und unendlichen Graphen / D. Kőnig. – Leipzig: Akademische Verlagsgesellschaf, 1936. – 258 s.
17. Colmerauer A. Un système de communication enfrançais / Colmerauer Alain, Henry Kanoui, Robert Pasero et Philippe Roussel // Rapport préliminaire de fin de contrat IRIA, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université Aix–Marseille II, France.
18. Малпас Дж. Реляционный язык ПРОЛОГ и его применение / Дж. Малпас. – Москва : Наука, 1990. – 464 с.
19. Братко И. Программирование на языке ПРОЛОГ для искусственного интеллекта / И. Братко ; пер. с англ. – Москва : Мир, 1990. – 560 с.
20. Куклин В. М. От математической логики к языкам программирования искусственного интеллекта / В. М. Куклин // CS&CS, 2017. – Issue 1 (5). – С. 40–52.
21. Мински М. Фреймы для представления знаний / М. Мински ; пер. с англ. – Москва : Энергия, 1979. – 151 с.
22. Church A. Introduction to mathematical logic. Vol. 1 / A. Church. – Princeton : Princeton University Press, 1956. – 485 p.
23. Zadeh Lotfi A. Fuzzy sets / Lotfi A. Zadeh // Information and Control, 1965. – Vol. 8. – P. 338–353.
24. Kosko B. Fuzzy systems as universal approximation / B. Kosko // IEEE Transactions on Computers, 1994. – Vol. 43. – № 11. – P. 1329–1333.
25. Jang J. S. R. ANFIS: adaptive–network–based fuzzy inference system / J. S. R. Jang // IEEE transactions on systems, man, and cybernetics, 1993. – Vol. 23. – № 3. – P. 665–685.

Додаткова література

1. Newell A. The chess machine: an example of dealing with a complex task by adaptation / A. Newell // ACM. Proceedings of the 1955 Western joint computer conference, 1955. – P. 101–108.
2. Newell A. GPS, program that simulates human thought / A. Newell, H. Simon // Defense Technical Information Center, 1961. – Vol. 4. – № 10. – P. 109–124.
3. Shannon C. E. A Mathematical Theory of Communication / C. E. Shannon // The Bell System Technical Journal, 1948. – Vol. 27. – P. 379–423; 623–656.
4. Turing A. M. On Computable Numbers, with an application to the Entscheidungsproblem / A. M. Turing // Proc. London Math. Soc. Ser. 2, 1937. – Vol. 42. – P. 230–265.

5. Newell A. Programming the Logic Theory Machine / A. Newell, F. C. Shaw // Proceedings of the Western Joint Computer Con–ference, 1957. – P. 230–240.
6. McCarthy J. Lisp 1.5 Programmer's Manual / J. McCarthy, P. Abrahams, D. Edwards et al. – MIT Press, Cambridge, Massachusetts, 1962.
7. Chang C. L. Symbolic Logic and Mechanical Theorem Proving / C. L. Chang, R.C.T. Lee. – New York : Academic Press, 1973. – 331 p.
8. Pospesel H. Introdaction to Logic: Predicate Logic. Englewood Cliffs / H. Pospesel. – New Jersey : Prentice – Hall, 1976.
9. Maslov S. An inverse method of establishing deducibility in classical predicate calculus / S. Maslov // Dokl. AN SSSR. – 1964. – Vol. 159. – P. 17–20; Maslov S. Proof-search strategies for methods of the resolution type / S. Maslov // Machine Intelligence. – 1971. – № 6. – P. 77−90.
10. van Vaalen J. An extension of unification to substitutions with an application to automatic theorem proving / J. van Vaalen // IJCAI, 1975. – Vol. 4. – P. 77–82.
11. Sickel S. A search technique for clause interconnectivity graphs / S. Sickel // IEEE Trans. On Computers, 1976. – № 8. – P. 823–835.
12. Martelli A. From dynamic programming to search algorithms with functi-onal costs / A. Martelli, U. Montanari // IJCA, 1975. – Vol. 4. – P. 345–350; Optimizing decision trees through heuristically guided search // CACM, 1978. – Vol. 21. – № 12. – P. 1025–1039.
13. Круглов В. В. Искусственные нейронные сети / В. В. Круглов, В. В. Борисов. – Москва : Горячая линия. – Телеком, 2002. – 382 с.
14. Малпас Дж. Реляционный язык ПРОЛОГ и его применение / Дж. Малпас. – Москва : Наука, 1990. – 463 с.
15. Гроп Д. Методы идентификации систем / Д. Гроп. – Москва : Мир, 1979. – 302 с.
16. Куклин В. М. Взгляд на будущее планетарной цивилизации / В. М. Куклин // Universitates: Наука и просвещение, 2003. – № 4 (16). – С. 18–22.
17. Kuklin V. Will the artificial intelligence help us? [Electronic resource] / V. Kuklin // Computer science and cybersecurity, 2016. – Issue 4 (4). – P. 35–41. – Way of access : http://periodicals.karazin.ua/cscs/article/view/7837/7310.pdf.

ПРАКТИЧНІ ЗАНЯТТЯ

ПРАКТИЧНЕ ЗАНЯТТЯ № 1
БАЙЄСОВА СИСТЕМА

1. Структура правила виведення Байєсової системи.

Умовна ймовірність p (A | B) дорівнює відношенню спільної ймовірності p (A∩B) до ймовірності події B за умови, що вона не дорівнює 0.

p (A | B) * p (B) = p (A∩B) і аналогічно p (B | A) * p (A) = p (B∩A),

оскільки p (A∩B) = p (B∩A), то отримаємо співвідношення.

2. Теорема Байєса p (A | B) * p (B) = p (B | A) * p (A)

Для подій $B \cap A$ и $B \cap \neg A$ справедливо $B = (B \cap A) \cup (B \cap \neg A)$, тоді

$$p(B) = p((B \cap A) \cup (B \cap \neg A)) = p(B \cap A) + p(B \cap \neg A) = p(B|A) * p(A) + p(B|\neg A) * p(\neg A)$$

3.Принцип устрою Байєсової системи логічного висновку.

Таблиця 1

P = апостер.	(Py *	P) /	(Py *	P +	Pn *	(1 – P))
Оцінка ймовірності результату при позитивній відповіді	Ймовірність ствердної відповіді при наявності результату	Апріорна ймовірність результату	Ймовірність ствердної відповіді при наявності результату	Апріорна ймовірність результату	Ймовірність позитивної відповіді при відсутності результату	Апріорна ймовірність відсутності результату

1. Розробити Байєсову систему перевірки на допустимість даної особи на виборну посаду.

2. Розробити систему для вирішення питання: Чи він (вона) закохався?

Результат – закохався.

Апріорна ймовірність результату P – передбачається, що основні умови для цього виконані.

Деякі питання, на які дається відповідь (позитивний – ствердний)

1. Коли проходить – озирається?
2. Намагається підійти ближче?
3. Усміхається при зустрічі?
4. Ніяковіє?

ПРАКТИЧНЕ ЗАНЯТТЯ № 2

ТЕОРИЯ ДЕМПСТЕРА (DEMPSTER)−ШАФЕРА (SHAFER)

Міра істинності визначається інтервалом ***Довіра <Міра істинності <Правдоподібність***. Довіра – це сума мас свідоцтв, що підтримують гіпотезу. Правдоподібність = 1 – сума мас свідоцтв, що відкидають гіпотезу.

Розглянемо приклад. Можна припустити, що у студента є науковий керівник, який допоможе зробити кар'єру?

Таблиця 2

Гіпотеза	Маса свідоцтв	Довіра	Правдоподібність
Байдуже. Тобто керівника немає	0	0	0
Він з кимось радиться, формально керівник є, шанси зробити кар'єру малі	0,2	0,2	0,5
Керівник є, і шанси зробити кар'єру є	0,5	0,5	0,8
Універсальні. Він має керівника, але невідомо, допоможе він в кар'єрі, чи ні	0,3	1,0	1,0

Цей метод дає можливість, виходячи з свідчень експертів, оцінити міру істинності кожного твердження, зазначеного в таблиці.

Проведіть обговорення в групі. Заповніть таблицю і визначте Довіру і Правдоподібність.

ПРАКТИЧНЕ ЗАНЯТТЯ № 3

ПЛАН ДЛЯ РОБОТА

1. Структура П-правила в STRIPS для побудови схеми управління роботом.

Модель дії: взяття кубика зі столу

Передумови – кубик на столі, рука робота не зайнята, зверху на кубику нічого немає. Результат – рука тримає кубик.

pickup (x)

P (precondition – передумова): ONTABLE (x), HANDEMPTY, CLEAR (x).

D (delete list – список викреслювання): ONTABLE (x), HANDEMPTY, CLEAR (x).

A (add formula – формула додавання): HOLDING (x).

Підстановка відповідності в цьому випадку дає – **pickup** (x) може бути застосоване лише якщо x = B. Тоді новий опис стану: CLEAR (C), ON (C, A), ONTABLE (A), HOLDIBG (B).

Розгляньте інші умови.

2. Алгоритм дії робота в структурі STRIPS:

1. Ціль розбивається відразу ж на підцілі (пов'язані зрозуміло).

2. Вибір з чого починати, взагалі кажучи, випадковий.

3. Перевірка передумов. Перевіряються відповідність правил з набору даної підцілі.

Правило застосовується (активується), якщо його можна виконати.

4. Пошук уніфікацій.

5. Наслідки застосування правила повинні бути неодмінно узгоджені з виконанням правила.

6. Виникає новий стан (нова база даних).

7. Теперь потрібно шукати нове правило, передумови якого не суперечать новій базі даних і визначаються першою підціллю.

ПРАКТИЧНЕ ЗАНЯТТЯ № 4

ЛОГІКА ВИСЛОВЛЮВАНЬ

Формалізуємо завдання найпростішого логічного висновку.

База знань (складається з двох правил):

Правило 1: Якщо «відпочинок – влітку» і «людина – активна», то «їхати в гори»,

(відпочинок літом $\wedge$ людина_активна $\rightarrow$їхати_в_гори)

Правило 2: Якщо «любить сонце», то «відпочинок влітку»,

(любить_сонце $\rightarrow$відпочинок _влітку)

Вхідні дані – «людина активна» і «любить сонце».

Прямий висновок,

термінальне правило: все літерали повинні опинитися в базі даних.

Формальний запис

(«відпочинок – влітку» $\wedge$ «людина – активна») $\rightarrow$«їхати в гори»,

«Любить сонце», $\rightarrow$ «відпочинок влітку»,

«людина активна»

«любить "сонце"»

Прибираємо імплікації

~ («відпочинок – влітку» $\vee$ «людина – активна») $\vee$ «їхати в гори»,

а також

~ «любить сонце» $\vee$ «відпочинок влітку»,

«людина активна»

«любить "сонце"»

перепишемо 1речення

~ «людина – активна» $\vee$ ~ «відпочинок – влітку» $\vee$ ~ «їхати в гори»,

резолюція

«людина активна» $\vee$ ~ «людина – активна» $\vee$ ~ «відпочинок – влітку» $\vee$ ~ «їхати в гори»,

результат
~ «Відпочинок – влітку»$\vee$ ~ «їхати в гори»,
Резолюція (2 пропозиція)
«любить “сонце”»$\vee$ ~ «любить сонце» $\vee$ «відпочинок влітку»,
результат «відпочинок влітку» потрапляє в базу даних.
Резолюція залишку першого речення
«відпочинок влітку» $\vee$ ~ «відпочинок – влітку» $\vee$ ~ «їхати в гори»,
результат «їхати в гори» потрапляє в базу даних
Усі все літерали потрапили в базу даних. Виконана термінальне умова.

ПРАКТИЧНЕ ЗАНЯТТЯ № 5
ТЕОРІЯ ПРЕДИКАТІВ

Формальні методи перетворення речень

1. **Закони Моргана**

$$\sim (x + y) = \sim x * \sim y, \sim (x * y) = \sim x + \sim y.$$

2. **Як вводиться сколемівська функція**?

Можна перейти від $(\forall y)[(\exists x)P(x,y)]$ до $(\forall y)[P(g(y),y)]$ за допомогою сколемівської функції g(y). Це можливо, якщо область визначення (дії) квантора існування і область значень сколемівської функції (яка, до того ж, повинна існувати) співпадають.

