МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ
Національний авіаційний університет

# МЕТОДИ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Лабораторний практикум
для студентів напряму підготовки
6.050101 «Комп'ютерні науки»

Київ 2012

МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ
Національний авіаційний університет

# МЕТОДИ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

Лабораторний практикум
для студентів напряму підготовки
6.050101 «Комп'ютерні науки»

Київ 2012

УДК 004.8(076.5)
ББК З 813я7
М 545

Укладач: *А.С. Савченко*

Рецензенти: *Абакумов В.Г.* – д-р техн. наук, проф. (Національний технічний університет України «КПІ»);
*Беркман Л.Н.* – д-р техн. наук, проф. (Державний університет інформаційно-комунікацій-них технологій);

*Затверджено на засіданні вченої ради Національного авіаційного університету (протокол № 6/12 від 05.11.2012 р.).*

**Методи та системи штучного інтелекту:** Лабораторний практикум для студентів напряму підготовки 6.050101 «Комп'ютерні науки» / Уклад.: А.С.Савченко. – К.: НАУ, 2012. – 40 с.

М 545

Викладено сутність мови логічного програмування *Prolog*, особливості написання програм у середовищі *Visual Prolog*, зокрема, досліджено механізм управління пошуком цілі, реалізацію числових обрахунків, оброблення лінійних списків та механізм наслідування. Розглянуто принципи побудови експертних систем та роботи нейронної мережі, а також особливості семантичного моделювання в системах штучного інтелекту.

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

## ВСТУП

Лабораторні роботи виконуються відповідно до навчальної програми дисципліни «Методи та системи штучного інтелекту» напряму підготовки 6.050101 «Комп'ютерні науки».

Мета виконання лабораторних робіт – набуття студентами практичних навичок та закріплення ними теоретичних знань про особливості написання програм у середовищі *Visual Prolog*, принципи побудови експертних систем, програмування роботи нейронної мережі та особливості семантичного моделювання в системах штучного інтелекту.

Вивчення матеріалів дисципліни побудовано відповідно до вимог кредитно-модульної системи оцінювання знань. Програмою дисципліни передбачено виконання чотирьох лабораторних робіт першого модулю і чотирьох робіт другого. На виконання та захист кожної роботи відводиться чотири академічні години, крім першої роботи першого модуля, на яку заплановано шість академічних годин.

За цей час студент повинен:

- одержати у викладача індивідуальний варіант завдання і виконати його;
- підготувати протокол звіту про лабораторну роботу;
- відповісти на контрольні запитання.

Звіт про виконання лабораторної роботи має містити: титульний аркуш, мету роботи, основні теоретичні відомості, порядок виконання лабораторної роботи, висновки.

До оформлення звіту ставляться такі вимоги:

- робота оформлюється на аркушах формату А4, або в окремому зошиті з лабораторних робіт;

– на титульному аркуші мають бути вказані назва дисципліни, тема роботи, ким виконано роботу (прізвище, ім'я, по батькові, номер групи, факультет), ким прийнято роботу;
– основна частина звіту має містити вхідні дані, текст програми, отримані результати, висновки.

Лабораторний практикум складено на основі матеріалу лекцій, які читаються для студентів напряму підготовки 6.050101 «Комп'ютерні науки» в Національному авіаційному університеті.

## Модуль 1. ПОДАННЯ ЗНАНЬ У СИСТЕМАХ ШТУЧНОГО ІНТЕЛЕКТУ

Модуль охоплює такі теми: «Основні поняття та означення штучного інтелекту», «Способи подання інтелектуальної задачі та методи пошуку розв‘язків», «Подання знань у системах штучного інтелекту», лабораторні роботи 1.1 – 1.4.

Лабораторна робота 1.1

## ДОСЛІДЖЕННЯ СЕРЕДОВИЩА ПРОГРАМУВАННЯ VISUAL PROLOG

**Мета роботи** – ознайомитись із середовищем програмування *Visual Prolog*, вивчити структуру *Prolog*-програми та набути навичок складання програм мовою *Prolog*.

### 1. Основні теоретичні відомості

У мові *Prolog* основною є декларативна компонента, тому вона призначена не стільки для оброблення даних, як для оброблення фактів і декларативних правил. *Prolog* є мовою числення предикатів. Сукупність фактів, що описують визначену предметну галузь, складають базу знань. Після запиту логічні методи забезпечують отримання нових фактів з фактів, поданих у базі знань, тобто намагаються довести істинність логічного виразу.

### 1.1. Середовище програмування *Visual Prolog*

Запуск середовища *Visual Prolog* виконується таким чином: *Пуск ⇨Програми ⇨Visual Prolog 5.2 ⇨Visual Prolog 32*. Вигляд вікна середовища *Visual Prolog* представлений на рис. 1.1.

**Запуск і тестування програми.** За допомогою команди меню *File* | *New* (*F7*) створюється новий файл *noname.pro*. Для виконання програми достатньо у вікні надрукувати текст

```
GOAL Write("Привіт!"), nl.
```

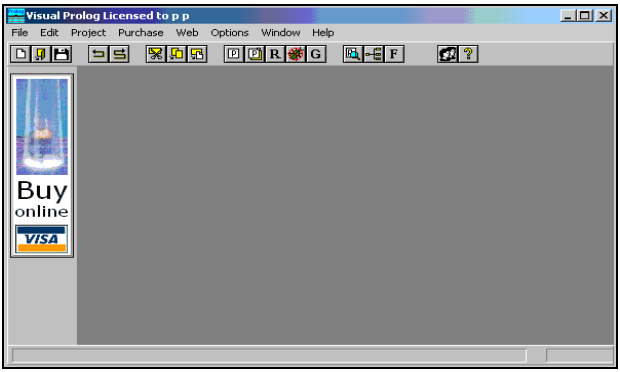


Рис. 1.1. Вигляд основного вікна середовища *Visual Prolog*

Активацією команди *Project* | *Test Goal* (*Ctrl+G*) виконується GOAL. Результат виконання команди GOAL наведено на рис. 1.2.

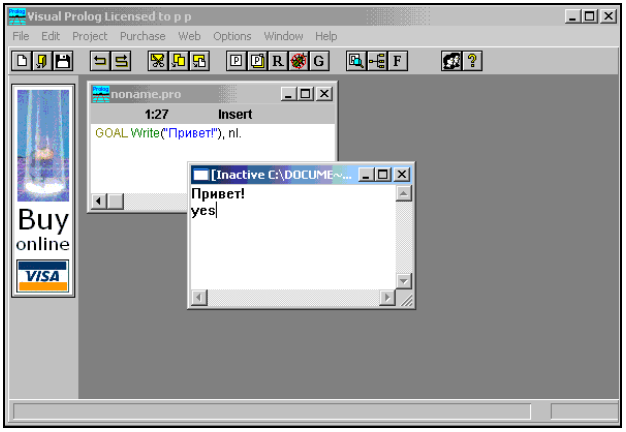


Рис. 1.2. Вигляд тестового вікна виконання команди GOAL

Результат виконання програми буде виведений в окремому вікні (*Inactive ...*), яке необхідно закрити перед тим, як тестувати наступну програму.

**Оброблення помилок.** У випадку виявлення помилок у програмі середовище *Visual Prolog* відобразить вікно *Errors* (*Warnings*). Вигляд списку виявлених помилок показано на рис. 1.3.

Подвійне натискання на напис помилки дозволяє переходити на її місце у вихідному коді програми.

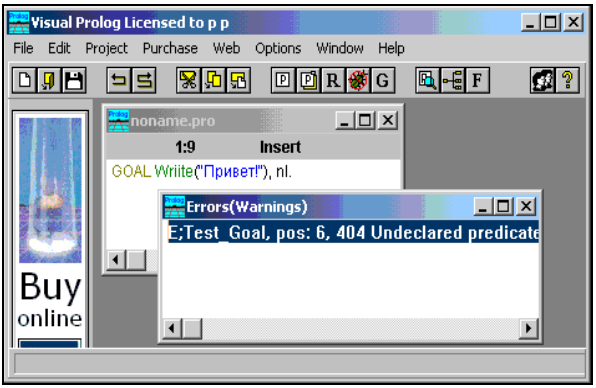


Рис. 1.3. Вигляд вікна з виявленими помилками

### 1.2. Структура *Prolog*-програми

Програма може включати такі розділи:

CONSTANTS

опис констант;

DOMAINS

опис імен та структур об'єктів;

PREDICATES

опис предикатів, тобто назв відносин, що існують між об'єктами (ім'я, тип, кількість аргументів);

DATABASE

опис зовнішньої бази даних;

CLAUSES

опис фактів і правил для конкретної предметної області. Використовуються речення двох типів, кожне з яких завжди закінчується крапкою:

- *речення-факти* – являють собою одиночну істинну ціль, наприклад, man(ivan);
- *речення-правила* – це твердження, що складаються з *головної цілі* і *хвостових цілей*, які істинні за деяких умов. Наприклад: mother(X,Y):–parent(X,Y),woman(X);

GOAL

- опис цілі (запиту); включається безпосередньо в програму.

### 1.3. Складання програми

Складемо програму, яка описує генеалогічне дерево деякої родини (рис. 1.4).

