О. М. Спірін

ПОЧАТКИ ШТУЧНОГО ІНТЕЛЕКТУ

Навчальний посібник
для студентів фізико-математичних спеціальностей
вищих педагогічних навчальних закладів

Житомир - 2004

УДК 681.142.2(075.5)
ББК 73я73
С72

Рекомендовано Міністерством освіти і науки України як навчальний посібник для студентів фізико-математичних спеціальностей вищих педагогічних навчальних закладів, протокол №14/18.2-717 від 16.04.2003р.

Рецензенти:

Рамський Ю.С., кандидат фіз.-мат. наук, професор кафедри основ інформатики і обчислювальної техніки Національного педагогічного університету ім. М.П. Драгоманова;

Ляшенко Б.М., кандидат фіз.-мат. наук, доцент кафедри інформатики Житомирського державного університету імені Івана Франка;

Лисогор І.А., методист лабораторії інформатики та обчислювальної техніки Житомирського ОІППО.

Спірін О.М.

С72 Початки штучного інтелекту: Навчальний посібник для студ. фіз.-мат. спец-тей. вищих пед. навч. закл-ів – Житомир: Вид-во ЖДУ, 2004. - 172 с.: рис.

ISBN 966-8456-00-9

Посібник містить теоретичні відомості з мови логічного програмування Prolog (версія Turbo Prolog 2.0), основ штучного інтелекту та експертних систем; приклади виконання завдань, тексти лабораторних робіт (контрольні питання, завдання, вимоги до захисту лабораторних робіт). Посібник може бути використаний на факультативних заняттях з інформатики у загальноосвітніх школах.

Для студентів фізико-математичних спеціальностей вищих педагогічних навчальних закладів.

УДК 681.142.2(075.5)
ББК 73я73

ISBN 966-8456-00-9

Передмова

Навчальний посібник написаний відповідно до чинної програми з інформатики для спеціальностей "математика і фізика", "фізика і математика" вищих педагогічних навчальних закладів. Розгляд окремих питань виходить за рамки програмних вимог щодо вивчення початків штучного інтелекту. Це дає можливість використовувати посібник як складову частину навчально-методичного забезпечення курсу інформатики вузу, а також для читання спецкурсів, проведення факультативної та гурткової роботи у закладах освіти.

Основу навчального посібника склав курс лекцій з основ штучного інтелекту курсу інформатики, який читався автором на фізико-математичному факультеті Житомирського державного університету імені Івана Франка на спеціальностях "математика і фізика", “математика і інформатика”.

У процесі написання посібника автор керувався такими положеннями:

1. Формування знань, умінь та навичок з початків штучного інтелекту в курсі інформатики вузу є складовою частиною завдання формування початків інформаційної культури студентів фізико-математичних факультетів. Названі компоненти повинні бути сформовані як на теоретичному рівні, що дозволяє орієнтуватися у загальних питаннях з проблем штучного інтелекту: історія розвитку, теоретичні основи, класичні проблеми та задачі, сучасний стан, тенденції та перспективи розвитку; так і на рівні практичної реалізації окремих задач з проблем штучного інтелекту на основі існуючих технологій.

2. Розгляд питань подання знань у системах штучного інтелекту (логічні методи, семантичні мережі, фрейми; логічний вивід) проводиться опосередковано у процесі вивчення однієї з мов логічного програмування (у даному випадку мова Пролог, її діалект Турбо-Пролог 2.0).

3. Вивчення експертних систем, їх розробки можливе як з використанням існуючих оболонок експертних систем, так і опосередковано - у процесі розгляду (або на основі) мови логічного програмування Пролог.

4. Ефективне використання посібника студентами під час самостійного вивчення даного курсу. Тому приклади програм є завершеними програмними продуктами, які коректно працюють після введення у середовище Турбо-Прологу версії 2.0 фірми Borland International. Крім того, кожна наведена у посібнику програма супроводжується відповідними коментарями.

Посібник містить основні відомості про системи штучного інтелекту, експертні системи: бази знань, механізми виводу в експертних системах, розробку експертних систем засобами мови Пролог.

У посібнику описаний теоретичний та практичний матеріал до виконуваних лабораторних робіт, що включає різнорівневі завдання для студентів.

Тексти програм на Турбо-Пролозі написані так, щоб полегшити їх розуміння тими, хто володіє українською мовою. Окремі програми або їх частини записані англійською мовою для випадків, коли використовуються вбудовані (стандартні) предикати Турбо-Прологу версії 2.0 або деякі процедури є загальноприйнятими і часто вживаються у практиці програмування на Пролозі.

ЛОГІЧНЕ ПРОГРАМУВАННЯ

§1. ЗАГАЛЬНІ ВІДОМОСТІ

1.1. Логічне і процедурне програмування

Логічне програмування - це відносно новий перспективний напрямок сучасного програмування, що виник у рамках робіт зі створення штучного інтелекту. Свою назву логічне програмування одержало від математичної логіки і базується на одному з її розділів - логіці предикатів першого порядку.

Основна мета створення логічного програмування - підвищення "інтелектуальності" комп'ютерів. Однак роль логічного програмування цим не обмежується: крім усього іншого, воно було взято за концептуальну основу відомого японського проекту ЕОМ 5-го покоління. Мета даного проекту - створення систем обробки інформації, що базуються на знаннях і можуть давати експертні консультації користувачам.