(~(∃x)P(x) еквівалентно *(∀x)[~P(x)]; (~∀x)[P(x)]*
еквівалентно *(∃x)[~P(x)]*

3. Процедури перетворення ППФ у обчисленні предикатів [7]

Розглянемо техніку перетворень ***правильно побудованих формул*** (ППФ) в набір простих виразів для організації комутативної бази даних.

$$(\forall x)\{P(x) \Rightarrow \{(\forall y)[P(y) \Rightarrow P(f(x,y))] \wedge \sim(\forall y)[Q(x,y) \Rightarrow P(y)]\}\}$$

– вхідна ППФ

3.1. ***Виключення символів імплікації*** (~ X1 $\vee$ X2 замість X1 $\Rightarrow$X2)

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x,y))] \wedge \sim(\forall y)[\sim Q(x,y) \vee P(y)]\}\} .$$

3.2. ***Обмеження області дії заперечення*** (застосування не більше ніж до однієї атомної формули). Використовуємо закони Моргана (~ (X1$\vee$ X2) еквівалентно ~ X1 $\wedge$~ X2; (~ (X1$\wedge$ X2) еквівалентно ~ X1 $\vee$ ~ X2), еквівалентності виразів з кванторами (~ (∃x) P (x) еквівалентно (∀x) [~ P (x)]; *~(∀x) [P(x)]* еквівалентно *(∃x) [~P(x)])* та інші еквівалентності.

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x,y))] \wedge (\exists y)[Q(x,y) \wedge \sim P(y)]\}\} .$$

3.3. ***Розділення змінних.*** В межах області дії квантора змінна, яку зв'язують цим квантором, є німа змінна. Її всюди можна замінити будь-якою іншою (що не використовується в іншому значенні) змінною в межах області дії квантора, при цьому значення істинності цієї ППФ не зміниться. Стандартизація змінних в межах ППФ означає перейменування німих змінних з тією метою, щоб кожен квантор мав свою, властиву тільки йому, німу змінну. Так, замість $(\forall x)[\ P(x) \Rightarrow (\exists x)Q(x)\ ]$ отримаємо $(\forall y)[\ P(y) \Rightarrow (\exists y)Q(y)\ ]$.

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x,y))] \wedge (\exists w)[Q(x,w) \wedge \sim P(w)]\}\}\ .$$

3.4. ***Виключення кванторів існування.*** Можна перейти від $(\forall y)[(\exists x)P(x,y)]$ до $(\forall y)[P(g(y),y)]$ за допомогою так званої сколемівської функції g (y). Це можливо, якщо область визначення (дії) квантора існування і область значень сколемівської функції (яка, до того ж, повинна існувати) збігаються. У разі, якщо квантор існування не входить в область дії кванторів спільності, то застосовують сколемівську функцію без аргументів, тобто константу А: $(\exists x)P(x)$ замінюється на $P(A)$.

$$(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x,y))] \wedge [Q(x,g(x)) \wedge \sim P(g(x))]\}\}\ .$$

3.5. ***Перетворення в префіксну форму.*** Перенесення квантора спільності в початок формули (тоді вона буде складатися з префікса = ланцюжка кванторів і бесквантворної формули = матриці).

$$(\forall x)(\forall y)\{\sim P(x) \vee \{[\sim P(y) \vee P(f(x,y))] \wedge [Q(x,g(x)) \wedge \sim P(g(x))]\}\}\ .$$

3.6. ***Приведення до кон'юнктивної нормальної форми***. Будь-яка матриця може бути записана як кон'юнкція скінченної множини диз'юнкцій літералів. Використовуються, наприклад, еквівалентності $A \vee (B \wedge C) = A \vee B \wedge A \vee C$, а також $A \wedge (B \vee C) = A \wedge B \vee A \wedge C$.

$$(\forall x)(\forall y)\{[\sim P(x) \vee \sim P(y) \vee P(f(x,y))] \wedge [\sim P(x) \vee Q(x,g(x))] \wedge [\sim P(x) \vee \sim P(g(x))]\}\ .$$

3.7. ***Виключення кванторів спільності.*** Припущення, що можна не показувати явно квантори спільності приводить до спрощення. Тоді все буде приведено до точної відповідності з кон'юнктивною нормальною формою.

$$[\sim P(x) \vee \sim P(y) \vee P(f(x,y))] \wedge [\sim P(x) \vee Q(x,g(x))] \wedge [\sim P(x) \vee \sim P(g(x))]\ .$$

3.8. ***Виключення символів кон'юнкції.*** Оскільки матриця представлена в кон'юнктивній нормальній формі, можна звільнитися від явної присутності символів кон'юнкції. Запис множиною ППФ (кожна з яких є диз'юнкцією літералів) тобто множиною речень.

$$\sim P(x) \vee \sim P(y) \vee P(f(x,y))\ ,$$
$$\sim P(x) \vee Q(x,g(x))\ ,\quad \sim P(x) \vee \sim P(g(x))\ .$$

3.9. ***Перейменування змінних.*** Змінні повинні бути різними в різних виразах списку.

$$\sim P(x1) \vee \sim P(y) \vee P(f(x1,y)),$$
$$\sim P(x2) \vee Q(x2,g(x2)),$$
$$\sim P(x3) \vee \sim P(g(x3)).$$

Питання до теми

1. Навіщо потрібна операція приведення до кон'юнктивної нормальної форми будь-якого виразу в логіці предикатів?
2. У чому сенс перейменування змінних для теорії предикатів?
3. Поясніть сенс кванторів спільності та існування, визначте області їх дії.
4. Чи можна використовувати диз'юнктивну нормальну форму?
5. У чому відмінності опису факту і правила в логіці предикатів?
6. Який з кванторів спільності та існування має відношення до сколемівських функцій?
7. Що таке сколемівська константа?
8. Як виключити символи імплікації в теорії предикатів?
9. Якою процедурою правило зводиться до системи речень?

ПРАКТИЧНЕ ЗАНЯТТЯ № 6

СПРОСТУВАННЯ НА ОСНОВІ РЕЗОЛЮЦІЇ

ЗАВДАННЯ. Для всіх x і y, якщо x є батьком y, а y батьків z, то x є дідусем чи бабусею z. У кожного є батько.

S: $(\forall x)(\forall y)\{[P(x,y) \wedge P(y,z)] \Rightarrow G(x,z)\}$ – аксіома 1; – аксіома 2.

Цільова ППФ: $(\exists x)(\exists y)G(x,y)$ – існують такі особи x і y, що x – дідусь або бабуся особи y.

> Тут також перша аксіома – це правило. Друга – це факт. Застосовуючи факт $\sim P(x,y) \vee \sim P(y,z) \vee G(x,z)$ до правила $P(f(w),w)$, отримаємо при $y = w$ і $x = f(w)$, $\sim P(w,z) \vee G(f(w),z)$, а потім знову застосуємо факт $P(f(w),w)$, при заміні w на $f(w)$ і $z = w$, отримаємо відповідь $G(f(f(w)),w)$ навіть не знаючи, яке було поставлене запитання. Тобто, інтелектуальна система схильна навчатися сама навіть без примусу (тобто без отримання запитань).

Але задаємо все ж запитання $(\exists x)(\exists y)G(x,y)$. Дерево спростування.

Таблиця 3

	$\sim P(x,y) \vee \sim P(y,z) \vee G(x,z)$	$P(f(w),w)$	$P(f(w),w)$
$\sim G(u,v)$	$\sim P(u,v) \vee \sim P(y,v)$	$\sim P(u,f(v))$	***NIL***

Неважко бачити, що це неконструктивна теорема, яка говорить тільки про існування розв'язку.

Модифікація дерева спростування.

Таблиця 4

	$\sim P(x,y)\vee \sim P(y,z)\vee G(x,z)$	$P(f(w),w)$	$P(f(w),w)$
$\sim$ G(u,v) $\vee$ G(u,v)	$\sim P(u,v)\vee \sim P(y,v)\vee G(u,v)$	$\sim P(u,f(v))\vee G(u,v)$	$G(f(f(v)),\ v)$

1. Сколемівська функція вводиться для виключення квантора існування в аксіомі 2, причому вона вказує ім'я батька для свого аргументу.

2. В процесі уніфікації змінні замінюються на інші і можуть виникнути ситуації, коли розміщені в певному порядку змінні можуть виявитися функціями самих себе. Але це проблема постановки завдання, або ж треба шукати сенс в такій відповіді.

3. При розв'язанні може бути кілька спростувань і кілька відповідей, при цьому одна може бути досить загальною. Але немає способу заздалегідь з'ясувати чи є ця відповідь задовільною або загальною.

4. Якщо в процесі уніфікації змінні замінюються на константи і ці константи ніяк не визначені, то можна знову у відповіді перейти до змінних.

Питання до теми

1. Що таке уніфікація і як вона застосовується в логіці предикатів?
2. Якщо дерево спростування в теорії предикатів дає в кореневій вершині порожню множину, де шукати відповідь?
3. Навіщо використовують метод резолюції в логіці предикатів?
4. Що таке спростування на основі резолюції?
5. Чи можна виключати тавтологію?

ПРАКТИЧНЕ ЗАНЯТТЯ № 7

ЗВОРОТНА СИСТЕМА ПРОДУКЦІЙ ДЛЯ ГІПЕРГРАФІВ

Питання (мета):

$(\exists x)(\exists y)[КІШКА(x)\wedge СОБАКА(y)\wedge \sim БОЇТЬСЯ(x,y)]$

− чи є такі кішка і собака, причому кішка не боїться собаки?

Факти: СОБАКА (ДЖИМ);

~ГАВКАЄ (ДЖИМ);

КРУТИТЬ_ХВОСТОМ (ДЖИМ);

НЯВКАЄ (МУРКА).

Правила:

[КРУТИТЬ_ХВОСТОМ (x1) $\wedge$ СОБАКА (x1)] $\Rightarrow$ ПЕСТИТЬСЯ (x1)

[ПЕСТИТЬСЯ (x2) ГАВКАЄ (x2)] $\Rightarrow$ ~БОЇТЬСЯ (y2, x2)

СОБАКА (x3) $\Rightarrow$ ТВАРИНА (x3)

КІШКА (x4) $\Rightarrow$ ТВАРИНА (x4)

НЯВКАЄ (x5) $\Rightarrow$ КІШКА (x5)

Узгоджені графи розв'язків. Якщо поставлено питання, про можливість існування учасників певної дії або події, і щодо цих учасників щось відомо, то дана задача має сенс. Причому якщо можливе узгодження всіх аргументів (або значень змінних), немає протиріч, то такий розв'язок становить практичний інтерес[45]. У таких задачах основний акцент зроблено на узгодженні (після уніфікації) аргументів. Виявляється, існують підстановки, що уніфікують літерали.

{МУРКА / x}, {МУРКА / x5}; {МУРКА / y2};
{ДЖИМ / y}; {ДЖИМ / x2}; {ДЖИМ / x1},
при якому все узгоджено і відповідь теж:
КІШКА (МУРКА) $\wedge$ СОБАКА (ДЖИМ) $\wedge$ ~БОЇТЬСЯ (МУРКА, ДЖИМ).

Умова зупинки. Умовою зупинки (термінальною умовою) є факт узгодження такого графа, який повинен неодмінно закінчуватися в вершинах фактів. Звернемо увагу, що деякі правила, де згадується тварина, не знадобилися.

ПРАКТИЧНЕ ЗАНЯТТЯ № 8

РЕЗОЛЮЦІЯ ВСЕРЕДИНІ ГРАФІВ І / АБО

Обмежена цільова резолюція (ОЦР). Розглянемо приклад в рамках зворотної системи продукцій.