Між членами родини існують відношення «*A* родич *B*». Крім того, кожному об'єкту притаманна властивість «чоловік», «жінка». Запишемо ці співвідношення у вигляді програми.

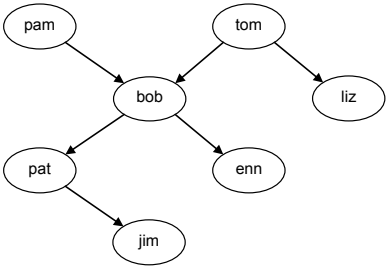


Рис. 1.4. Приклад генеалогічного дерева родини

У секції domains вказуємо тип аргументів, які будуть використовуватись.

```
DOMAINS
    name=symbol
    age=integer
```

У секції предикатів описуємо існуючі відношення *родич (ім'я родича, ім'я дитини)*, *чоловік (ім'я, вік)*, *жінка (ім'я, вік)*.

Указуємо режим детермінізму фактів: determ – факт може бути в одному екземплярі або не бути зовсім; nondeterm (режим за замовчуванням) – фактів може бути скільки завгодно, навіть жодного; single – факт лише один. Як правило використовується для лічильників, при оголошенні лічильнику присвоюється початкове значення. Single-факти або ще їх називають факти-змінні, допускають руйнівне присвоювання і відіграють роль змінних з «глобальною» областю видимості зі справжніх імперативних мов. Використовуючи їх, можна максимально пришвидшити роботу програми з підрахунку чого-небудь. Тип факту-змінної може бути не лише число, рядок, символ.

```
PREDICATES
    nondeterm  parent(name, name)
    nondeterm  man(name, age)
    nondeterm  woman(name, age)
```

У секції clauses описуємо всі існуючі відношення у родині. Порядок опису відношень не має значення, але відношення одного типу мають бути записані підряд, інакше програма вкаже на поми-

лку. Тобто, спочатку можна записати всі відношення «родич», потім всі відношення «чоловік», а потім – «жінка»:

```
CLAUSES
    parent(pam, bob).
    parent(tom, bob).
    parent(tom, liz).
    parent(bob, pat).
    parent(bob, enn).
    parent(pat, jim).

    man(tom, 67).
    man(bob, 45).
    man(jim, 1).
    woman(pam, 60).
    woman(liz, 40).
    woman(pat, 23).
    woman(enn, 18).
```

Для перевірки програми, необхідно запустити її на виконання. Список типових помилок:

- неописаний предикат – предикат не вказаний у секції predicates, але використовується у тілі програми;
- предикат описаний, але не використовується, – предикат описаний у секції predicates, але не використовується у тілі програми;
- немає крапки у кінці речення в секції clauses;
- неправильно заданий тип аргумента у секції domains;
- змінна, описана у заголовку правила, не використовується в тілі правила.

Для коментування використовують такі конструкції:

% рядковий коментар;

/* блочний коментар */.

### 1.4. Складання запитів

Запит у *Prolog* називається ***ціллю*** (goal), яку необхідно вивести з наявних фактів. Цілі бувають прості, складні, константні, зі змінними.

**Прості запити**. У *простій константній цілі* аргументи предикатів задано у вигляді констант, тому відповідь може бути лише «так» або «ні». Наприклад:

```
GOAL
    parent(pam, bob).          Yes
Чи є Пам родичем Боба?         Відповідь – так.
```

У *простих запитах зі змінними* замість аргументів можуть стояти змінні – це послідовність літер або цифр, які обов'язково починаються з великої літери. Наприклад:

| | |
|---|---|
| GOAL<br>parent(Who, bob). | X=pam<br>X=tom<br>2 Solution |
| Хто батьки Боба? | Два розв‘язки: Пам і Том. |
| GOAL<br>parent(X, Y). | Хто кому родич?<br>Будуть виведені всі батьки та їх діти. |

Для виведення лише однієї з кількох змінних використовується *анонімна змінна*, яка позначається символом нижнього підкреслення «_». Наприклад:

| | |
|---|---|
| GOAL<br>parent(_,Child). | Вивести лише імена дітей. |

**Складні запити.** Складні запити містять в собі декілька цілей, розділених комою. Цілі можуть бути константними або зі змінними. Наприклад:

| | |
|---|---|
| GOAL<br>parent(Who,bob),<br>man(Who,_). | Хто родич Боба?<br>Чи є цей родич чоловіком?<br>Тобто: Хто батько Боба? |

### 1.6. Складання правил

Для зменшення коду програми та спрощення опису предметної галузі застосовують речення-правила, які дозволяють визначати нові відношення через уже існуючі. Такі речення складаються з головної (зліва) і хвостової (справа) частин. У хвостовій частині відношення записують через кому (,) – відношення «І», або крапку з комою (;) – відношення «АБО».

Запишемо правило для відношення *батько (ім’я батька, ім’я дитини)* через існуючі відношення *родич (ім’я родича, ім’я дитини)* та *чоловік (ім’я, вік)*.

Правило читається таким чином: «*X* буде батьком *Y*, якщо *X* – родич *Y*, і *X* – чоловік» і записується як:

```
father (X,Y):–parent(X,Y),man(X,_).
```

Граф даного правила показано на рис. 1.5. Позначення: квадрат – особа чоловічої статі, трикутник – особа жіночої статі, коло – стать не вказана.

Доповнюємо програму новим предикатом *father* та правилом:

```
PREDICATES
    nondeterm father(symbol, symbol)
CLAUSES
    father(X, Y):–parent(X, Y), man(X, _).
```

У відповідь на запит «father(F, Ch) – хто кому батько?» буде видано два розв‘язки.

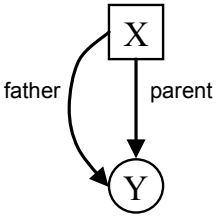


Рис. 1.5. Граф для відношення «батько»

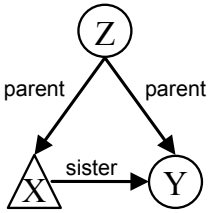


Рис. 1.6. Граф для відношення «сестра»

Особливості предметної області слід враховувати при складанні правил. Наприклад, для відношення *сестра* (*ім’я сестри, ім’я родича*) слід врахувати, що на місці першого і другого аргументів не може бути одна й та ж людина, тобто не можна бути сестрою самій собі – це помилка (рис.1.6). Тому треба явно вказати, що це різні об’єкти, тобто $X<>Y$:

```
sister(X,Y):-parent(Z,X),parent(Z,Y),woman(X,_),X<>Y.
```

**2. Порядок виконання роботи**

2.1. Відповідно до п.п. 1.1–1.2 дослідити середовище програмування *Visual Prolog*.

2.2. Згідно з п.п. 1.3 зобразити генеалогічне дерево власної родини (не менше 10 осіб – три покоління) та скласти за ним програму. База фактів має містити відомості про особу (прізвище, ім’я, рік народження, стать) та родинні відношення. Використати предикати: родич, чоловік, жінка, подружжя.

2.3. За п.п. 1.5 скласти правила для відношень згідно з варіантом.

2.4. До кожного правила зобразити відповідний граф та зробити запис природною мовою.

2.5. Згідно з п.п. 1.4 скласти запити різних типів.

**3. Звіт.** Звіт має містити результати виконання п.п. 2.1–2.6.

**4. Варіанти.** Записати правила та зобразити графи для таких відношень.

1) матір, онучка, племінник;
2) батько, син, кузина;
3) мачуха, племінниця, дядько;
4) матір, дідусь, тітка;
5) батько, , бабуся, кузен;
6) онук, брат, матір;
7) пасинок, дитина, племінниця.

**Контрольні запитання**

1. Поясніть структуру та призначення секцій в *Prolog*-програмі.
2. Поясність, як *Prolog* розрізняє константи та змінні?
3. Поясність, як складаються речення-правила?
4. Що таке ціль у мові *Prolog* та як записуються запити різних типів?

Лабораторна робота 1.2

## ДОСЛІДЖЕННЯ МЕХАНІЗМУ УПРАВЛІННЯ ПРОЦЕСОМ ПОШУКУ ЦІЛІ

**Мета роботи** – дослідити механізм пошуку у мові *Prolog* та навчитись використовувати вбудовані предикати управління пошуком.

### 1. Основні теоретичні відомості

У мові *Prolog* реалізована стратегія пошуку по дереву в глибину. Зіставлення з ціллю виконується обходом дерева зверху вниз і зліва направо. Якщо процес зіставлення невдалий, програма повертається до найближчого альтернативного варіанта, при цьому зв'язані змінні звільняються.

У процесі пошуку розв'язку *Prolog* може виконувати такі операції: зіставлення зі зразком, зв'язування змінних, відкат і звільнення змінних.

### 1.1. Механізм пошуку

Механізм пошуку цілі у мові *Prolog* розглянемо на прикладі.

CLAUSES

father(jon, lisa).
father(bob, tom).
father(bob, pat).
man(jon). man(tom).
man(bob).
woman(sui).woman(pat).

GOAL

father(bob,X),
woman(X).

Хто дочка Боба?

Спочатку *Prolog* починає доводити першу підціль. Зразок father (bob,X) вдається зіставити з другим реченням father (bob,tom). Вільна змінна X зв'язується зі значенням tom. Підціль доведено.