Відомо, що для того, щоб скласти програму розв'язку деякої задачі мовою процедурного типу, необхідно розробити її алгоритм і у подальшому транслювати його в набір базових конструкцій (цикли, розгалуження, процедури-підпрограми та ін.) за допомогою досить об'ємного набору операторів обраної мови. Відповідно до цього, програміст повинен спочатку продумати весь план розв'язку задачі, а потім деталізувати його до найменших подробиць. Крім того, процедурні мови, у своїй більшості, дозволяють оперувати тільки тими типами даних, які можуть зберігатися у пам'яті ЕОМ, і тільки тими операціями, які вона у змозі виконувати. Як правило, ці типи даних елементарні, операції над ними досить примітивні, тому складання навіть простої програми на мовах низького рівня (до яких відносяться процедурні) перетворюється у довгий копіткий процес і призводить до великої кількості помилок, пошук і виправлення яких часто вимагає навіть більше часу, ніж безпосереднє написання програми.

При використанні мови логічного програмування основна увага приділяється описанню структури прикладної задачі, а не розробці наказів комп'ютеру про те, що йому потрібно робити. Тут відсутнє традиційне поняття алгоритму розв'язку задачі. Для того, щоб розв'язати деяку задачу, необхідно досить детально описати дані умо-

ви задачі і, наприклад, шляхом запитів до програми одержати потрібні відповіді. При цьому програміст не вказує, "як" перевірити твердження, сформульоване у вигляді запиту, - програма сама або доведе істинність твердження, або повідомить про те, що твердження є хибним, або що про нього нічого не можна сказати на основі даних умов.

Тому логічне програмування вимагає іншого стилю мислення, відмови від прийнятих програмістських стереотипів: замість того, щоб задати складну послідовність команд, які б вказували машині на виконання тих чи інших дій для розв'язку задачі, необхідно описати її зміст у термінах об'єктів і відношень між ними. Отже, *замість алгоритму розв'язку задачі програміст складає її логічну специфікацію.* Для певної мови логічного програмування специфікація задачі (описання проблеми) розглядається як фактична програма. Зауважимо, що таким підходом до розв'язування задач визначається *суть декларативного програмування*.

Найбільш повно і строго ідеї логічного програмування реалізувались у мові програмування Пролог. Про це говорить і назва мови - ПРОграмування на основі ЛОГіки. Розробка мови Пролог, первісний варіант якої було створено А.Кольмерое у 1972 р. у Марсельському університеті, зробила логічне програмування практичним інструментом, доступним широкому загалу програмістів. Потрібно зазначити, що Пролог не єдина мова програмування, на якій можна писати програми з використанням концепцій логічного програмування[1].

1.2. Короткі відомості з логіки предикатів

Теоретичною основою мови Пролог є розділ формальної логіки - логіка предикатів першого порядку.

Атомарні формули

У логіці предикатів елементарним об'єктом, що володіє істинносним значенням є *атомарна формула*. Атомарна формула складається зі символічного позначення предикату та термів, що виступають у ролі аргументів цього предикату. У загальному випадку позначення предикату - це ім'я відношення, що існує між аргументами.

Предикат - така частина виразу, зміна якої не може не викликати за собою серйозної зміни смислу і навіть може привести до безглуздя.

Аргумент - це така частина виразу, яка може змінюватись у тих межах, доки вираз залишається можливим для розуміння.

[1] Наприклад, DUC, ESP, HCPRVR, Парлог, KL1 тощо.

Атомарна формула записується як позначення предикату, за яким у дужках розміщується декілька аргументів. Кожен аргумент - це терм. Загальний вигляд атомарної формули:

$P(t_1, t_2,..., t_n)$, де P - позначення предикату, $t_1, t_2,..., t_n$ - терми.

Терм - це або константа, або змінна, або використання функції. "Використання функції" записується як символічне позначення функції, за яким в дужках розміщується список аргументів. Кожен аргумент функції сам є термом. Загальна форма використання функції:

$f(t_1, t_2,..., t_n)$, де f - позначення функції, $t_1, t_2,..., t_n$ - терми.

У логіці предикатів прийняті наступні позначення:

- літери a, b, c означають константи;
- літери x, y, z означають змінні;
- літери f, g, h означають функції;
- літери P, Q, R означають предикати.

Правильно побудовані формули

Правильно побудована формула (ППФ) отримується у результаті комбінування атомарних формул із логічними з'єднувачами.

Таблиця логічних з'єднувачів:

Символ	Смисл	Символ	Смисл
~ або $^-$	"ні" (заперечення)	$\Rightarrow$	слідування (імплікація)
& або $\wedge$	"і"	$\Leftrightarrow$	"тоді і тільки тоді, коли"
$\vee$	"або"		

Правильно побудована формула визначається наступним чином:

- атомарна формула є ППФ;
- якщо A і B - ППФ, то ППФ будуть і формули:

ППФ	Читається так
$\overline{A}$	"не A"
$A \& B$	"A і B"
$A \vee B$	"A або B"
$A \Rightarrow B$	"A імплікує B" ("якщо A, то B")
$A \Leftrightarrow B$	"A тоді і тільки тоді, коли B"
$\exists x \ A$	"існує x таке, що A"
$\forall x \ A$	"для довільного x, A"

Тут $\exists x$, $\forall x$ - квантори змінних. Якщо в атомарній формулі як терм виступає змінна, то це значить, що у даному місці може бути більш ніж один елемент з області, яка інтерпретується атомарними формулами. Для ППФ квантор $\exists x$ означає, що в області інтерпретації *існує хоча б один елемент*, який при підстановці на місце змінної, зробить дану ППФ істинною. Аналогічно, квантор $\forall x$ означає, що ППФ буде істинною при підстановці довільного елемента.

