Міністерство освіти і науки України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

ОПОРНИЙ КОНСПЕКТ ЛЕКЦІЙ З КУРСУ

«Логічне програмування»

Освітньо-кваліфікаційний рівень – бакалавр
Спеціальність "Комп'ютерні системи та мережі"

Тернопіль – 2013

Тема 1: Логіка та обчислення висловлювань.

1.1 Поняття висловлювань та основні логічні операції

Висловлювання (у математичній логіці) – це речення, в якому стверджується або заперечується властивості певного об'єкту. Кожне висловлювання приймає одне з двох знань істинне (1) або хибне (0). Наприклад, Земля обертається навколо сонця. Тернопіль столиця України.

Висловлювання позначаються великими літерами A, B, C, X, Y, Z.

Логічна змінна (булава змінна), яка приймає одне з значень true, false називається константою.

Основними логічними операціями над логічними змінними є наступні:

⌉ - заперечення;

Λ – кон'юкція (логічне множення);

V – диз'юкція (логічне додавання);

→ - імплікація (якщо то);

~ - еквівалентність (↔).

Вирази, які отримуються при використанні логічних змінних і логічних зв'язків називаються логічними, або пропозиційними формулами.

Приведемо таблиці істинності:

A	B	Λ	V	→	~
I	X	X	I	X	X
I	I	I	I	I	I
X	I	X	I	I	X
X	X	X	X	I	I

A називається посилкою (антецедент);

B називається заключенням (консеквент);

A~B (A тоді і тільки тоді, коли B).

Поряд дій при обчисленні по формули такий:

- заперечення;
- кон'юкція;
- диз'юкція;
- імплікація і еквівалентність.

(Виконання в порядку слідування, але при необхідності можна змінити дану пріоритетність за допомогою дужок).

2. Основні аксіоми та закони алгебри логіки.

Аксіоми алгебри логіки

1 V A = 1;

0 Λ A = 0;

A V A = A;

A Λ A = A;

0 V A = A;

1 Λ A = A;
A V ⎤A = 1;
A Λ ⎤A = 0.

Аксіома подвійного заперечення
⎤⎤(A) = A.

Закони комутативності
A V B = B V A;
A Λ B = B Λ A.

Закони асоціативності
A V B V C = A V (B V C);
A Λ B Λ C = A Λ (B Λ C).

Закони дистрибутивності
A V (B V C) = (A Λ B) V (A Λ C);
A Λ (B Λ C) = (A V B) Λ (A V C);

Закони де Моргана
A V B = B Λ A; (A B) – стрілка Пірса; !!!
A Λ B = B V A. (A B) – штрих Шефера. !!!

Закони поглинання
A V A V B = A;
A Λ A Λ B = A.

3. Нормальні форми. Тотожні формули.

Бувають диз'юктивна та кон'юктивна нормальні форми (ДНФ та КНФ). Формула **алгебри висловлювань** задано диз'юктивно - кон'юктивній формі. ДНФ тоді, якщо вона представлена у вигляді диз'юкції кон'юкцій елементарних висловлювань та їх заперечень.

Формула задана в кон'юкції диз'юкцій (КНФ), якщо вона представлена у вигляді в кон'юкцій диз'юктивних змінних та їх заперечень. Наприклад,

ДНФ (диз'юкції кон'юкцій та їх заперечень)
⎤Z V XY V XYZ
КНФ (кон'юкції диз'юкцій та їх заперечень)
(⎤XY V Z)(⎤X V XY⎤Z)
Приведемо алгоритм приведення будь-яких формул до ДНФ і КНФ.

- на першому етапі позбуваються від зв'язок імплікації та еквівалентності

імплікація $X \to Y = ⎤X \vee Y$;
еквівалентність X ~ Y = (⎤X V Y) (X V ⎤Y).

- приводимо заперечення до незалежних до незалежних змінних. Розглянемо заперечення до кон'юкції і диз'юкції двох змінних (до добутку XY).

¬(XY) = ¬X V ¬Y;

¬(X V Y) = ¬X Λ ¬Y.

- розкриваємо дужки згідно першого закону (розподільного) для переведення до ДНФ або навпаки перетворюємо суми в добутки по другому розподільному закону для приведення до КНФ.

(X V Y) Z = XZ V YZ;

XY V Z = (X V Z) (Y V Z).

- при наявності двох заперечень кожен раз від них позбавляємося. Наприклад,

(z → t) (x V z) V ¬(x(y V z))t. Привести даний вираз використавши правила алгебри логічних змінних до ДНФ.

(¬z V t) (x V z) V ¬(x(y V z))t = (¬z V t) (x V z) V (¬x V ¬(y V z))t = (¬z V t) (x V z) V (¬x V ¬y Λ ¬z)t = (¬zx V ¬zz V tx V tz) V (¬x V (¬y Λ ¬z))t = (¬zx V ¬zz V tx V tz) V (¬xt V ¬yt ¬zt) = ¬zx V tx V tz V ¬xt V ¬y¬zt = ¬zx V t (¬x V x) V tz V ¬y¬zt = ¬zx V t V tz V ¬y¬zt = ¬zx V t (1 V z V ¬y¬z) = ¬zx V t.

4. Досконалі нормальні форми

ДНФ є: ДДНФ і ДКНФ.

ДДНФ – алгебри висловлювань називається така диз'юктивна нормально форма (ДНФ), яка:

- всі доданки вміщують співмножником всі змінні без заперечень або з запереченням, але не разом;

X ¬Y V ¬X ¬Y

- відсутні повторення складових та співмножників.

ДКНФ називається така кон'юктивна нормальна форма (КНФ), в якій:

- кожний співмножник вміщує складову без заперечень, або з запереченнями, але не разом;

- відсутні повторення співмножників та складових.

(X V ¬Y V Z) (¬X V ¬Y V Z) (X V Y V Z)

5. Алгоритм приведення до ДДНФ

а) Приведення до ДДНФ.

Алгоритм представлений таким чином:

6. спочатку приводимо до ДНФ;

7. складові, яких співмножниками входять не всі змінні помножують на одиниці, які представлені у вигляді диз'юкцій кожної недостатньої змінної з її запереченням;

8. розкриваємо дужки згідно першого розподільного законі. Приклад,

X V ¬YZ = X (Y V ¬Y)(Z V ¬Z) V ¬YZ(X V ¬X) = XYZ V XY¬Z V X¬YZ V X¬Y¬Z V X¬YZ V ¬X¬YZ = XYZ V XY¬Z V X¬YZ V X¬YZ V X¬Y¬Z V ¬X¬YZ.

б) Приведення до ДКНФ.

Алгоритм представлений таким чином:

9. спочатку приводимо до КНФ;

10.до співмножників, які вміщують не всі змінні додають нулі, які представлені у вигляді кон'юкцій змінних яких не вистачає з їх запереченням;

11.згідно другого розподільного закону приводять всі співмножники до сум першого ступені виключаючи повторення співмножників. Приклад,

(X V ⌉Y)(X V Z) = (X V ⌉Y V Z⌉Z)(X V Y⌉Y V Z) = (X V ⌉Y⌉Z)(X V Y V Z)(X V ⌉Y V Z).

6. Пропозиціональні формули

Висловлювання – це твердження відносно якого можна сказати істинне воно чи хибне. Із простих висловлювань за допомогою логічних операцій будуються більш складні. В обчисленні висловлювань розглядаються логічні операції ⌉ Λ V → ~. Ці логічні операції називаються пропозиціональними логічними зв'язками.

Логічні змінні, які позначаються латинськими літерами називаються пропозиціональними змінними.

Алфавіт мови: обчислення висловлювань, яке складається із пропозиціональних змінних, логічних зв'язків і дужок.

Правильно побудувати вирази мови обчислення висловлювань називаються пропозиціональними формулами. Всі пропозиціональні змінні є формулами.

Якщо і ,є формулами, то формули є також вираз

Пропозиціональна формула, яка істинна при будь-яких значеннях змінних, які входять до неї, тобто істинна при будь-яких інтеграціях називаються тавтологією.

Пропозиціональна формула, яка хибна при будь-яких значеннях змінних називається протиріччям.

Пропозиціональна формула, яка приймає істинні значення при деяких інтерпретаціях називається виконуваною.

Тавтологія – P V ⌉P

Протиріччя – P Λ ⌉P

P Λ G - протиріччя

7. Логічний наслідок і логічний вивід

Виявлення того факту, що із множини висловлювань (формул обчислень) логічно слідує інше висловлювання є основною задачею обчислень. Говорять, що із множини формул $(F_1, F_2,\ldots, F_n) \to F$, якщо $F_1 \Lambda F_2 \Lambda \ldots \Lambda F_n$ є тавтологією. Наприклад,

8. Якщо будувати протиатомні сховища (A), то інші держави будуть відчувати себе в небезпеці (B) і наш народ отримає хибне представлення про свою безпеку (C).

9. Якщо інші країни будуть відчувати себе в небезпеці (B), то вони можуть почати примітивну війну (D).

10. Якщо наш народ отримає хибне представлення про свою безпеку (C), то він ослабить свої зусилля, які направленні на збереження миру (E).

11. Якщо не будувати протиатомні сховища ($\neg$A), то ми ризикуємо мати колосальні втрати у випадку війни (F).

12. Відповідно або інші країни можуть почати примітивну війну (D) і наш народ ослабить зусилля (E) або ми ризикуємо мати колосальні втрати у випадку війни (F).

Складемо пропозиціональну формулу, яка відповідає кожному із висловлювань посилок і висловлювань наслідку (5).

1) (A $\rightarrow$ (B & C));

2) (E $\rightarrow$ D);

3) (C $\rightarrow$ E);

4) ($\neg$A $\rightarrow$ F);

5) (D & E) V F).

Для того, щоб довести, що з твердження 1, 2, 3, 4 випливає 5 необхідно доказати, що формулу ((A $\rightarrow$ (B & C) & (E $\rightarrow$ D) & (C $\rightarrow$ E) & ($\overline{A}$ $\rightarrow$ F) $\Rightarrow$ (D & E) V F).

1) Замінимо в всіх висловлюваннях імплікацію: (A $\rightarrow$ (B & C) $\equiv$ $\overline{A}$ V (B & C)

$\overline{(\overline{A} \text{ V } (B \& C) \& (\overline{E} \text{ V } D) \& (\overline{C} \text{ V } E) \& (A \text{ V } F) \text{ V } (D \& E) \text{ V } F)}$.

2) Використаємо такі еквівалентності: $\overline{(P \& Q)} = \overline{P} \text{ V } \overline{Q}$; $\overline{P \text{ V } Q} = P \& Q$.

((A & ($\overline{B}$ V$\overline{C}$) V (E & $\overline{D}$) V (C & $\overline{E}$) V ($\overline{A}$ & $\overline{F}$) V (D & E) V F).

3) P & (Q V R) $\equiv$ (P & Q) V (P & R)

(A & $\overline{B}$) V (A & $\overline{C}$) V (E & $\overline{D}$) V (C & $\overline{E}$) V ($\overline{A}$ & $\overline{F}$) V (D & E) V F

Допустимо, що дана формула хибна, тобто всі диз'юктивні числа повинні бути хибними і кінцеве F повинно бути хибними. Але тоді A повинно бути істинне, бо $\overline{A}$ & $\overline{F}$ повинно бути хибним. Тоді B повинно бути істинне, бо A & $\overline{B}$ повинно дорівнювати 0.

Аналогічно D має = 1.

Висловлювання E є хибним, бо D & E = 0.

C = 0., бо C & $\overline{E}$ = 0.

В результаті отримаємо протиріччя, бо A має дорівнювати 0 і дорівнювати 1.

Дана формула ні при якій інтерпретації не може бути хибною. Отже вона є тавтологією. Отже висловлювання 5 є істинним.

8. Метод резолюції

Інше міркування, яке дає ключ до розуміння методу логічного виводу, називається метод резолюції. Для того, щоб доказати, що (F_1, F_2,…, F_n) $\rightarrow$ F необхідно доказати, що формула F_1 & F_2 &…& F_n $\rightarrow$ F є тавтологією. Для цього

достатньо довести, що її заперечення є протиріччям повинна бути така формула $\overline{(F1 \& F2 \& \ldots \& Fn)} \vee F$

Тобто протирічною є множина формул, яка складається із посилок $F_1, F_2, \ldots, F_n$ і F.

Робоча програма

Тема 1. Вступ до логіки

Базові поняття логіки.

Реальне поняття дає опис певних об'єктів. О. р. противопоставляется определению номинальному, выражающему требование (предписание, норму), каким должны быть рассматриваемые объекты.

Особливості математичної логіки.

Зміст поняття і основні типи визначення понять.
Класифікація міркувань.

Тема 2. Види міркувань

Дедуктивні міркування.
Складні міркування.
Безпосередній висновок.
Опосередковане заключення.
Прості силогізми.
Складні висновки.
Індуктивні міркування.
Заключення по аналогії.

Тема 3. Дедуктивні системи

Логічний вивід і логічне програмування. Співвідношення між змістовними і формальними теоріями. Формалізація мислення і формальні системи. Аксіоматичний метод і формальні теорії.

Тема 4. Логіка і обчислення висловлювань.

Логічні операції над логічними змінними. Алгебра логіки. Пропозиціональні формули. Логічний наслідок і логічний вивід. Метод резолюцій. Обчислення висловлювань. Логічний наслідок і формальний вивід.

Лекція 2.

Обчислення предикатів

2.1 Відношення і предикат

Не всі висловлювання і не всі логічні міркування можуть бути описані на мові висловлювань. В деяких випадках висловлювання відносяться до властивостей об'єктів або відношення між об'єктами. Крім цього необхідно стверджувати, що будь-які або деякі об'єкти мають певні властивості і знаходяться в певних відношеннях.

Предикат виступає індикатором відношення.

Наприклад, введемо відношення порядку на множині дійсних чисел – «>», то якщо 2>3 предикат значення «0».

Нехай задана множина *ПМ*. Розглядаємо декартовий добуток.

$ПМ^n = ПМ_1 \times ПМ_2 \times \ldots \times ПМ_n$

Ця множина всіх впорядкованих послідовностей із n елементів $(x_1, x_2, \ldots, x_n)$ таких що $x_i \in ПМ$.

n – місне відношення множини *ПМ* називається підмножиною $ПR \subset ПМ^n$

Відношення «Більше» це підмножина множин всіх впорядкованих пар чисел.

Зауваження

Іноді розглядаємо множину

$$ПМ^n = ПМ_1 \times ПМ_2 \times \ldots \times ПМ_n,$$

де $ПМ_1, ПМ_2, \ldots, ПМ_n$ - мають різну природу.

Наприклад, пряма і площина можуть належати різним множинам, але вони можуть знаходитись у відношенні паралельно.

Двохмісні відношення називаються бінарними.

Наприклад, відношення >, <, = - бінарні відношення на множинні дійсних чисел.

Одномірні відношення (унарні) називаються **ознаками**, або **властивостями**.

Властивості – належність до деякої множини або класу.

n – місний предикат $P(x_1, x_2, \ldots, x_n)$ –це функція такого вигляду $P: ПМ^n \rightarrow ПВ$

$ПВ \in \{0, 1\}$

2.2 Квантори

Нехай заданий предикат $P(x_1, x_2, \ldots, x_n)$ на множині $ПМ^n$. $x_1, x_2, \ldots, x_n$ називається вільними змінними.

Нехай заданий вираз

$\forall X_i P(x_1, x_2, \ldots, x_n), x \in ПМ^n$.

$\forall$ - квантор загальності.

Для всіх X_i, які належать до множини $ПМ^n$ предикат P є істинним.

$\exists X_i P(x_1, x_2, \ldots, x_n)$, $X_i \in ПМ^n$.

$\exists$ - квантор існування

Існує таке X_i , що предикат P набуває значення істина.

Якщо множина $ПМ^n$ є скінченна це означає, що є скінченна кількість незалежних змінних і якщо вона скінчена, то справедливий такий вираз

$\forall X P(x_1, x_2, \ldots, x_n) = P(x_1) \& P(x_2) \& \ldots P(x_n)$.

$\exists X P(x_1, x_2, \ldots, x_n) = P(x_1) \& P(x_2) \& \ldots P(x_n)$.

Перехід від виразів

$P(x_1, x_2, \ldots, x_n) \rightarrow \forall X_i (P(x_1, x_2, \ldots, x_n))$.

$P(x_1, x_2, \ldots, x_n) \rightarrow \exists X_i (P(x_1, x_2, \ldots, x_n))$.