*Prolog* переходить до доведення другої підцілі – woman (tom). Друга підціль не може бути доведена, оскільки такого факту немає. Неуспіх. Але розв'язок є. Просто перше зв'язування виявилося невдалим.

*Prolog* звільняє зв'язану змінну і повертається (відкочується) до доведення першої підцілі, оскільки ще є речення, з якими можна зіставляти.

Друге зіставлення зразка father (bob, X) з реченням father (bob, pat) зв'язує X зі значенням pat. Доведенням другої підцілі – woman (pat) є восьме речення. Пролог знайшов розв'язок: X = pat, 1 Solution. Дерево пошуку цілі зображено на рис. 1.7.

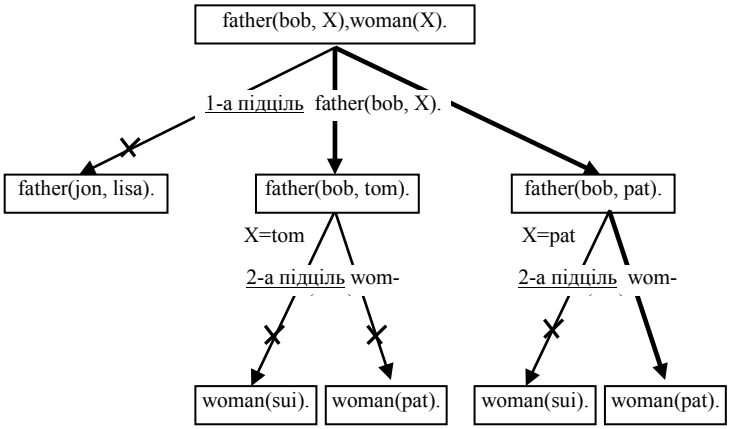


Рис. 1.7. Приклад дерева пошуку

Доведення першої підцілі father(bob, X) передбачає три варіанти. Пролог переглядає їх почергово: перший з них відкидає, другий – підходить, тому цю гілку «розгортає» до кінця. Доведення другої підцілі передбачає два варіанти. Таким чином, виникає шість можливих варіантів розв'язку. Хрестиком позначено помилкові шляхи, тобто відкат.

### 1.2. Управління пошуком за допомогою предиката *fail*

Управління пошуком розв'язків у *Prolog* реалізовано за допомогою механізму пошуку з поверненням (*backtracking*), за якого система намагається відшукати всі можливі розв'язання задачі. Механізм виведення програми запам'ятовує ті точки процесу уніфікації, у яких не були використані всі альтернативні рішення, а потім повертається в ці точки і шукає рішення по іншому шляху.

Предикат fail називають ***відкатом*** після невдачі. Він викликає штучне неуспішне завершення пошуку, що дозволяє отримати всі можливі розв'язки задачі. Особливо актуальне його використання для внутрішніх цілей, наприклад:

```
PREDICATES
    nondeterm father(symbol, symbol)
CLAUSES
    father(jon, lisa).
    father(bob, tom).
    father(bob, pat).
GOAL
    father (X,Y),
    write(X," is ",Y,"'s father\n"), fail.
```

### 1.3. Управління пошуком за допомогою предиката *cut*

Предикат ***відсікання*** cut позначається за допомогою символа **!**. Він дозволяє отримати доступ лише до частини даних, усуваючи подальші пошукові дії. У загальному випадку можна записати:

$$R :- A, B, !, C$$

Це означає, що якщо для цілей $A$ і $B$ знайдено хоча б один розв'язок, то подальше перебирання можливих варіантів значень $A$ і $B$ не потрібне. Наприклад, представлена нижче програма надрукує лише імена хлопчиків:

```
PREDICATES
    nondeterm child(symbol)
    show
GOAL
    write(«Це хлопчики:»),nl, show.
CLAUSES
    child(«Петро»).child(«Іван»).child(«Олег»).
    child(«Маша»).child(«Оля»).child(«Катя»).
    show :- child(X), write(X), nl, X=«Олег»,!.
```

**2. Порядок виконання роботи**

2.1. Для програми з лабораторної роботи 1.1 скласти запит відповідно до варіанта так, щоб були знайдені всі правильні розв‘язки.

2.2. Скласти дерево пошуку для запиту відповідно до варіанта. Перекреслити гілки, які призводять до відкату. Позначити шляхи, що приводять до правильного розв‘язку.

2.3. У разі потреби використати в програмі предикати fail і cut, пояснити їх призначення.

**3. Звіт.** Звіт має містити результати виконання п.п. 2.1–2.4.

**4. Варіанти.** Скласти запит для пошуку:

1) знайти батька, у якого двоє дітей;
2) знайти племінницю через предикати кузина і батьки;
3) знайти матір, у якої дві дитини;
4) знайти тітку з двома племінниками;
5) знайти батьків для певної особи через предикат син або дочка;
6) знайти бабусю з трьома онуками;
7) знайти онуків для певної особи через предикат бабуся або дідусь;

**Контрольні запитання**

1. Поясніть, як реалізується пошук у мові *Prolog*.
2. Поясніть, які виконуються операції при пошуку цілі.
3. Поясніть призначення предиката *fail*.
4. Поясніть призначення предиката відсікання.

Лабораторна робота 1.3

# ДОСЛІДЖЕННЯ РЕАЛІЗАЦІЇ ЧИСЛОВИХ ОБРАХУВАНЬ У VISUAL PROLOG

**Мета роботи** – вивчити методи організації рекурсивних числових обрахувань у *Visual Prolog*.

## 1. Основні теоретичні відомості

У *Visual Prolog* використовується ряд убудованих функцій для обчислення арифметичних виразів, наприклад: >, <, =, X mod Y, sqrt(X), sin(X) та ін. Наприклад, для виведення на екран даних лише про осіб віком від 20 років і старше, можна використати таку конструкцію

```
CLAUSES
    person(ivan, 24, mam).
    person(anna, 18, woman).
GOAL
    person(A,B,N),B>20.
```

Можна використовувати також власні предикати, наприклад, вираз plus(X,«1»,Z):-next(X,Z) означає, що результатом додавання одиниці до *X* буде число, наступне за *X,* тобто *Z*.

Рекурсія у мові *Prolog* важлива, оскільки в ній немає циклічних конструкцій. ***Рекурсія*** – це зведення задачі до неї ж, але меншої розмірності. Процес продовжується доти, доки її можна буде розв'язати безпосередньо.

Спосіб, за якого розв'язок складається з двох фаз: перша фаза – розбиття на підзадачі, друга фаза – розв'язок підзадач, називається ***низхідною стратегією***. Він простий, але не завжди ефективний. Головний недолік – велика витрата пам'яті. Кожен рекурсивний виклик передбачає виділення ділянки пам'яті для усіх змінних.

Стратегія розв'язання, починаючи з простого випадку, піднімаючись угору по ієрархії підзадач, поки не буде вирішено початкову задачу, називається ***висхідною***. Тоді рекурсивний виклик – останній оператор у правилі, і запам'ятовувати значення змінних немає потреби. Всі відомості передаються як параметри.

Розглянемо відмінності стратегій на прикладі задачі про суму натуральних чисел.

Приклад 1. Розв'язання задачі про суму натуральних чисел за допомогою низхідної рекурсії. Якщо рекурентна формула для обчислення суми $S_n = S_{n-1} + n$, а початкове значення $S_0 = 0$, тоді програма буде виглядати так.

```
PREDICATES
    nondeterm Sum(integer, integer)
CLAUSES
    Sum(0,0).
    Sum(N,S):-N1=N-1, Sum(N1,S1), S=S1+N, N1>=0.
GOAL
    Sum(3,N).
```

Предикат Sum включає два аргументи: перший – розмір задачі, другий – результат.

Перше речення – розв'язок для найпростішого випадку ($N$=0), результат дорівнює нулю.

Друге правило складається з чотирьох підцілей: перша – зменшення розмірності задачі, друга – рекурсивне звернення зі зменшеним значенням аргумента $N$, третя – формування спільного розв'язку зі щойно отриманих, четверта – перевірка на закінчення виконання.

Схему розв'язання для випадку *Sum*(3, *S*), показано на рис. 1.8.

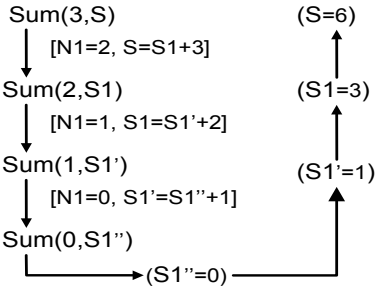


Рис. 1.8. Етапи розв'язання задачі про суму чисел методом низхідної рекурсії

Приклад 2. Розв'язання задачі про суму натуральних чисел за допомогою висхідної (хвостової) рекурсії. Для обчислення суми до початкового значення S0=0 додається число N0=0, яке збільшується на одиницю з кожним рекурсивним викликом. Коли значення N0 дорівнюватиме N, рекурсивні виклики припиняються. Тобто маємо запис:

```
Sum(N,S):-Sm(N,S,0,0).
```

Водиться допоміжна процедура з двома додатковими параметрами:

S0=0 – початкове значення суми,

N0=0 – лічильник кількості рекурсивних викликів

Далі записується основна частина програми.

```
Sm(N,S,N0,S0):-
    N0<=N,!,          % перевірка на закінчення
    N1=N0+1,          % модифікація лічильника
    S1=N0+S0,         % обчислення поточного значення суми
    Sm(N,S,N1,S1).    % хвостова рекурсія
```

Після закінчення рекурсивних викликів у S1 буде останнє обчислене значення суми. Для пересилання його в змінну S використовується така конструкція предиката:

```
Sm( _,S,_,S).
```

Для виведення даних у вигляді таблиці можна використати таку конструкцію.

```
pr_table:-write("--------------------"), nl,
          write("1   I   2   I   3   I"), nl,
          write("----------------------"), nl, fail.
pr_table:-Sum(1,S1),Sum(2,S2),Sum(3,S3),
          write(S1," I ",S2," I ",S3," I"),nl,fail.
pr_table:-write("--------------------"), nl.
```

Запит на виведення таблиці записується в секції goal: pr_table.