Зауваження 1.

1.1. Імплікацію $A \Rightarrow B$ можна замінити як $\overline{A} \vee B$ (і навпаки). Цікаво, що при хибному А і довільному (хибному або істинному) В все твердження (імплікація) істинне. Іншими словами, "з хибного припущення слідує, що завгодно (можливий і хибний, і істинний висновок)";
1.2. Для формальних тверджень використовують певні правила, наприклад, $\overline{\overline{A}} = B$; $\overline{(A \vee B)} = \overline{A} \wedge \overline{B}$; $\overline{(A \wedge B)} = \overline{A} \vee \overline{B}$.

Фразова форма

Фразова форма логіки предикатів - це спосіб запису формул, при якому використовуються тільки з'єднувачі $\&, \vee, \overline{\ }$.

Позитивна чи негативна атомарна формула називається *літералом*. Кожна фраза - це множина літералів, сполучених символом $\vee$. Негативні літерали розміщуються у кінці кожної фрази, а позитивні - на початку.

Фраза, що містить тільки один позитивний літерал, називається *фразою Хорна*.

Наприклад, $C \vee \overline{E} \vee \overline{F} \vee \overline{G}$.

Фрази можна записувати з використанням зворотної стрілки імплікації, що стоїть між позитивними і негативними літералами. Літерали, що розміщені по різні сторони від стрілки, сполучаються комами.

Наприклад,

$C \vee D \vee \overline{E} \vee \overline{F}$ можна записати як $C, D \leftarrow E, F$ (не фраза Хорна);

$C \vee \overline{E} \vee \overline{F} \vee \overline{G}$ можна записати як $C \leftarrow E, F, G$ (фраза Хорна). Дійсно, враховуючи "Зауваження 1", маємо:

$C \vee \overline{E} \vee \overline{F} \vee \overline{G} = C \vee \overline{(E \wedge F \wedge G)} = C \Leftarrow (E \wedge F \wedge G)$.

Правило резолюції

Правило резолюції вказує, як одержати нову фразу з двох даних фраз. Доведено, що під час використання лише правила резолюції

можна вивести довільний наслідок із множини аксіом, які подані у фразовій формі.

Правило резолюції діє наступним чином.

Дві фрази можуть бути резольвовані між собою, якщо одна з них містить позитивний літерал, а інша - відповідний негативний літерал з одним і тим же позначенням предикату і однаковою кількістю аргументів, і якщо аргументи обох літералів можуть бути уніфіковані (тобто погоджені) один з одним.

При цьому, *константа уніфікується з константою*, якщо вони однакові і стоять на однакових позиціях у переліку аргументів відповідних предикатів; *змінна уніфікується з довільною константою*, і якщо в одній і тій же фразі змінна зустрічається більше одного разу, то резольвента буде містити дану константу на тих місцях, де розглядувана змінна знаходилась у фразі, що резольвується.

Зауваження 2.

У фразовій формі не використовується явна квантифікація змінних. Однак, неявно всі змінні квантифіковані. Наприклад, у фразі $Q(x,y) \vee \overline{R}(x,y)$ розуміється $\forall x \forall y (Q(x,y) \vee \overline{R}(x,y))$.

Розглянемо приклади відповідних теорій, побудованих із двох фраз:

1. Фраза $P(a,b) \vee \overline{Q}(c,d,e)$ і фраза $Q(c,d,x) \vee \overline{R}(c,d,x)$. Резольвента - $P(a,b) \vee \overline{R}(c,d,e)$ стає новою (третьою) фразою даної теорії.

2. Фраза $P(a) \vee \overline{Q}(b,c)$ і фраза $Q(c,c) \vee \overline{R}(b,c)$ не резольвуються (аргументи предикатів Q і $\overline{Q}$ не уніфікуються).

1.3. Розв'язання задач на Пролозі

Пролог - мова, що швидко розвивається. За останні роки з'явилося більше десятка його нових діалектів; розроблені об'єктні розширення Прологу, в якому є засоби для роботи з абстрактними типами даних. Виявилося, що на Пролозі можливе розв'язання задач з використанням підходів, далеких від математичної логіки, зокрема, функціонального і об'єктно-орієнтованого програмування.

Пролог, який відноситься до реляційних мов високого рівня, суттєво відрізняється від процедурних мов програмування (Бейсік, Паскаль, Фортран тощо) як за типами даних, так і за базовими конструкціями, та розрахований на ефективне розв'язання широкого кола задач певного класу, реалізація яких процедурними мовами значно ускладнюється. Але Пролог не є універсальною мовою програмування, тому існує ряд класів задач, для розв'язку яких використання Прологу є малоефективним. До таких задач слід віднести обчислю-

вальні задачі: засоби виконання обчислювальних дій Прологу досить слабкі і носять переважно допоміжний характер.