Називається навішування квантору на змінну x_i . При цьому число вільних змінних зменшується на 1.

Вираз, який не має вільних змінних називається **висловлюванням**. Нехай заданий P(x) це означає, що x ділиться на 5 без остачі тоді висловлювання, що існує таке x, що цей предикат буде справедливий $\exists X P(x)$.

Це означає, що змінна x залежить від множини $ПМ^n$ на якій вона розглядається. Якщо $ПМ^n$ - множина цілих чисел, то значення цього висловлювання є істинне. Але якщо x Є (0,5), то це висловлювання хибне. Якщо множина утворюється арифметичною прогресією, яка задається різницею d = 5, тоді справедливий такий вираз ∀ X P (x).

2.3 Мова логіки предикатів

Розглянемо таке твердження

«Друг мого друга – мій друг». Задамо відношення R (x, y). Дехто x є другом, а R задає відношення «друзі». Будемо вважати, що дане відношення є комутативним R (x, y) = R (y, x). То щоб x і y були друзями Василя:

R (x, Василь)

R (y, Василь).

Використаємо логічні зв'язки (Λ та →) і запишемо:

R (x, y) Λ R (y, Василь) → R (x, Василь).

Для того, щоб показати, що x та y не коректні індивідууми, а можуть приймати біуь-які значення, то використаємо ∀.

∀ x ∀ y (R (x, y) Λ R (y, Василь) → R (x, Василь). (1)

Це буде означати, що для любих двох друзів, якщо другий із них є другом Василя, то і перший є його другом. Якщо вважати, що x є другом y, то і y є другом x: ∀ x ∀ y (R (y, x) → R (x, y)).

Звідси повинно випливати (1)

Але це не обов'язково повинно виконуватися для любих друзів один з яких дружить з Василем, але може бути знайдена пара друзів, які володіють такою властивістю. В такому випадку використовуємо квантор ∃

∃ x ∃ y (R (x, y) Λ R (y, Василь) → R (x, Василь).

Це означає, що якщо знайдуться двоє друзів, другий із яких дружить з Василем, то з ним дружить і перший.

∀ x ∀ y ∀ z (R (y, x) Λ R (y, z)) → R (x, z).

Це твердження в загальному випадку є хибним.

Твердження полягає в тому, що для будь-якої пари x, y знайдеться z, що якщо вона дружить з другим із пари, то z буде дружити із I

∀ x ∀ y ∃ z (R (y, x) Λ R (y, z)) → R (x, z).

Yjdt

Структура логіки предикатів (ЛП) - алфавіт, правила побудови формул із символів алфавіту, правила дедуктивного виведення з аксіом нових формул (доведення теорем), правила інтерпретації.

Мова логіки предикатів - це система символів, що створюють алфавіт. До нього належать символи, введені в логіці висловлювань, і нові символи, які позначають терміни, введені в логіці предикатів.

Алфавіт:

- символи, що позначають елементарні (прості) висловлювання (формули, формальні вирази) P, Q;
- символи, які позначають істиннісні значення висловлювань - "/", V;
- символи, що позначають предметні (індивідні) змінні - x, y, г,... п (множинність предметних змінних може бути безмежною);
- символи, які позначають предметні константи (постійні) - а, в, с, а*,... п;
- символи, що позначають д-місні предикати - Р, і?, РҐ Р Р о
- символи, котрі позначають предметні функції - Рі9 Р2, Р (верхній індекс позначає місність предметних функцій, а нижній визначає їх кількість);
- символ, що позначає терм г;
- символ, який позначає предикатну змінну X;
- символ, що позначає відношення предикації <=;
- символ, який позначає квантор загальності V;
- символ, що позначає квантор існування 3;
- символи, котрі позначають пропозиційні зв'язки (логічні сполучники, логічні постійні): кон'юнкція л, диз'юнкція v, імплікація ->, еквівалентність =, заперечення ->;
- технічні символи: (- ліва дужка;) - права дужка.

2.4 Синтаксис мови обчислень предикатів

Для того, щоб описати мову формальної теорії необхідно задати множину її символів (словник) і правила побудови виразів (формул) мови.

Конструкції мови – це логічні зв’язки, квантори, предметні змінні, предметні константи, функціональні символи, предикатні символи, терми, висловлювання, атомарні формули (елементарні) і формули.

Так як мова висловлювання частковим випадком мови предикатів, то висловлювання є формулами мови обчислення предикатів.

Символи мови:

1. логічні зв’язки $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\equiv$.
2. Квантори $\forall$, $\exists$.
3. Предметні змінні (літери латинського алфавіту, малі)
4. Предметні константи (літери латинського алфавіту, великі)
5. Функціональні символи f, f_1, g …крім цього можуть використовуватись sin, cos, ln, …..
6. Предикатні символи P, Q, R….
7. Терми $f^n(t_1, \ldots, t_n)$ – це константи або предметні змінні, або вирази такого типу f^n, n – місний функціональний символ $t_1, \ldots, t_n$ – терміни. Якщо n=0 , то термін називається предметною або функціональною константою.
8. Атомарні (елементарні) формули – вирази виду: $P(t_1, \ldots, t_n)$, де P – парний предикатний символ, а t_n – термін. Якщо n=0, то сумарна формула співпадає із пропозиціональною змінною.
9. Формули обчислення предикатів – це вирази: $\neg F$, $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, $(F_1 \rightarrow F_2)$, $(F_1 \equiv F_2)$, де F_1, F_2 – формули. $\forall X_i F$, $\exists X_i F$, X_i – предметна змінна.
10. Літерами – називаються формулами такого виду P, $\neg$P, де P – атомарна формула.

Лекція 3.

Тема 3. Моделі представлення знань у ШІ

Знання як інформаційна одиниця

Поняття знань і підходи до їх подання

Знання – це відображення об'єктивних властивостей та зв'язків матеріального світу.

Знання – це систематизована інформація, яка може певним чином поповнюватися і на основі якої отримують нову інформацію, нові знання.

На сучасному етапі можна виділити таку парадигму (система ключових принципів – символьний підхід) вербально-дедуктивну, яка виражається через такі чинники:

- будь-яка інформаційну одиниця додається вербально у вигляді явно сформульованих тверджень та фактів.

Основним механізмом отримання нової інформації – висновок від загального до часткового. Вербально-дедуктивне додання є неповним:

- Дедуктивний метод не є єдино можливим;
- Далеко не всі знання є вербальними.

Тому ще існує конекціоністський підхід. Це підхід на основі поля знань.

Однорідне поле знань це сукупність простих однорідних елементів, які обмінюються між собою інформацією. Нові знання з'являються на основі певних процедур, які відзначаються над полем знань. Знання програми зберігаються вербально, при умові якщо за певний час система дозволяє отримати вербальний опис понять. Знання зберігає не вербально, якщо застосування фіксованого набору процедур дає змогу отримати нове поняття або факт без їх вербального опису.

Знання в інтелектуальній системі подаються **у вигляді**: <F, R, P>

F – сукупність фактів (зберігаються в явному вигляді);

R – сукупність правил виводу;

P – сукупність процедур, які визначають яким чином застосовувати правила.

База знань – це сукупність усіх знань у пам'яті системи БЗ поділяються на екстепсіональну і інтепсіональну.

Екстенсіональна – це сукупність усіх явних фактів.

Інтенсіональна – це сукупність усіх правил виводу та процедур за допомогою яких з існуючих фактів можна вивести нові твердження. Наприклад:

Ф1 – заєць їсть траву;

Ф2 – вовк їсть м'ясо;

Ф3 – заєць є м'ясо.

П1 – якщо А є м'ясо, а В їсть м'ясо, то В їсть А.

П2 – якщо В їсть м'ясо, то В є хижаком.

П3 – якщо В їсть А, то А є жертвою.

П4 – якщо хижак вмирає, то жертва швидко розмножується.

П5 – якщо жертва вимирає, то хижак вимирає.

Властивості знань.

- Внутрішня інтерпритованість. Кожна інформаційна одиниця має певне унікальне ім’я.
- Структурованість. Знання повинні мати гнучку структуру. Одні інформаційна одиниці мають включатися до інших.
- Зв’язність. Між інформаційними одиницями повинні встановлюватися різні типи зв’язків (причинно наслідкових, просторових).
- Сематична метрика. Між інформаційна одиницею повинен встановлюватися зв’язок, який характеризує інформаційна близькість.
- Активність. Історично склалося, що дані є пасивні, а команди – активні. Виконання команд в інтелектуальній системі повинно ініціюватися текучим станом БЗ.

На даний час не існує жодної системи в світі, яка задовольняє всі властивості.

Типи знань

- Базові елементи, об’єкти реального світу.
- Твердження і визначення. Вони базуються на тазових елементах і наперед розглядаються як достовірні.
- Концепції. Це узагальнення базових об’єктів.
- Відношення. Елементарні властивості базових елементів.
- Теореми. Теореми не представляють собою ніякої користі без експертних правил, їх використання.
- Алгоритми. Необхідні для розв’язку визначених задач.
- Стратегії та евристики. Це вродженні або набуті правила поведінки.
- Метазнання. Це знання про знання. Метазнання визначають довір’я до знань.

Моделі представлення знань.

Для маніпулювання знаннями реального світу за допомогою комп’ютера використовують їх моделювання. При розробці моделей представлення враховують такі фактори, як однорідність, простота розуміння. Однорідне представлення приводить до спрощення управління знаннями і логічними висновками. За існуючою класифікацією Класичними моделями є:

- продукційна;
- фреймова;
- модель семантичної мережі;
- логічна модель.

5. Представлення знань у вигляді правил (продукційна модель)

У продукційній моделі знання представлені сукупністю правил вигляду "ЯКЩО-ТО". ЕС з базами знань, заснованими на цій моделі, називаються продукційними системами. Кожне продукційне правило складається з двох частин. Перша посилка правила (антецедент) складається з елементних пропозицій, сполучених логічними зв'язками І, АБО, НЕ і іншими. Друга частина називається висновком (консеквент) і включає в себе одне або декілька пропозицій, які утворять рішення, що видається правилом або підлягаючу виконанню дію. Приклад продукційного правила:

ЯКЩО двигун не заводиться
І стартер двигуна не працює, Антецедент

ТО несправна система електроживлення стартера автомобіля
Консеквент

У цьому випадку є дві пропозиції (правила), сполучені в антецеденті логічною зв'язкою "І". Кожне правило містить атрибути і значення.

Атрибут	**Значення**
Двигун	Не заводиться
Стартер двигуна	Не працює
Система електроживлення стартера	Несправна

Правило спрацьовує, якщо факти з робочої пам'яті при зіставленні співпали з його антецедентом, після чого правило вважається відпрацьованим.

Продукційні системи діляться на два протилежних типи з прямими висновками і зворотними. Типовим представником першого типу є MYCIN, призначена для медичної діагностики, ЕС ORS, що використовується для рішення задач проектування, відноситься до другого типу [7,8].

Системи продукцій з прямими висновками мають три складових компоненти:

- базу правил, що складається з набору продукцій (правил висновку);
- базу даних, що містить безліч фактів;
- інтерпретатор для логічного виведення на основі даних і знань.

Бази правил і даних утворюють базу знань (БЗ), а інтерпретатор відповідає механізму логічного висновку.

Прямий висновок часто називають висновком, керованим даними або висновком, керовані антецедентам, оскільки висновок відшукують за відомими фактами (гіпотезам), з яких він слідує, Висновок реалізовується циклами "розуміння виконання", причому в кожному циклі частина вибраного правила,

що виконується оновлює БЗ. У результаті БЗ, що міститься перетворюється від вихідного до цільового.

Продукційні системи з прямим висновком непридатні для рішення крупномасштабних задач, оскільки описане циклічне зіставлення істотно сповільнює швидкість висновку із збільшенням числа правил.

У системах продукцій із зворотними висновками за допомогою правил будують дерево І - АБО, зв'язуюче в єдине ціле факти і висновки. Логічним висновком тут є оцінка дерева на основі фактів в БЗ. При цьому спочатку висувають деяку гіпотезу, а потім механізм висновку в процесі роботи повертають назад, переходячи до пошуку фактів, підтверджуючих цю гіпотезу. Якщо вона виявилася правильною, то вибирають слідуючу гіпотезу деталізуючу першу і що є по відношенню до неї підцілью. Далі відшукують факти, підтверджуючі виконання підлеглої гіпотези. Такий зворотний пошук називають керованими цілями, або керованим консеквентами, і застосовують у випадках, коли цілі відомі і нечисленні. Перевага зворотних висновків в тому, що оцінюються лише частини дерева, які мають відношення до висновку. Однак якщо твердження або заперечення неможливе те породження дерева позбавлене значення.

Існують ще і двонаправлені висновки, в яких спочатук оцінюють невеликий об'єм отриманих даних і вибирають гіпотезу прямим висновком, а потім для зворотного висновку запитують дані, необхідні для прийняття рішення про виконання даної гіпотези. На основі використання двонаправлених висновків можна реалізувати більш могутню і гнучку ЗОСПР.

Переваги продукційних моделей представлення знань:

- простота конструювання і розуміння окремих правил їх поповнення і модифікації, а також механізму логічного висновку.
- одноманітність структури, придатної для побудови ЕС різних предметних областей;
- гнучкість родово-видової ієрархії понять, виражена в неясній формі (зміна ієрархії досягається шляхом зміни правила);
- модульність.

Недоліки продукційних моделей представлення знань:

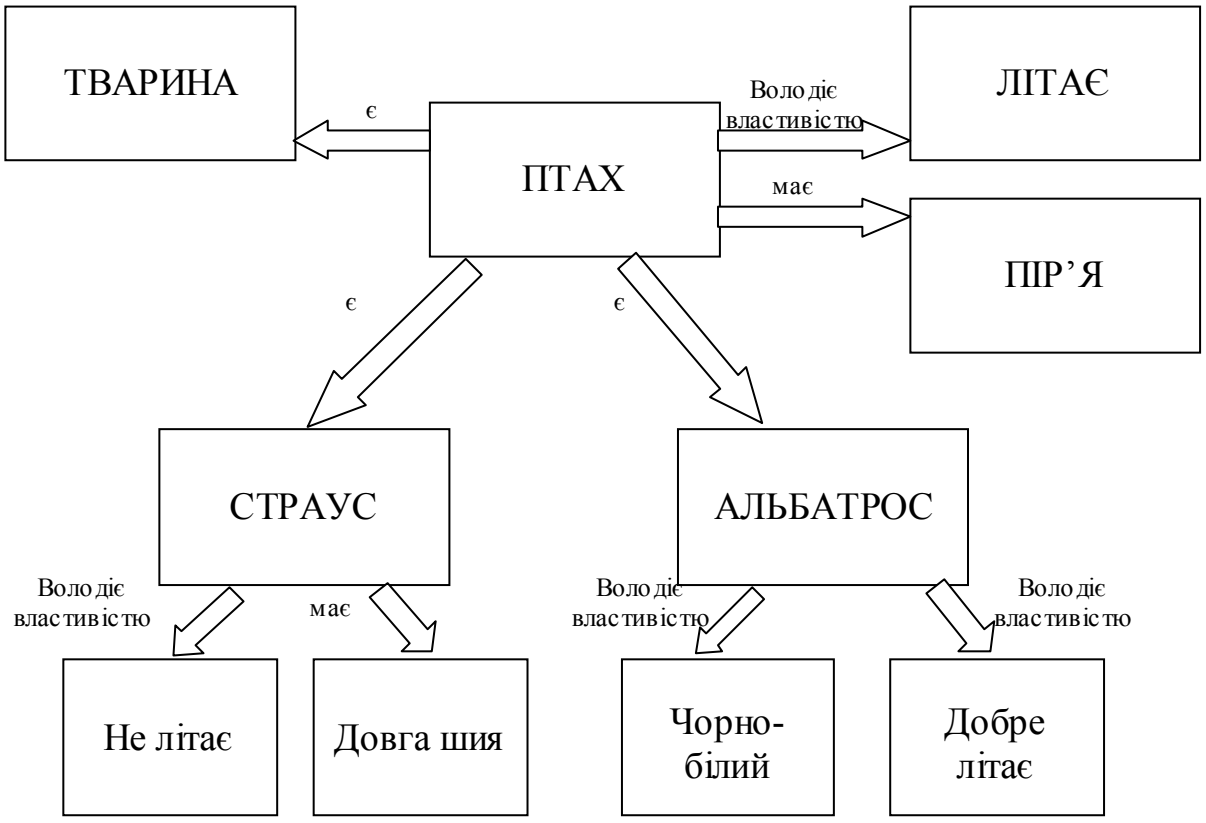
- труднощі управління продукціним висновком через складність оцінки цілісного образу знань і відсутність гнучкості в логічному висновку
- низька ефективність обробки знань через тривалу непродуктивну перевірку застосовності правил;
- складність представлення родово-видовий ієрархії понять в явній формі.