**2. Порядок виконання роботи**

2.1. Використовуючи програму з лабораторної роботи 1.1, скласти запит, який виводить у вигляді таблицю дані про осіб, які задовольняють певні умови (за варіантами).

2.2. Скласти програму для обчислення суми чисел, заданих за варіантом.

2.3. Подати результати обчислення суми чисел методом хвостової та низхідної рекурсії у вигляді таблиць. Також у вікно вивести ім’я розробника.

**3. Звіт.** Звіт повинен містити результати виконання п.п.2.1–2.2.

**4. Варіанти.** Вивести дані про осіб, які задовольняють умови:

1) чоловіки віком від 35 років і старші;
2) чоловіки з іменем Іван;
3) жінки з певним прізвищем ;

4) жінки молодші 18 років;
5) Особи віком від 20 років;
6) Лише імена жінок від 40 років і старші;
7) Лише імена та вік чоловіків з певним прізвищем.

Знайти суму чисел:
1) парних чисел: $2 + 4 + 6 + \ldots + n$;
2) квадратів чисел ряду $Sum = 1^2 + 2^2 + \ldots + n^2$;
3) чисел, кратних 5: $Sum = 5 + 10 + 15 + \ldots + n$;
4) чисел Фібоначі: $Sum = 1+1+2+3+5+8+ \ldots + n$;
5) непарних чисел: $Sum = 1 + 3 + 5 + \ldots + n$;
6) чисел, кратних 3: $Sum = 3 + 6 + 9 + \ldots + n$;
7) чисел, кратних 7: $Sum = 7 + 14 + 21 + \ldots + n$.

**Контрольні запитання**
1. Що таке рекурсія? Яке її призначення у мові *Prolog*?
2. У чому суть низхідної стратегії рекурсії?
3. Яка особливість хвостової рекурсії?
4. Яка стратегія рекурсії є більш ефективною?

Лабораторна робота 1.4

## ДОСЛІДЖЕННЯ РЕАЛІЗАЦІЇ ЛІНІЙНИХ СПИСКІВ ТА ЇХ ОБРОБЛЕННЯ

**Мета роботи** – набуття практичних навиків роботи зі списками у *Visual Prolog*

### 1. Основні теоретичні відомості

Списки у *Visual Prolog* – це множина елементів одного типу, які записуються у вигляді послідовності елементів у квадратних дужках, розділених комою: [1,3,5], [jam,apple,jin].

Списки є динамічними структурами, тобто можуть містити довільну кількість елементів, яка може змінюватися в процесі роботи програми. Окремим випадком списку є список, що складається з одного елемента – [x] і порожній список – [ ].

Кожен непорожній список може бути розділений на *голову* (Head) – перший елемент списку і *хвіст* (Tail) – інші елементи спи-

ску, що позначається символом у вигляді вертикальної риски (|). Можна виділяти перший елемент, або перший і другий :

L=[Head|Tail] або L=[Head_1,Head_2|Tail].

Список є рекурсивною структурою даних. Головний спосіб оброблення списку – це перегляд та оброблення кожного його елемента, доки не буде досягнуто кінця.

Алгоритм оброблення зазвичай складається з двох речень: перше вказує, що робити зі звичайним списком (списком, який можна розділити на голову і хвіст), друге – що робити з порожнім списком.

Приклад 1. Друкування списку по рядках.

```
DOMAINS
    list = integer*                 /* позначення списку*/
PREDICATES
    print(list)
CLAUSES
    print([]).      /* якщо список порожній, то нічого не робити */
    print([H|T]):-
                    write(H),/* надрукувати значення голови списку*/
                    nl,             /*перехід на наступний рядок */
                    print(T).       /* надрукувати хвіст списку */
GOAL
    print([2,5,8]).
```

Приклад 2. Визначення довжини списку. Припустімо, що довжина списку = довжині хвоста списку + один. Довжина порожнього списку ([ ]) = нулю.

```
DOMAINS
    list = integer*
PREDICATES
    len(list, integer)
CLAUSES
    len([], 0).                     /* довжина порожнього списку */
    len([ _|Tail],N):-len(Tail,N1), /* обчислення довжини хвоста  */
    N=N1+1.                 /* обчислення загальної довжини списку */
GOAL
    len([2,5,3,9,6],N).
```

Приклад 3. Визначення належності елемента списку. Порівнюємо перший елемент списку із шуканим елементом *X*. Якщо вони збігаються, то припиняємо виконання і відповідь буде «так».

У противному випадку перевіряємо наявність елемента *X* у хвості списку. Хвіст *T* рекурсивно передаємо у процедуру member(X, T) поки не буде досягнуто кінця списку. Якщо елемента не знайдено – відповідь «ні»:

```
DOMAINS
    list = integer*
PREDICATES
    nondeterm member(integer, list)
CLAUSES
    member(X, [X | _ ]).
    member(X, [ _ | T ]):- member(X, T).
GOAL
    member(8,[3,6,9]).
```

Приклад 4. Видалення елемента *X* зі списку. Якщо *X* є першим елементом списку, то розв‘язком є хвіст списку. У противному випадку треба видалити елемент *X* із хвоста і долучити до отриманого списку голову. Тобто переносимо голову списку *H* у результуючий список, а хвіст *T* рекурсивно передати у процедуру delete(X,T,L), поки не буде досягнуто кінця списку:

```
DOMAINS
    list = integer*
PREDICATES
    nondeterm delete(integer, list, list)
CLAUSES
    delete(X, [X | T ], T).
    delete(X, [ H | T ],[H | L]):- delete(X,T,L).
GOAL
    delete(X,[3,6,9],[6,9]).
```

Якщо у списку декілька однакових елементів, то у цій програмі буде видалено лише перший з таких елементів.

Приклад 5. Злиття двох списків. Якщо перший список порожній, то результат – другий список. В іншому випадку необхідно злиття хвоста першого списку з другим списком та додавання до отриманого списку голови першого списку:

```
DOMAINS
    list = integer*
PREDICATES
    merge(list, list, list)
CLAUSES
    merge([], L2, L2).
    merge([H | T], L2, [H | L3]):- merge(T,L2,L3).
GOAL
    merge([1,2],X,[1,2,4,5,6]).
```

Приклад 6. Об'єднання двох списків. Об'єднання відрізняється від злиття тим, що елементи списку не мають повторюватись. Тому, якщо голова першої множини належить до другої множини, то її треба додавати до отриманого списку:

```
DOMAINS
    list = integer*
PREDICATES
    union(list, list, list)
CLAUSES
    union([], L2, L2).
    union([H | T], L2, L3) :-
    member(H, L2),      /*перевіряємо належність голови H
                              до другого списку L2*/
    union(T, L2, L3),!.
    union([H | T], L2, [H | L3]) :-
    union(T, L2, L3).
GOAL
    union([1,2],[1,2,4,5,6],X).
```

**2. Порядок виконання роботи**

2.1. Проаналізувати наведені приклади оброблення списків. Скласти запити різного вигляду.

2.2. Виконати завдання згідно з варіантом. Подати результати роботи складеної програми.

**3. Звіт.** Звіт повинен містити результати виконання п.п. 2.1–2.2, тексти програм та результати їх тестування на різних запитах.

**4. Варіанти.** Скласти програму для виконання операції зі списком:

1) видалення *N*-го елемента зі списку: del(N,L1,L2);

2) розділення списку *L*1 на два списки компаратором *K*: *L*2 містить числа менші за *K*, *L*3 – числа більші за *K:* Split(K,L1,L2,L3);

3) обчислення кількості додатних *N*1 і від'ємних чисел *N*2 у списку *L*: count(L,N1,N2);

4) розділення списку *L*1 на два списки: *L*2 містить парні числа (*N* mod 2 = 0), *L*3 – непарні: Split(L1,L2,L3);

5) видалити зі списку елементів з парними номерами: del(L1,L2);

6) вставка елемента *X* на *N*-е місце у невпорядкованому списку: insert(X,N,L1,L2);

7) вставка елемента *X* у список перед елементом *Y*: insert(X,Y,L1,L2).

**Контрольні запитання**

1. Як позначається операція відділення голови списку від хвоста? Які бувають структури відділення?
2. Як позначається список?
3. Як обробляються списки мовою *Prolog?*

## Модуль 2. ПРОЕКТУВАННЯ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ

Модуль складається з таких тем: «Вирішувачі проблем, засновані на знаннях», «Сучасні тенденції та підходи до створення систем штучного інтелекту», лабораторні роботи 2.1 – 2.4.