Пролог придатний для задач, де головне значення мають складно структуровані нечислові дані, а також для задач, де важливу роль відіграє пошук розв'язку серед багатьох варіантів.

Процес розв'язання задачі на Пролозі проходить у декілька етапів.

1. Аналізуються дані умови задачі (або ще кажуть, предметна область): факти, функції, відношення. Вибираються позначення для них.
2. Описуються природною мовою з точки зору істинності відношення та функції.
3. Описані відношення оформлюються як аксіоми у вигляді фраз (фактів, правил), зрозумілих Прологу для роботи з ними. Множина таких фраз складає програму, що моделює структуру предметної області. Програма вводиться у робоче поле компілятора Турбо-Пролог.
4. Формулюються та виконуються запити до введеної програми, завдяки чому досягається результат розв'язку задачі.

Варто зазначити, що за замовчуванням Пролог працює з припущенням про замкнутість світу (теорії, предметної області). Це означає наявність заперечення деякої фрази, якщо вона не представлена у програмі. Тобто, на довільний запит по перевірці істинності фрази дається повідомлення Yes або No. Альтернативою замкнутості є припущення про відкритість світу, за яким у разі відсутності деякої фрази у програмі вважається, що дана фраза не істинна і не хибна, тобто недоказова.

Назвемо деякі області, де використовується мова логічного програмування Пролог:

- створення динамічних реляційних баз даних;
- переклади з однієї мови на іншу;
- реалізація експертних систем і оболонок експертних систем;
- управління виробничими процесами;
- доведення теорем і пакети штучного інтелекту;
- розробка швидких прототипів прикладних програм;
- створення природно-мовних інтерфейсів для існуючих систем.

§2. РОБОТА З ПРОГРАМОЮ НА ТУРБО-ПРОЛОЗІ

2.1. Робоче середовище Турбо-Пролог версії 2.0

Після запуску системи Турбо-Пролог версії 2.0 (файл prolog.exe) і появи повідомлення про ініціалізацію системи на екрані з'являються головне меню та чотири вікна:

Редагування - для вводу тексту програми;
Діалогу - для вводу запитів і видачі результатів;
Повідомлень - для видачі повідомлень компілятора;
Трасування - для видачі інформації про хід виконання програми.

Так як розмір вікон може змінюватися, то активні вікна можуть перекривати пасивні.

Команди головного меню:

File - Команди для управління файлами
Edit - Редагування програми
Run - Запуск програми
Compile - Компіляція програми
Options - Параметри компіляції
Setup - Зміна параметрів системи

Розглянемо детальніше деякі команди головного меню.

File - Це меню містить дев'ять команд обробки файлів.

Load Завантажує файл для обробки.
Існуючий текстовий файл завантажується у буфер редактора, після чого цей файл (початковий текст програми) можна редагувати, компілювати або запускати на виконання. Після виконання даної команди система запитує ім'я файлу, задати яке можна або вводом з клавіатури або, натиснувши *Enter* у відповідь на запрошення *File Name*, перейти курсором у відповідний каталог та на потрібний файл. З клавіатури можна також ввести маску, за якою буде виведено для вибору відфільтрований список файлів. Поява повідомлення *"Save current text (y/n):" ("Зберегти поточний текст (так/ні):")* говорить про те, що користувач намагається завантажити новий файл або вийти з системи, попередньо не виконавши операцію збереження зміненого початкового тексту програми. Далі необхідно натиснути відповідну клавішу.

Pick Виводить список файлів, що завантажувались на даному сеансі роботи для їх повторного вибору та обробки.
Ім'я найвищого файлу зі списку відповідає файлу, що знаходиться в даний момент у буфері редактора. Вибір елементу -

-load file-- генерує запрошення *File Name*, дії з яким аналогічні діям у команді *Load*.

New file Очищає вікно редагування для введення та редагування тексту нової програми.
За замовчуванням для нового файлу встановлюється ім'я WORK. pro, яке пізніше при збереженні можна змінити, вибравши команду *Write to*.

Save Зберігає в робочому каталозі файл програми.

Write to Записує у новий файл (каталог) текст програми з вікна редагування.
Текст програми, що редагується зберігається у новому файлі або перезаписується у файл, що вже існує. В останньому випадку система запитує підтвердження необхідності його перезапису.

Directory Надає можливість вибору логічного пристрою та каталогу для перегляду файлів.

Change dir Змінює поточний каталог.

OS shell Викликає операційну систему (ОС).
Управління передається операційній системі. Для повернення у Турбо-Пролог необхідно в ОС виконати команду *exit*.

Edit - Створення нової та модифікація тексту існуючої програми.
Працює як текстовий редактор. При роботі у редакторі є можливість для виклику вмонтованої допомоги (*F1*). Крім редагування вихідного тексту поточної програми можна відредагувати будь-який інший файл, не знищуючи при цьому текст поточного файла. Крім того, можна перенести у текст програми, що редагується, потрібний фрагмент (або весь текст) програми з іншого файла. Обидві ці можливості виконуються викликом з вікна редагування допоміжного редактора (режими *Xedit*, *Xcopy*).