Більшість існуючих комерційних експертних систем продукційні .

Виведення висновку

Тема 4. Модель семантичної мережі.

На думку фахівців, семантична мережа це найбільш загальний спосіб представлення знань. Модель семантичної мережі відображає сукупність об'єктів предметної області і відносин між ними. При цьому об'єктам відповідають вершини (або вузли) мережі, а відносинам з'єднуючі їх дуги. Як об'єкти можуть виступати події, процеси, дії узагальнені поняття або властивості суб'єктів, що використовуються для. описи класів останніх. Вершини мережі сполучаються дугою, якщо відповідні об'єкти предметної області знаходяться в якому-небудь відношенні наприклад; <бути> (бути елементом класу), <мати>, <володіти властивістю> (мати значення властивості) і інші. На мал. 4.1 приведений приклад моделі семантичної мережі



Мал 4.1

Як і в моделі заснованій на фреймах, в семантичній мережі можуть бути відображені родово-видові відношення, які. дозволяють реалізувати успадкування властивостей від об'єктів-батьків. Тому семантичні мережі придбавають всі Переваги і Недостатки представлення знань фреймовими моделями. Основна нестача семантичних мереж збільшення часу пошуку зростанням розмірів мережі складність обробки виключень

Семантичні мережі

Термін семантична означає “значеннєва”, а сама семантика — це наука, що установлює відносини між символами й об'єктами, що вони позначають, тобто наука, що визначає зміст знаків.

Семантична мережа — це орієнтований граф, вершини якого — поняття, а дуги — відносини між ними.

Як поняття звичайно виступають абстрактні чи конкретні об'єкти, а відносини — це зв'язку типу: “це” (“АКО - A-Kind-Of”, “is”), “має частиною” (“has part”), “належить”, “любить”. Характерною рисою семантичних мереж є обов'язкова наявність трьох типів відносин:

- клас - елемент класу (квітка - троянда);
- властивість - значення (колір - жовтий);
- приклад елемента класу (троянда - чайна).

Можна запропонувати кілька класифікацій семантичних мереж, зв'язаних з типами відносин між поняттями.

По кількості типів відносин:

- Однорідні (з єдиним типом відносин).
- Неоднорідні (з різними типами відносин).

По типах відносин:

- Бінарні (у який відносини зв'язують два об'єкти).
- N-арні (у який є спеціальні відмови, що зв'язують більш двох понять).

Найбільше часто в семантичних мережах використовуються наступні відносини:

- зв'язку типу “частина — ціле” (“клас — підкласи, елементом-безліччю, і т.п.);
- функціональні зв'язки (обумовлені звичайно дієсловами “робить”, “впливає”...);
- кількісні (більше, менше, дорівнює...);
- просторові (далеко від, близько від, за, під, над...);
- тимчасові (раніш, пізніше, протягом...);
- атрибутивні зв'язки (мати властивість, мати значення);
- логічні зв'язки (И, ЧИ, НЕ);
- лінгвістичні зв'язки й ін.

Проблема **пошуку рішення** в базі знань типу семантичної мережі зводиться до задачі пошуку фрагмента мережі, що відповідає деякої підмережі, що відбиває поставлений запит до бази.

Приклад 1.3

На мал. 4.2 зображена семантична мережа. Як вершини отут виступають поняття “людин”, “т. Іванов”, “Волга”, “автомобіль”, “вид транспорту” і “двигун”.

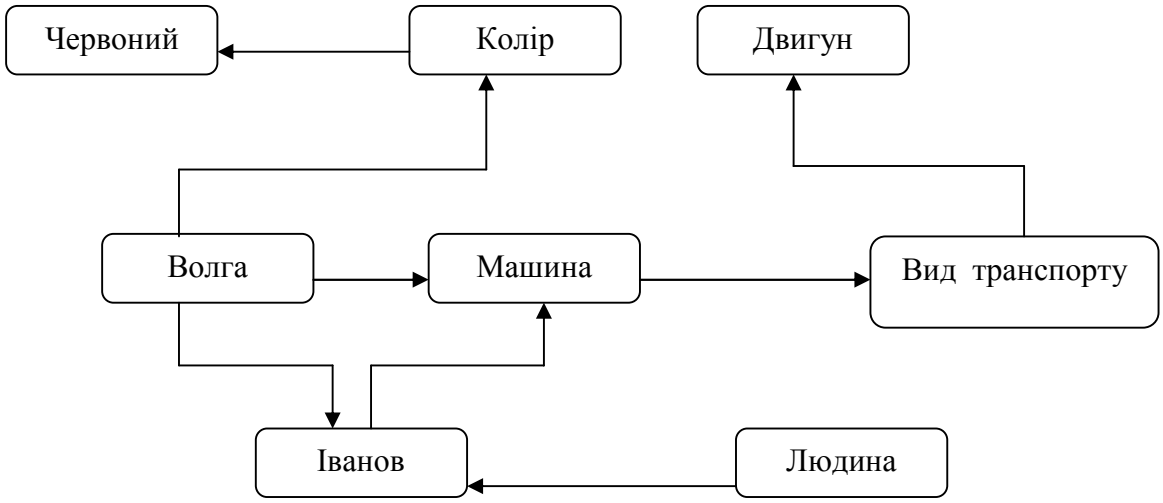


Рис. 4.2. Семантична мережа

Дана модель представлення знань була запропонована американським психологом Куиллианом. Основною її перевагою є те, що вона більш інших відповідає сучасним представленням про організацію довгострокової пам'яті людини [Скрегг, 1983].

Недоліком цієї моделі є складність організації процедури пошуку висновку на семантичній мережі.

Для реалізації семантичних мереж існують спеціальні мережні мови, наприклад NET [Цейтин, 1985], мова реалізації систем SIMER+MIR[Осипов, 1997] і ін. Широко відомі експертні системи, що використовують семантичні мережі як мову представлення знань — PROSPECTOR, CASNET, TORUS [Хейес-Рот і ін.,

Виведення висновку

Застосування

http://www.slideshare.net/DickAhles/semantic-networks-for-business

Тема 5. Фреймова і логічна моделі

Фреймова модель представлення знань заснована на теорії фреймів Марвіна Мінського [38, 39], що описує абстрактну психологічну модель пам'яті людини і його свідомість. Фреймове представлення знань є альтернативним по відношенню до продукційному, оскільки дозволяє в явній формі зберігати, родово-видовую ієрархію понять в базі знань.

Фреймом називається структура для опису стереотипної ситуації, що складається з характеристик цієї ситуації та їх значень. Характеристики

називаються слотами, а значення заповнювачами слотів. Слот може містити конкретне значення, одну або декілька продукцій (евристик), за допомогою яких це значення визначається ім'я процедури обчислення його по заданому алгоритму, а також декілька значення. Іноді слот включає в себе компонент, званий фасетом, який задає діапазон або перелік його всіляких значень. Сукупність фреймів, що моделює яку-небудь предметну область, являє собою ієрархічну структуру в яку фреймі сполучаються за допомогою родово-видових зв'язків. На верхньому рівні ієрархії знаходиться фрейм, що містить найбільш загальну інформацію, істинну для всіх інших фреймов. Фрейм, володіють здатністю наслідувати знання характеристик своїх батьків з більш високим рівнем ієрархії.

Розрізнюють статичні і динамічні системи фреймів. У статичних системах фрейми не можуть змінюватися, а в динамічних це допустиме.

Слот має наступну структуру:

$$\{<\text{ім'я слота}>;<f_1><v_1>;..;<f_n><v_n>;<q_1>;..;<q_m>\}, \qquad (2.1)$$

де f_i $(1 \le i \le n)$ – імена атрибутів, характерних для даного слоту;
v_i – значення цих атрибутів або множини значень;
q. $(1 \le j \le m.)$ різні посилання на інші слоти.

Приклад 2.1.

{ <АВТОМОБІЛЬ-ТАХ1> < НОМЕРНИЙ ЗНАК > < А 26-50 ХА, К 08-20 ХА>,< СТАН>< СПРАВНИЙ, В РЕМОНТІ>; <ТИП> < ГАЗ-24, ГАЗ-24>; <КОЛОНА >; < ГАРАЖ>; <ВОДІЙ>}.

Слот, що ілюструється має ім'я АВТОМОБІЛЬ-ТАХ1 і містить знання про два автомобілі однакового типу з вказаними в прикладі "номерами" один з яких справний, а другий в ремонті. Деякі інші відомості про ці об'єкти можна знайти в слотах про іменами КОЛОНА, ГАРАЖ і ВОДІЙ. Наприклад, слот ВОДІЙ може мати вигляд

{< ВОДІЙ >;<ПІП> < ІВАНОВ ІВАН ПЕТРОВИЧ, КРАВЧУК СЕМЕН ІВАНОВИЧ >; < НОМЕРНИЙ ЗНАК > < А 26-50 ХА, К08-20 ХА>)

При переході від слота АВТОМОБИЛЬ-ТАХ1 до слоту ВОДІЙ можна встановити, що Кравчук Семен Іванович в теперішній момент не на лінії, а в гаражі або у відпустці і що саме; він водій автомобіля К 08-20 ХА, який в даний час знаходиться в ремонті.

Структура фрейму має наступний вигляд:

{ < ім'я фрейма > < ім'я слота > < значення слота > <ім'я слота > <значення слота >…}. (2.2)

Помітимо, що кожний фрейм це як би готова структура, яка при певному заповненні слотів значеннями перетворюється в опис конкретного факту, події, явища або процесу Тому розрізнюють фрейми-прототипи, що зберігають знання по предметній області, і конкретні фрейми, які поповнюють ці одиниці знань реальними відомостями.

Приклад 2.2.

{< ВІДРЯДЖЕННЯ> < ХТО > < СИДОРОВ В.І. > < КУДИ> < КИЇВ, АН УКРАЇНИ> < КОЛИ > < 1.03.1992 > < НА ЯКИЙ ТЕРМІН> <ДВА ТИЖНІ > < З КИМ> < значення слота>}.

Приведений фрейм має ім'я ВІДРЯДЖЕННЯ з всіма заповненими слотами:, крім слота З КИМ, роль якого в даному прикладі не обов'язкова.

У мові фреймов основною операцією є пошук за зразком. Зразок являє собою фрейм, в якому заповнені не всі структурні одиниці, а тільки ті, по яких серед фреймів, що зберігаються в робочій пам'яті, будуть відшукуватися потрібні фрейми. Зразок може, наприклад, містити ім'я фрейма і ім'я деякого слота зі значенням у фреймі. Процедура пошуку за таким зразком. проверяет наявність в пам'яті ЕОМ фрейма з даним ім'ям і даними значеннями слота в зразку. Якщо в зразку вказане ім'я деякого слота і його значення, то процедура пошуку. обеспечивает вибірку з пам'яті всіх фреймів, в яких міститься слот з такими ім'ям і значенням, як у зразка.

У конкретних реалізаціях фреймових мов на процедуру пошуку за зразком накладаються певні обмеження, які все ж дають користувачеві досить широкий простір для маніпулювання знаннями До поширених фреймових мов відносяться FRL, KRL, FMS, KEE, KRINE, LOOPS і інші, що представляють середовище для розробок і досліджень в області інженерії знань. Системи програмування, засновані на фреймах, часто називають об'єктно-орієнтованими, так, як кожний фрейм відповідає деякому об'єкту предметної області, а слоти містять описуючий цей об'єкт дані, є в слотах знаходяться значення ознак об'єкта. Фрейм може бути наданий у вигляді списку властивостей або в формі записок якщо використати властивості бази даних

Переваги фреймових моделей:

- ефективність структурованого опису складних понять і рішення задач з різними способами висновку внаслідок] різної виродженості родо-видових зв'язків;
- можливість економії робочої пам'яті комп'ютера, оскільки значення слотів зберігаються в єдиному примірнику і. включаються тільки в одну фрейм, що містить слот з даним ім'ям;
- допустимість обчислення значення будь-якого слота за допомогою відповідних процедур або евристично.

Недоліки фреймових моделей:

Відносно висока їх складність, що збільшує трудомісткість внесення змін в родово-видову ієрархію і знижуюча швидкість робота механізму висновку утруднені процеси обробки виключень, а також управління завершеністю і постійністю цілісного образу знань.

Логічна модель

У основі представлення знань логічною моделлю лежить мова математичної логіки, що поєднує в собі логіку висловлювання і числення, предикатів і що дозволяє формально описувати поняття предметної області і зв'язку між ними.

Висловлюванням називається вираження, в якому затверджується або заперечується наявність яких-небудь властивостей об'єкта. Предикатом іменують функцію, визначену на будь-якій безлічі, але приймаючу. только два значення "істина (I)" або "хибне (X)" і призначену для вираження властивостей об'єктів або зв'язків між ними.

Найбільш простою мовою логіки є числення висловлювання (логіка 0-го порядку), в якому відсутні змінні. Для іменування об'єктів предметної області випадкові константи,. Логическиє пропозиції або висловлювання утворять атомарні формули. Висловлювання логічно виходить із заданих посилок, якщо воно істинне завжди, коли істинні посилки. приклад, в складному висловлюванні

"ЯКЩО тварина має пір'я
I відкладає яйця,
ТО ця тваринна птах

висловлювання "ця тварина птах" логічно виходить з двох висловлювань. Як висновок воно. истинно завжди, коли істинні обидві посилки. Правила висновку називаються несуперечливими, якщо їх висновки логічно виходять з посилок. Сукупність правил висновку, що дозволяють вивести все висловлювання., які логічно виходять з посилок називається. полной.

Числення висловлювання не є досить виразний засіб для обробки знань, оскільки в ньому не можуть бути представлені пропозиції, що включають в себе змінні з кванторами. Розширенням числення висловлювання виявляє числення предикатів, з кванторами (логіка 1-го порядку) в якому для вираження відносин між об'єктами можуть користуватися пропозиція не тільки з константами, але і із змінними.

Мова логіки 1-го порядку задається синтаксисом [29] що допускає чотири типи виразів: константи, змінні предикатні імена, функціональні імена.

Константи служать іменами індивідуумів (на відміну від імен сукупностей) об'єктів, людей або подій Наприклад, символи КнігаЗ, 0негин1,

Татьяна2. Файл15 и.т.д, являють собою констакти разом з доданими числами для вказівки цілком певних предметів, людей, процесів.

Змінні означають імена сукупностей, такого, як "книга", "чоловік", "подія" Наприклад, символ Книга3 являє собою цілком певний акземпляр, а символ "книга" означає або "поняття книги", або безліч "всіх книг". Символами, x, y, z,...відмічаються імена сукупностей (певних множин або понять)

Предикатні імена (предикатні константи),. задають правила з'єднання констант і змінних (процедури, операцій, граматики). Для предикатних імен використовуються символи на зразок наступних: СТОЛИЦЯ, ЛОБОВ, ПИСЬМЕННИК, ПЛЮС, ПОМНОЖИТИ.

Функціональні імена (функціональні константи) являють собою такі ж правила, як і предикати, але для відмінності від предикатних записуються малими буквами: столиця, любов, писати, плюс, помножити.