### Лабораторна робота 2.1

### ДОСЛІДЖЕННЯ МЕХАНІЗМУ НАСЛІДУВАННЯ У VISUAL PROLOG

**Мета роботи –** навчитись реалізовувати механізм наслідування на фреймах у *Visual Prolog*

#### 1. Основні теоретичні відомості

***Механізм наслідування*** – це метод подання багатьох станів знань, за якого конкретний стан може наслідувати інформацію від

загального стану. Фраза ставиться у відповідність стану шляхом додавання імені стану в заголовок фрази (у вигляді першого її аргумента), всі інші аргументи не змінюються.

Головним завданням при використанні механізму наслідування є встановлення ієрархії станів знань. Взаємозв'язок станів можна розглядати як дерево, у якому кожен стан зображається вузлом. Нехай є два стани – А і Б. Якщо стан Б – це вузол дерева, породжений станом А, то Б успадкує всі фрази, пов'язані з А, за винятком тих, які явно спростовуються у Б.

Розглянемо реалізацію механізму наслідування у *Prolog* на прикладі фреймів. На рис. 2.1 схематично зображено об'єкти предметної галузі «Птахи» та відношення між ними. Спочатку опишемо предметну галузь за допомогою фреймів.

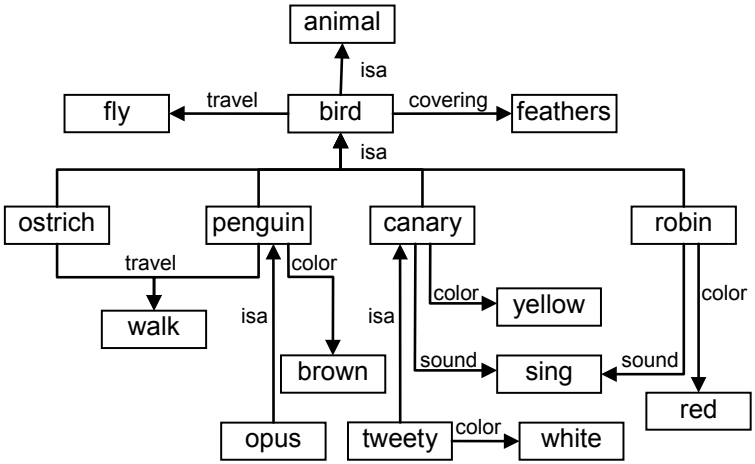


Рис. 2.1. Схематичне зображення фрагменту предметної галузі «Птахи»

Кожен фрейм являє собою набір відношень, а комірка isa визначає ієрархію фреймів (рис. 2.2).

У першій комірці кожного фрейму міститься ім'я вузла, наприклад, name(tweety) або name(bird).

У другій комірці визначається відношення наслідування між цим вузлом та його родичами.

У третій комірці міститься список властивостей, які описують цей вузол. У цьому списку можна використовувати будь-які предикати.

В останній комірці міститься список виключень та прийнятих за замовчуванням значень цього вузла. Його елементи також можуть бути словами або предикатами, що описують властивості.

| name: bird |
|---|
| isa: animal |
| properties: travel(flies) |
| default: |

| name: penguin |
|---|
| isa: bird |
| properties: color(brown) |
| default: travel(walks) |

| name: canary |
|---|
| isa: bird |
| properties: color(yellow)<br>sound(sing) |
| default: size(small) |

| name: tweety |
|---|
| isa: canary |
| properties: |
| default: color(white) |

Рис. 2.2. Фрейм для бази знань про птахів

Кожен предикат frame зв'язує імена комірок, списки властивостей та прийняті за замовчуванням значення. Це дозволяє розрізняти типи знань та приписувати їм різну поведінку в ієрархії наслідування. Хоча за такої реалізації підкласи можуть наслідувати властивості обох списків, для конкретних додатків можуть виявитися корисними й інші подання. Наприклад, можна наслідувати лише значення, що використовуються за замовчуванням, або будувати третій список, який містить властивості самого класу, а не його екземплярів. Такі значення називають ***значеннями класу***. Наприклад, можливо, щоб клас canary визначав вид співочого птаха. Ця властивість не має унаслідуватися підкласами або екземплярами: tweety – не є видом співочих птахів.

За допомогою предиката frame з чотирма аргументами запишемо відношення, показані на рис. 2.2.

```
frame(name(bird),
        isa(animal),
        [travel(flies)],
        []).
 frame(name(penguin),
        isa(bird),
        [color(brown)],
        [travel(walks)]).
```

```
frame(name(canary),
        isa(bird),
        [color(yellow), sound(sing)],
        [size(small)]).
frame(name(tweety),
       isa(canary),
       [],
       [color (white)]).
```

Для перевірки відповідності типів предиката frame, зокрема, для перевірки того, що в третій комірці фрейму міститься список, елементи якого набувають значень з певного діапазону властивостей, можна використовувати вбудовані можливості мови *Prolog*.

Визначивши для фрейму (рис. 2.2) повну множину описів і відношень наслідування, розробимо процедури для отримання властивостей з цього подання:

```
get(Prop, Object):-
     frame(name(Object),_,List_of_properties,_),
     member(Prop, List_of_properties).
get(Prop, Object):—
     frame(name(Object),_,_,List_of_defaults),
     member(Prop, List_of_defaults).
get(Prop, Object):—
     frame(name(Object),isa(Parent),_,_),
     get(Prop, Parent).
```

Якщо структура фреймів допускає множинне наслідування властивостей, то в подання та стратегію пошуку необхідно внести такі зміни. У поданні фрейму аргумент, зв'язаний з предикатом isa, має містити список суперкласів об'єкта. Кожен суперклас цього списку – це батьківський клас об'єкта, вказаного в першому аргументі предиката frame. Якщо opus належить до класу пінгвінів penguin і є персонажем мультфільму (cartoon_char), то це можна подати як:

```
frame(name(opus),
        isa([penguin, cartoon_char]),
       [color(black)],
        []).
```

Перевірити властивості об'єкта opus можна за допомогою повернення по ієрархії isa до обох суперкласів penguin та cartoon_char. Між третім і четвертим предикатами get у попередньому прикладі необхідно додати визначення:

```
get(Prop,Object):-
        frame(name(Object), isa(List), _,_),
        get_multiple(Prop, List).
```

Предикат get_multiple можна визначити таким чином:

```
get_multiple(Prop, [Parent|_]):-
        get(Prop, Parent).
get_multiple(Prop,[_|Rest]):-
        get_multiple(prop, Rest).
```

У цій ієрархії властивості класу penguin і його суперкласів будуть перевірені до початку перевірки властивостей класу cartoon_char.

З кожною коміркою фрейму можна зв'язати будь-яку процедуру. Як параметр предиката frame можна додати правило *Prolog* або список таких правил. Для цього все правило необхідно взяти в дужки та включити цю структуру до списку аргументів предиката frame. Цей список правил, у якому кожне правило взято в дужки, має стати параметром предиката frame, який визначає відповідь залежно від значення *X*, що передається фрейму opus.

**2. Порядок виконання роботи**

2.1. Скласти програму, яка описує предметну галузь, зображену на рис. 2.1 за допомогою фрейму (рис. 2.2), та реалізує механізм наслідування у *Prolog*.

2.2. Скласти структурну схему об'єктів та зв'язків між ними для предметної галузі за варіантом, аналогічно до рис. 2.1.

2.3. Подати фрейм, що описує предметну галузь, аналогічно до рис. 2.2.

2.4. Скласти програму, яка реалізує механізм наслідування у *Prolog*.

**3. Звіт.** Звіт має містити результати виконання п.п. 2.1–2.4, приклади роботи програм.

**4. Варіанти.** Предметну галузь студент обирає самостійно.

**Контрольні запитання**

1. Особливості фреймового представлення знань.
2. Що таке наслідування у *Prolog*?
3. Як реалізується наслідування на фреймах?

Лабораторна робота 2.2

# ДОСЛІДЖЕННЯ ПРИНЦИПІВ ПОБУДОВИ ЕКСПЕРТНИХ СИСТЕМ

**Мета роботи** – вивчити принципи побудови та навчитись розробляти експертні системи

## 1. Основні теоретичні відомості

Експертні системи можна поділити на два типи: засновані на фактах та засновані на правилах. Нижче наведено експертну систему на фактах, де знання про предметну галузь (у вигляді фактів) зберігаються у зовнішній базі даних. У разі зміни предметної галузі модифікується лише база даних, не змінюючи програму.

Розглянемо приклад експертної системи, яка дозволить за наявністю певних ознак визначити вид риби. список ознак, які однозначно характеризують кожен з чотирьох видів риб, наведено у табл. 2.1.