Run - Створена за допомогою редактора програма компілюється та виконується.
Після успішного виконання цієї команди активізується діалогове вікно. Якщо в оперативній пам'яті знаходиться скомпільована програма, текст якої не модифікувався після останньої компіляції або останнього виконання, то чергова компіляція (перекомпіляція) не виконується.

Compile- Компілюється програма, текст якої містить вікно редагування.

Memory Скомпільована програма (отриманий код) розміщується в оперативній пам'яті.

OBJ file Скомпільована програма має формат об'єктного файла, що записується в OBJ директорії.
Ця директива може використовуватися у випадку, коли створюється проект і перед його компоновкою потрібно відредагувати (або відкомпілювати) декілька модулів. При компіляції у файл.OBJ (якщо програма є частиною проекту) на початку вихідного тексту програми необхідно написати ключове слово *project* разом з іменем проекту.

EXE file Скомпільована програма має формат виконуваного файла, що записується в EXE директорії.
Автоматично встановлюються зв'язки (компоновка). Якщо програма, яка компілюється, є частиною проекту, то процес встановлення зв'язків задається описом проекту.

Project Компілюється проект зі всіма модулями.
Спочатку компілюються всі модулі, які входять до складу проекту, а потім встановлюються зв'язки між ними.

Link only Встановлюються зв'язки файла (файл тільки компонується).
Компонується об'єктний файл, який відповідає вихідному тексту, що знаходиться у даний момент у редакторі. Якщо поточний вихідний текст є частиною проекту (містить директиву *project*), то компонується весь проект.

Options- Наступні команди дозволяють визначити параметри:

Link options Встановлення зв'язків під час компіляції.
Edit PRJ file Редагування файлу проекту.
Compiler directives Директив компілятора.

Setup - Надає можливість змінити наступні параметри системи:

Colors Визначення кольорів елементів вікон, головного меню, рядка значень функціональних клавіш тощо.
Window size Встановлення розміру і розміщення вікон.
Directories Визначення робочих каталогів.
Miscellaneous Адаптер, екран, клавіатура (функції службових клавіш та їх комбінацій), рядок функціональних клавіш для різних режимів роботи компілятора.
Load SYS file Завантаження файла SYS з параметрами.
Save SYS file Запис поточних параметрів у файлі SYS.

2.2. Програма на Турбо-Пролозі

Об'єкти та структури даних

Програми на мові Пролог оперують даними двох основних типів: числами та атомами. *Атом* - це рядок символів, що позначає деякий абстрактний об'єкт. Атом починається з рядкової (малої) літери і складається з літер, цифр та символу підкреслення. Якщо атом повинен містити інші спеціальні знаки або починатися з прописної літери, то він береться в апострофи.

Числа можуть бути дійсними та цілими. Числа і атоми можуть об'єднуватися у конгломерати – структури та списки.

Структура - це конструкція, що складається з імені структури і набору її елементів, які відокремлюються комами і беруться у круглі дужки. Елементами структур можуть бути числа, атоми, змінні, інші структури та списки. Кількість елементів структури фіксована і не може змінюватися при виконанні програми.

Списки являють собою об'єднання елементів певного виду, які відокремлюються комами і беруться у квадратні дужки. Списки відрізняються від структур тим, що не мають імен, і тим, що кількість їх елементів може змінюватися при виконанні програми.

Елементами структур можуть бути змінні.

Змінна у мові Пролог, на відміну від процедурних мов програмування, не розглядається як виділена ділянка пам'яті ЕОМ; вона служить для позначення об'єкту, на який не можна посилатися по імені. Її можна вважати локальним іменем деякого об'єкту. Значення змінній може присвоїтися у традиційному розумінні тільки локально, в межах однієї фрази програми. Змінна повинна починатися з прописної (великої) літери або символу підкреслення і містити тільки символи літер, цифр і підкреслення.

Змінна, що складається тільки з символу підкреслення, є анонімною і використовується у тому випадку, коли ім'я змінної не суттєве.

Приклади даних:

- атоми: хома, 'Хома', 10_00, '2_5';
- числа: 1, 1.5, 0.00006, 1.2E-23;
- структури: має(петро, 'ЕОМ'), вчиться(ольга, 10_Б);
- списки: [1,3,5,8], [петро, хома, леся, дмитро], [к11_А,к11_Б, к11_В];
- змінні: X, Хома, _, _час.

Іноді об'єкти даних у Пролозі (числа, атоми, змінні) називають *термами*, а структури - *складеними термами*. Терм має ім'я - *функтор*, а кількість його аргументів визначає *арність* терму. Наприклад, атом хома є термом з функтором хома і арністю 0, структура має(петро, 'ЕОМ') є термом з функтором має і арністю 2.

Типи даних

Для компілятора використовуються базові та специфічні типи даних.

Базові типи даних:

char	окремий символ (8-бітовий ASCII- символ), що міститься в апострофах. Символ ASCII позначається початковим символом \, за яким записується код символу або сам символ;
integer	ціле число у діапазоні від -32768 до 32767;
real	дійсне число у діапазоні [+1E-307; +1E+308] - для додатніх чисел і [-1E+308; -1E-307] - для від'ємних;
symbol	послідовність літер, цифр і знаків підкреслення, яка починається з рядкової (малої) літери або міститься у лапках;
string	послідовність[1] символів, що міститься у лапках;
file	ім'я файла.