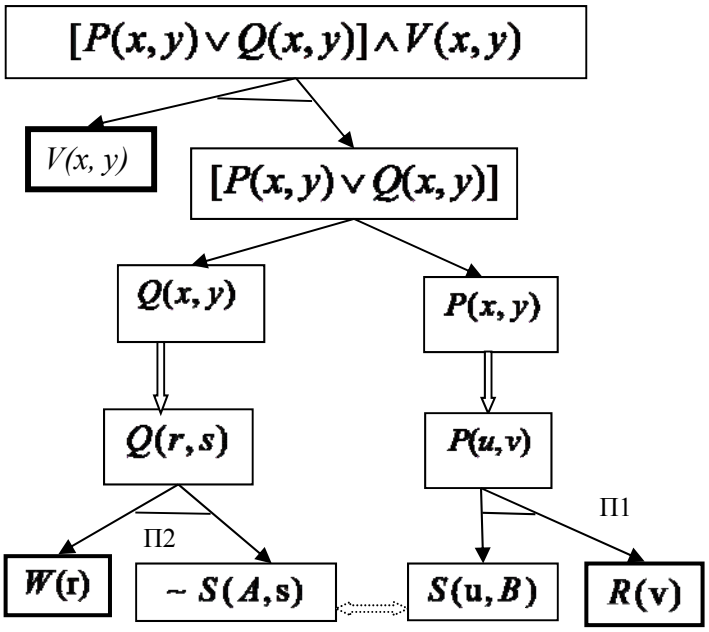
Мета (ціль): $[P(x,y) \vee Q(x,y)] \wedge V(x,y)$

Правила П1: $[R(v) \wedge S(u,B)] \Rightarrow P(u,v)$

П2: $[\sim S(A,s) \wedge W(r)] \Rightarrow Q(r,s)$

Факти: $R(B) \wedge W(B) \wedge V(A,B) \wedge V(B,B)$

Або у вигляді графа:



[45] Потрібно тільки відзначити, що він може бути іноді імовірнісним, вимагати для повної визначеності додаткових умов.

Рис. 1. Неузгоджений граф І / АБО (зворотна система продукцій)

1. Розглянемо спочатку узгодження гілок. Звернімо увагу на ліву гілку графа. В цьому випадку є два варіанти через існування двох фактів $V(A,B)$ і $V(B,B)$. Перший варіант, коли ми з них вибираємо $V(A,B)$. Тоді $x = A, y = B$ і права гілка узгоджується з іншими при $u = A, v = B$, але середня не узгоджується, бо виходить, що треба б задовольнити фактом $W(B)$, а у нас виходить $W(A)$. Граф узгоджується на фактах $R(B) \wedge V(A,B)$ завиключення середньої гілки.

2. Якщо ж виберемо для лівої гілки $V(B,B)$, тоді $x = u = B, y = v = B$, і середня гілка узгоджується з нашим вибором при $x = r = B, y = s = B$, а для узгодження правої гілки з цим вибором і фактами $R(B)$ и $W(B)$ отримаємо $u = B, v = B$. Тобто граф узгоджується лише на фактах $R(B) \wedge W(B) \wedge V(B,B)$.

3. Після застосування операції в формі значка, який представлений двосторонньою подвійною стрілкою, що означає резолюцію, і після виключення тавтології при ідентифікації граф перетворюється в такий спосіб:

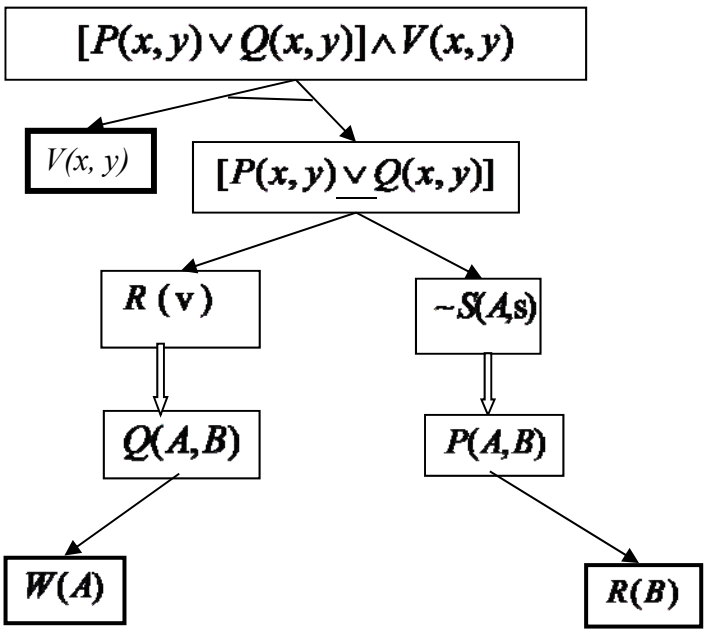


Рис. 2. Граф І / АБО після застосування ОЦР

Ця резолюція, що показана раніше у вигляді обопільної подвійної стрілки, називається обмеженою цільової резолюцією. Рішення очевидно при ідентифікації, але узгоджується граф тільки для фактів $R(B) \wedge V(A,B)$ при виключенні середньої гілки графа. Тобто, без застосування резолюції і з урахуванням її застосування граф узгоджується для неповного числа фактів, причому в різних комбінаціях.

Узагальнення цільового виразу – розділення змінних. Спробуємо зберегти загальність виведення подібно до того, як це роблять в логіці предикатів, розділяючи змінні. Можна спробувати ***відразу ввести різні змінні в цільовий вираз розглянутого вище прикладу***, переписавши його у вигляді

Мета (ціль): $[P(x1, y1) \wedge V(x1, y1)] \vee [Q(x2, y2)] \wedge V(x2, y2)]$

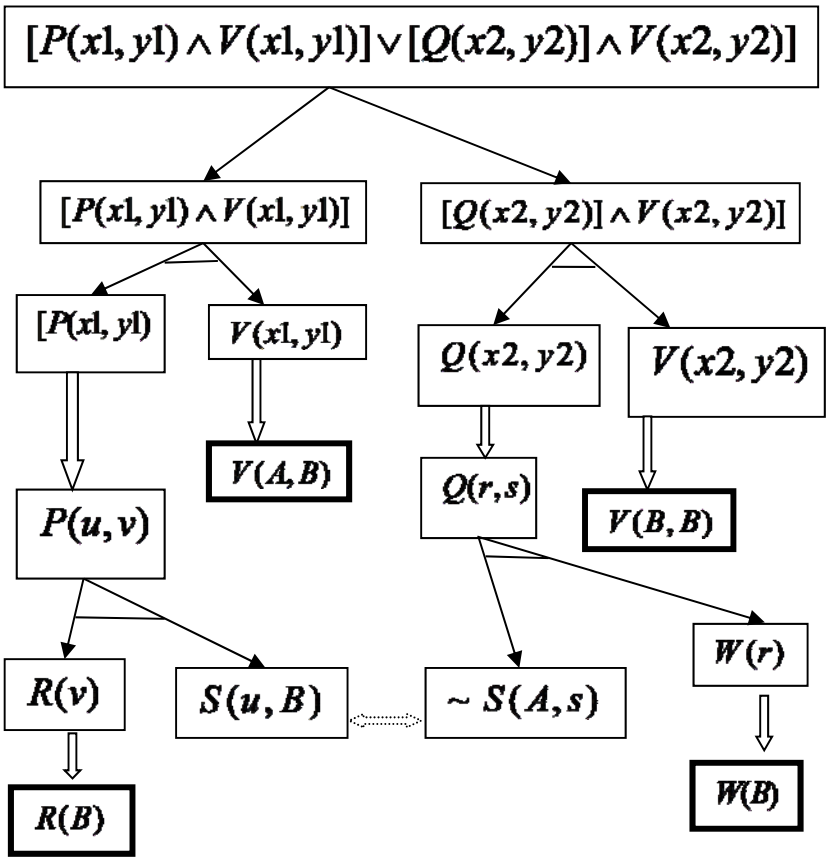


Рис. 3. Граф І / АБО з узагальненням цільового виразу в разі застосування ОЦР

Нагадаємо правила

$$\text{П1: } [R(v) \wedge S(u, B)] \Rightarrow P(u, v)$$

$$\text{П2: } [\sim S(A, s) \wedge W(r)] \Rightarrow Q(r, s)$$

Розглянемо формальні розв'язки на графі до застосування уніфікації і в разі її застосування.

1. До застосування резолюції: уніфікація {B / y1, A / x1, B / x2, B / y2} дає (див. Виділені кінцеві літерали) початковий набір фактів. Тобто $x1 = u = A, y1 = B; x2 = r = B, y2 = s = B$, і в цьому випадку вдалося знайти відповідність, і задача формально розв'язана: ліва гілка задовольняє фактам $R(B) \wedge V(A, B)$, а права – фактам $W(B) \wedge V(B, B)$. Зрозуміло, сформулювавши таким чином мету (ціль), насправді ми розділили її на дві, причому невиконання однієї не заважало виконанню іншої і навпаки. Тому ми показали, що ліва гілка задовольняє двом фактам, а права –

іншим двом. Здавалося б, ми не знайшли спільного розв'язку. Але погляньмо, що відбувається при застосуванні резолюції.

2. Після застосування резолюції: В цьому випадку для застосування резолюції потрібно провести уніфікацію $u = A, s = B$. Цікаво, що ця уніфікація не суперечить розглянутим вище уніфікаціям для двох незалежних гілок. Таким чином, проведення обмеженої резолюції призвело до об'єднання розв'язків і виконання вимог задоволення мети всіма фактами і узгодження всіх гілок графа.

Таким чином останній приклад показує, як важливо правильно сформулювати задачу. Оскільки при правильному формулюванні, розкриваючи дужки цільової функції, і в представленій кон'юнкції двох, взагалі кажучи, незалежних цілей (для цього змінені назви змінних на рис. 4), отримуємо два незалежних розв'язки, кожен з яких задовольняє лише частині фактів (але разом вони задовольняють всьому набору фактів). Однак всередині графа видно можливість проведення резолюції (тут ОЦР) двох літералів. Після цієї операції, отримуємо потрібну відповідь і доводимо теорему. Отже, по-перше, **потрібно уважно ставиться до формулювання задачі і по-друге – слід уважно розглядати внутрішні можливості процедур спрощення побудованого графа.**

ПРАКТИЧНЕ ЗАНЯТТЯ № 9

РОБОТА З ФРЕЙМАМИ

У глобальній базі даних є два – явний та функціональний, – фрейма.

а) явний фрейм

Таблиця 5

Посилка_10		
елем	:	(елем _із посилок)
відправник:	:	(Вася_1)
одержувач:	:	(Маша_10)
об'єкт:	:	(Цукерка_10)})

б) функціональний фрейм

Таблиця 6

Посилка_111		
елем	:	(елем _із посилок)
відправник:	:	*одержувач* (Посилка _10)
одержувач:	:	(Петро_111)
об'єкт:	:	(Цукерка_10)})

Тут очевидно, що Маша_10 і є одержувач (Посилка _10), але питання може бути поставлене інакше – чи надсилала Маша що-небудь кому-небудь? Це питання може бути представлене третім функціональним фреймом:

Таблиця 7

z (x)		
елем	:	*(*елем _із посилок*)*
відправник:	:	(Маша_10)
одержувач:	:	x
об'єкт:	:	y(x)

Підстановка {(x, Петро_111), (y, Цукерка_10)} – «Маша відправила цукерку Петрові».

ПРАКТИЧНЕ ЗАНЯТТЯ № 10

МАТЕМАТИКА ЧЕРЧА – ФОРМАЛІЗМ ЛЯМБДА–ОБЧИСЛЕННЯ

1. Редукції

α-перетворення: заміна в виразі $\lambda x.e$ імені змінної x на будь-яке інше (яке не використовується в цьому виразі) ім'я з одночасною заміною всіх вільних входжень цієї змінної в вираз e.

δ-редукція: вираз +1 4 може бути перетворено до виразу 5, а вираз OR TRUE FALSE – до виразу TRUE (тут OR – функція логічного «або»).

β-редукція: відповідає застосуванню функції, представленої лямбда-виразом, до аргументу,

η-перетворення: $\lambda x.E\ x$ еквівалентно функції E.

Якщо до виразу можна застосувати одну з редукцій, то він називається таким що редуціюється, або **редексом** (від redex – reducible expression). Якщо жодного редекса в виразі немає, вираз знаходиться в **нормальній формі**. Процес «налагодження» зводиться до перетворення вихідного виразу до його нормальної форми.

2. Методи

Канонічний порядок редукцій: нехай перетворення відбуваються таким чином, що редукція завжди застосовується до найлівішого **з найбільш внутрішніх редексов** – (АПР аплікативний порядок редукцій). Тобто перш за все «обчислюється» значення аргументу виклику, а потім вже відбувається підстановка цього значення в тіло, що викликається. Таким чином, цей порядок редукцій відповідає **енергійному способу обчислення** значення функцій в функціональному програмуванні.