*Таблиця 2.1*

| Номер ознаки | Ознака | Значення ознаки | Сом | Плотва | Окунь | Щука |
|---|---|---|---|---|---|---|
| 1 | Маса | > 20 кг | + | | | |
| 2 | | < 20 кг | + | + | + | + |
| 3 | Тіло | Довге вузьке | | | | + |
| 4 | | Широке | | + | + | |
| 5 | Луска | Звичайна | | | + | + |
| 6 | | Сріблясте | | + | | |
| 7 | Малюнок на тілі | Темні смуги | | | + | |
| 8 | Особливості | Має вуса | + | | | |

Експертна система складається із сукупності модулів. База знань містить знання про предметну галузь. База даних (фактів) містить дані, уведені користувачем. Блок інтерфейсу забезпечує діалог з користувачем. Механізм логічного виведення забезпечує пошук необхідних знань і фактів.

Опис предметної галузі зберігається в базі знань у вигляді двох предикатів:

1) ознаки та її номеру:

sign(numb_s, stri), де numb_s = integer, str = symbol;

2) риби, що розпізнається, та списку номерів ознак, які однозначно її характеризують:

rule(fish, list_sign), де fish = symbol, list_sign = numb_s*.

Тобто в секції clauses записуємо:

```
sign(1, маса більше 20 кг).
sign(2, маса менше 20 кг).
sign(3, тіло довге вузьке).
sign(4, тіло широке).
sign(5, луска звичайна).
sign(6, луска сріблясta).
sign(7, темні смуги на тілі).
sign(8, має вуси).
rule(сом, [1, 2, 8]).
rule(плотва, [2, 4, 6]).
rule(окунь, [2, 4, 5, 7]).
rule(щука, [2, 3, 5]).
```

Виконання програми починається запуском предиката run та з вибірки першого правила, яке містить назву тварини та список ознак rule(Fish, List). Предикат check(List) перевіряє наявність ознак. Якщо все ознаки наявні, то видається назва виду риби. Далі, у разі потреби, видається стан бази даних, тобто список відповідей користувача. Потім проводиться очищення бази даних від попередніх відповідей:

```
run:- rule(Fish, List),
      heck(List), !,
      write("Fish is  ", Fish), nl,
      status_db,
      clear_facts.
```

Якщо розв'язку не знайдено, то видається повідомлення:

```
run:- write(«\n  не можу визначити»), nl,
     write(«\n що це за риба»), nl
      status_db,
      clear_facts.
```

Предикат контролю має рекурсивний характер. Обирається голова списку, перевіряється, чи немає в БД відомостей про поточну ознаку. В іншому випадку організується діалог з користувачем.

```
check([ ]).    % якщо список порожній, то рекурсія припиняється
check([Head| Rest]):-
      yes(Head), ! ,          % у базі даних є ця ознака
      check(Rest).            % перевірка хвоста списку
```

```
check([Head | _ ]):-
    no(Head), !,  % у базі даних указується на відсутність ознаки
    fail.                    % відкат до наступного правила
check([Head| Rest]):-
    sign(Head, Text),
    ask(Head, Text),
    check(Rest).             % перевірка хвоста списку
```

Організація діалогу з користувачем

```
ask(Head, Text):-
    write(«Запитання:»,Text, «yes/no?»), nl, % виведення запиту
                                                на екран
    readln(Reply),   % запам'ятовування відповіді в Reply
    frontchar(Reply, First, _), % перша літера відповіді
    remember(Head, First). % запам'ятовування відповіді
```

У базі даних експертної системи спочатку немає ніяких фактів про поточну ситуацію. Тому в програму треба включати блок операторів, який дозволить у ході діалогу з користувачем, отримувати інформацію і записувати її в базу даних:

```
database
    yes(numb_s)
    no(numb_s)
```

Відповіді будуть зберігатись у вигляді наявності або відсутності ознаки з відповідним номером.

Предикати поповнення бази даних:

```
remember(Head, 'y'):- assertz(yes(Head)).
remember(Head, 'n'):- assertz(no(Head)), fail.
                          %у випадку негативної відповіді – відкат
```

Предикат очищення бази даних після видачі відповіді:

```
clear_facts:-
     write("\n\n Для виходу натисніть будь-яку клавішу \n"),
    retractall( _ ),
    readchar( _ ).
```

Правило status_db призначене для перевірки стану бази даних у вигляді списку відповідей користувачів на задані запитання:

```
status_db:- !,
    write(""),
    resdln(Reply),
    frontchar(Reply,'y',_ ),
    print_positive,
```

```
        print_negative,
        write(" Завершення  ").
```

Виведення позитивних фактів з бази даних:

```
print_positive:-
        write(" Позитивна відповідь " ), nl,
        write("---------------------"), nl, fail.
print_positive:-
        xpositive(X,Y),
        write(X, " ", Y),nl
        fail.
print_positive:-
        write("--------------"), nl.
```

Другий предикат необхідний, щоб підціль print_positive була успішно доведена. Аналогічно виводяться відомості про негативні відповіді.

**2. Порядок виконання роботи**

2.1. Скласти програму для експертної системи предметної галузі, заданої за варіантом.

2.2. Провести тестування експертної системи на декількох екземплярах предметної галузі.

2.3. Побудувати дерево виведення для одного екземпляра з предметної галузі.

**3. Звіт.** Звіт має містити таблицю ознак екземплярів предметної галузі, текст програми експертної системи, результати виконання за п.п. 2.2, 2.3.

**4. Варіанти.** Предметну галузь студент обирає самостійно.

**Контрольні запитання**

1. З яких модулів складається експертна система?
2. Які недоліки має продукційна експертна система?
3. У чому відмінність динамічної експертної системи від статичної?
4. Як зберігаються знання в експертній системі на предикатах?
5. Для чого необхідно записувати відповіді користувача у базу даних?
6. Які предикати використовуються для додавання та видалення фактів у базі даних?

Лабораторна робота 2.3

# ДОСЛІДЖЕННЯ МЕХАНІЗХМУ РОБОТИ НЕЙРОННОЇ МЕРЕЖІ

**Мета роботи –** закріплення теоретичних знань та набуття практичних навичок роботи з найпростішою нейронною мережею

## 1. Основні теоретичні відомості

*Персептрон Розенблата* вважається першою моделлю нейронної мережі й класикою для вивчення принципу функціонування.

У найпростішому вигляді персептрон складається із сукупності чутливих (сенсорних) елементів (*S*-елементів), на які надходять вхідні сигнали. *S*-елементи випадковим чином пов'язані із сукупністю асоціативних елементів (*A*-елементів), вихід яких відрізняється від нуля лише тоді, коли збуджене досить велике число *S*-елементів, які впливають на один *A*-елемент. *A*-елементи з'єднані з реагуючими елементами (*R*-елементами) зв'язками, коефіцієнти підсилення (*w*) яких змінні і змінюються в процесі навчання.

Зважені комбінації виходів *R*-елементів становлять реакцію системи, яка вказує на належність розпізнаваного об'єкта до певного образу.

Якщо розпізнаються лише два образи, то в персептроні встановлюється лише один *R*-елемент, який має дві реакції – додатну (+1) і від'ємну (–1).

Алгоритм роботи персептрона включає такі етапи:

1) визначення початкових значень ваги зв'язку між *S*-, *A*- та *R*-елементами;
2) визначення сумарного сигналу на вході *A*-елемента;
3) визначення сигналу на виході *A*-елемента;
4) визначення значення сигналу на вході та виході *R*-елемента;
5) адаптація персептрона.

Розглянемо принцип функціонування персептрона з бінарними *S*- і *A*- елементами і біполярним *R*-елементом (рис. 2.3) на прикладі розпізнавання зображень літер «Н» і «П» (рис. 2.4, а, б) на рецепторному полі з дев'яти елементів (рис. 2.4 в). У разі подання зображення літери «Н» на виході *R*-елемента необхідно отримати сигнал «–1», літери «П» – сигнал «+1».

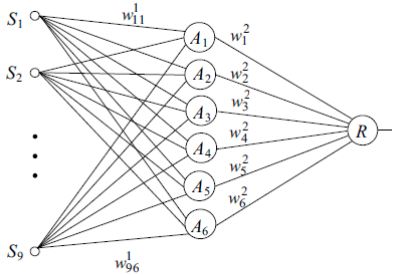


Рис. 2.3. Елементарний персептрон

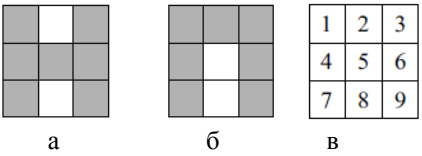


Рис. 2.4. Зображення літер «Н» (а) і «П» (б) та рецепторне поле (в)

### 1.1. Визначення початкового значення ваги зв'язку

Для роботи персептрона необхідно задати початкові значення ваги зв'язку між бінарними $S$- і $A$-елементами ( $w^1_{ij}$, $i=\overline{1,9}$, $j=\overline{1,6}$ ) та між $A$-елементами і біполярним $R$-елементом ( $w^2_k, k=\overline{1,6}$ ). Ці значення отримують за допомогою генератора випадкових чисел з кінцевої множини {0,1…0,9}. Дані, отримані для розгляданого прикладу, наведено у табл. 2.2 та 2.3 відповідно.