Специфічні типи даних:

ref	- цифрові посилання бази даних;
place	- в пам'яті або в розширеній системі пам'яті або у файлі;
dbasedom	- тип даних, що генерується для термів у глобальній (зовнішній) базі даних;
bt_selector	- селектор бінарного дерева;
db_selector	- визначений користувачем селектор зовнішньої бази даних;
bgi_ilist	-

Алфавіт

Для Турбо-Прологу *алфавіт* складається з таких символів:

1. англійські і російські[2] літери
2. цифри 0,1,2,...,9
3. знаки арифметичних дій +, -, *, /, mod, div
4. знаки відношень <, >, =, <=,>=,<>,><
5. дужки (,), [,]
6. знаки для коментарів /*, */, %
7. обмежувачі . , ; ' " :-
8. спеціальні знаки (символи) !, ?, #, @, _, ~, є, ї, ґ та ін.

---

[1] Дані типу symbol на відміну від даних типу string запам'ятовуються у внутрішній символьній таблиці, яка розміщується в оперативній пам'яті. Для побудови цієї таблиці необхідний додатковий час, але її використання забезпечує найбільш швидкий пошук.

[2] При використанні в іменах атомів деяких українських символів (ї,і,є,ґ) виникає помилка компіляції. Тому будемо їх уникати або замінювати подібними російськими.

Синтаксис програми

Програма на мові Пролог складається з множини фактів та правил і її можна розглядати як мережу відношень, що існують між термами. Факти та правила називаються *фразами програми*.

Факт - це твердження про те, що виконується деяке конкретне відношення або описується деяка властивість об'єкту (фраза без умов). Факти є структури, кожна з яких завершується символом "крапка". Вони являють собою ті дані, з якими оперує програма. Факт, що складається зі структури з одним елементом, описує деяку властивість. Факт, що має арність більшу 1, описує відношення між об'єктами.

Правило - це факт, істинність якого залежить від інших фактів. Правило є конструкція виду:

```
висновок :-
умова 1, умова 2,..., умова n.
```

Тут висновок, умова 1, умова 2,..., умова n - структури; терм висновок є заголовком правила, терми умова 1, умова 2,..., умова n - підцілями правила. Формальний запис правила інтерпретується так: "Твердження висновок є істинним, якщо одночасно істинні твердження умова 1, умова 2,..., умова n" або "Для того, щоб твердження висновок було істинним, необхідно, щоб були істинними твердження умова 1, і умова 2, і т.д. і умова n".

Сукупність фактів і правил, які записані у програму у вигляді термів з одним і тим же функтором і мають одну і ту ж кількість аргументів, визначає певний *предикат*.

Найпростіша Пролог-програма - це множина фактів. Множину фактів ще називають базою даних програми.

Крім фактів і правил, що визначаються програмістом, можна використовувати й інші, так звані, стандартні предикати, які виконують різноманітні допоміжні дії.

Отже, замість десятків різноманітних операторів у процедурних мовах Пролог має один вид оператора - правило.

Структура програми

Для програми на Турбо-Пролозі необхідна наявність певних розділів компілятора - оголошення програмних секцій domains, predicates, clauses тощо);.

Програма на Турбо-Пролозі може складатися з таких розділів:

constants	Оголошення атомів-констант. Розділ не обов'язковий.
domains	Оголошення типів даних для аргументів предикатів. Зміст розділу може бути відсутнім, але для кращої читабельності цю директиву включають у програму.
global domains	Оголошення глобальних типів. Розділ може бути відсутнім.

database	Оголошення предикатів динамічної бази даних. Розділ може бути відсутнім.
global database	Оголошення предикатів глобальної динамічної бази даних. Розділ може бути відсутнім.
predicates	Оголошення предикатів. Якщо типи даних деяких предикатів перевизначені, то необхідно їх попередньо описати у розділі domains. Розділ обов'язковий.
global predicates	Оголошення глобальних предикатів для різних модулів проекту. Розділ може бути відсутнім.
goal	Визначення цілі (мети). При наявності у даному розділі тексту запиту компілятор, виконуючи програму, автоматично намагається погодити цей запит. Якщо у програмі відсутній даний розділ, то користувач може вводити запити до програми.
clauses	У цей розділ записуються фрази, тобто міститься сам текст програми. Розділ, як правило, необхідний.

Опції компілятора (записуються перед розділами програми):

code	Опція зміни максимально можливого розміру програми, що розміщується у оперативній пам'яті. Для ЕОМ з невеликою оперативною пам'яттю необхідно зменшувати місце для програмного коду, залишаючи більше місця для стеку. Число, що слідує за code задає параграфи: 1 параграф 16 байтів. За замовчуванням величина code - 1024 (16К).
diagnostics	Видача користувачу діагностичної таблиці зі списком предикатів та їх найважливішими властивостями. Таблиця видається за умови, що у програмі є розділ goal.
include	Надає можливість уставити текст файла іншої програми у програму, що компілюється. Ім'я файла задається у форматі DOS. Ця опція дозволяє написати спільні для різних програм частини один раз та підключати їх до цих програм при компіляції.
nobreak	Відміняє опитування клавіатури на появу переривання (Ctrl+C, Ctrl+Break) під час виконання кожного предикату програми.
nowarnings	Заглушує попередження компілятора про те, що у програмі змінна зустрічається один раз або що вона не зв'язана.
project	Використовується при модульному програмуванні. В описанні проекту визначаються всі модулі.
shorttrace	Включає скорочене трасування. Програма при цьому виконується у режимі "за кроками". Програма оптимізується.

trace — Повне трасування. Внутрішня оптимізація відключається. Для трасування у певному місці програми записується на її початку trace і відключається трасування trace(off); у потрібному місці програми трасування включається trace(on). Для трасування "за кроками" необхідно скористатися командою функціонального меню (клавіша F10).