Предикат n-ї парності це предикатне ім'я разом про певним числом ТЕРМІВ і n аргументами. Унарні і бінарні предикати мають відповідно один і два аргументи. ТЕРМОМ є всяка змінна і будь-яка функціональна. форма. Приведені нижче приклади -це логічні моделі представлення фактів за допомогою так званих предикатних атомарних формул:

ЛЮБОВ (Онегін1, Татьяна2); "Онегін любить Тетяну";
СТОЛИЦЯ (Київ): "Київ -столиця"

Введемо бінарні предикати:
КОНКР (значение1, значение2) "є елементом деякого типу";
СУБ'ЄКТ (y, x) - "змінна x є суб'єкт події y
ЕЛЕМ (x, y)-елемент;x належить безлічі y

Наступними прикладами ілюструються правильно побудовані предикатні формули (ППФ), що включають в себе кванторы існування (∃) і спільність (∀)

"Ваня посилає. книгу Маше":
ПОСИЛКА(ВаняЗ, Маша1, Книга15); (2.3)

"Кожний студент вчиться":
∀ x (.СТУДЕНТ (x) ⊃ ВЧИТЬСЯ (x)); (2.4)

Деякі студенти вчаться"
∃ x (СТУДЕНТ (x) ∧ ВЧИТЬСЯ(x)); (2.5)

"Жоден студент не вчиться":
¬(∃ x (СТУДЕНТ (x) ∧ ВЧИТЬСЯ (x)); (2.6)

Будь-який унарний предикат можна перетворити в бінарний, використовуючи предикати КОНКР(х, у) і СУБ'ЄКТА (у, х). Наприклад, речення (2.4) можна представити еквівалентним йому реченням

∀ х(КОНКР(х, студент) ⊃ КОНКР (х, вчиться)),

а речення (2.6) допускає декілька еквівалентних речень:

а) ⌉(∃ х (КОНКР (х, студент) ∧ КОНКР(х, вчиться)));

б) ⌉(∃ х ∃ у (КОНКР(х, студент) ∧КОНКР(у, вчиться) ∧ СУБ'ЄКТ (у, х))

в) (∀х ∀у (КОНКР(х, студент) ∨⌉ КОНКР(х, вчиться) ∨ СУБ'ЄКТ (у, х)).

Нескладно уявити собі в перетворення m-арних предикатів в твір бінарних. Наприклад, тернарний предикат (2.3) виразиться пропозицією

ВІДПРАВНИК(посилка, Ваня3) ∧ ОДЕРЖУВАЧ(посилка, Маша1) ∧ ОБ'ЄКТ (посилка, книга15).

Відносини між значеннями і аргументами m-арного предиката можна виразити також функціональними формами (функціями). Наприклад, квант знань (2.3) представляється в термінах функцій наступною формою:

РІВНО (відправник (Посилка8), Ваня3) ∧ РІВНО (одержувач (Посилка8), Маша1) ∧ РІВНО (об'єкт (Посилка8), Кніга15) ∧ ЕЛЕМ (Посилка8, посилки).

Предикат РІВНО описує відношення рівності. Приведені вище вирази використовують визначені на безлічі "посилок" функції, значення які представляють конкретизації, що стосуються ПОСИЛКИ8.

Фразу тієї ж синтаксичної форми, що і (2.3), запишемо з кванторами:

"Ваня посилає щось кожному" (всім одне і те ж):

∃у ∀ х ПОСИЛКА (Ваня3, х, у).

Ваня посилає щось кожному" (не обов'язково всім одне і те ж):

а) ∀х ∃ у ПОСИЛКА (Ваня3, х, у);

б) ∀х ∃ у ∃посилка (ВІДПРАВНИК (Ваня3) ∧ ОДЕРЖУВАЧ (посилка, х) ∧ ОБ'ЄКТ (посилка, у));

в)∀х ∃у ∃z (ВІДПРАВНИК z (Ваня3) ∧ОДЕРЖУВАЧ(z, х) ∧ОБ'ЄКТ(z,у) ∧ ЕЛЕМ(z, посилки)).

Ці три представлення відповідають трьом формам раніше приведеної фрази "Ваня посилає книгу Маші".

Приведені вище приклади показують, що вибір числа предикатів, їх аргументів, констант і змінних в значній мірі залежить від людини-дослідника

або розробника знанняорієнтованої системи. Числення предикатів ніяк не обгрунтовує такий вибір.

Семантичне значення предикатних формул повинно відповідати істинам предметної області, а визначення -семантичних значень компонент і формул обчислень предикатів базується на понятті інтерпретації логічної формули [29].

До відмітних позитивних бісів логічних моделей представлення знань можна віднести єдиність теоретичного обгрунтування і можливість реалізації системи формально точних визначень і висновків. Головне достоїнство логічних моделей можливість безпосереднього програмування механізмів логічного висновку і маніпулювання знаннями. Логічна інтерпретація формальних граматик і їх правил висновку привела до виникнення мов логічного програмування найбільш відомим з яких є ПРОЛОГ. Недостатком логічних моделей вважають істотні обмеження, що пов’язані з непредставленістю специфічних видів знань у мові логіки предикатів. У звязку з цим введені некласичні (модельна і часова) логіки, а також альтернативні представлення (мережеве і об’єктне)

В межах об’єктного представлення визначаються роздуми з замовчуваннями, “здорового глузду” і нестрогі модифіковувані роздуми, формалізація яких освітлена в праці.

Тема 6. Основи програмування на Пролозі та робота з базами даних (БД).

6.1. Середовище програмування

Після введення команди PROLOG і запуску системи Турбо-Пролог на екрані з'являються чотири вікна [12]:

1.Вікно редагування (для введення початкової програми).
2.Вікно діалогу (для введення запитів і видачі результатів).
3.Вікно повідомлень (для видачі повідомлень).
4.Вікно трасування (для видачі інформації про виконання програми).

У найнижчому рядку виводиться інформація про можливості, що надаються функціональними клавішами. Вище розташовується рядок підказки, в якому з'являються службові вказівки.

Головне меню містить сім команд:

1.**Run** Запуск програми .
2.**Compile** Компіляція програми .
3.**Edit** Редагування програми .
4.**Options** Зміна параметрів компіляції .
5.**Files** Команди управління файлами.
6.**Setup** Зміна системних параметрів .
7.**Quit** Вихід з системи.

За допомогою клавіші **ESC** ви можете вийти з будь-якого меню або підменю і повернутися в головне меню.

Розглянемо послідовно приведені команди.

Run Створена за допомогою редактора програма компілюється і виконується. Якщо в оперативній пам'яті вже знаходиться скомпільована програма, то компіляція не проводиться.

Compile Створена за допомогою редактора програма компілюється. Процес і результати компіляції залежать від вибраних параметрів.

Edit Служить для створення і модифікації початкової програми.

Options Вказуються три підпункти, які визначають результати подальшої компіляції:

Memory Скомпільована програма поміщається в оперативну пам'ять;
OBJ file Скомпільована програма має формат об'єктного файла;
EXE file Скомпільована програма має формат завантажувального файла. Встановлення зв'язків відбувається автоматично.

Files Це меню містить одинадцять команд обробки файлів:

Load Завантажує файл для обробки;
Save Запам'ятовує активний робочий файл;
Directory Вибирає зміст;
Print Друкує вміст робочого файла;
Copy Копіює файл;
Rename Перейменовує будь-який файл;

File-Name Перейменовує робочий файл;
Module List Формує список модулів для виконання;
Zap file in editor Стирає вміст робочого файла;
Erase Стирає будь-який файл.
Operating System Викликає MS-DOS з поверненням.
Quit Ця команда служить для виходу із системи.

Команди редактора

Значення	Керуючі клавіші	Додаткові клавіші
Команди переміщення курсора		
На символ ліворуч	Ctrl-S	<-
На символ праворуч	Ctrl-D	->
На слово ліворуч	Ctrl-A	Ctrl- <-
На слово праворуч	Ctrl-F	Ctrl- ->
Вгору на рядок	Ctrl-E	↑
Вниз на рядок	Ctrl-X	↓
Вгору на сторінку	Ctrl-R	PgUp
Вниз на сторінку	Ctrl-C	PgDn
На початок рядка	Ctrl-OS	Home
До кінця рядка	Ctrl-QD	End
На початок блоку	Ctrl-QB	
До кінця блоку	Ctrl-QK	
Команди стирання		
Режим вставки/заміни	Ctrl-V	Ins
Стерти лівий символ		Backspace
Стерти символ, на який вказує курсор	Ctrl-G	Del
Стерти рядок	Ctrl-Y	Ctrl-Backspace
Команди роботи з блоками		
Відмітити початок блоку	Ctrl-KB	
Відмітити кінець блоку	Ctrl-KK	
Скопіювати блок	Ctrl-KC	F5
Пересунути блок	Ctrl-KV	F6
Стерти блок	Ctrl-KY	F7
Зчитати блок з диска	Ctrl-KR	F9
Записати блок на диск	Ctrt-KW	
Ввімкнути/вимкнути маркування блоку	CtrI-KH	
Інші команди		
Викликати довідку по редактору		F8
Перейти до рядка		F2
Закінчити редагування		F10 або ESC
Ввімкнути/вимкнути	Ctrl-QI	

автоматичну вставку		
Пошук	Ctrl-QF	F3
Продовження пошуку	Shift-F3	
Пошук і заміна	Ctrl-QA	F4

6.2. Структура програми

Програма на Турбо-Пролозі може включати до 5 розділів [7,9,12]:

```
domains
        < опис доменів >
database
        < опис динамічних БД >
predicates
        < опис предикатів >
goals
        < цілеві твердження >
clauses
        < твердження і речення >
/ * коментарі * /
```

6.3. Опис стандартних доменів

Існує шість типів даних (стандартних доменів), опис яких приведений в таблиці 2.1.

Таблиця 2.1

Тип даних	Ключове слово	Діапазон	Приклад
символьний	char	всі символи	‘a’, ‘b’, ...
цілий	integer	–32768 – 32767	будь-які цілі числа
дійсний	real	1E-307 – 1E+308	всі дійсні числа
стрічки	string	послідовні символи <= 250	“today”, “mazda”
символічний	symbol	1)* 2)**	dog, street “it_is_a_dos”
файли	file	допустимі в DOS ім’я файлів	‘a:\edit.pro’,0

1) * Послідовність букв, цифр і знаків підкреслювання (першою повинна бути маленька буква);
2) ** послідовність довільних символів, взятих в подвійні лапки.

Для опису доменів використовується 4 формати:

1 формат : name = d, де name - імена об'єктів, d - ключове слово, що визначає тип стандартного домена.

Приклад: dog, street = symbol.

2 формат : mylist = element *, де mylist - ім'я списку, element - це або ім'я раніше описаного домена, або один із стандартних типів доменів.

Приклад: list1 = symbol* person = symbol
list2 = person*

3 формат : myCompDom = f1 (D11, ..., D1m);
fn (Dn1, ..., Dnm);, де f1-fn – функтори.

Цей формат використовується для опису структур.

4 формат : file = name1; name2; ...;nameN;, де nameN - символічні імена файлів.

В програмі може бути тільки один опис файлу.

Приклад:

```
domains
  person, thing = symbol
predicates
  likes (person, thing)
clauses
```

6.4. Опис предикатів

В цьому розділі вказуються імена предикатів і область їх аргументів або стандартних типів доменів.

likes (person, thing) ◄► likes (symbol, symbol)

Предикати служать як для опису фактів, так і для опису правил, які оперують з фактами. Предикати можуть бути як з аргументами, так і без них. В Турбо-Пролозі є стандартні предикати, яких не потрібно описувати.

6.5. Опис речень

Речення представляють собою факти або правила. Речення повинно закінчуватися крапкою. Формально речення може бути представлене в наступному вигляді:

```
relation (object, ..., object) :-
        relation (object, ..., object) and
        .....
        relation (object, ..., object) or
        .....
```

Речення для одного предикату йдуть одне за одним.

6.6. Деякі особливості структури програми

В програмі обов’язковий перелік розділів предикатів і речень (predicates, clauses). Можливе використання глобальних доменів і предикатів. При компіляції підтримується тільки одна мета. Вміст розділу domains може бути відсутнім, але для зручності читання і зрозумілості програми заголовок domains рекомендується завжди включати в текст програми перед розділом clauses. Якщо програма, що розробляється, призначена для роботи в пакетному режимі, розділ goal не може бути опущений. У програмі можуть бути присутніми ще два розділи, що забезпечують визначення глобальних доменів і предикатів:

global- domains

global predicates.

Визначення типів даних і предикатів в цих розділах дозволяє забезпечити міжмодульний інтерфейс.

6.7. Приклади використання предикатів

Опишемо родинне дерево за допомогою предикатів, граф якого представлений на рисунку 2.1.

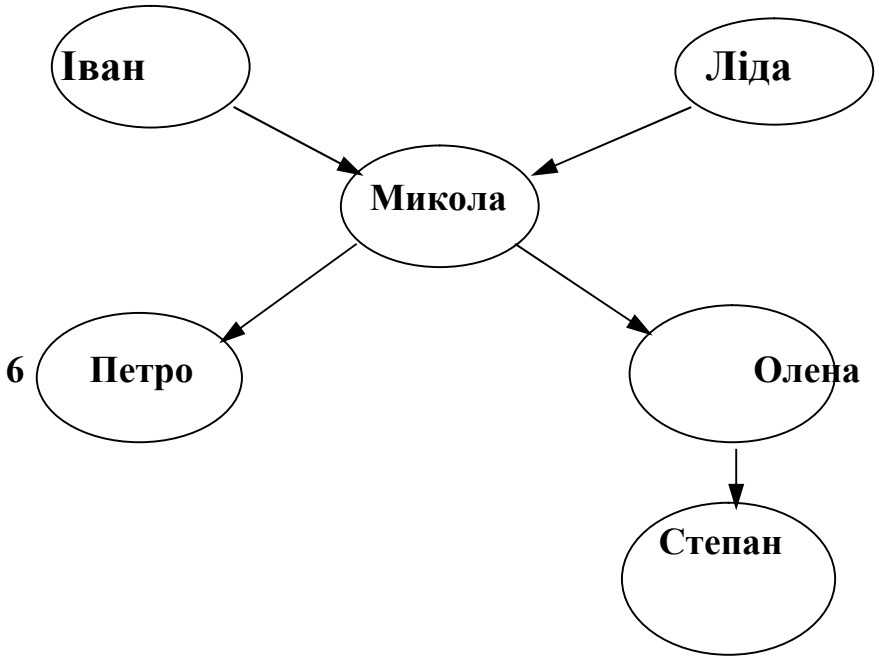
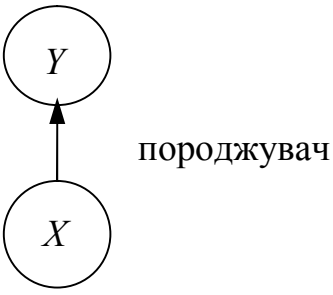


Рисунок 2.1 - Граф родинного дерева

Опишемо зв’язки між членами родини за допомогою предиката “породжувач”.

породжувач (Олена, Степан)
породжувач (Іван, Микола)
породжувач (Ліда, Микола)
породжувач (Микола, Петро)
породжувач (Микола, Олена)

predicates
породжувач (*X*,*Y*)
.
.



або

```
domains
  name1, name2 = symbol                    породжувач (X, Y)
predicates                                 породжувач (X,...)
   породжувач (      name1, name2)         породжувач (...,Y)
                .                          породжувач (...,...)
```

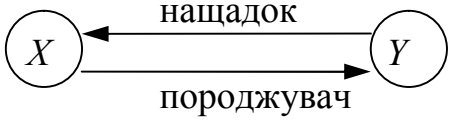
Для опису родинних зв'язків можна додати наступні предикати:

```
predicates
  жінка (name1)
  чоловік (name2)
clauses
  чоловік (Іван)
  чоловік (Микола)
  чоловік (Петро)
  чоловік (Степан)
  жінка (Ліда)
  жінка (Олена)
```

Приклад:

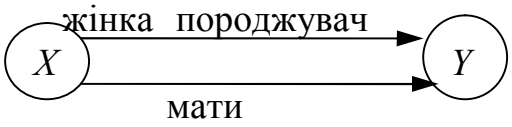
породжувач (*X*, ...),
чоловік (*X*).

Розширимо відношення "породжувач" зворотнім відношенням "нащадок". Це правило можна сформулювати наступним чином: Якщо *Y* нащадок *X*, то *X* - породжувач *Y*.



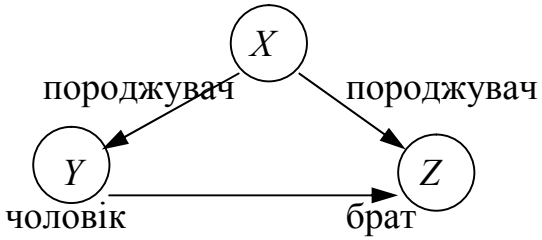
нащадок (*X,Y*) :- породжувач(*Y,X*)

Визначимо відношення "мати".



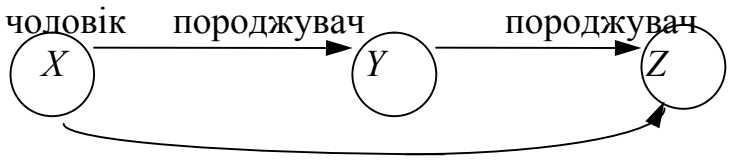
мати (*X,Y*) :- породжувач (*X,Y*), жінка (*X*).