*Таблиця 2.2*

**Ваги $w^1_{ij}$ зв'язків персептрона між $S$- і $A$-елементами**

| $w^1_{ij}$ | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $S_7$ | $S_8$ | $S_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $A_1$ | 0,3 | 0,2 | 0,1 | 0,6 | 0,5 | 0,4 | 0,9 | 0,6 | 0,7 |
| $A_2$ | 0,2 | 0,1 | 0,3 | 0,4 | 0,5 | 0,6 | 0,7 | 0,1 | 0,9 |
| $A_3$ | 0,3 | 0,5 | 0,1 | 0,6 | 0,5 | 0,4 | 0,9 | 0,4 | 0,7 |
| $A_4$ | 0,4 | 0,3 | 0,2 | 0,1 | 0,8 | 0,7 | 0,6 | 0,6 | 0,9 |
| $A_5$ | 0,5 | 0,3 | 0,3 | 0,6 | 0,1 | 0,2 | 0,9 | 0,2 | 0,7 |
| $A_6$ | 0,6 | 0,5 | 0,4 | 0,1 | 0,2 | 0,3 | 0,8 | 0,5 | 0,8 |

*Таблиця 2.3*

**Ваги $w^2_k$ зв'язків персептрона між $R$- і $A$-елементами**

| $w^2_k$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ |
|---|---|---|---|---|---|---|
| $R$ | 0,3 | 0,8 | 0,6 | 0,9 | 0,8 | 0,1 |

### 1.2. Визначення сумарного сигналу на вході $A$-елемента

Якщо на вхід персептрона подається зображення літери «Н» активуються всі $S$-елементи, крім другого і восьмого. Одиничні сиг-

нали з виходів активованих бінарних *S*-елементів через зв'язки, вагові коефіцієнти яких задано в табл. 2.2, надходять на входи *A*-елементів. Сумарний вхідний сигнал на вході *i*-го *A*-елемента визначається співвідношенням

$$U_{\text{вх. }Ai} = \sum_{j=1}^{9} U_{\text{вих. }Sj} w_{ji}^{1}, \quad j = \overline{1,6},$$

де $U_{\text{вх. }Ai}$ – сигнал на вході *i*-го *A*-елемента; $U_{\text{вих. }Sj}$ – сигнал на виході *j*-го *S*-елемента; $w_{ji}$ – вага зв'язку між *j*-м *S*-елементом та *i*-м *A*-елементом.

Для першого *A*-елемента маємо

$$U_{\text{вх. }A1} = \sum_{j=1}^{9} U_{\text{вих. }Sj} w_{j1}^{1} = 1 \cdot 0,3 + 0 \cdot 0,2 + 1 \cdot 0,1 + 1 \cdot 0,6 + 1 \cdot 0,5 + 1 \cdot 0,4 +$$

$$+1 \cdot 0,9 + 0 \cdot 0,6 + 1 \cdot 0,7.$$

Далі обчислюються сигнали на входах інших *A*-елементів.

Аналогічні дії виконуємо для випадку подачі на вхід персептрона зображення літери «П». У такому випадку будуть задіяні всі *S*-елементи, крім п'ятого і восьмого. Отримані результати заносимо в табл. 2.4.

*Таблиця 2.4*

**Величини сигналів на входах *A*-елементів**

| Зображення | Сигнали на входах *A*-елементів | | | | | |
|---|---|---|---|---|---|---|
| | $U_{\text{вх. }A1}$ | $U_{\text{вх. }A2}$ | $U_{\text{вх. }A3}$ | $U_{\text{вх. }A4}$ | $U_{\text{вх. }A5}$ | $U_{\text{вх. }A6}$ |
| Літера «Н» | 3,5 | 3,6 | 3,5 | 3,7 | 3,3 | 3,2 |
| Літера «П» | 3,2 | 3,2 | 3,4 | 3,2 | 3,6 | 3,5 |

### 1.3. Визначення сигналу на виході *A*-елемента

Вихідний сигнал *A*-елементів визначається співвідношенням

$$U_{\text{вх. }A} = \begin{cases} 1, \text{ якщо } U_{\text{вх. }A} \geq \Theta, \\ 0, \text{ якщо } U_{\text{вх. }A} < \Theta, \end{cases}$$

де $\Theta$ – поріг.

Для спрощення припустимо, що пороги $\Theta_i$ для всіх *A*-елементів однакові: $\Theta_1 = \Theta_2 = ... = \Theta_6 = \Theta$. Значення порога необхідно вибирати таким чином, щоб з подачею різних зображень активізувались різні множини $M_1$ і $M_2$, причому такі, що не перетинаються:

$$M_1 \cap M_2 = 0. \qquad (2.1)$$

У цьому прикладі для ефективної роботи нейронної мережі поріг слід обирати між 3,2 та 3,7. Умова (2.1) буде виконуватись, якщо Θ = 3,5 і від подання зображення літери «Н» будуть активізуватись елементи $A_1$, $A_2$, $A_3$ і $A_4$, а з поданням зображення літери «П» – елементи $A_5$ і $A_6$.

### 1.4. Визначення сигналу на вході та виході *R*-елемента

Спираючись на дані табл. 2.3 розрахуємо сигнали $U_{\text{вх. }R\text{Н}}$, $U_{\text{вх. }R\text{П}}$ на вході *R*-елемента з поданням зображень літери «Н» і «П»:

$$U_{\text{вх. }R\text{Н}} = \sum_{j=1}^{6} U_{\text{вих. }Ai} w_i^2 = 1\cdot 0,2 + 1\cdot 0,8 + 1\cdot 0,6 + 1\cdot 0,9 + 0\cdot 0,8 + 0\cdot 0,1 = 2,5.$$

$$U_{\text{вх. }R\text{П}} = \sum_{j=1}^{6} U_{\text{вих. }Ai} w_i^2 = 0\cdot 0,2 + 0\cdot 0,8 + 0\cdot 0,6 + 0\cdot 0,9 + 1\cdot 0,8 + 1\cdot 0,1 = 0,9.$$

Зі значенням порогу *R*-елемента $\Theta_R = 1,7$ та поданням зображення літери «Н» на виході персептрона буде сигнал «+1», а з поданням зображення літери «П» – «–1», що не відповідає початковим умовам задачі. Тому необхідно налаштовувати персептрон.

### 1.5. Адаптація персептрона

Для налаштування (адаптації) персептрона використаємо α-систему підкріплення за величини сигналу підкріплення η=0,1 і з поданням послідовності зображень «Н», «П», «Н», «П», … у моменти часу $t_1$, $t_2$, $t_3$, … . Процес адаптації ваги зв'язків між *R*- і *A*-елементами наведено в табл. 2.5.

У другому стовпчику табл. 2.5 при $t=t_0$ наведено початкові значення ваги зв'язків та величини сигналів $U_{\text{вх. }R\text{Н}}$, $U_{\text{вх. }R\text{П}}$ на вході *R*-елемента з поданням відповідного зображення.

За першої появи зображення літери «Н» у момент часу $t_1$ (позначено $t^H_1$) через наявність помилкового сигналу на виході персептрона коригуються ваги активних зв'язків $w_1^2$, ..., $w_4^2$ на величину η=0,1. Це коригування зменшує сумарний вхідний сигнал $U_{\text{вх. }R\text{Н}}$ до величини 2,1.

*Таблиця 2.5*

**Адаптація ваги зв'язків персептрона за допомогою α-системи підкріплення**

| Вагові коефіцієнти та вихідні сигнали | Моменти часу | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $t_0$ | $t_1^{\text{Н}}$ | $t_2^{\text{П}}$ | $t_3^{\text{Н}}$ | $t_4^{\text{П}}$ | $t_5^{\text{Н}}$ | $t_6^{\text{П}}$ | $t_7^{\text{П}}$ | $t_8^{\text{П}}$ | $t_9^{\text{П}}$ |
| $w_1^2$ | 0,2 | 0,1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $w_2^2$ | 0,8 | 0,7 | 0 | 0,6 | 0 | 0, | 0 | 0 | 0 | 0 |
| $w_3^2$ | 0,6 | 0,5 | 0 | 0,4 | 0 | 0,3 | 0 | 0 | 0 | 0 |
| $w_4^2$ | 0,9 | 0,8 | 0 | 0,7 | 0 | 0,6 | 0 | 0 | 0 | 0 |
| $w_5^2$ | 0,8 | 0 | 0,9 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| $w_6^2$ | 0,1 | 0 | 0,2 | 0 | 0,3 | 0 | 0,4 | 0,5 | 0,6 | 0,7 |
| $U_{вх.\,R\text{Н}}$ | 2,5 | 2,1 | – | 1,7 | – | 1,4 | – | – | – | – |
| $U_{вх.\,R\text{П}}$ | 0,9 | – | 1,1 | – | 1,3 | – | 1,4 | 1,5 | 1,6 | 1,7 |

Припустімо функціонування *R*-елемента описується співвідношенням:

$$U_{\text{вих.}\,R} = \begin{cases} +1, \text{ якщо } U_{\text{вх.}\,R} \geq \Theta_R, \\ -1, \text{ якщо } U_{\text{вх.}\,R} < \Theta_R, \end{cases}$$

де $\Theta_R$ – поріг *R*-елемента, тоді для досягнення правильної реакції *R*-елемента на зображення літери «Н» необхідні два повторні коригування ваги зв'язків $w_1^2$, ..., $w_4^2$. Результати цих коригувань наведено у табл. 2.5 у п'ятому та сьомому стовпчиках при $t = t_3^{\text{Н}}$ і $t = t_5^{\text{Н}}$.