Зауваження 3.

а) При записі розділів програми необхідно враховувати, що:
- константи, домени та предикати повинні бути оголошені до того, як вони будуть використовуватися;
- програма може мати не більше одного розділу *goal*;
- всі глобальні оголошення повинні бути зроблені до локальних;
- розділи *database* можуть мати імена, але задане ім'я як назва розділу в програмі може бути використане лише один раз;

б) Щоб спостерігати протокол трасування одночасно з виконанням програми "за кроками", необхідно, щоб вікна компілятора (див. стор. 10) не перекривались, а вікно трасування було необхідного розміру.

2.3. Виконання програми

Виконання програми на мові Пролог є доведенням істинності (погодженням) деякого логічного твердження у межах даної сукупності фактів і правил.

Логічне твердження, що доводиться, формулюється у вигляді набору предикатів, з'єднаних між собою символом "кома" (логічне "і") або символом "крапка з комою" (логічне "або"). Таке твердження з точки зору Пролог-програми є *запитом*, який необхідно:

- під час запуску компілятора (команда *Run* головного меню) ввести у діалогове вікно після повідомлення *Goal:* (Ціль:),
- або записати як частину тексту програми у розділі goal.

Виконання запиту – це єдина можливість виконати програму і відшукати відповідь до задачі, що розв'язується.

Використання константи як аргументу у предикатах запиту дає змогу задати вхідні дані до програми, а використання змінної – отримати вихідні дані. Втакий спосіб дані можна ввести тільки до початку роботи програми, а одержати результати – після закінчення.

Для вводу, виводу даних у процесі роботи програми використовуються стандартні предикати вводу-виводу, серед яких:

readint (X), readreal (X), readln (X) - для введення з поточного пристрою вводу відповідно цілих, дійсних чисел та рядка символів;

write (Зн1, Зн2, Зн3,...) – виводить задані значення $Зн_1$, $Зн_2$, $Зн_3$,... на поточний пристрій виводу, де $Зн_1$, $Зн_2$, $Зн_3$,... - константи або змінні;

nl – переводить вихідний потік на наступний рядок поточного пристрою виводу.

Предикати вводу-виводу називають позалогічними. Їх дії у процесі розв'язку логічних цілей породжують побічні ефекти. *Побічний ефект* - це така дія предикату, яка не анулюється при поверненні у разі роботи механізму повернення Турбо-Прологу (стор. 20).

Приклад 1. Відомо те, що Петро та Ольга навчаються у 10 класі, Хома та Леся - у 9 класі. Один учень знає іншого, якщо вони вчаться в одному класі. Вважається, що певний учень не може знати сам себе.

```
domains                                              /* pr_1. pro */
   учень = symbol
   клас  = integer
predicates
   вчиться ( учень, клас)
   знае ( учень, учень)
clauses
   вчиться ( хома , 9 ).
   вчиться ( петро , 10 ).
   вчиться (ольга, 10 ).
   вчиться ( леся , 9 ).
   знае ( X , Y ) :-
        вчиться ( X , Клас ) ,
        вчиться ( Y , Клас ) ,
        X <> Y.
```

Коментар. У програмі оголошуються предикати вчиться/2 і знае/2, аргументи учень типу "рядок символів" і клас типу "ціле число". Умова того, що учень X навчається в одному й тому ж класі з учнем Y забезпечується однаковим іменем змінної у відповідних умовах правила.

Організація запитів

Після того, як компілятор виконає команду Run головного меню, можна ввести у діалогове вікно запит до програми.

Простий запит складається з імені предикату, за яким у дужках міститься список аргументів.

Якщо введено *запит з константами*, тобто у запит не входять змінні, то компілятор дає одну з відповідей Yes або No. Якщо *у запит входять змінні*, то компілятор намагається відшукати всі такі їх значення, при яких запит буде істинним. При відповіді змінній присвоюється знайдене значення або (у випадку, коли потрібне значення змінної не знайдено) видається повідомлення No Solution.

Якщо необхідно одержати *тільки одну відповідь* на запит із змінними, потрібно при формулюванні запиту використати відсікання.

Якщо *у запит входить анонімна змінна* (_), то це рівносильне наказу компілятору знехтувати значення аргументу. Така змінна буде погоджена будь з чим, але не буде мати ніякого значення, тобто вихідні дані за допомогою такої змінної не виводитимуться.

Складений запит утворюється з декількох простих запитів, з'єднаних між собою такими знаками: "кома" або сполучником (and) - логічне "і"; "крапка з комою" або сполучником (or) - логічне "або".