Нормальний порядок редукцій (НПР) – застосування редукції до найлівішого з найбільш зовнішніх редексів – підстановка аргументу в тіло функції відбувається до того, як перетворення будуть проводитися над самими аргументами. Це схоже на процес «**лінивих» обчислень в функціональних мовах програмування** (де обчислення аргументу здійснюється лише один раз – при першому зверненні до аргументу; надалі відбувається звернення до вже обчисленого значення). Тут процес перетворення виразу, який служить аргументом застосування функції, відбуватиметься стільки раз, скільки разів аргумент з'являється у визначенні цієї функції.

(Λx.λy.y) ((λx.x x) (λx.x x)) при застосуванні НПР використовуємо β-редукцію, вважаючи весь вираз редексом. Дійсно, вираз є застосуванням функції (λx.λy.y) до деякого аргументу. Однак аргумент x не використовується в тілі функції – λy.y, так що незалежно від того, що являє собою цей аргумент, результатом такої β-редукції буде вираз λy.y. При застосуванні АПР – зациклюємось.

3. Задачі і вправи

1. Порівняння двох методів

(Λx. * X x) (+3 4)

а) .при застосуванні АПР спочатку – δ-редукція до виразу аргументу отримаємо (λx. * x x) 7, потім β-редукція перетворює цей вислів до (* 7 7) і потім δ-редукція приведе вираз до нормальної форми 49.

б) Тепер НПР, першою буде β-редукція до всього вислову (* (+3 4) (+3 4)) і ще три кроки δ-редукції, (* 7 (+3 4)), потім (* 7 7) і, нарешті, 49. Кроків в НПР більше.

2. Питання про зв'язану або вільну змінну

λy. (λy. + 2 y) (+ 1 y), тут хотілося б відразу застосувати α-перетворення, наприклад, λz. (λy. + 2 y) (+ 1 z). Але треба визначати, які входження змінної є вільними, а які зв'язаними. Тобто помилитися можна і при виконанні початкового α-перетворення.

Інший вираз λy. (Λx.λy. + x y) у в зовнішньому лямбда-виразі одне входження змінної y є вільним, а інше – зв'язаним.

3. Бета-редукція

Запишемо функцію, яка отримує значення більше на одиницю (ця процедура обчислення в лямбда-обчисленні називається бета-спрощення або бета-редукція)

$$(\lambda x.x+1)3 \Rightarrow 4$$

4. Карінг

Для функції двох аргументів $(\lambda x.y.x + y)$ застосуємо до одного аргументу

$$(\lambda x.y.x + y)1 \Rightarrow (\lambda y.1 + y).$$

Можна записати інакше (ця процедура називається карінг[46])

$$(\lambda x.y.x + y) \Rightarrow (\lambda x.(\lambda y.x + y))$$

очевидно

$$(\lambda x.(\lambda y.x + y))1 \Rightarrow (\lambda y.1 + y).$$

5. Спрощення запису

Вираз (+ 1) еквівалентно *(λx. + 1 x)*.

ПОЯСНЕННЯ: може використовуватися ***префіксний запис*** операцій, тобто замість звичного 3 + 5 можна записати + 3 5. Всі функції в ***розширеному лямбда-обчисленні*** будуть карінговими, тобто вираз + 3 також має сенс і є застосування функції + до константи 3, в результаті якого виходить функція збільшення цілого аргументу на 3. Функції застосовуються в порядку ***зліва направо***, тобто вираз *fxy* розуміють як застосування функції f до аргументу x, і застосування отриманого результату до аргументу y.

ПРАКТИЧНЕ ЗАНЯТТЯ № 11

ЗАВДАННЯ ЩОДО ВИКОРИСТАННЯ ЛІСП[47]

1. Функції без імені (анонімні функції) – форма визначення лямбда-функцій, що записується у вигляді: (lambda (parameters) body). Вид цього виразу нагадує звичайну форму визначення функції defun, тільки без імені. Такий особливий вид функції можна використовувати скрізь замість імені функції.

```
((Lambda (x) (* x x)) -6)
==> 36
```

Аналогічно визначимо і відразу застосуємо анонімну функцію двох аргументів:

```
((Lambda (x y) (+ x y)) 2 3)
```

---

[46] Карирування або каринг (англ. Currying) - перетворення функції від багатьох аргументів на набір функцій, кожна з яких з одним аргументом. Можливість такого перетворення вперше зазначена в працях Готтлоба Фреге, систематично вивчена Мойсеєм Шейнфінкелем, а найменування отримало за іменем Хаскелла Каррі – розробника комбінаторної логіки. Існування функцій карингу еквівалентно логічному твердженню.

[47] Варіант мови ЛІСП – СКІМ, див. Потапенко В. А. ЛІСП – СКІМ [Електронний ресурс] / В. А. Потапенко. –Режим доступу: http://www.lisp.ru/page.php?id=22.

```
==> 5
```

2. Написати функцію LEN, яка повертає довжину списку, переданого функції як параметр. Зокрема (LEN ‘(A B C)) поверне 3; (LEN NIL) поверне 0. Довжина порожнього списку дорівнює 0, довжина списку L дорівнює 1 + довжина списку (`cdr L`).

Розв’язання в організації рекурсії по довжині списку-аргументу:

```
(Defun LEN (list)
    (Cond ((eq list nil) 0)
        (T (+ 1 (LEN (cdr list))))
    )
)
```

*тут **cdr** – список без першого елемента, **list** створює список, LENGTH (або LEN) повертає як значення довжину списку. Для визначення нових функцій в Ліспі використовується спеціальна форма **defun**, вираз **eq** порівнює два символи і повертає **t**, якщо вони однакові, і повертає **nil** в іншому випадку, **t** – відповідний йому вираз буде обчислюватися в тих випадках, коли жодна інша умова не виконується.*

3. Написати функцію REV, яка бере список як аргумент і повертає обернений список в якості результату. (REV NIL) поверне NIL; (REV ‘(A B C)) поверне (C B A); (REV ‘(A B (C) D (T))) поверне ((T) D (C) B A).

Метод розв’язання – метод параметру, що накопичується :

```
(Defun REV (list h-arg)
    (Cond ((eq list nil) h-arg)
        (T (REV (cdr list) (cons (car list) h-arg)))
    )
)
```

*тут **cons** – об’єднує елементи (зазвичай два) в список, **car** - перший елемент списку (list 1 2) це (1 2), умовне речення (cond (<перевірка-1> <дію-1>) (...) (...)) обчислюються послідовно зліва направо поки не зустрінеться NIL.*

4. Записати функцію «факторіал», яка по аргументу n повертає число n! Завдання може бути вирішене за допомогою простої рекурсії.

```
(Defun Факторіал (n)
    (cond
        ((= N 0) 1)
        (T (* n (Факторіал (- n 1))))
    )
)
```

5. Зробити програму, яка запитує ім’я користувача, а потім виводить йому вітання. Використовуємо операції всередині СКІМа: **read** для читання імені, display для виведення на дисплей, newline для завершення

рядка. Придумуємо самі нові операції: привіт для вітання з одним параметром – ім'ям користувача; користувач для отримання імені користувача.

Текст програми.

```
(Define (привіт ім'я)
  (Display "Привіт,")
  (Display ім'я)
  (Display "!")
  (Newline))
(Define (користувач)
  (Write "Представтеся:")
  (Read))
(Привіт (користувач))
```

ПРАКТИЧНЕ ЗАНЯТТЯ № 12

ПРИКЛАД РОЗВ'ЯЗАННЯ ЗАДАЧІ З КОМЕНТАРЯМИ[48]

УМОВИ

Олексій, Вітя і Ігор після уроків знайшли на підлозі в кабінеті фізики маленьку гирьку. Кожен з них, розглядаючи знахідку, висловив два припущення. Олексій сказав: «Це гирька з латуні, і важить вона, швидше за все, 5 г». Вітя припустив, що гирька зроблена з міді і важить 3 г. Ігор же вважав, що гирька не з латуні і вага її – 4 г. Учитель фізики зрадів, що пропажа знайшлася, і сказав хлопцям, що кожен з них має рацію лише наполовину. З якого металу – латуні (L) або міді (M) – виготовлена гирька, і яка її вага? У відповіді запишіть список, що включає першу букву назви металу, а потім цифру, відповідну вазі гирьки, наприклад, (L 4).

РОЗВ'ЯЗАННЯ
(З коментарями)

Завдання будемо вирішувати засобами Common LISP – сучасного діалекту мови ЛІСП, стандартизованого ANSI. Ідея розв'язання – ***сформуємо повну множину можливих розв'язків і виберемо з неї ті, які задовольняють заданим умовам***. Програму реалізуємо в функціональному стилі.

Визначимо функцію, яка формує простір можливих відповідей. Для визначення функції скористаємося формою `defun` – формою введення функції користувача.

Спрощений синтаксис форми:

```
(defun name (parameter*)
  body-form*
```

[48] *Common Lisp* –діалект мови *ЛІСП*.

Як ім'я створюваної функції name можна використовувати практично будь-який символ. Зазвичай імена функцій містять тільки літерні символи і дефіси, але інші символи дозволені і використовуються в певних угодах про імена. Наприклад, функції, які перетворять один вид значення в інший, іноді використовують -> в імені. Найважливіша угода про імена - зазвичай будують складені імена з дефісами, а не з підкресленнями або зміною регістра. Таким чином, `tes-widget` краще відповідає стилю ЛІСП, ніж `testWidget`. Список параметрів функції (`parameter*`) визначає змінні, які будуть використовуватися для зберігання аргументів, переданих функції при її виклику. Якщо функція не приймає аргументів, вказують порожній список (). Параметри функцій поділяються на кілька класів: обов'язкові, необов'язкові, множинні і ключові. Тіло форми DEFUN складається з будь-якого числа виразів ЛІСПу `body-form`. При виконанні функції вони будуть обчислюватися в зазначеному у визначенні порядку, а значення останнього виразу повертається як значення функції. Крім того, для негайного повернення з будь-якої точки функції використовується спеціальний оператор `RETURN-FROM`.

Для виконання завдання визначимо функцію `cross` з двома параметрами-списками `a` і `b`. Список `a` при виконанні функції буде містити всі можливі варіанти матеріалу гирі `(m 1)`, а список `b` – всі можливі варіанти ваги `(3 4 5)`, зазначені в задачі. В результаті виклику функції буде отримано множину всіх можливих відповідей, представлене списком, що складається з двоелементного списку – можливих розв'язків. У двоелементною списку перший елемент визначає матеріал, а другий – вагу `(M 5)`.

```
(defun cross (a b)
…..
 )
```

Щоб сформувати тіло функції розглянемо другий можливий варіант оголошення функції – функції без імені, або анонімної функції.

Анонімні функції часто використовують в практичному програмуванні, в разі, коли тільки в одному місці програми потрібно визначити функцію для передачі її як аргумент в іншу функцію. Для визначення функції без імені використовується форма визначення лямбда-функцій, що записується у вигляді: `(lambda (parameters) body`. Вид цього виразу нагадує звичайну форму визначення функції `defun`, тільки без імені. Такий особливий вид функції можна використовувати скрізь замість імені функції. Для прикладу визначимо лямбда-функцію, що підносить свій єдиний аргумент до квадрату і відразу ж застосуємо її до аргументу:

```
((lambda (x) (* x x)) -6)
          ==> 36
((lambda (x y) (+ x y)) 2 3)
          ==> 5
```

Хоча так можна застосовувати LAMBDA-функції, але таким чином їх майже не використовують. Анонімні функції можуть бути корисні, коли вам потрібно передати функцію як аргумент іншої функції, а функція, яку вам потрібно передати, досить проста, щоб висловити вбудований аргумент.

Розглянемо приклад з ***вбудованою функцією.*** Функція sort сортує список, зазначений в якості першого аргументу за правилом, заданому другим аргументом. Другий аргумент повинен бути функцією, що приймає два параметри і повертає узагальнене логічне значення – ІСТИННЕ, якщо аргументи розташовані у правильному порядку і ХИБНЕ – якщо в неправильному. Для запобігання обчислень аргументи повинні бути «заквотривовані» – перший аргумент-список звичайним блокуванням: `'(4 1 6 7)` – апостроф перед списком запобігає його обчисленню, а другий аргумент – функціональним блокуванням `#' імя_функції` (`#'` працює як традиційне блокування, але вказується тільки для функцій).