Визначимо відношення "брат".



брат (*Y,X*) :- породжувач (*X,Y*)
породжувач (*X,Z*),
чоловік (*Y*).

Визначимо відношення “дід”.



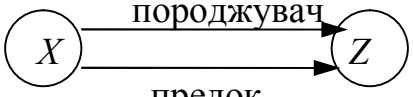
дід

дід (*X*,*Y*) :- породжувач (*X*,*Y*),
породжувач (*X*,*Z*),
чоловік (*X*).

6.8. Рекурсивне визначення правил

Розглянемо відношення “предок” і визначимо його через відношення “породжувач”. Всі відношення можна виразити з допомогою 2-х правил:

* 1-е правило буде визначати безпосереднього найближчого предка;
* 2-е - далеких предків.



предок

Рисунок 2.2.

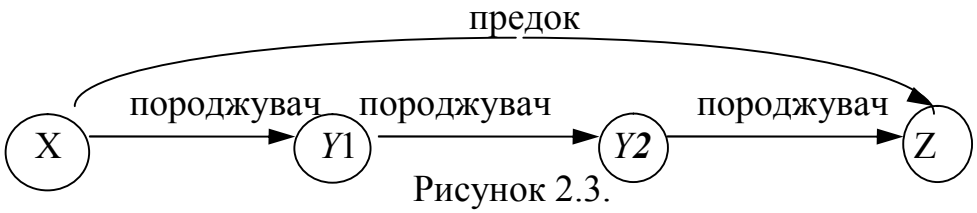


Рисунок 2.3.

Перше правило може сформулювати так: для всіх *X* і *Y* , *X* предок *Z*, якщо *X* породжувач *Z*.

предок (*X, Z*) :- породжувач (*X, Z*) (рисунок 2.2)

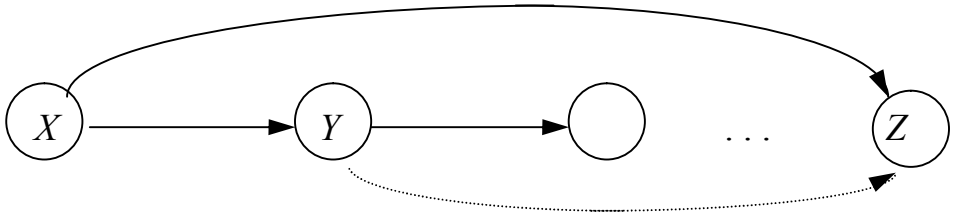
2-е правило складне. Один із способів визначення далеких родичів міг би бути таким, як показано на рисунку 2.3 і відношення “предок” визначалось би множиною речень:

предок (*X*, *Z*) :- породжувач (*X*, *Z*).

предок (*X*, *Z*) :- породжувач (*X*, *Y*), породжувач (*Y* , *Z*).

предок (*X*, *Z*) :- породжувач (*X*, *Y*1), породжувач (*Y*1, *Y*2), породжувач (*Y*2, *Z*).

Ця програма працює лише у визначених об’єктах. Вона буде знаходити предків лише до визначеної глибини родинного дерева. Але можна визначити відношення предок через нього самого: для всіх *X* і *Z*, *X* є предок *Z* якщо є *Y* такий що: *X* - породжувач *Y* і *Y* предок *Z*.



предок (*X*, *Z*) :- породжувач (*X*, *Y*), предок (*Y*, *Z*).
Об’єднавши два правила одержимо:
предок (*X*, *Z*) :- породжувач (*X*, *Z*) (рекурсивне визначення).

Програма “Конструктор слів” є прикладом закінченої програми з використанням предикатів і тверджень [9]. Її метою є пошук і друк синоніма деякого слова. Наприклад, синонімом слова brave є daring.

```
Програма “Конструктор слів”
domains
  word, syn = symbol
predicates
  synonym(word, syn)
clauses
  synonym(brave, daring).
  synonym(honest, truthful).
  synonym(modern, new).
  synonym(rare, uncommon).
goal
  synonym(brave, X),
  write("A synonym for 'brave' is"), nl,
  write("", X, ""), nl.
```

Програма “Президенти” демонструє використання різних типів об'єктів. Твердження даної програми містять відомості про шість президентів США [9]. Предикат president(name, party, state, birth_year, year_in, year_out) /* президент(ім'я, партія, штат, рік народження, початковий рік перебування при владі, кінцевий рік перебування при владі) */ має об'єкти типу символьного рядка і типу цілого числа, як це видно з розділу програми domains. Останні три об'єкти предиката president - цілі числа, їх доменами є відповідно birth_year, year_in і year_out. Оголошення доменів в розділі програми domains виглядає досить просто: birth_year, year_in, year_out = integer. Кома в цьому оголошенні служить як розділювач при переліку імен доменів одного типу.

```
Програма “Президенти”
domains
  name, party, state = symbol
  birth_year, year_in, year_out = integer

predicates
  president(name, party, state,
            birth_year, year_in, year_out)

clauses
```

```
  president(eisenhower, republican, texas,
        1890, 1953, 1961).
  president(kennedy, democrat, massachusets,
        1917, 1961, 1963).
  president(johnson, democat, texas,
        1908, 1963, 1969).
  president(nixon, republican, california,
        1913, 1969, 1974).
  president(ford, republican, nebraska,
        1913, 1974, 1977).
  president(carter, democrat, georgia,
        1924, 1977, 1981).

goal
  president(X, democrat, S, Yb, Yi, Yo),
  write("Name: ", X), nl,
  write("State: ", S), nl,
  write("Year of birth: ", Yb), nl,
  write("Year in: ", Yi), nl,
  write("Year out: ", Yo), nl.
```

Лекція 7.
Тема 7. Повторення і рекурсія

7.1. Програмування операцій, що повторюються

Є два способи реалізації правил, що виконують одну і ту ж задачу багатократно: повторення і рекурсія [7-9]. При повторенні використовується такий механізм, як відкат, а для реалізації рекурсії застосовується самовиклик. Відкат - це механізм Турбо-Прологу, який використовується для знаходження додаткових фактів і правил, необхідних для обчислення мети, якщо поточна спроба її обчислення виявилась невдалою. Правила, які реалізують повтор і рекурсію, повинні містити засоби керування їх виконанням. Загальну схему організації повторення можна представити у такому вигляді:

1) rule :-
   <предикати і правила>,
   fail.
2) rule :-
   <предикати і правила>,
   !.

Для управління повторенням використовуються вбудовані предикати:

Fail - це невдача. Він здійснює відкат, так що предикати і правила виконуються ще раз.

! або cut - це предикат, що означає відсічення, і він встановлює бар'єр, який заважає виконувати відкат до всіх альтернативних рішень поточної підцілі.

Загальний вид організації рекурсії наступний:

```
rule :-
        <предикати і правила>,
        rule.
```

Правила повтору і рекурсії забезпечують однаковий результат, хоч алгоритми їх виконання різні. Кожен із них в конкретній ситуації має свої переваги. Рекурсія потребує великих системних ресурсів. Існують наступні методи для організації операцій, що повторюються:

1) відкат після невдачі;
2) метод відсікання і відкату;
3) правила повтору, що визначаються користувачем;
4) узагальнене рекурсивне правило.

7.2. Методи повторення

7.2.1. Метод відкату після невдачі

Цей метод повторення використовується для управління обчисленням мети при пошуку всіх можливих її рішень. Приклад програми: вивести на екран назву всіх міст України:

```
domains
        name = symbol
predicates
        cities (name)
        show_citses        1                2
goal
        write ("Назва міст України"), nl, show_cities.
clauses
        cities ("Київ").
        cities ("Харків").
        cities ("Дніпропетровськ").
        cities ("Львів").
        show_cities :-
                cities (N),
                write (N), nl,
                fail.
```

Ціль складається із 2-х підцілей: 1 і 2. Спочатку виконується перша ціль безумовно, друга ціль є правилом. Предикат cities обмежує змінну N першою назвою міста - "Київ". Потім це значення виводиться на екран. Предикат fail

визиває відкат до наступного твердження, яке може забезпечити обчислення цілі. Fail забезпечує відкат як для невдалих, так і для вдалих правил.

Розглянемо ще один приклад програми “Службовці” на використання предикату fail [9].

```
        Програма “Службовці”
        domains
  name, sex, department = symbol
  pay_rate = real
predicates
  employee(name,sex,department,pay_rate)
  show_male_part_time
  show_data_proc_dept
        goal
          write("Службовці чоловічої статі з почасовою оплатою"), nl, nl,
          show_male_part_time.
        clauses
        employee("John Walker   ","M","ACCT",3.50).
        employee("Tom Sellack   ","M","OPER",4.50).
        employee("Betty Lue     ","F","DATA",5.00).
        employee("Jack Hunter   ","M","ADVE",4.50).
        employee("Sam Ray       ","M","DATA",6.00).
        employee("Sheila Burton ","F","ADVE",5.00).
        employee("Kelly Smith   ","F","ACCT",5.00).
        employee("Diana Prince  ","F","DATA",5.00).
        show_male_part_time :-
          employee(Name, "M", Dept, Pay_rate),
          write(Name, Dept,  "$", Pay_rate), nl,
          fail.
        show_data_proc_dept :-
          employee(Name, _, "DATA", Pay_rate),
          write(Name, "$", Pay_rate), nl,
          fail.
```

Програма містить дані про службовців компанії, працюючих неповний робочий день. Предикат бази даних має вигляд employee(name, sex, department, pay_rate)

/* службовець(ім'я, стать, відділ, почасова_оплата) */

Наступне правило видає вміст всієї бази даних:

```
show_all_part_time_employees: -
employee(Name, Sex, Dept, Pay_rate),
write(Name, "  ", Sex, "  ", Dept, "  ", Pay_rate), nl,
fail.
```

Змінні Name (ім'я), Sex (стать), Dept (відділ), Pay_rate (почасова оплата) не означені, і, отже, всі вони можуть набути різних значень. Якби це правило було

метою програми, то результатом її виконання був би список всіх службовців з неповним робочим днем. Відзначимо, що необхідно отримати список, який містить дані тільки про службовців чоловічої статі. Для цього потрібно, щоб процес зіставлення значень Sex був успішним тільки для тверджень, що містять M в позиції другого об'єкта. Під час внутрішньої уніфікації постійне значення M порівняне тільки з M. Правило, що накладає цю умову на вибірку даних, має вигляд:

```
show_male_part_time: -
employee(Name, "M", Dept, Pay_rate),
write(Name, Dept, "$ ", Pay_rate), nl,
fail.
```

Альтернативною формою умови вибірки по ознаці статі є предикат рівності Sex = "M". Використовуючи цей предикат рівності, можна побудувати інше правило, що має той же самий результат:

```
show_male_part_time: -
employee(Name, Sex, Dept, Pay_rate),
Sex="M",
write(Name, Dept, "$ ", Pay_rate), nl
fail.
```

Зауважте, що для цих правил деякі підцілі можуть виявитися неуспішними через неможливість задовольнити умові ознаки статі. Відкат виникне до моменту, коли інформація, що міститься в факті, буде видана на екран. Предикат fail не потрібен, якщо умови правил неможливо виконати, тобто підціль буде неуспішною сама по собі. Предикат fail включений в правило для того, щоб викликати відкат, якщо умови правила будуть виконані і все правило виявиться успішним.

7.2.2. Метод відсікання і відкату

В деяких випадках необхідно мати доступ тільки до визначеної частини даних, які обмежуються деякою умовою. Розглянемо той же приклад, тільки по-іншому визначимо правило show_cities:

```
show_cities :-
        cities (N),
        write (N), nl,
        make (N), !.
        make (N) :- N = “Могилів -Подільський”.
```

Предикат ! забезпечує відсікання. Обчислення цього предикату завжди вдале. Цей предикат встановлює бар’єр, який забороняє виконати відкат до всіх альтернативних рішень поточної підцілі. Для виводу міст, що повторюються, в даному випадку “Тернопіль”:

```
show_cities :-
        cities (N),
        N = “Тернопіль”,
        write (N), nl,
        fail.
```

Якщо необхідно видати тільки перше місто з іменем “Тернопіль”, то це можна організувати наступним чином:

```
show_cities :-
        cities (N),
        N = “Тернопіль”,
        write (N), nl,
        !.
```

Пояснити метод відкату і відсікання можна на простій ситуації, в якій предикати бази даних містять декілька імен, як це має місце для предикатів child (дитина) в програмі, що формує список імен дітей. Відзначимо, що необхідно видати список імен дітей до імені Diana включно.

Програма “Імена дітей”:

```
domains
  person = symbol
predicates
  child(person)
  show_some_of_them
  make_cut(person)
goal
  write( "Хлопчики і дівчатка" ), nl, nl,
  show_some_of_them
clauses
child(" Tom  ").
```

```
child(" Beth  ").
child(" Jeff  ").
child(" Sarah  ").
child(" Larry  ").
child(" Peter  ").
child(" Diana  ").
child(" Judy  ").
child(" Sandy  ").

show_some_of_them:-
  child(Name),
  write("   ", Name), nl,
  make_cut(Name),!.
make_cut(Name):- Name="Diana".
```

Предикат cut використовується для того, щоб виконати відсікання у вказаному місці. Предикат fail використовується для продовження відкатів і доступу до послідовності імен бази даних до елемента з ім'ям Diana. Таким чином, поставлену задачу виконує відповідна комбінація предикатів cut і fail. Ця комбінація називається методом відкату і відсікання. У програмі про імена предикатом бази даних є child (person). Для цього предиката є 9 альтернативних тверджень.

Правило, що забезпечує генерацію всіх імен (а не тільки деяких), має вигляд:

```
show_them_all: -
child(Name),
write("    ", Name), nl,
fail.
```

Воно засноване на методі відкату після невдачі. При цьому для того, щоб використати предикат cut, необхідно визначити деяку умову, яка може бути і простою, як в нашому прикладі, і дуже складною. У цьому випадку достатньою є умова Name=Diana. Правило, що визначає цю умову, має вигляд:

```
make_cut(Name): - Name="Diana".
```

Це правило з подальшим предикатом cut (!) утворить правило make_cut (зробити відсікання). Тепер виконання програми буде неуспішним, а відкати будуть повторюватися доти, поки Name не виявиться рівним Diana. У цей момент результат виконання правила make_cut буде успішним, що в свою чергу спричинить виконання предиката cut. Таким чином, комбінуючи правила відсікання з методом відкату після невдачі, можна отримати правило відсікання і відкату:

```
show_some_of_them: -
child(Name),
write("   ", Name), nl,
make_cut(Name), nl.
```

```
make_cut(Name): - Name="Diana".
```

7.2.3. Метод повтору, що визначається користувачем

Вигляд правила повтору наступний [7-9]:

```
repeat.
repeat :- repeat.
```

Перший repeat є твердженням, яке оголошує предикат repeat істинним. Перший предикат не створює підцілей, тому він завжди істинний. Оскільки є ще один варіант для того правила, то вказівник відкату встановлюється на перший repeat.

Другий repeat - це правило, яке використовує само себе як компоненту. Другий repeat викликає третій repeat, і цей виклик обчислюється успішно, так як перший repeat задовільняє підцілі repeat, тому правило завжди успішне. Предикат repeat буде обчислюватися успішно при кожній новій спробі його виклику після відкату. Наведене правило - це рекурсивне правило, яке ніколи не буває неуспішним. Це правило використовується в якості компоненти інших правил. Розглянемо приклад програми, яка зчитує стрічку, введену з клавіатури, і дублює її на екран. У разі введення користувачем стрічки “stop” програма закінчується.

```
domans
        name = symbol
predicates
        write_message
        repeat
        do_echo
        check (name)
goal
        write_message.
        do_echo.
clauses
        repeat
        repeat :- repeat
        write_message :-
                nl, write (“Введіть, будьласка, імена”), nl,
                    write (“Я повторю їх”), nl,
                    write (“Щоб мене залишити, введіть stop”), nl.
        do_echo :-
                repeat
                readln (Name),
                write(Name), nl,
                check (Name), !.
```

```
check (stop) :-
        nl, write (“Допобачення”).
check ( - ) :- fail.
```

Параметр do_echo - це правило повтору, що визначається користувачем. Першою компонентою цього правила є repeat, який викликає повторне виконання всіх наступних за ним компонент. Предикат readln зчитує стрічку з клавіатури, предикат write видає цю стрічку на екран, остання компонента check має два варіанти: перший варіант забезпечує перевірку введеного повідомлення на рівність “stop”, при цьому курсор зсувається на початок стрічки і видається повідомлення “До побачення” і процес повторення закінчується виконанням предиката відсічення !.