Після моменту часу $t = t_5^{\text{Н}}$, оскільки виконується співвідношення (2.1), із вхідної послідовності можуть бути виключені зображення літери «Н» і подаватися лише зображення літери «П».

Результати коригування ваги зв'язків $w_5^2$, $w_6^2$, які визначають сигнал на вході *R*-елемента з поданням зображення літери «П», наведено в останньому рядку табл. 2.5.

Оскільки в цьому прикладі коригування вхідного сигналу *R*-елемента при поданні зображення літери «П» виконується лише за допомогою ваги двох зв'язків, причому, після другого коригування

вага зв'язку $w_5^2$ набуває максимального значення і надалі збільшуватись не може, то процес навчання нейронної мережі правильної реакції на друге зображення більш тривалий і закінчується лише при $t = t_9^{\text{П}}$.

**2. Порядок виконання роботи**

2.1. Розробити структуру елементарного персептрона згідно з варіантом.

2.2. Виконати розрахунки для запропонованого персептрона за п.п. 1.1–1.4. Обґрунтувати вибір величини порога для *A*- та *R*-елементів.

2.3. Провести навчання персептрона методом α-підкріплень за п.п. 1.5. Обґрунтувати вибір величини кроку в алгоритмі адаптації.

**3. Звіт.** Звіт повинен містити результати розрахунків за п.п. 2.1–2.3 та висновки про роботу.

**4. Варіанти.** Розробити персептрон, здатний розпізнавати першу літеру прізвища та імені студента.

**Контрольні запитання**

1. Пояснити принцип роботи штучного нейрона на прикладі персептрона Розенблата.
2. Які етапи включає алгоритм роботи персептрона?
3. Що таке адаптація персептрона?

Лабораторна робота 2.4

## ДОСЛІДЖЕННЯ СЕМАНТИЧНОГО МОДЕЛЮВАННЯ У СИСТЕМАХ ШТУЧНОГО ІНТЕЛЕКТУ

**Мета роботи –** навчитись моделювати предметну галузь за допомогою семантичної сітки.

**1. Основні теоретичні відомості**

Розглянемо реалізацію механізму наслідування у *Prolog* для простої мови *семантичних сіток*. Для спрощення не будемо враховувати важливу різницю між класами та їх екземплярами.

У семантичній сітці, показаній на рис. 2.5, вузли виражають такі об’єкти, як конкретна канарка tweety, та класи ostrich (страус), penguin (пінгвін), robin (дрізд), bird (птах) та animal (тварина).

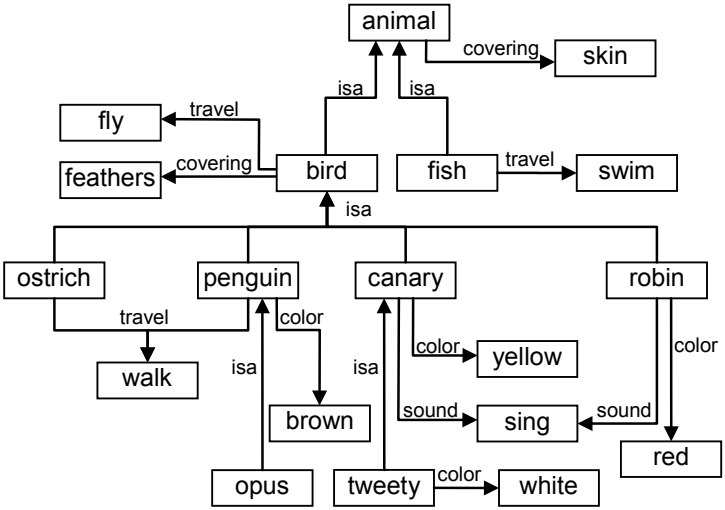


Рис. 2.5. Фрагмент семантичної сітки, що описує деяких птахів та риб

Відношення isa зв’язує класи різних рівнів ієрархії наслідування. У цій сітці реалізовані канонічні форми подання даних.

Предикат isa(Type, Parent) вказує на те, що об’єкт Type є підтипом Parent, а предикат hasprop(Object, Property, Value) описує властивість об’єктів.

Предикат hasprop вказує на те, що об’єкт Object має властивість Property зі значенням Value. При цьому Object та Value – це вузли, а Property – ім’я зв’язку, що їх об’єднує.

Фрагмент списку предикатів, які описують ієрархію деяких птахів та риб, показану на рис. 2.5:

```
isa(canary, bird).
isa(ostrich, bird).
isa(bird, animal).
isa(opus, penguin).
isa(robin, bird).
isa(penguin, bird).
isa(fish, animal).
isa(tweety, canary).
```

```
hasprop(tweety, color, white).
hasprop(canary, color, yellow).
hasprop(bird, travel, fly).
hasprop(ostrich, travel, walk).
hasprop(robin, sound, sing).
hasprop(bird, cover, feathers).
hasprop(robin, color, red).
hasprop(penguin, color, brown).
hasprop(fish, travel, swim).
hasprop (penguin, travel, walk).
hasprop(canary, sound, sing).
hasprop(animal, cover, skin).
```

Створимо рекурсивний алгоритм пошуку, що дозволяє визначити, чи притаманні деякому об'єкту семантичної сітки вказані властивості. Властивості зберігаються в сітці на найвищому рівні, на якому вони істинні. За допомогою наслідування об'єкт або підклас набуває властивостей свого суперкласу. Так, наприклад, властивість fly належать до об'єкта bird і всіх його підкласів. Виключення розміщуються на окремому рівні виключень. Так, страус ostrich і пінгвін penguin ходять (walk), але не літають (fly).

Предикат hasproperty починає пошук з конкретного об'єкта. Якщо інформація напряму не пов'язана з цим об'єктом, він переходить по зв'язку isa до суперкласів. Якщо суперкласу вже не існує, і предикат hasproperty не знайшов потрібної властивості, пошук завершується невдачею. Предикат hasproperty виконує пошук у глибину в ієрархії наслідування.

```
hasproperty(Object, Property, Value):-
      hasprop(Object, Property, Value).
hasproperty(Object, Property, Value):-
      isa(Object, Parent),
      hasproperty(Parent, Property, Value).
```

**2. Порядок виконання роботи**

2.1. За фрагментом семантичної сітки, показаної на рис. 2.5, скласти програму у *Visual Prolog*.

2.2. Скласти фрагмент семантичної сітки, що описує предметну галузь задану за варіантом.

2.3. На основі семантичної сітки з п.п. 2.3 скласти програму.

**3. Звіт.** Звіт має містити результат виконання п.п. 2.1, 2.3–2.4, приклади роботи програм.

**4. Варіанти.** Предметну галузь студент обирає самостійно.

**Контрольні запитання**

1. Що таке семантичне моделювання?
2. Як реалізується наслідування на семантичних сітках?
3. Наведіть відмінність семантичних сіток та фреймів для опису предметної галузі.
4. Яка відмінність наслідування на фреймах та семантичних сітках?

## СПИСОК ЛІТЕРАТУРИ


1. *Адаменко А.* Логическое программирование и Visual Prolog / А. Адаменко, А. Кучуков. – СПб.: БХВ-Петербург, 2003. – 982 с.
2. *Братко И.* Язык Prolog (Пролог): алгоритмы искусственного интеллекта. – 3-е изд.: пер. с англ. / И. Братко. – М.: Вильямс, 2001. – 640 с.
3. *Джексон П.* Введение в экспертные системы / П. Джексон. – М.: Вильямс, 2001. – 624 с.
4. *Ин Ц.* Использование Турбо-Пролога: пер. с англ. / Ц. Ин, Д. Соломон – М.: Мир, 1993. – 608 с.
5. *Круглов В.В.* Искусственные нейронные сети. Теория и практика. – 2-е изд. стериотип. / В.В. Круглов, В.В. Борисов. – М.: Горячая линия – Телеком, 2002. – 382 с.
6. *Цуканова Н.И.* Логическое программирование на языке Visual Prolog: Учеб. пособие для вузов / Н.И. Цуканова, Т.А. Дмитриева. – М.: Горячая линия-Телеком, 2008. – 144 с.
7. *Янсон А.* Турбо-Пролог в сжатом изложении: пер. с нем. / А. Янсон– М.: Мир, 1991. – 95 с.
8. *Люгер Джордж Ф.* Искусственный интеллект: стратегии и методы решения сложных проблем. – 4-е изд.: пер. с англ. / Джордж Ф. Люгер. – М.: Вильямс, 2003. – 864 с.

**ЗМІСТ**

*Навчально-методичне видання*

# МЕТОДИ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛКТУ

Лабораторний практикум
для студентів напряму підготовки
6.050101 «Комп'ютерні науки»

Укладач САВЧЕНКО Аліна Станіславівна

В авторській редакції

Підп.до друку . Формат 60x84/16. Папір офс.
Офс. друк. Ум. друк. арк. 8. Обл.-вид. арк. 8.
Тираж пр. Замовлення № . Вид. № .

Видавець і виготовлювач
видавництво Національного авіаційного університету «НАУ-друк»
03058. Київ-58, проспект Космонавта Комарова, 1.

Свідоцтво про внесення до Державного реєстру ДК № 977 від 05.07.2002