Наприклад, сортування списку чисел за зростанням і убуванням:

```
sort '(4 1 6 7) #'(lambda (x y) (< x y)))
 ==> (1 4 6 7)
(sort '(4 1 6 7) #'(lambda (x y) (> x y)))
 ==> (7 6 4 1)
```

Крім того, анонімні функції можуть бути результатом роботи функції – можна створювати генератори функцій. Напишемо звичайну функцію, яка будує і повертає в якості результату функцію яка збільшує свій аргумент на задане число:

```
(defun make-incr (x) (lambda (y) (+ x y)))
```

В даному визначенні використано ще одне важливе застосування LAMBDA-виразів – їх використання для створення замикань (closures) – функцій, які захоплюють частину середовища виконання, в якому вони були створені. Технічне позначення функції, що посилається на вільну змінну в своєму контексті – замикання, тому що функція ніби «змикається» над змінною. Так, наша функція `make-incr` повертає функцію, що має посилання на змінну, яка перестане існувати після виходу з `make-incr`. Функція виглядає дивно, але вона працює саме так, як нам потрібно – якщо викликати `make-incr` з аргументом `2`, ми отримаємо анонімну функцію, яка збільшує значення свого аргументу на `2`. Викличемо цю функцію для створення «збільшувача» аргумента на 2 і присвоїмо результат виклику - створену анонімну функцію змінної:

```
(Setq 2+ (make-incr 2))
  #<FUNCTION: LAMBDA (Y) (+ X Y)>
```

Тепер 2+ є анонімна функція, яка збільшить свій аргумент на 2. Після того, як ми отримали об'єкт-функцію, її можна виконати. COMMON LISP надає дві функції для виконання функції через об'єкт-функцію: `FUNCALL` і APPLY. Вони відрізняються тим, як вони отримують аргументи, які будуть передані функції, що викликається. Введемо `FUNCALL`, це функція, яка використовується тоді, коли під час написання коду ви знаєте кількість аргументів, які ви будете передавати функції. Першим аргументом FUNCALL є об'єкт функції що запускається, а решта – аргументи які, передаються цій функції. Викличемо наш об'єкт-функцію для заданого аргументу, наприклад:

```
(Funcall 2+ 3)
==> 5
(Funcall # '(lambda (x) (* 4 x)) 4)
==> 16
```

Функція APPLY також дозволяє застосовувати об'єкт-функцію до аргументу. Подібно FUNCALL, її першим аргументом є об'єкт функції. Але після першого аргументу, замість перерахування окремих аргументів, вона приймає список. APPLY застосовує функцію до значень у списку. Це дозволяє вам переписати попередній код таким чином:

```
(Apply 2+ '(3))
```

```
==> 5
```

У цьому випадку виклик APPLY не особливо корисний, але він незамінний, коли список аргументів стає відомий тільки під час виконання. APPLY не дбає про те, чи використовує функція необов'язкові, залишкові або іменовані об'єкти – список аргументів створюється шляхом об'єднання всіх аргументів, і результуючий список повинен бути правильним списком аргументів для функції з достатньою кількістю аргументів для обов'язкових параметрів і відповідними іменованими параметрами.

Є ще багато зручних функцій для роботи з функціональними об'єктами. Розглянемо функцію MAPCAR. Перший аргумент MAPCAR – функція, яку необхідно застосувати, а наступні аргументи -списки, чиї елементи будуть поставляти аргументи для цієї функції. Вона завжди повертає список і не вимагає уточнення типу результату. Іншими словами функція MAPCAR застосовується до елементів аргументів-списків, беручи від кожного списку по елементу. Результат кожного виклику функції збирається в новий список. наприклад:

```
(Mapcar 2+ '(1 2 3 4 5))
==> (3 4 5 6 7)
(Mapcar # '(lambda (n) (+ n 10))' (1 2 3 4 5))
==> (11 12 13 14 15)
(Mapcar # '+ (list 1 2 3) (list 10 20 30))
==> (11 22 33)
```

В останньому прикладі об'єкт функція, відповідна до математичної операції додавання застосовується до двох аргументів. Вони витягуються кожен зі свого списку, сформованого за допомогою форми list: вираз (list 1 2 3) сформує список '(1 2 3), а вираз (list 10 20 30) – список '(10 20 30). Зазначена в прикладі форма виконає поелементне складання цих списків в порядку проходження їх елементів. Розглянемо тепер вираз такого виду:

```
(Mapcar (lambda (x) (mapcar (lambda (y) (list x y)) '(1 2
3))' (a b c))
==> (((A 1) (A 2) (A 3)) ((B 1) (B 2) (B 3)) ((C 1) (C
2) (C 3)))
```

Тут спільно працюють дві анонімні функції, причому одна є замиканням. Зовнішня анонімна функція по черзі отримує елементи зовнішнього списку '(a b c), передає їх внутрішній анонімній функції, яка по черзі об'єднує кожен елемент цього списку з кожним елементом внутрішнього списку '(1 2 3). Формується відповідь – список складної структури – список з трьох списків, кожен елемент якого є списком з двох елементів. Для «вирівнювання» спискової структури застосуємо до неї функцію append, що об'єднує списки. Для цього зручно скористатися функцією apply, перетвореної до виду об'єкта-функції: 1

```
(Apply # 'append' (((A 1) (A 2) (A 3)) ((B 1) (B 2) (B 3))
((C 1) (C 2) (C 3))) )
==> ((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

У цьому виразі функціональне блокування # 'append можна записати і таким чином: (function append):

```
(Apply (function append) '(((A 1) (A 2) (A 3)) ((B 1) (B 2)
(B 3)) ((C 1) (C 2) (C 3)) ))
==> ((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Таким чином, ми отримали список необхідної структури. Для зручності зберемо це в одну функцію:

```
;;; Розв'язання на мові Common Lisp
;; Визначимо набір функцій для розв'язання задачі
; Функція, за допомогою якої сформуємо
; простір можливих рішень
(Defun cross (a b)
    (apply
        (Function append)
        (Mapcar (lambda (x) (mapcar (lambda (y)
(list x y)) b)) a)
    )
)
; приклад виклику
(Cross '(m l)' (3 4 5))
==> ((M 3) (M 4) (M 5) (L 3) (L 4) (L 5))
```

Сформуємо наступні функції, потрібні нам для розв'язання задачі. Розробимо функцію, яка буде виконувати перевірку на рівність двох аргументів і повертати 1, коли аргументи рівні, і 0, в разі, якщо вони не рівні. Функція заснована на використанні базової функції мови ЛІСП `cond`, яка виконує перевірку умови. Форма `cond` містить деяку (можливо, нульову) кількість підвиразів, які є списками форм. Кожен підвираз містить форму умови і нуль або більше форм для виконання.

```
(Cond (test-1 consequent-1-1 consequent-1-2 ...)
      (Test-2)
      (Test-3 consequent-3-1 ...)
      ...)
```

Відбирається перший підвираз, чия форма умови обчислюється в `не-NIL`. Всі інші підвирази ігноруються. Форми відібраного підвиразу послідовно виконуються. Якщо більш точно, `COND` обробляє свої підвирази зліва направо. Для кожного підвиразу, обчислюється форма умови. Якщо результат NIL, `COND` переходить до наступного підвиразу. Якщо результат T, то підвираз обробляється як список форм. Після виконання списку форм, `COND` повертає управління без обробки підвиразів, що залишились. Оператор `COND` повертає результат виконання останньої форми зі списку. Якщо цей список порожній, тоді повертається значення форми умови. Якщо COND повернула управління без обчислення будь-якої гілки (всі умовні форми обчислювалися в NIL), повертається значення NIL. Для того, щоб виконати останнє підвираз, в разі якщо раніше нічого не виконалося, зазвичай використовують T для форми умови.

```
(Cond ((> 2 3) 1) (t 0))
==> 0
(Cond ((<2 3) 1) (t 0))
```

```
==> 1
```

Таким чином, ця перевірочна функція буде такою:

```
; Перевірочна функція, повертає 1 якщо аргументи
; рівні і 0, якщо нерівні
(Defun? (X y)
(Cond ((eq x y) 1) (t 0))
)
; приклад виклику
(? 'A' b)
==> 0
(? 'A' a)
==> 1
```

За допомогою цієї функції напишемо наступну тестуючу функцію `check`, яка порівнює два можливих рішення – двохелементні списки v a, і повертає кількість збігів в них. Для цього за допомогою базових функцій ЛІСП `car` (отримання голови списку) і `cadr` (отримання хвоста списку) з двохелементних списків витягуються відповідні частини, потім передаються в функцію?, що визначає наявність збігу (результат 1) або відсутність його (результат 0) і підсумовуються результати цих перевірок.

Дана перевірочна функція буде такою:

```
; Перевірочна функція, що порівнює два можливих
; рішення, повертає кількість збігів
(defun check (v a)
    (+
        (? (car v) (car a))
        (? (cadr v) (cadr a))
    )
)
; Приклад виклику
(check '(m 2) '(m 2))
==> 2
(check '(m 1) '(m 2))
==> 1
(check '(m 1) '(l 2))
==> 0
```

За допомогою тестуючої функції `check` створимо функцію `is`, яка перевіряє чи є заданий двохелементний список виду (матеріал вага) розв'язком системи. Перевірка відбувається відповідно до умов завдання: у висловлюваннях «гирька з латуні і важить 5 г», «гирька зроблена з міді і важить 3 г», «гирька не з латуні і важить 4 г» істинна тільки одна частина. Функція як аргумент отримує двоелементний список виду (мате-

ріал вага) `var` і повертає NIL, якщо список-аргумент не є розв'язком і сам список `var`, якщо він є розв'язком задачі. Для цього список-аргумент `var` віддається функції перевірки `check`, яка його порівнює зі списками, що представляють висловлювання, задані в умові завдання. Далі результат порівнюється з 1 – і якщо одиниця була отримана в результаті всіх трьох перевірок, то аргумент функції `var` є розв'язком задачі і повертається в якості відповіді. Перевірку на те, що у всіх трьох випадках були отримані одиниці зручно організувати за допомогою звичайної логічної функції `and`. Дана перевірочна функція буде мати такий вигляд:

```
    ;  Перевірочна функція, яка оцінює можливий
розв'язок,
    ; повертає NIL, якщо розв'язок не підходить і
    ; сам розв'язок, якщо підходить
    defun is (var)
        (cond
            ((and
                (= 1 (check var '(L 5)))
                (= 1 (check var '(M 3)))
                (= 1 (check var '(M 4)))
            ) var)
        )
    )
    ; Приклад виклику
    (is '(m 3))
    ==> NIL
    (is '(m 5))
    ==> (M 5)
```

За допомогою функції `is` можна написати рекурсивну функцію, що переглядає всю множину можливих розв'язків і повертає перший знайдений. Для цього функція тестує перший елемент зі списку можливих рішень. Якщо цей елемент є розв'язком задачі, то він буде повернутий як результат функції. Якщо не є, то буде вирахувано хвіст списку можливих розв'язків – залишок, без вже перевіреного елемента, і переданий на вхід функції як аргумент. Такий рекурсивний виклик закінчиться в разі, якщо або буде знайдено розв'язок задачі, або закінчаться елементи в списку – функція обчислення хвоста списку поверне значення NIL.

Ця рекурсивна функція пошуку розв'язків може бути такою:

```
    ; Рекурсивна функція, яка шукає розв'язок,
    ; що задовольняє вимогам задачі, серед
    ; списку можливих розв'язків
    (defun solve (lst)
        (cond
```

```
            ((null lst) nil)
            ((is (car lst)))
            (t (solve (cdr lst)))
        )
    )
```

Результат застосування цієї функції до списку всіх можливих розв'язків і буде розв'язком нашої задачі:

```
; Приклад виклику – розв'язок задачі
 (solve (cross '(m l) '(3 4 5)))
  ==> (M 5)
```

ПРАКТИЧНЕ ЗАНЯТТЯ № 13
НАВЧАННЯ НЕЙРОННИХ СИСТЕМ

Алгоритм зворотного поширення помилки – це прямий прохід, при якому обчислюється відгук мережі на поданий вхідний сигнал; зворотний прохід, при якому, відповідно до отриманого сигналу похибки, модифікуються ваги всіх нейронів від вихідного шару до вхідного.