Правило repeat широко використовується в якості компоненти інших правил. Прикладом цього може служити програма “Луна”, яка зчитує рядок, введений з клавіатури, і дублює його на екран. Якщо користувач введе “stop”, то програма завершується.

Програма “Луна”

```
domains
  name = symbol
predicates
  write_message
  repeat
  do_echo
  check(name)
goal
  write_message,
  do_echo.
clauses
  repeat.
  repeat: - repeat.
write_message:-
  nl, write( "Введіть, будь ласка, імена"), nl,
  write("Я повторю їх"), nl,
  write( "Щоб зупинити мене, введіть stop"), nl, nl.
do_echo:-
  repeat,
  readln(Name),
  write(Name), nl,
  check(Name),!.
check(stop):-
  nl, write(" - OK, bye!").
check(_):- fail.
```

Правило repeat є першим в розділі тверджень програми “Луна”. Друге правило виводить інформацію для користувача. Третє правило do_echo є

правилом повтору, визначеним користувачем. Його перша компонента є repeat:

```
do_echo: -
repeat,
readln(Name),
write(Name), nl,
check(Name),!.
```

Твердження repeat спричиняє повторне виконання всіх наступних за ним компонент. Предикат readnl(Name) зчитує рядок з клавіатури, write(Name) видає його на екран. Останнє підправило check(Name) має два можливих значення. Одне визначається підправилом:

```
check(stop): -
nl, write(" - OK, bye!").
```

Якщо рядок, що вводиться користувачем має значення “stop”, то правило буде успішним. При цьому курсор зсувається на початок наступного рядка, на екрані з'являється повідомлення "OK, bye!" (до побачення), і процес повторення завершується. Символ відсікання (!) служить для припинення відкатів, якщо умова check виконана. Інше значення check(Name) визначається підправилом:

```
check(Name): - fail.
```

Якщо значення рядка відмінне від “stop”, то результат виконання правила буде fail. У цьому випадку станеться відкат до правила repeat. Таким чином, do_echo є кінцевим правилом в ланцюгу повторень, умова виходу з якої визначається предикатом check. Завдяки тому, що правило repeat є компонентою, правило do_echo стає кінцевим правилом повтору.

7.2.4. Узагальнене рекурсивне правило

Узагальнене рекурсивне правило можна представити у вигляді [7-9]:

<ім’я правила рекурсії> :-

1) <список предикатів>,
2) <предикат умови виходу>,
3) <список предикатів>,
4) <ім’я правила рекурсії>,
5) <список предикатів>.

Перша компонента - група предикатів, в якій успіх чи невдача на рекурсію не впливає. Друга компонента - це предикат умови виходу, успіх чи невдача цього предиката або дозволяє продовжити рекурсію, або викликає її зупинку. Третя компонента аналогічна першій. Четверта компонента - це саме рекурсивне правило. Успіх цього правила викликає рекурсію. П’ята компонента - це список предикатів, успіх чи невдача яких не впливають на рекурсію.

Розглянемо просту рекурсію. Правило, що містить само себе в якості компоненти, називається правилом рекурсії. Правила рекурсії, як і правила повтору, реалізовують повторне виконання задач. Вони ефективні, наприклад, при формуванні запитів до бази даних, а також при обробці таких доменних структур, як списки. Приклад правила рекурсії:

```
write_string: - /* видати рядок */
write( "МИ - ЦЕ ВЕСЬ СВІТ"), nl,
write_string.
```

Це правило складається з трьох компонент. Перші дві видають рядок "МИ – ЦЕ ВЕСЬ СВІТ" і переводять курсор на початок наступного рядка екрана. Третя – це саме правило. Оскільки воно містить само себе, то, щоб бути успішним, правило write_string повинно задовольняти само собі. Це приводить знову до виклику операції видачі на екран рядка і зміщення курсора на початок нового рядка екрана. Процес продовжується нескінченно, і в результаті рядки видаються на екран нескінченне число разів.

Однак, у разі виникнення нескінченної рекурсії, число елементів даних, що використовуються рекурсивним процесом, безперервно росте і в деякий момент стек переповниться. На екрані з'явиться повідомлення про помилку. Виникнення переповнення під час виконання програми для користувача небажане, оскільки в результаті можуть виявитися загубленими істотні дані. Уникнути подібної ситуації можна шляхом збільшення розмірів стека, для чого служить опція Miscellaneous settings (інші установки параметрів) в меню Setup (установка).

Програма "Повернися" демонструє просте правило рекурсії, в яке включена умова виходу.

```
domains
  char_data = char

predicates
  write_prompt
  read_a_character

goal
  write_prompt,
  read_a_character

clauses
write_prompt: -
  write( "Будь ласка, введіть символи."), nl, nl
  write( "Для завершення введіть #    "), nl, nl.
```

Програма циклічно зчитує символ, введений користувачем: якщо цей символ не #, то він видається на екран, якщо цей символ #, то програма завершується. Правило рекурсії має вигляд:

```
read_a_character: -
  readchar(Char_data),
  Char_data <> '#',
  write(Char_data),
  read_a_character.
```

Перша компонента правила є вбудований предикат Турбо-Прологу, що

забезпечує зчитування символа. Значення цього символа присвоюється змінній Char_data. Наступне підправило перевіряє, чи є введений символ символом #. Якщо ні, то підправило успішне, символ видається на екран і рекурсивно викликається read_a_character. Цей процес продовжується доти, поки внутрішня перевірка не виявить недопустимий символ #. У цей момент обробка зупиняється і програма завершується.

Узагальнене правило рекурсії містить в тілі правила само себе. Рекурсія буде кінцевою, якщо в правило включена умова виходу, що гарантує закінчення його роботи. Тіло правила складається із тверджень і правил, що визначають задачі, які повинні бути виконані. Правило рекурсії повинно містити умову виходу, в іншому випадку рекурсія нескінченна.

Розглянемо приклад генерації всіх цілих чисел, починаючи з 1 і закінчуючи 7. Нехай ім'я правила буде write_number(Number). Для цього прикладу перша компонента структури загального правила рекурсії не використовується. Другою компонентою, тобто предикатом умови виходу, є Number < 8. Коли значення Number дорівнює 8, правило буде успішним і програма завершиться. Третя компонента правила оперує з числами. У цій частині правила число видається на екран і потім збільшується на одиницю. Для збільшених чисел буде використовуватися нова змінна Next_Number. Четверта компонента - виклик самого правила рекурсії write_number(Next_number). П'ята компонента, представлена в загальному випадку, тут не використовується.

Програма генерації ряду чисел використовує наступне правило рекурсії:

```
write_number(8).
write_number(Number): -
Number < 8,
write(Number), nl,
Next_Number = Number + 1,
write_number (Next_number).
```

Текст програми “Генерація ряду” наступний:

```
Програма “Генерація ряду”
domains
  number = integer

predicates
  write_number(number)

goal
  write("Here are the numbers:"),
  nl, nl,
  write_number(1),
  nl, nl,
  write("          All done, bye!").

clauses
```

```
write_number(8).
write_number(Number) :-
        Number < 8,
        write("          ", Number), nl,
        Next_number = Number + 1,
        write_number(Next_number).
```

Програма починається зі спроби обчислити підціль write_number(l). Спочатку вона зіставляє підціль з першим правилом write_number(8). Оскільки 1 не дорівнює 8, то зіставлення неуспішне. Програма знову намагається зіставити підціль, але вже з головою правила write_number(Number). На цей раз зіставлення успішне внаслідок того, що змінній Number присвоєне значення 1. Програма порівнює це значення з 8; умовою виходу є рівність Number=8. Оскільки 1 менше 8, то підправило успішне. Наступний предикат видає значення, присвоєне Number. Змінна Next_Number набуває значення 2, а значення Number збільшується на 1.

У цей момент правило write_number викликає само себе з новим значенням параметра рівним 2 і присвоєним Next_Number. Відмітимо, що необов'язково викликати правило, використовуючи те ж ім'я змінної, яке є в голові правила. Це всього лише позиція в списку параметрів, яка є істотною при передачі значень. Фактично, якщо не передавати значення Next_Number, то приріст основного числа програми неможливий. При рекурсивному виклику голови правила програма знову намагається виконати підціль write_number(8). Програма продовжує виконувати цикл зіставлення, присвоєння і видачі значень Number доти, поки значення Number не стане рівним 8. В цей момент мета виконана, правило успішне і програма завершується після видачі повідомлення All done, bye!

Важливою властивістю правила рекурсії є його розширюваність. Наприклад, воно може бути розширене для підрахунку суми ряду цілих чисел.

Програма "Сума ряду 1" використовує правило рекурсії для обчислення суми ряду цілих чисел від 1 до 7:

S(7) = 1+2+3+4+5+6+7 = 28

або

S(7) = 7+6+5+4+3+2+1 = 28

Правило рекурсії програми виконує обчислення за звичайною схемою додавання:

```
1       Початкове значення
+ 2     Наступне значення
4       Часткова сума
+ 3     Наступне значення
6       Часткова сума
```

Правило рекурсії має вигляд

```
sum_series(1,1).                    /* сума ряду */
sum_series(Number, Sum): -
Number > 0,
```

```
        Next_number = Number - 1,
        sum_series(Next_number, Partial_Sum),
        Sum = Number + Partial_Sum.
```

Дане правило має чотири компоненти і одне додаткове нерекурсивне правило. Відмітимо, що остання компонента правила рекурсії - це правило Sum з Partial_Sum (часткова сума) як змінною. Це правило не може бути виконане доти, поки Partial_Sum не набуде деякого значення [9].

```
        Програма “Сума ряду”
        domains
  number, sum = integer
predicates
  sum_series(number, sum)
goal
          sum_series(7,Sum),
          write("Сума ряду:"), nl, nl,
          write("   S(7) = ", Sum), nl.
        clauses
          sum_series(1,1).
          sum_series(Number, Sum) :-
            Number > 0,
            Next_number = Number - 1,
            sum_series(Next_number, Partial_Sum),
            Sum = Number + Partial_Sum.
```

Програма “Сума ряду” починається зі спроби виконати підціль sum_series(7, Sum). Спочатку програма намагається зіставити підціль з підправилом sum_series(1,1). Зіставлення невдале. Потім вона намагається зіставити підціль з sum_series(Number, Sum). На цей раз зіставлення завершується успішно з присвоєнням змінній Number значення 7. Далі програма порівнює значення Number, рівне 7, з 0, тобто перевіряється умова виходу. Оскільки 7 більше 0, то зіставлення успішне, програма переходить до наступного підправила. У цьому правилі змінній Next_number присвоюється значення 6, тобто значення Number - 1. Потім правило викликає саме себе у вигляді sum_series(6, Partial_Sum). Наступним підправилом є правило Sum, що містить вільну змінну Partial_Sum. Однак, щойно був викликаний рекурсивний процес, тому правило Sum не може бути викликане. Тепер програма намагається зіставити незмінне правило sum_series (1,1) з sum_series (6, Partial_Sum). Процес зіставлення неуспішний, оскільки непорівнянний жоден з параметрів. У результаті програма переходить до наступного правила з головою sum_series (Number, Sum), присвоюючи змінній Number значення 6. Цей циклічний процес зіставлення продовжується доти, поки не буде отримано sum_series(1, Partial_Sum). Тепер це правило зіставляється з sum_series(1,1), а Partial_Sum приписується значення 1. При зіставленні правила з головою правила змінна Sum набуває значення 1. Процес зіставлення продовжується далі, внаслідок чого

Next_number набуде значення 0 (1-1). На наступному етапі зіставлення змінна Number набуває значення 0 і за умовою виходу правило виявляється неуспішним. Таким чином, процес зіставлення "стрибає" до правила Sum. Під час процесу зіставлення змінна Partial_Sum була вільна, а програма запам'ятовувала значення Number для подальшого використання. Але правило SUM продовжує означувати змінну Sum, присвоюючи їй послідовно значення 1, 3, 6, 10, 15, 21 і 28. Кінцеве значення Sum є 28.

Програма “Факторіал” використовує правило рекурсії для обчислення і друку факторіала цілого числа. (Факторіал числа N записується як N!. Окличний знак - це поширене позначення факторіала.) N! - є добуток всіх цілих чисел від 1 до N:

N! = N * (N-1) * (N-2) *. .. 2 * 1

Приклади:

1! = 1

21 = 2 * 1 = 2

3! = 3*2*1 = 6

4! = 4*3*2*1 = 24

5! =5*4*3*2*1 = 120

6! = 6*5*4*3*2*1 = 720

7! = 7*6*5*4*3*2*1= 5040

Структура правила рекурсії для обчислення факторіала точно така ж, як і для правила рекурсії попередньої програми. Для підсумовування ряду використовувалося послідовне підсумовування. Воно виконувалося за допомогою рекурсії. Для обчислення факторіала використовується послідовне множення. Його значення утвориться після зчитування значень відповідних змінних з стека, що використовуються як список параметрів для останнього підправила. Це останнє підправило викликається після завершення рекурсії. Правило рекурсії для обчислення факторіала наступне:

```
factorial (1,1): - !.
factorial(Number, Result): -
Next_number = Number -1,
factorial(Next_number, Partial_factorial),
Result = Number*Partial_factorial.
```

Внаслідок роботи програми отримаємо 7! = **5040.**

Програма “Факторіал”

```
domains
  number, product = Integer
predicates
  factorial(number, product)
goal
  factorial(7, Result),
  write(" 7! =,      "Result), nl.
clauses
factorial(l, l): - !.
```

```
factorial(Number, Result): -
  Next_number = Number -1,
  factorial(Next_number, Partial_factorial),
Result = Number * Partial_factorial.
```

Лекція 8.

Тема 8. Списки, операції і арифметичні вирази.

8.1. Списки

Списки є набором об'єктів одного і того ж доменного типу [9,10]. Об'єктами списку можуть бути цілі і дійсні числа, символи, символьні стрічки і структури. Відмінною рисою списків є відносне положення елементів у списку. Порядок грає важливу роль в процесі співставлення.

Приклади списків:

[1, 5, 9, 13] ["Так", "Ні", "Незнаю"]

[2.5, 7.8, 3.6]

Кількість елементів в списку називається довжиною. Список, який не містить елементів, називається пустим або нульовим [ ], довжина цього списку дорівнює нулю.

Непустий список можна розглядати як такий, що складається з 2-х частин:

1) перший елемент списку - голова;

2) інша частина списку - хвіст.

Голова є елементом списку, а хвіст представляє собою хвіст сам по собі. Якщо список складається із одного елемента, то його можна розділити на голову і пустий список.

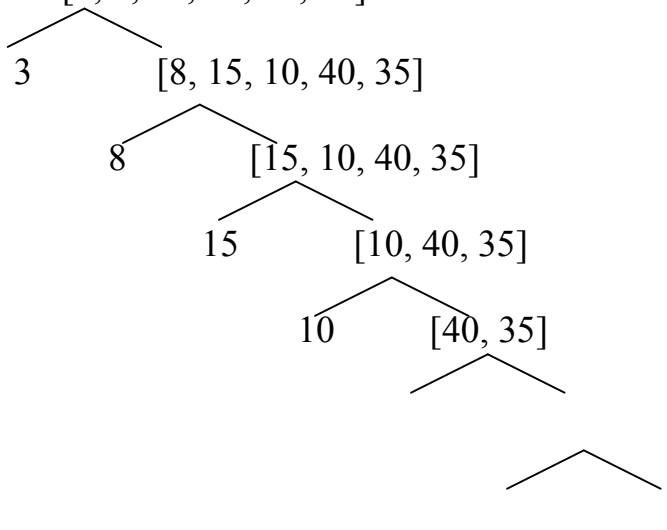
[8] → 8+ [ ]

[1, 3, 8] → 1+ [3, 8]

Список можна графічно представити у вигляді лінійного графу чи у вигляді бінарного дерева.

[3, 8, 15, 10, 40, 35].

У вигляді бінарного дерева:



40 [35]

[35] [ ]

У вигляді лінійного графу (рисунок 7.1).

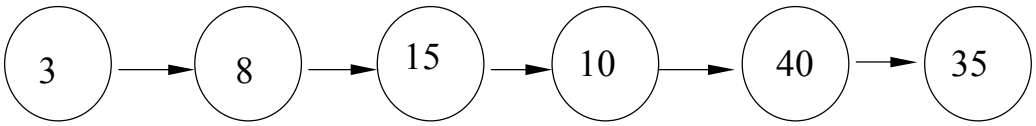


Рисунок 7.1 - Лінійний граф

8.2. Опис списків

Для того, щоб можна було використовувати список в програмі, необхідно:
1) описати домен елементів списку

```
b = a *                    b = integer *
a = integer
```

2) описати предикат списку (предикат, який використовує список)

```
predicates
        list (b)
```

3) присвоєння значень елементам списку здійснюється в розділі речень чи мети:

```
clauses
        list ([7, 8, 9, 13]).
goal
        list (X)
        list ( _ , _ , x, _ )
```

Операція ділення списку на голову і хвіст позначається за допомогою вертикальної лінії:

```
        [H | T]
```

голова — хвіст

Приклад виводу значень списку на екран.

```
domains
        a = b *
        b = integer
predicates
        print_list (a)
clauses
        print_list ([ ]).
        print_list ([ H | T]) :-
                write (H), nl,
                print_list (T).
```

8.3. Операції над списками

7.3.1. Пошук елементів у списку

```
domains
        l = symbol *
        n = symbol  елемент
predicates                  список
        member (n, l)

clauses
    1)  member (N, [ N | _ ]).
    2)  member (N, [ _ | T ]) :-
            member (N,T).
```

Перший варіант правила передбачає наявність відповідності між об'єктом пошуку і головою списку. Хвіст в даному випадку не суттєвий. Дану операцію можна використовувати в наступних випадках:

1) визначити приналежність елемента до списку;
2) знайти всі елементи списку.

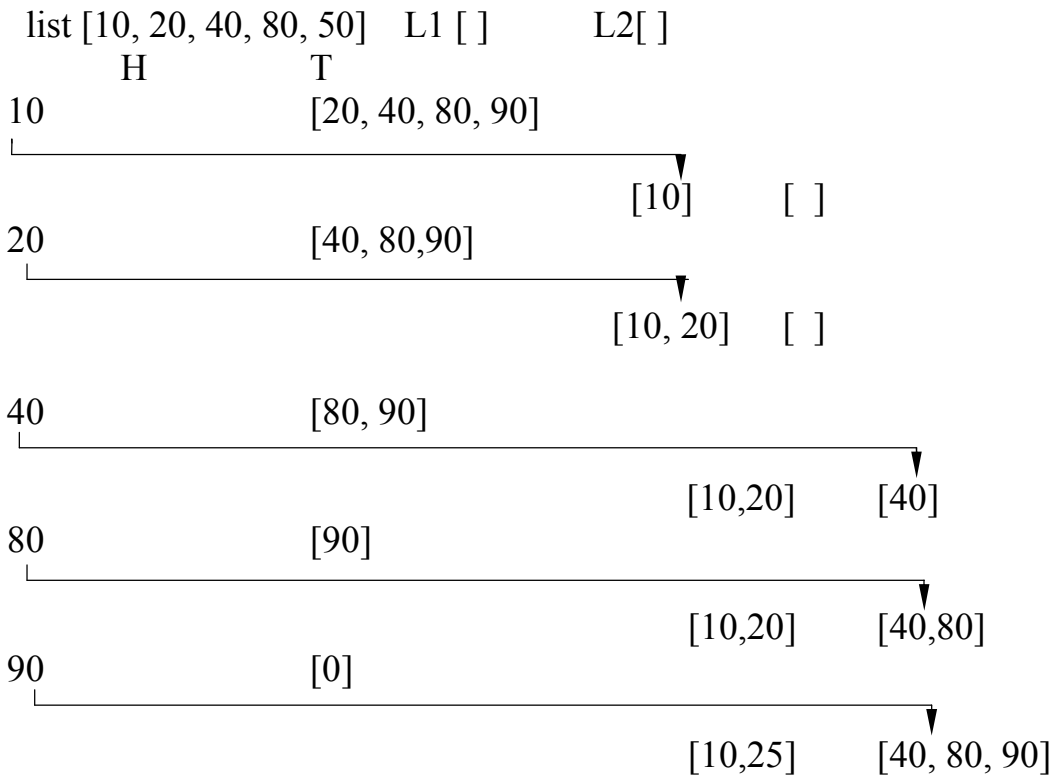
Наприклад:

1) member (5, [7, 8, 9, 5, 10]).
2) member (X, [. . . .]).

8.3.2. Ділення списків

```
domains
        middle = integer
        list = integer *
predicates
        split (middle, list, list, list)
clauses
    1)  split (middle, [H | T], [H | L1], L2) :-
            H <= middle,
            split (middle, T, L1, L2).

    2)  split (middle, [H | T],L1,[H | L2) :-
            split (middle, T, L1, L2),
            H > middle.
        split ( _ , [ ], [ ], [ ]).
            middle = 25
```



8.3.3. Приєднання списку

```
domains
        list = integer *
predicates
        append (list, list, list)
clauses
    1)  append ([ ], L, L).
    2)  append ([N | L1],L2, [N | L3]) :-
            append (L1, L2, L3).
```

Перше правило виконується для випадку, коли перший список пустий, то другий список дорівнює L. Якщо L1 не пустий, то за другим правилом один елемент із L1 передається в L3 до цих пір, поки L1 не стане пустий.

1) ([. . .], [. . .], *X*) - одержимо *X* як об'єднання списків L1 і L2, по суті L3.
2) (*X*, *Y*, [. . .]) - видає всі можливі комбінації *X* і *Y*.
3) ([. . .], *X*, [. . .]) - використовується для визначення різниці 2-х списків.
4) (*X*, [травень | *Y*], [січ. , . . ., грудень]) - розділення.

X = [січ. , . . . , квіт.]
Y = [трав., . . . , груд.].

Весь процес приєднання списку можна представити у вигляді сукупності дій: список L3 спочатку пустий, елементи L1 переміщуються в L3, елементи L2 переміщуються в L3.

Приклад: L1 = [4, 5, 8] L2 = [9, 13].

Спочатку Турбо-Пролог намагається задовільнити перше правило. Для того, щоб задовільнити перше правило, необхідно, щоб L1 був пустим.

Турбо-Пролог намагається задовільнити друге правило, розкручуючи ланцюжок рекурсій до цих пір, поки L1 не стане пустим. При цьому елементи списку L1 послідовно пересилаються в стек. Коли L1 пустий, то є можливим застосування першого правила. Третій список ініціалізується другим. Такий процес можна пояснити за допомогою двох станів append до і після застосування першого варіанта правила

append ([ ], [9, 13], . . .) до
append ([ ], [9, 13], [9, 13]) після

В даний момент процедури уніфікації Турбо-Прологу повністю задовільняють це правило і Турбо-Пролог починає закривати рекурсивні виклики цього правила. Вилучені при цьому із стеку елементи розміщуються один за одним в якості голови до першого і третього списку. Необхідно замітити, що елементи вилучаються в зворотньому порядку, і що значення вилученого із стеку елемента присвоюються змінній N одночасно у L1 і L3: [N I L1] [N I L3]. Кроки цього процесу можна представити наступним чином:

append ([ ], [9, 13], [9, 13])
append ([8], [9, 13], [8, 9, 13])
append ([5, 8], [9, 13], [5, 8, 9, 13])
append ([4, 5, 8], [9, 13], [4, 5, 8, 9, 13])

Розглянемо 6 операцій: добавлення елемента, знищення, приналежність елемента, сортування списку, приєднання двох списків, виділення підсписку.

8.3.4. Сортування списків

```
domains
        number = integer
        list = member *
predicates
        insert_sort (list, list)
        insert (number, list, list)
        arc_order (number, number)
clauses
    1)  insert_sort ([ ], [ ]).
    2)  insert_sort ([X | T], sorted_l) :-
            *insert_sort (T, sorted_T),
            insert (X, sorted_T, sorted_L).
        insert (X, [Y | Sorted_L], [Y | Sorted_L1]) :-
            arc_order (X, Y), !,
            insert (X, Sorted_L, Sorted_L1).
```

```
insert (X, Sorted_L, [X | Sorted_L]).
arc_order (X, Y) :- X > Y.
```

При сортуванні використовуються засоби розбиття списку на голову і хвіст і сортування хвоста, який також є списком.

Нехай задана мета: insert_sort ([4, 7, 3, 9], s).

Спочатку Турбо-Пролог застосовує приведене правило до вихідного списку. Вихідний список в цей момент ще не визначений. Для задоволення першого правила insert_sort (L1,L2) L1 і L2 повинні бути пустими, другий варіант правила трактує список як комбінацію голови і його хвоста. Внутрішні уніфікаційні процедури Турбо-Прологу намагаються зробити L1 пустим. Вилучення елементів L1 починається з голови і здійснюється рекурсивно. Ця частина здійснює знищення елементів із списку в стек.

По мірі того, як Турбо-Пролог намагається задовільнити перше із правил, проходять рекурсивні виклики insert_sort, при цьому значеннями *X* послідовно стають всі елементи вихідного списку, які потім розміщуються в стек. В результаті цього вихідний список стає пустим. Тепер перший варіант правила insert_sort проводить обнулення вихідного списку і таким чином правилу надається форма insert_sort ([j,i]). Надалі Пролог намагається задовільнити друге правило з тіла правила insert_sort - правило insert. Змінній *X* спочатку присвоюється перше взяте зі стеку значення 9 і правило insert приймає наступну форму: (insert,9[ ],).

Тепер задовільнимо друге правило із тіла insert_sort, тобто відбувається повернення на одне коло рекурсії. Із стеку береться наступний елемент 3 та тестується перший елемент insert. Це правило неуспішне, так як 3>9-? та неуспішний перший варіант insert. При допомозі другого варіанту insert 3 встановлюється в другий список зліва від 9.

insert(3,[9],[3,9])

Далі відбувається повернення до insert_sort, який має вигляд insert_sort(3,[9],[3,9]), на наступному колі рекурсії відбувається вибір із стеку 7. 7>3 - це правило успішне, тому елемент 3 заноситься в стек та insert викликається рекурсивно ще раз, але уже з хвостом списку 9. Так як правило 7>9 не успішне, то використовується другий варіант insert (завершується успішно), далі відбувається повернення на попереднє коло рекурсії, в результаті 7 поміщається в вихідний список 3 і 9.

При поверненні ще на одне коло рекурсії insert_sort і із списку вилучається 4.

8.3.5. Компонування даних у список

findall (Variable, Predicate, List) - збирає дані із БД у список.

List - ім’я змінної вихідного списку.

Predicate - ім’я предиката, який використовується для вибірки даних.

Variable - ім’я об’єкта із предиката, який використовується в якості елемента списка.

Цей предикат дозволяє перенести всі значення Variable, які визначені в Predicate, в список List в тому порядку, в якому вони зустрічаються в програмі. Потім List можна використовувати як завгодно.

Predicate - це або звичайний предикат, або предикат БД.

Приклад:

```
domains
        points = real
        list = points *
        name = string
database
    football (name, points)
clauses
        football (“динамо”, 113.5).
        . . . . . .

        findall (Points, football ( _ , Points), list).
```

Лекція 9.

Тема 9. Приклади програмування в Турбо-Пролозі

9.1. База даних футбольної команди

База даних футбольної команди складається з наступних модулів [9]:

1. Головного модуля.
2. Модуля для введення даних.
3. Модуля для вилучення даних.
4. Модуля для вибірки даних.
5. Модуля закінчення роботи з програмою.

Головний модуль програми do_mbase є одночасно і метою програми:

```
do_mbase :–
assert_database,
makewindow(1, 7, 7, “PRO FOOTBALL DATABASE”, 0, 0, 25, 80),
menu,
clear_database.
```

Модуль засилає в базу інформацію з тверджень player, створює вікно, висвічує меню і очищає БД по закінченні роботи програми.

Початковий вміст БД задається за допомогою тверджень з використанням статичних предикатів. Правилом для занесення в базу цієї інформації служить:

```
assert_database :–
p1ayer(P_name, T_name, P_number, Pos, Ht, Wt, Exp, College),
assertz(dplayer(P_name, T_name, P_number, Pos, Ht, Wt, Exp, College)),
fail.
assert_database :– !.
```

Цей предикат використовує метод відкату після невдачі для перебору всіх тверджень player.

Предикат очищення бази даних такий:

```
c1ear_database :–
retract(dp1ayer(_,_,_,_,_,_,_,_)),
fail.
clear_database :– !.
```

Оскільки об'єкти тверджень dplayer в цьому правилі не представляють інтересу, то використовуються анонімні змінні.

Меню призначене для зручності користувача у виборі програмних функцій. Для забезпечення екранного простору, який відповідає вимогам, створюється вікно на весь екран (25 стрічок, 80 колонок). Модуль menu висвічує чотири доступні користувачу опції:

1. Add a player to database (занесення в БД нової інформації про гравців).
2. Delete a player from database (вилучення інформації про гравців).
3. View a player from database (видача інформації на екран).
4. Quit from this program (закінчення роботи з програмою).

Модуль menu в основному складається з предикатів write, які висвічують на екрані перераховані вище опції. Зірочки використовуються тут для виділення простору, що містить меню. У модулі присутні предикати write, що створюють цей бордюр із зірочок, і предикат, що запитує у користувача ціле число в діапазоні від 1 до 4.

Модуль menu повністю відповідає поставленим до нього в проекті програми вимогам:

```
menu :–
repeat,
clearwindow,
write("*****************************************"), nl,
write("*                                       *"), nl,
write("*     1.    Add a player to database     *"), nl,
write("*     2.    Delete a player from database      *"), nl,
write("*     3.    View a player from database        *"), nl,
write("*     4.    Quit from this program       *"), nl,
write("*                                       *"), nl,
write("*****************************************"), nl, nl,
write("Please enter your choice,1, 2, 3 or 4:"),
readint(Choice), nl,
process(Choice),
Choice = 4,
!.
```

Модуль для введення даних (побудований на основі правила process(1)) створює вікно для тексту, вимагає у користувача ввести дані з клавіатури, зчитує їх і заносить в БД нове твердження dplayer. Далі модуль забирає створене вікно і повертає управління головному меню.

Окреме, менше за розміром, вікно створюється для забезпечення діалогу з програмою. За предикатом makewindow, що створює це вікно, йдуть предикати

write, readln і readint, які інформують користувача про те, які дані він повинен ввести, і зчитують ці дані з клавіатури:

```
process(1) :–
makewindow(2, 7, 7, “Add Player to DATABASE”, 2, 20, 18, 58),
shiftwindow(2),
write(“Enter player name:”),
read1n(P_name),
write(“Enter team:”).
read1n(T_name),
```

Аналогічно, за допомогою write, readln і readint вводяться номер гравця, його позиція, ріст, вага, стаж виступів і університет.

За предикатами write, readln і readint слідує предикат assertz. Цей предикат вміщує нові твердження dplayer услід за тими, що вже є.

Об'єктами цього нового твердження є значення, присвоєні змінним P_name, T_name, P_number і т.д.

```
assertz(dp1ayer(P_name, T_name, P_number, Pos, Ht, Wt, Exp, College)),
write(P_name, “has been added to the database.”),
nl, !,
write(“Press space bar.”),
readchar(_),
removewindow.
```

Останні рядки сигналізують про закінчення процесу введення і забирають додаткове вікно.

Модуль для вилучення даних побудований на основі правила process(2). Це правило, так само як і правило process(l), створює своє власне вікно, запитує у користувача ім'я гравця і вилучає з БД твердження, що містить інформацію про нього. Після очищення вікна управління знову передається головному меню.

Услід за предикатами, що створюють вікно і зсувають його, йдуть предикати, що запитують ім'я гравця. Введене користувачем значення присвоюється змінній P_name:

```
process(2) :–
makewindow(3, 7, 7, “Delete Player from DATABASE”, 10, 30, 7, 40),
shiftwindow(3),
write(“Enter name to DELETE:”),
read1n(P_name).
```

Наступна частина правила здійснює операцію вилучення твердження з БД, посилає коротке повідомлення про це користувачеві, чекає натиснення ним довільної клавіші і забирає з екрана додаткове вікно.

```
retract(dp1ayer(P_name, _, _, _, _, _, _, _)),
write(P_name, “ has been deleted from the database.”),
nl, !,
write(“Press space bar.”),
readchar(_),
removewindow.
```

Для вилучення з бази вибраного користувачем твердження застосований предикат retract. Оскільки будь-яка інша інформація про гравця, крім його імені, не представляє інтересу в даній операції, то на місці всіх інших об'єктів стоять анонімні змінні.