Вважатимемо, що індекс i – належить до нейрона у вхідному шарі, індекс j – до нейрона в прихованому шарі і індекс k – до нейрона у вихідному шарі; навчальна множина C у вигляді набору прикладів (x_n, d_n), де n – це номер навчального прикладу; позначимо похибку на виході нейрона для навчального прикладу n як $e_k(n)$; реальний відгук y_k, очікуваний відгук d для нейронів вихідного шару.

$$e_k = d_k - y_k . \tag{1}$$

«Енергію» похибки можна записати у вигляді

$$E = \frac{1}{2}\sum_k e_k^2 , \tag{2}$$

де підсумовування здійснюється за всіма нейронам вихідного шару.

Модифікація ваг при зворотному проході обчислень повинна бути спрямована на зменшення величини E

З огляду на $y_k = \varphi(v_k) = \varphi(\sum_{j=0}^{m} w_{kj} y_j)$ похідну сигналу похибки по деякій синаптичній вазі в мережі можна записати як:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial w_{ji}} . \tag{3}$$

де $\frac{\partial E}{\partial e_j} = \frac{1}{2}\frac{\partial \sum_j e_j^2}{\partial e_j} = e_j$, $\frac{\partial e_j}{\partial y_j} = -1$, $\frac{\partial y_j}{\partial v_j} = \varphi'(v_j)$, $\frac{\partial v_j}{\partial w_{ji}} = y_i$,

отримаємо $$\frac{\partial E}{\partial w_{ji}} = -e_j \varphi'(v_j) y_i . \qquad (4)$$

Метод градієнтного спуску $\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$, η – константа швидкості навчання.

Нехай $\Delta w_{ji} = \eta \delta_j y_i$, тоді $\delta_j = -\frac{\partial E}{\partial v_j} = -\frac{\partial E}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial v_j} =$ – локальний градієнт нейрона *j*. Таким чином, для обчислення модифікації ваги w_{ji} досить обчислити локальний градієнт нейрона *j*.

Розглянемо локальний градієнт нейрона *j* в прихованому шарі мережі.

$$\delta_j = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} = -\frac{\partial E}{\partial y_j}\varphi'(v_j) . \qquad (5)$$

Оскільки $$E = \frac{1}{2}\sum_k e_k^2 , \quad \frac{\partial E}{\partial y_j} = \sum_k e_k \frac{\partial e_k}{\partial y_j} = \sum_k e_k \frac{e_k}{v_k}\frac{v_k}{y_j} ,$$

$$e_k = d_k - y_k = d_k - \varphi_k(v_k) ,$$

$v_k = \sum_{j=0}^{m} w_{kj} y_j$, отримаємо $\frac{\partial e_k}{\partial v_k} = -\varphi_k'(v_k)$, $\frac{\partial v_k}{\partial y_j} = w_{kj}$, тоді

$$\frac{\partial E}{\partial y_j} - \sum_k e_k \varphi_k'(v_k) w_{kj} = -\sum_k \delta_k w_{kj} \quad \text{и} \quad \delta_j = \varphi_j'(v_j)\sum_k \delta_k w_{kj} . \qquad (6)$$

Модифікація ваг виконується після обчислення відгуку мережі на кожен з навчальних прикладів. Спочатку обчислюються локальні градієнти для всіх нейронів, починаючи з вихідного шару, а потім, відповідно до отриманих градієнтів, обчислюються модифікації всіх ваг мережі.

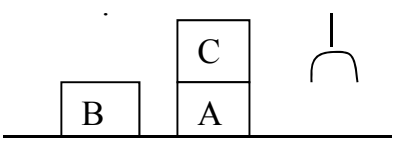
Процес навчання припиняється, якщо похибка для всіх навчальних прикладів не перевищує деякого прийнятного значення, або якщо перевищено максимально допустиму кількість ітерацій.

ПРАКТИЧНЕ ЗАНЯТТЯ № 14

ТЕХНОЛОГІЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ (СИСТЕМА RSTRIPS) ПРИ КОНФЛІКТІ ЦІЛЕЙ

Розглянемо ситуацію, коли кубик С стоїть на кубику А, а поверхня кубика В вільна, причому кубики А і В стоять на столі. Рука вільна.

HANDEMPTY, CLEAR (B), CLEAR (C), ONTABLE (B), ONTABLE (A), ON (C, A)



Мета для робота: поставити кубик С на кубик В, а кубик А зверху на кубик С:

ON (C, B) ON (A, C)

Побудуємо послідовність вибору стратегії:

1. Ціль явно складова, тому вона розбивається відразу ж на підцілі (пов'язані зрозуміло). Перша підціль − ON (C, B), а друга підціль − ON (A, C).

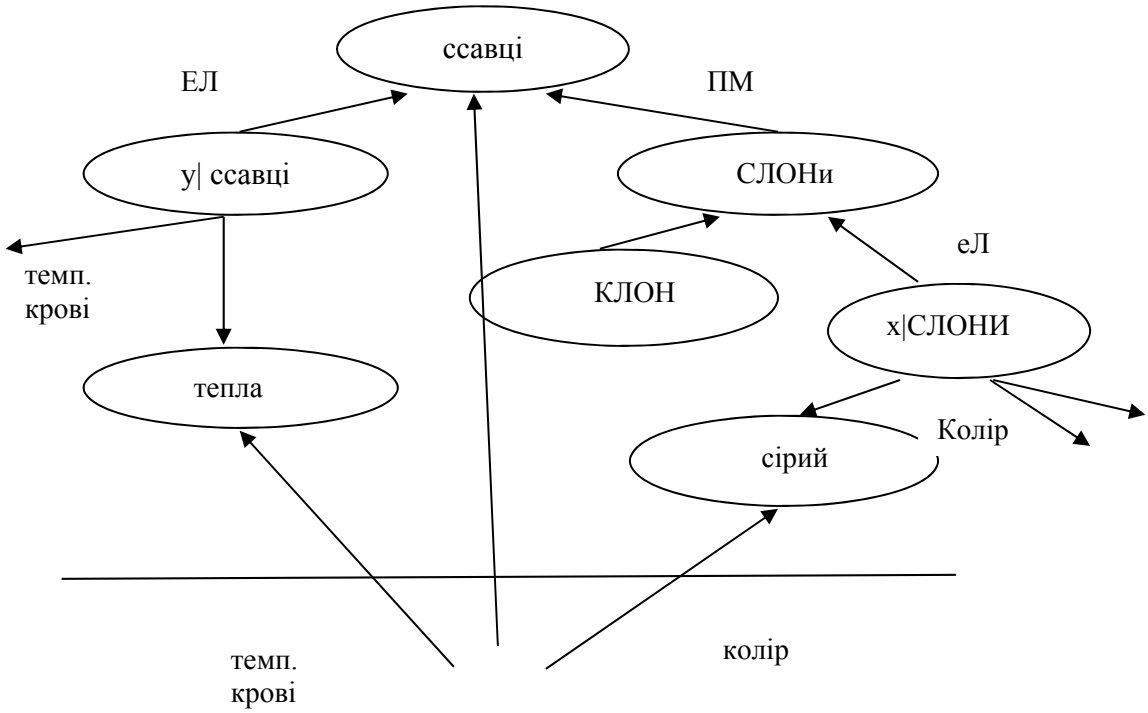
2. Вибор з чого починати, взагалі кажучи, випадковий, але, тепер почали з другої підцілі.

ПРАКТИЧНЕ ЗАНЯТТЯ № 15

СЕМАНТИЧНА МЕРЕЖА З НЕЯВНОЮ КОНСТАНТНОЮ ВЕРШИНОЮ

Розглянемо випадок, коли константна вершина «захована» в мережі фактів. Це варіант «неявної» константної вершини. Тому для її пошуку потрібно піднятися в вершину «ссавці» (зазвичай програма при неявних константних вершинах стартує по дугах ЕЛ або ПМ, які забезпечують успадкування, щоб потім опускатися по віялу ланцюгів) і пройтися по всіх ланцюгах, які виходять з цієї вершини. Покажіть, як досягається відповідність мережі цільовій мережі фактів у цьому випадку.

Приклад:



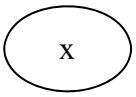


Рис. 4. Пошук відповідності мережі цільової зі змінною цільовою вершиною і мережі фактів з неявною константною вершиною

Змінні цільові вершини – це зазвичай константні вершини, причому константи сколемівські, тобто такі, які слід визначити.

Константні вершини – це певні реальні об’єкти.

Стратегія розв’язання задач наступна:

Пошук константних вершин в мережі фактів

(А) здійснюється пошук константних вершин в мережі фактів, які знаходяться в ланцюзі, що з’єднує змінну цільову вершину або

(Б) піднімаючись в множину, звідки виділено змінний цільовий об’єкт (змінна цільова вершина пов’язана з множиною, яка дозволяє операцію успадкування) і потім опускаючись за іншими напрямками, відшукуються константні вершини.

ТЕСТ
(виберить вірні відповіді)

1. Що таке в теорії експертних систем комутативність системи продукцій?

а) Застосування правил повинно відбуватися в порядку їх слідування, починаючи з першого.

б) Застосування правил повинно відбуватися в зворотному порядку їх слідування, починаючи з останнього.

в) Правила можна застосовувати в довільному порядку.

г). Правила потрібно застосовувати вибірково, заздалегідь заданому списку.

2. Що таке уніфікація і як вона застосовується в логіці предикатів?

а) Процедури створення нового речення з двох інших можливі, якщо початкові значення змінних збігаються.

б) Речення при цьому не повинні взагалі містити змінних.

в) Процедури створення нового речення з двох інших можливі, якщо змінні в них переозначають так, щоб вони збігалися.

г). Змінні в реченнях при цьому можуть бути будь-якими.

3. В чому відмінності опису факту і правила в логіці предикатів?

а) Факт – це окремі літерали, а правило – це два літерала, пов’язаних імплікацією (або еквівалентним оператором).

б) Форма опису, взагалі кажучи, однакова.

в) Факти можуть бути пов’язані кон’юнкцією, а правила – тільки імплікацією.

г) Правила не містять кон’юнкції, а факти містять.

4. Якою процедурою правило зводиться до системи речень?

а) Заміною імплікації на інші зв’язки.

б) Резолюцією.

в) Спрощуванням на основі резолюції.

г) Уніфікацією.

5. Що таке термінальна умова?

а) Умова початку процедури розв’язання.

б). Умова зупинки процедури розв’язання.

в). Умова, яка дозволяє призупинити розрахунок і задати питання користувачеві.

г). Умова, при виконанні якої друкується відповідь на питання користувача.

6. Як можна в логіці висловлювань реалізувати термінальну умову?

а) Коли всі цілі виявляться в оновленій базі даних.

б). За допомогою процедур резолюції
в). При використанні всіх правил.
г). Повною взаємною уніфікацією цілей і фактів.

7. Які є підстави вважати реляційні бази даних системами штучного інтелекту?
а) Ніяких.
б) Через використання логічних зв'язків.
в) Через можливість застосування заздалегідь заданого алгоритму.
г) Через можливість незалежного від користувача пошуку.

8. Який з кванторів спільності та існування має відношення до сколемівських функцій?
а) Жоден.
б) Квантор спільності.
в) Квантор існування.
г) Обидва квантори.

9. Навіщо потрібна операція приведення до кон'юнктивної нормальної форми будь-якого виразу в логіці предикатів?
а) Для явного використання повсюдно кон'юнкції.
б) Для виключення за замовчуванням диз'юнкції.
в) Для виключення імплікації.
г) Для виключення за замовчуванням кон'юнкції.

10. Що таке сколемівська константа?
а) Фіктивна константа.
б) Цілком певна константа, що має певне задане значення.
в) Константа, значення якої слід перевірити.
г) Деяка константа, яка існує, але поки вона не визначена.

11. Наведіть приклади комутативних систем подання знань.
а) Графи.
б) Експертні системи на основі теорії предикатів.
в) Семантичні мережі
г) Взагалі всі системи штучного інтелекту.

12. Для чого використовують метод резолюції в логіці предикатів?
а) Для спрощення процедур обчислення.
б) Для отримання з двох речень третього.
в) Для уніфікації.
г) Для виключення кон'юнкції.

13. Що таке спростування на основі резолюції?
а) Використання заперечення результату процедури резолюції.

б) Використання заперечення цільової функції в структурі операцій на основі резолюції

в) Використання заперечення речень для проведення процедури резолюції.

г) Доказ того, що резолюція двох речень має значення «хибне».

14. Чи можна використовувати диз'юнктивну нормальну форму?

а) Не можна, ні за яких умов.

б) Можна тільки в теорії графів І / АБО.

в) Можна тільки в мові Пролог.

г) Взагалі кажучи, можна у всіх випадках.

15. Поясніть, що таке глобальна база даних.

а) Це велика база даних.

б) База даних, об'єднана з базою знань.

в) Кілька пов'язаних баз даних.

г) Незастосовне поняття.

16. Поясніть відмінності поведінки системи STRIPS і системи RSTRIPS при взаємодії цілей.

а) STRIPS не перебудовує плани, а RSTRIPS це робить.

б) RSTRIPS не відрізняється практично від STRIPS, це більш акуратна версія.

в) RSTRIPS повністю запам'ятовує план, а STRIPS – частково.

г) Вони в принципі не відрізняються.

17. Як перейти до нечіткості в описі чітких характеристик.

а) Вважати події випадковими.

б) Використовувати імовірнісний підхід.

в) Знайти ступінь належності характеристик певного предмета до даного типу характеристик.

г) Визначити наявність певних характеристик в загальному наборі характеристик предмета або явища.

18. Як провести процедуру переходу до чіткості в алгоритмах нечіткої логіки?

а) Визначити максимуми результуючої характеристичної функції.

б) Визначити мінімуми результуючої характеристичної функції.

в) Знайти середньозважене значення аргументу результуючої характеристичної функції.

г) Знайти середнє значення результуючої характеристичної функції.

19. Що таке в алгоритмі дії робота в структурі STRIPS Precondition (передумова)?

а) Початковий стан світу для виконання наступного кроку плану.

б) Початкове положення виконавчих механізмів для виконання наступного кроку плану.

в) Повний набір даних для реалізації процедури побудови планів.

г) Незмінні в процесі виконання плану набори даних.

20. Що таке в алгоритмі дії робота в структурі STRIPS delete list (список викреслювань)?

а) Видаляються дані після даного кроку виконання плану.

б) Видаляються дані про виконавчі механізми після даного кроку виконання плану.

в) Видалені дані на попередньому кроці виконання плану.

г) Видаляються дані на наступному кроці виконання плану.

21. Що таке в алгоритмі дії робота в структурі STRIPS add formula (формула додавань)?

а) Додаються дані про виконавчі механізми після даного кроку виконання плану.

б) Додаються дані про стан світу після даного кроку виконання плану.

в) Додані дані про стан світу на попередньому кроці виконання плану.

г) Додаються користувачем дані для наступного кроку виконання плану.

22. Вкажіть формальні підстави вважати реляційні бази даних системами штучного інтелекту.

а) Немає.

б) Через використання логічних зв’язок.

в) Система подібна до системи зв’язаних фреймів.

г) Є система таблиць.

23. Імплікація, заперечення, диз’юнкція і кон’юнкція. Що з них використовується в описі процедур логіки предикатів?

а) Все з переліченого.

б) Тільки диз’юнкція і кон’юнкція.

в) Диз’юнкція, заперечення.

г) Імплікація, заперечення, диз’юнкція.

24. Імплікація, заперечення, диз’юнкція і кон’юнкція. Що з них використовується в графах І /АБО?

а) Все з переліченого.

б) Тільки диз’юнкція і кон’юнкція

в) Імплікація, заперечення

г) Імплікація, заперечення, диз’юнкція.

25. Імплікація, заперечення, диз’юнкція і кон’юнкція. Що з них використовується в мові Пролог?

а) Все з переліченого.

б) Тільки диз’юнкція і кон’юнкція.

в) Імплікація, заперечення.

г) Імплікація, заперечення, диз’юнкція.

26. Наведіть практичну форму структури П-правила в STRIPS для побудови схеми управління роботом.

а) Precondition (передумова) і add formula (формула додавань), a delete list (список викреслювань) не виписують, хоча і враховують неявно.

б). Precondition (передумова) і delete list (список викреслювань).

в). Precondition (передумова), add formula (формула додавань), delete list (список викреслювань).

г). Precondition (передумова), delete list (список викреслювань) об'єднують з add formula (формула додавань).

27. Як виключити символи імплікації в теорії предикатів?

а) Використати заперечення і диз'юнкцію.

б) Використати заперечення і кон'юнкцію.

в) Просто викреслити.

г) Це зробити неможливо.

28. Що таке уніфікація в логіці предикатів?

а) Заміна змінних.

б) Прирівнювання однієї змінної до іншої в різних літералах при процедурі резолюції.

в) Прирівнювання однієї змінної до іншої в різних літералах при процедурі диз'юнкції.

г) Прирівнювання однієї змінної до іншої в різних літералах при процедурі кон'юнкції.

29. Що таке «Пряма система продукцій»?

а) Використовуючи факти, а також частково або повністю правила, шукати цільову функцію.

б) Узгодити всі цілі, правила і факти.

в) Для деяких цілей, використовуючи частково або повністю правила, домогтися узгодження з фактами.

г) Для всіх цілей, використовуючи частково або повністю правила, домогтися узгодження з фактами.

30. Что таке «Зворотна система продукцій»?

а) Використовуючи факти, а також частково або повністю правила шукати цільову функцію.

б) Узгодити всі цілі, правила і факти.

в) Для деяких цілей, використовуючи частково або повністю правила, домогтися узгодження з фактами.

г) Для всіх цілей, використовуючи частково або повністю правила, домогтися узгодження з фактами.

31. Термінальна умова для прямої системи продукцій – це:

а). Задане число кроків (незалежно від результату)

б). Повне заповнення (всі факти з початкового набору увійшли в базу).
в) а) і б) одночасно.
г) Або а), або б).

32. Термінальна умова для зворотної системи продукцій – це:
а) Повне узгодження отриманого після виключення зайвих фактів графа розв'язку.
б) Неповне узгодження отриманого графа.
в) Перевірка однозначності розв'язку.
г) Одночасне виконання пунктів а) і в).

33. У зворотній системі продукцій факти повинні бути узгоджені:
а) Всі.
б) Найважливіші.
в) Жоден.
г) Деякі.

34. В теорії Байєса використовують умовну ймовірність:
а) Для незалежних подій.
б) Для залежних подій.
в) Для подій, пов'язаних певною умовою.
г) Для подій певного типу.

35. База даних в експертних системах до розв'язання задачі містить:
а) Тільки факти.
б) Факти і правила.
в) Правила.
г) Факти, правила і цілі.

36. База знань в експертних системах до розв'язання задачі містить:
а) Тільки факти.
б) Факти і правила.
в) Тільки правила.
г) Факти, правила і цілі.

37. Виключати літерали в складносурядному реченні в теорії предикатів можна:
а) Якщо літерал з визначеності має значення «хибне».
б) Якщо літерал з визначеності має значення «істинне».
в) Якщо літерал має невизначене значення.
г) Якщо літерал зайвий.

38. Виключення кванторів існування можливе:
а) При використанні сколемівської функції.
б) Завжди, просто треба їх прибрати.

в) Якщо є інші змінні.
г) Якщо речення не має сенсу.

39. Виключати частини речення в складносурядному реченні в теорії предикатів можна:
а) Якщо ця частина речення зайва.
б) Якщо ця частина речення з певністю має значення «хибне».
в) Якщо ця частина речення має невизначене значення.
г) Якщо це тавтологія.

40. У теорії предикатів цілі приводять до форми:
а) Речення.
б) Правил.
в) Фактів.
г) Залишають без змін.

41. Виключати частину речення в складносурядному реченні в теорії предикатів можна:
а) Якщо ця частина речення має невизначене значення
б) Якщо ця частина речення з певністю має значення «хибне»
в) Якщо ця частина речення є частиною іншого речення
г) Якщо ця частина речення зайва.

42. Чи тотожні подання задач в теорії предикатів та на графі І / АБО?
а) Абсолютно.
б) При застосуванні графів частина розв'язків отриманих в теорії предикатів може бути втрачена.
в) Розв'язки на графі відмінні від розв'язків в теорії предикатів.
г) При застосуванні теорії предикатів частина розв'язків, отриманих на графі, може бути втрачена.

43. Що таке «тіло» в виразах мови Пролог?
а) Список цілей – це тіло.
б) Все речення – це «тіло».
в) «Тіло» – це форма запису якого завгодно речення в мові Пролог.
г) Цей термін не використовується.

44. Що таке «голова» у виразах мови Пролог?
а) «Голова» речення – це список цілей.
б) «Голова» – це квантор існування.
в) «Голова» – це форма запису якого завгодно квантору в мові Пролог.
г) «Голова» – це, зокрема, консеквент в структурі правила.

45. У які види речень мови ПРОЛОГ входить «голова»?
а) В Факти і Правила.
б) Тільки в Правила.

в) В Питання і Правила.
г) У всі види речень.

46. У які види речень мови ПРОЛОГ входить «тіло»?
а) В Факти і Правила.
б) Тільки в Правила.
в) В Питання і Правила.
г) У всі види речень.

47. Чи забезпечує Пролог одиничність розв'язку?
а) Абсолютно.
б) Не забезпечує.
в) Не забезпечує, але дозволяє знайти деякі розв'язки.
г) Знаходить взагалі всі можливі розв'язки.

48. Фрейм складається з:
а) Бінарних предикатів.
б) Предикатів вищих порядків.
в) Окремих літералів.
г) Всіх видів речень.

49. Успадкування властивостей (A–Kind–Of) в системі фреймів
а) A–Kind–Of слот вказує на інший фрейм, звідки переносяться відомості аналогічних слотів.
б) A–Kind–Of слот шукає подібний слот іншого фрейма і об'єднує відомості.
в) A–Kind–Of слот шукає подібний слот іншого фрейма і замінює свої відомості.
г) A–Kind–Of слот шукає подібний слот іншого фрейма і замінює його відомості.

50. Жорстка форма кон'юнкції двох літералів в термінах характеристичних функцій для нечіткої логіки – це:
а) Добуток характеристичних функцій, які відповідають цим двом літералам.
б) Мінімальні значення з двох характеристичних функцій, які відповідають цим двом літералам.
в) Сума характеристичних функцій, які відповідають цим двом літералам.
г) Різниця характеристичних функцій, які відповідають цим двом літералам.

51. М'яка форма кон'юнкції двох літералів в термінах характеристичних функцій для нечіткої логіки.
а) Добуток двох характеристичних функцій, які відповідають цим двом літералам.

б) Мінімальне значення з двох характеристичних функцій, які відповідають цим двом.

в) Сума характеристичних функцій, які відповідають цим двом літералам.

г) Різниця характеристичних функцій, які відповідають цим двом літералам.

52. Жорстка форма диз'юнкції двох літералів в термінах характеристичних функцій для нечіткої логіки.

а) Сума мінус різниця двох характеристичних функцій, які відповідають цим двом літералам.

б) Максимальне значення з двох характеристичних функцій, які відповідають цим двом літералам.

в) Сума характеристичних функцій, які відповідають цим двом літералам.

г) Різниця характеристичних функцій, які відповідають цим двом літералам.

53. М'яка форма диз'юнкції двох літералів в термінах характеристичних функцій для нечіткої логіки.

а). Сума мінус різниця двох характеристичних функцій, які відповідають цим двом літералам.

б) Максимальне значення з двох характеристичних функцій, які відповідають цим двом літералам.

в) Сума характеристичних функцій, які відповідають цим двом літералам.

г) Різниця характеристичних функцій, які відповідають цим двом літералам.

54. Нечіткій нейрон «І» – це:

а) Кон'юнкція множини диз'юнкцій синаптичних ваг і зовнішніх сигналів кожного входу.