Призначенням модуля для вибірки даних (process(3)) є пошук даних, що містяться в БД. Цей модуль створює своє власне вікно, а потім запитує ім'я гравця. Якщо в БД знаходиться твердження, що містить введене ім'я, модуль проводить вибірку даних і виводить їх на екран в зручному форматі:

```
process(3) :–
makewindow(4, 7, 7, “View Window”, 7, 30, 16, 47),
shiftwindow(4),
write(“Enter name to view: ”),
read1n(P_name),
dplayer(P_name, T_name, P_number, Pos, Ht, Wt, Exp, College).
```

Предикат dplayer шукає потрібне твердження в БД і вибирає значення цілих та стрічкових типів по кожному пункту, що запитується. Цілий ряд предикатів write потім виводить отримані значення:

```
nl, write(“ NFL League Player”), nl,
nl, write(“ Player Name: ”, P_name),
nl, write(“ Team Name:  ”, T_name),
nl, write(“ Position:              ”, Pos),
nl, write(“ Player Number:      ”, P_number),
nl, write(“ Player's Height:      ”, Ht, “ ft-in”),
nl, write{“ Player's Weight:      ”, Wt, “ lb”),
nl, write(“ Player's NFL-exp:   ”, Exp, “ year(s)”),
nl, write(“ Player's College:     ” College),
nl, nl, !,
nl, write(“Press space bar”),
readchar(_),
removewindow.
```

Якщо в БД відсутнє твердження з введеним користувачем ім’ям гравця, програма видає повідомлення про помилку. Вікно для цього повідомлення повинно розташовуватися на видному місці, наприклад, в центрі екрану. Варіант третього правила (process (3)), що відповідає за видачу повідомлення про помилку, виглядає так:

```
process(3) :–
makewindow(5, 7, 7, “No Luck”, 14, 7, 5, 60),
shiftwindow(5),
write(“Can't find that player in the database.”), nl,
write(“Sorry, bye!”),
nl, !,
write(“Press space bar.”),
readchar(_),
removewindow,
shiftwindow(1).
```

Модуль закінчення роботи (process(4)) забезпечує нормальне закінчення сеансу роботи з БД. Цей модуль не створює свого власного вікна. Нове вікно тут зайве, оскільки повідомлення дуже короткі і не вимагають багато місця на екрані. Модуль, однак, вимагає від користувача чіткої відповіді на питання, чи хоче він закінчити роботу з програмою:

```
process(4) :–
write(“Are you sure want to quit (y/n)”),
readln(Answer),
frontchar(Answer, ‘y’, _), !.
```

Модуль реакції на помилку дозволяє реагувати на допущені користувачем помилки при введенні. Якщо користувач введе число, менше 1 або більше 4, буде успішним одне з правил:

```
process(Choice) :–
Choice < 1,
error.
process(Choice) :–
Choice > 4,
error.
```

Обидва правила викликають модуль error:

```
error :–
write(“Please enter a number from 1 to 4.”),
write(“(Press the space bar to continue)”),
readchar(_).
```

Лістинг програми “ База даних футбольної команди ” приведений нижче.

Програма “ База даних футбольної команди ”

```
domains
  p_name, t_name, pos, height, college  = string
  p_number, weight, nfl_exp             = integer

database
  dplayer(p_name, t_name, p_number, pos,
    height, weight, nfl_exp, college)
predicates
  repeat
  do_mbase
  assert_database
  menu
  process(integer)
  clear_database
  p1ayer(p_name,t_name,p_number,pos,height,weight,nfl_exp,college)
  error
goal
  do_mbase.
clauses
  repeat.
```

```
  repeat :– repeat.
/* База даних футбольної команди */
  player(“Dan Marino”, “Miami Dolphins", 13, “QB”,
    “6-3”, 215, 4, “Pittsburgh”).
  player(“Richart Dent”, “Chicago Bears”, 95, “DE”,
    “6-5”, 263, 4, “Tennessee State”).
  player(“Bernie Kosar”, “Cleveland Browns”, 19, “QB”,
    “6-5”, 210, 2, “Miami”).
  player(“Doug Cosbie”, “Dallas Cowboy”, 84, “TE”,
    “6-6”, 235, 8, “ Santa Clara”).
  player(“Mark Malone”, “Pittsburgh Steelers”, 16, “QB”,
    “6-4”, 223, 7, “Arizona State”).
/* кінець початкових даних */
  assert_database :–
    player(P_name, T_name, P_number, Pos, Ht, Wt, Exp, College),
    assertz(dplayer(P_name,T_name,P_number,Pos,Ht,Wt,Exp,College)),
    fail.
  assert_database :– !.
  clear_database :–
    retract(dplayer(_,_,_,_,_,_,_,_)),
    fail.
  clear_database :– !.
```

/*Діалог з цією БД здійснюється за принципом меню. При цьому використовуються віконні засоби Турбо-Прологу. Базуючись на запиті користувача, СУБД активізує відповідні процеси для задоволення цього запиту. Меню можна розширити за рахунок включення нових функцій. */

```
/* задання мети у вигляді правила */
  do_mbase :–
    assert_database,
    makewindow(l, 7, 7, “ PRO FOOTBALL DATABASE ”, 0, 0, 25, 80),
    menu,
    clear_database.
  menu :–
    repeat,
    clearwindow,
    write(“*****************************************”), nl,
    write(“*                                       *”), nl,
    write(“*   1.     Add a player to database      *”), nl,
    write(“*   2.     Delete a player from database       *”), nl,
    write(“*   3.     View a player from database         *”), nl,
    write(“*   4.     Quit from this program        *”), nl,
    write(“*                                       *”), nl,
    write(“*****************************************”), nl, nl,
    write(“Please enter your choice,1, 2, 3 or 4:”),
    readint(Choice), nl,
```

```
    process(Choice),
    Choice = 4,
    !.
/* Додавання інформації про гравця в БД */
  process(1) :–
    makewindow(2, 7, 7, “Add Player to DATABASE”, 2, 20, 18, 58),
    shiftwindow(2),
    write(“Enter player name: ”),
    readln{P_name),
    write(“Enter team: ”),
    readln(T_name),
    write(“Enter player number: ”),
    readint(P_number),
    write(“Enter position: ”),
    readln(Pos),
    write(“Enter height: ”),
    readln(Ht).
    write(“Enter weight: ”),
    readint(Wt),
    write(“Enter NFL exp: ”},
    readint(Exp),
    write(“Enter college: ”),
    readln(College),
    assertz(dplayer(P_name,T_name,P_number,Pos,Ht,Wt,Exp,College)),
    write(P_name, “ has been added to the database.”),
    nl, !,
    write(“Press space bar.”),
    readchar(_),
    removewindow.
/* Вилучення інформації про гравця з БД */
  process(2) :–
    makewindow(3,7,7,“ Delete Player from DATABASE ”, 10, 30, 7, 40),
    shiftwindow(3),
    write(“Enter name to DELETE:”),
    readln(P_name),
    retract(dp1ayer(P_name, _, _, _, _, _, _, _)),
    write(P_name, “ has been deleted from the database.”),
    nl, !,
    write(“Press space bar.”),
    readchar(_),
    removewindow.
/* Перегляд інформації про гравця */
  process(3) :–
    makewindow(4, 7, 7, “ View Window ”, 7, 30, 16, 47),
    shiftwindow(4),
```

```
    write("Enter name to view: "),
    readln(P_name),
    dplayer(P_name, T_name, P_number, Pos, Ht, Wt, Exp,College),
    nl, write(" NFL League Player"), nl,
    nl, write(" Player Name:      ", P_name),
    nl, write(" Team Name:        ", T_name),
    nl, write(" Position:         ", Pos),
    nl, write(" Player Number:    ", P_number),
    nl, write(" Player's Height:  ", Ht, "ft-in"),
    nl, write{" Player's Weight:  ", Wt, " lb"),
    nl, write(" Player's NFL-exp: ", Exp, "year(s)"),
    nl, write(" Player's College: " College),
    nl, nl, !,
    nl, write("Press space bar"),
    readchar(_),
    removewindow.
  process(3) :–
    makewindow(5, 7, 7, "No Luck", 14, 7, 5, 60),
    shiftwindow(5),
    write("Can't find that player in the database."), nl,
    write("Sorry, bye!"),
    nl, !,
    write("Press space bar."),
    readchar(_),
    removewindow,
    shiftwindow(1).
/* Вихід з діалогу */
  process(4) :–
    write("Are you sure want to quit (y/n)"),
    readln(Answer),
    frontchar(Answer,'y', _), !.
/* Неправильне звертання до БД */
  process(Choice) :–
    Choice < 1,
    error.
  process(Choice) :–
    Choice > 4,
    error.
  error :–
    write("P1ease enter a number from 1 to 4."),
    write("(Press the space bar to continue)"),
    readchar(_).
```

9.2. Експертна система вибору породи собак

Дана експертна система побудована на логіці першого порядку [9]. Головний модуль do_expert_job викликає модуль show_menu. Цей модуль пропонує користувачу вибрати програмну функцію. Відповідь користувача зчитується в цілочисельну змінну Choice(вибір), і виклик process(Choice) приводить до виконання відповідної програмної функції. Модулі process(0) і process(2) служать для виходу з програми. Модуль process(1) викликає модуль do_consulting. Різноманітні модулі, що викликаються do_consulting, видають породи собак, виконують виведення й оновлюють робочі дані. Модуль eval_reply забезпечує зручне завершення діалогу консультації.

Експертна система, що базується на логіці, складається з бази знань, що містить твердження логіки предикатів. Твердження мають одну з двох форм: rule або cond. База знань наведена нижче:

```
topic(“dog”).
topic(“short-haired dog”).
topic(“long-haired dog).
ru1e(l, “dog”,                “short-haired dog”,   [1]          ).
ru1e(2, “dog”,                “long-haired dog”,    [2]          ).
ru1e(3, “short-haired dog”, “English Bulldog”,      [3, 5, 7]    ).
ru1e(4, “short-haired dog”, “Beagle”,               [3, 6, 7]    ).
ru1e(5, “short-haired dog”, “Great Dane”,           [5, 6, 7, 8] ).
ru1e(6, “short-haired dog”, “American Foxhound”,    [4, 6, 7]    ).
ru1e(7, “long-haired dog”, “Cocker Spaniel”,        [3, 5, 6, 7] ).
ru1e(8, “long-haired dog”, “Irish Setter”,          [4, 6,]      ).
ru1e(9, “long-haired dog”, “Collie”,                [4, 5, 7]    ).
ru1e(10, “long-haired dog”, “St. Bernard”,          [5, 7, 8]    ).
cond(l, “short-haired                ”).
cond(2, “long-haired                 ”).
cond(3, “height under 22 inches      ”).
cond(4, “height under 30 inches      ”).
cond(5, “low-set tail                ”).
cond(6, “longer ears                 ”).
cond(7, “good natured personality    ”).
cond(8, “weight over 100 lb          ”).
```

Останній об'єкт у твердженні rule – список цілих чисел. Список містить номери умов, що характеризують кожну породу собаки в базі знань. Пропозиції cond містять всі можливі характеристики собак.

Експертна система для вибору породи собаки виглядає так:

```
domains
database
        xpositive(symbol, symbol)
        xnegative(symbol, symbol)
predicates
        do_expert_job
```

```
        do_consulting
        ask(symbol, symbol)
        dog_is(symbol)
        it_is(symbol)
        positive(symbol, symbol)
        negative(symbol, symbol)
        remember(symbol, symbol, symbol)
        clear_facts
goal
        do_expert_job.
clauses
        do_expert_job:-
                makewindow(1, 7, 7, "AN EXPERT SYSTEM", 1, 16, 22, 58),
                nl, write("    ************************************"),
                nl, write("    WELCOME TO A DOG EXPERT SYSTEM       "),
                nl, write("                                         "),
                nl, write("    This is a dog identification system."),
                nl, write("    Please answer the question about     "),
                nl, write("    the dog you would like by typing in  "),
                nl, write("    'yes' or 'no'.                       "),
                nl, write("    ************************************"),
                nl, nl,
                do_consulting,
                write("Press space bar."), nl,
                readchar(_),
                removewindow,
                exit.
        do_consulting:-
                dog_is(X), !, nl,
                write("the dog you have indicated is a(n) ", X, "."), nl,
                clear_facts.
        do_consulting:-
                nl, write("Sorry, I can't help you !"),
                clear_facts.
        ask(X, Y):-
                write("  Question:- ", X," it ", Y, "? "),
                readln(Reply),
                remember(X, Y, Reply).

        positive(X, Y):-
                xpositive(X, Y), !.
        positive(X, Y):-
                not(negative(X, Y)), !,
                ask(X, Y).
        negative(X, Y):-
```

```
        xnegative(X, Y), !.
remember(X, Y, yes):-
        asserta(xpositive(X, Y)).
remember(X, Y, no):-
        asserta(xnegative(X, Y)),
        fail.
clear_facts:-
        retract(xpositive(_, _)),
        fail.
clear_facts:-
        retract(xnegative(_, _)),
        fail.

dog_is("English Bulldog"):-
        it_is("short-haired dog"),
        positive(has, "height under 22 inches"),
        positive(has, "low-set tail"),
        positive(has, "good natured personality"), !.
dog_is("Beagle"):-
        it_is("short-haired dog"),
        positive(has, "height under 22 inches"),
        positive(has, "longer ears"),
        positive(has, "good natured personality"), !.
dog_is("Great Dane"):-
        it_is("short-haired dog"),
        positive(has, "low-set tail"),
        positive(has, "good natured personality"),
        positive(has, "weight over 100 lb"), !.
dog_is("American Foxhound"):-
        it_is("short-haired dog"),
        positive(has, "height under 30 inches"),
        positive(has, "longer ears"),
        positive(has, "good natured personality"), !.
dog_is("Cocker Spaniel"):-
        it_is("long-haired dog"),
        positive(has, "height under 22 inches"),
        positive(has, "low-set tail"),
        positive(has, "longer ears"),
        positive(has, "good natured personality"), !.
dog_is("Irish Setter"):-
        it_is("long-haired dog"),
        positive(has, "height under 30 inches"),
        positive(has, "longer ears"), !.
dog_is("Collie"):-
        it_is("long-haired dog"),
```

```
        positive(has, "height under 30 inches"),
        positive(has, "low-set tail"),
        positive(has, "good natured personality"), !.
dog_is("St. Bernard"):-
        it_is("long-haired dog"),
        positive(has, "low-set tail"),
        positive(has, "good natured personality"),
        positive(has, "weight over 100 1b"), !.
it_is("short-haired dog"):-
        positive(has, "short-haired"), !.
it_is("long-haired dog"):-
        positive(has, "long-haired"), !.
```

Ця програма видає початкове меню, пропонуючи користувачу вибір між consultation (консультацією) й exit from the system (виходом з системи). Якщо користувач вибирає консультацію, то між користувачем і системою відбувається діалог. Після цього користувачу повідомляється результат. Результатом є або вибрана порода, або повідомлення Sorry, I can’t help you (Вибачте, я не можу допомогти вам).

Література

1. Адаменко А.Н., Кучуков А.М. Логическое программирование и Visual Prolog. – СПб.: БХВ-Петербург, 2003. – 992 с.
2. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG, 3-е издание. : Пер. с англ. – М.: Издательский дом “Вильямс”, 2004. – 640 с.
3. Ин Ц., Соломон Д. Использование Турбо-Пролога. – М.: Мир, 1993. – 608 с.
4. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. – М.: Мир, 1990. – 235 с.
5. Стобо Дж. Язык программирования Пролог. – М.: Мир, 1993. – 368 с.
6. Змитрович А.И. Интеллектуальные информационные системы. – Минск, ТетраСистемс, 1997. – 367 с.
7. Метакидес Г., Нероуд А., Принципы логики и логического программирования. Москва, "Факториал", 1998, 288 с.
8. Марселлус Д. Программирование экспертных систем на Турбо Прологе: Пер. с англ. – М.: Финансы и статистика, 1994. – 256 с.
9. http://nlping.com/example4/example4.htm
10. http://llt.msu.edu/vol5num1/furstenberg/default.html