б) Диз'юнкція множини кон'юнкцій синаптичних ваг і зовнішніх сигналів кожного входу.

в) Кон'юнкція множини синаптичних ваг і зовнішніх сигналів кожного входу.

г) Диз'юнкція множини синаптичних ваг і зовнішніх сигналів кожного входу.

55. Нечіткій нейрон «АБО» – це:

а) Кон'юнкція множини диз'юнкцій синаптичних ваг і зовнішніх сигналів кожного входу.

б) Диз'юнкція множини кон'юнкція синаптичних ваг і зовнішніх сигналів кожного входу.

в) Кон'юнкція множини синаптичних ваг і зовнішніх сигналів кожного входу.

г) Диз'юнкція множини синаптичних ваг і зовнішніх сигналів кожного входу.

56. Регресія в планах для робота:

а) Перевіряє, чи відповідають поточному стану світу цілі на глибині одного кроку програми.

б) З'ясовує, чи відповідають поточному стану світу кінцеві цілі.

в) Дає оцінку поточного стану світу на підставі цілей на глибині одного кроку програми.

г) Дає оцінку поточного стану світу на підставі кінцевих цілей.

57. Маркер в планах для робота:

а) Показує поточний стан виконання або перегляду плану.

б) Не дозволяє видаляти елементи стану світу, розташовані вище, в передісторії.

в) Дозволяє застосовувати процедуру регресії.

г) Суто декоративний елемент програми.

58. Опишіть конфлікт цілей в планах робота.

а) Виконанню наступної підцілі заважає захищеність досягнутих цілей.

б) Неможливість досягнення наступної підцілі через захищеність попередньої вимагає все почати з початку.

в) Розв'язання задачі створення плану через захищеність попередньої цілі принципово неможливо.

г) Неможливість досягнення наступної підцілі через захищеність попередньої вимагає втручання користувача.

59. Опишіть реакцію STRIPS на конфлікт цілей в планах робота.

а) Не звертаючи увагу на маркер і вже досягнуті цілі, система домагається наступної цілі.

б) При помилковому значенні регресії прибирає результати попередніх і наступних кроків і проводить регресію решти програми.

в) Звертається до користувача.

г) Друкує помилку і зупиняє програму.

60. Опишіть реакцію RSTRIPS на конфлікт цілей в планах робота.

а) Не звертаючи увагу на маркер і вже досягнуті цілі, система домагається наступної цілі.

б) При помилковому значенні регресії прибирає результати попередніх і наступних кроків і проводить регресію решти програми.

в) Звертається до користувача.

г) Друкує помилку і зупиняє програму.

61. У чому Ви бачите історичне об'єднання штучних нейронних мереж і експертних систем прийняття рішень на основі логіки?

а) У створенні нейронів І та АБО.

б) У створенні нечіткої логіки.

в) У створенні мережі нейронів І і АБО, результати дії якої можна переглянути на роздруківці розв'язання.

г) Немає ніякого історичного об'єднання.

62. Що приховано від користувача в мові Пролог?

а) Процедура резолюції, що формує нові підцілі.

б) Процедура перегляду, яка відмовляється від частини рішення, яка не узгодить весь частковий граф розв'язку.

в) Процедура повернення до розгляду за допомогою зворотної дедукції іншої гілки графа.

г) Всі пункти а), б), і в) вірні.

63. Як в теорії предикатів при формуванні дерева спростування перейти від неконструктивної теореми до конструктивної?

а) Використати заперечення цільової функції.

б) Використати тавтологію, літерал якої є цільовою функцією.

в) Зробити заміну змінних в кожному літералі.

г) Це неможливо.

64. Яка система продукцій відповідає доведенню теорем?

а) Пряма.

б) Зворотна.

в) Двостороння.

г) Режим з поверненням.

65. Для встановлення практичної ймовірності діагнозу за ознаками в теорії Байєса треба знати:

а) Імовірність діагнозу при наявності ознаки.

б) Імовірність діагнозу за відсутності ознаки.

в) Апріорну ймовірність діагнозу.

г) Справедливі умови а), б), в) одночасно.

66. Якщо дерево спростування в теорії предикатів дає в кореневій вершині порожню множину, де шукати відповідь?

а) У процедурах уніфікації при формуванні дерева.

б) Ніде, це неконструктивна теорема.

в) У проміжних операціях.

г) У виборі правил.

67. Якщо в базовому наборі є однолітеральне речення, яка стратегія виявиться ефективною?

а) Стратегія пошуку в ширину.

б) Стратегія опорної множини.

в) Стратегія переваги одночленам.

г) Лінійна по входу стратегія.

68. Що подібно в теорії предикатів і в формалізмі мови ПРОЛОГ?

а) Використання резолюції.

б) Пряма продукція.

в) Спростування на основі резолюції.

г) Вони повністю подібні.

69. Чим різняться теорія предикатів і мова Пролог?

а) У Пролозі використовують імплікації, в теорії предикатів їх немає.

б) У Пролозі використовують кон'юнкцію і диз'юнкцію, в теорії предикатів – щось одне.

в) Вірні відповіді а) і б)

г) Нічим вони не відрізняються.

70. Які перетворення в теорії предикатів виконуються з фактами, правилами і цілями?

а) Всі вони перетворюються в однотипні речення.

б) У правилах зберігається імплікація.

в) Цілі приводять до нормальної диз'юнктивної форми.

г) Ніяк не перетворюються.

71. Навіщо потрібна операція приведення до диз'юнктивної нормальної форми будь-якого виразу в логіці предикатів?

а) Для явного використання повсюдно кон'юнкції.

б) Для виключення за замовчуванням диз'юнкції.

в) Для виключення імплікації.

г) Для виключення за замовчуванням кон'юнкції.

72. Наведіть приклади некомутативних систем подання знань.

а) Графи.

б) Експертні системи на основі теорії предикатів.

в) Бази даних.

г) Взагалі всі системи штучного інтелекту.

ЗАКЛЮЧЕННЯ

Люди, які обмінюються думками, повідомляючи один одному про думки третіх осіб, озираючись навколо, поступово створюють в своєму уявленні те, що слід і не слід робити, мораль, стандарти поведінки, взагалі все, чим вони керуються у своєму житті. Це і є метод формування наборів рішень на основі усвідомлення безлічі подій, явищ і думок. Таким чином формується неформалізоване знання окремої людини, знання, яке важко або навіть неможливо записати у вигляді теорій, алгоритмів і схем.

Але одночасно з цим, намагаючись зрозуміти механізми явищ, люди стали аналізувати їх деталі, шукати способи опису і прийшли до теорій, навчилися розраховувати і прогнозувати. Ці способи формування знань були побудовані на поняттях, абстракціях, дозволяли формалізувати опис явищ. Формалізація дозволила розвинути науку, методи описи явищ.

Спроби створення машин, які б могли як людина розібратися в характері процесів, спрогнозувати результати, пояснювати явища і виробляти рішення спочатку спиралися на сформульовані і формалізовані знання і такі ж формалізовані операції з ними. Це логічний висновок і розрахунки. Цей підхід був хороший для задач, що можуть обчислюватися, обчислюваних, вони дозволяють використовувати формалізовану базу даних і правила взаємодії її елементів.

Але для об’ємних, складних і масштабних завдань створення бази даних стало непідйомним завданням, яке вимагало, зокрема, великих витрат людської праці по формалізації даних і формування великої бази цих даних. Обсяги даних, які збирає людина за все життя, формуючи свою пам’ять, аналог бази даних машини, настільки великі, і настільки різноманітні, що формалізувати їх навряд чи вдасться. Крім того, ніхто і не візьметься за цю задачу. Тому підхід, заснований на формальному логічному висновку, що використовує формалізацію даних може бути привабливим лише для невеличкої кількості важливих завдань. Наприклад, таких завдань, де може знадобитися висока точність, де можна використовувати наукові теорії, де потрібно забезпечити усвідомлювану розробниками варіабельність і, звичайно, де витрати і час для створення таких програм будуть прийнятними.

Обчислювані завдання, де обмежена кількість варіантів, освоюються штучним інтелектом за допомогою машинного навчання. Програма AlphaZero за добу освоїла гри і перевершила людину. Необчислювані завдання, які не можна формально розрахувати, вирішуються інакше – обробляється і сприймається величезне число даних. В результаті виходить вже усвідомлення матеріалу за рахунок оптимізації реакцій і рефлексій, порівняння знайдених рішень з існуючими подібними рішеннями в базі даних на рівні більше-менше – саме так формується людське розуміння. Це, наприклад, реалізовано в програмі IBM Debater, де реакції програми підганяли під реакції людини (тест Тьюринга). Конкурс з такою програмою поки виграла людина (Хариш

Натарайан), але як вважали експерти за рахунок недоступною машині емоційної подачі матеріалу, а не науковою суворістю.

З розвитком нейронних мереж різного виду, з якісним збільшенням їх елементної бази, ускладненням їх архітектури, з'явилася можливість завантажити в пам'ять машини гігантську кількість сценаріїв поведінки, даних і варіантів вирішення приватних завдань, що повністю подібно до того, як людина сама усвідомлює навколишній світ. У цьому випадку як у людини, так і у машини відбувається виділення найбільш повторюваних сценаріїв поведінки і реакцій, тобто формується розуміння світу, що дозволяє більш-менш адекватно реагувати на поставлені запитання. Зрозуміло, що таке знання залишається неформалізованим, не цілком усвідомленим, інтуїтивним, заснованим на величезній кількості рефлексій, що знаходяться в інформаційній базі, це не є результатом застосування теорій, але зате корисно в житті і в поточній діяльності.

Зрозуміло також і те, що, задаючи питання такій машині і людині, можна зіткнутися з тим, що відповідь-рішення в принципі не обчислюється, не тільки тому, що не ясний спосіб розрахунку (тобто, невідомий алгоритм), але й тому, що рішення засноване на виборі найбільш прийнятного (причому оцінка цієї прийнятності також не формалізована, як перехід до чіткості при використанні нечіткої логіки) сценарію серед маси йому подібних. Неоднозначність рішень, що раніше турбувала математиків і стримувала процес збільшення числа елементів нейронних систем, через ентузіазм розробників, які погано усвідомлювали цю проблему, привела до виявлення нових можливостей. Але і машина – тут це звичайно нейронна мережа, – і людина, тим не менше цей вибір відповіді на цю необчислювану задачу зроблять, з чим суворим формалістам доведеться примиритися. Підключаючи машину до великої кількості людей, які спілкуючись з нею її навчають, можна через деякий час (тривалість якого оберно пропорційна числу учасників діалогів) можна усвідомити, що її відповіді стали мало відрізнятися від відповідей людей, які з нею спілкувалися і її тим самим навчали.

Але є проблема, на яку звернув нашу увагу відомий філософ В. В. Шкода: «такі мережі поступово приведуть до відмови від абстракцій, знання буде принципово іншим, не схожим на теперішнє». Ця обставина не може не хвилювати прихильників науки, що оперує абстракціями і символікою. Зайве захоплення таким неформалізованим штучним інтелектом може ще більше відсунути науку на периферію суспільного інтересу, а це призведе до зниження її матеріального і фінансового забезпечення.

Зрозуміло, що також, як це сталося у людині в процесі її інтелектуального розвитку, нейронні мережі з нашою допомогою поступово освоять формалізм опису: абстракції, символіку, методи і прийоми, але це трапиться трохи пізніше. А поки доведеться пережити захоплення погано формалізованими підходами «know-how», що використовують масштабні нейронні мережі, які швидко перехоплюють ініціативу у людині в різних областях діяльності.

Навчальний посібник

Куклін Володимир Михайлович

ПОДАННЯ ЗНАНЬ І ОПЕРАЦІЇ НАД НИМИ

Навчальний посібник

Коректор *О. В. Пікалова*
Комп'ютерне верстання *В. В. Савінкова*
Макет обкладинки *О. Д. Чорна*

Формат 60х84/16. Ум. друк. арк. 9,59. Наклад 300 пр. Зам. № 18/19.

Видавець і виготовлювач
Харківський національний університет імені В. Н. Каразіна,
61022, м. Харків, майдан Свободи, 4.
Свідоцтво суб'єкта видавничої справи ДК № 3367 від 13.01.2009

Видавництво ХНУ імені В. Н. Каразіна
Тел. 705-24-32