Можливі запити до програми:

Запит	Інтерпретація	Результат
вчиться (хома, 9)	Чи вчиться Хома у 9-му класі?	Yes
вчиться (ольга, 9)	Чи вчиться Ольга у 9-му класі?	No
вчиться (хома, К)	Де (у якому класі) вчиться Хома?	K=9 1 Solution
вчиться (X,10)	Хто вчиться у 10-му класі?	X= петро X= ольга 2 Solutions
знае (петро, X)	Кого знає Петро?	X= ольга 1 Solution
знае (X, Y)	Знайти тих, хто знає один одного?	X= хома, Y= леся X= петро, Y= ольга X= ольга, Y= петро X= леся, Y= хома 4 Solutions
вчиться (X, 9) , знае (леся, X)	Серед тих, хто вчиться у 9-му класі, відшукати тих, кого знає Леся?	X= хома 1 Solution
вчиться (X, 9) , знае (X, ольга)	Хто вчиться у 9-му класі і знає Ольгу?	No Solution
вчиться (X, 9) ; знае (петро, X)	Відшукати тих, хто вчиться у 9-му класі або тих, кого знає Петро?	X= хома X= леся X= ольга 3 Solution
вчиться (X, _) , знае (ольга, X)	Кого серед тих, які вчаться у будь-якому класі, знає Ольга?	X= петро 1 Solution

Зауваження 4.

Компілятор Турбо-Пролог може виконувати певні запити до програм, які не містять директиви goal, і подібно до деяких версій процедурних мов моделюється режим безпосередніх обчислень. Наприклад, незалежно від тексту пролог-програми можна обчислити значення виразу $\sqrt{3{,}2-2{,}95}$, виконавши у діалоговому вікні Турбо-Прологу запит X=sqrt(3.2-2.95). Одержимо відповідь X=0.5.

Операція співставлення

Основною операцією, що виконується над даними у Пролозі є *операція співставлення* двох термів, яка визначається так:

а) число співставляється тільки з рівним йому числом;

б) атом співставляється тільки з рівним йому атомом;

в) змінна співставляється будь з чим і при співставленні одержує як значення те, з чим вона співставляється;

г) структура співставляється з іншою структурою, якщо вони мають однакові функтори і однакову арність за умови, що всі їх аргументи попарно співставляються.

Розглянемо приклади виконання операції співставлення:

Вдале виконання	Невдале виконання
5 і 5; хома і хома; знає (хома, X) і знає (хома, петро) ; має (петро, 'ЕОМ') і має (петро, _)	хома і 'Хома' знає (хома, X) і знає (хома, Y)[1] має (петро, 'ЕОМ') і має (X, X) вік (леся, 15) і вік (леся,16)

Механізм повернення

Пролог доводить істинність тверджень (погоджує твердження) на основі *механізму повернення*.

Для виконання деякої сукупності підцілей (предикатів), які складають запит або входять як складові у правило, Пролог бере першу підціль і намагається довести її істинність (виконати підціль). Тому проглядаються всі факти і правила програми і шукається той факт, що співставляється з підціллю, або правило, заголовок якого з нею співставляється.

Якщо факт знайдено, то істинність підцілі вважається доведеною і береться наступна підціль. Якщо знайдено правило, то Пролог намагається у такий же спосіб довести істинність підцілей даного правила.

Якщо ж при пошуку виявлено, що є декілька варіантів доведення істинності підцілі (тобто декілька фактів або правил, що з нею співставляється), то Пролог автоматично відмічає так звану *точку повернення*. Якщо у деякий момент виконання програми чергова підціль не може бути виконана, то автоматично відбувається повернення до останньої відміченої точки повернення і Пролог намагається відшукати інший варіант доведення (інший шлях виконання програми).

[1] За умови, що змінні X та Y не конкретизовані (не мають певних значень) на момент виконання операції співставлення.

Варто зазначити, що Турбо-Пролог після першого доведення (Solution) вказаної сукупності підцілей намагається автоматично довести цю сукупність (запит) повторно, співставляючи першу підціль з новими фактами і правилами програми. Це призводить до того, що Турбо-Пролог автоматично знаходить не один, а всі можливі варіанти доведення твердження (можливі розв'язки), кількість яких вказується як N Solutions. Якщо істинність твердження, що доводиться не може бути визначено на основі фактів та правил програми, то Пролог видає повідомлення "No Solution".

У разі потреби роботу механізму повернення можна припинити, використовуючи відсікання. Відсікання використовується для запобігання непотрібних повернень і скорочення альтернативних шляхів у програмі, якщо наперед відомо, що той або інший шлях не приведе до розв'язку. Відсікання забезпечується предикатом "!", який дає вказівку компілятору не повертатися назад далі тієї точки, де стоїть даний предикат. Наприклад, доповнимо правило знае/2 так, щоб на запит знае(X, Y) одержати тільки одну відповідь (X= хома, Y= леся 1 Solutions):

```
знае (X, Y) :-
  вчиться (X, Клас) ,  вчиться (Y, Клас) , X <> Y, !.
```

Часто (наприклад, для побудови діалогу користувача з програмою) використовують предикат fail, за *яким Турбо-Пролог вимушений здійснити відкіт - невдало завершити процес погодження фрази програми*, що спонукає систему виконати погодження фрази програми ще раз. Використання відсікання і відкоту у певній фразі за умови успішного погодження всіх її предикатів, крім fail, забезпечує невдале виконання фрази та усунення всіх наступних відкотів.

Приклад 2. Запишемо програму, яка містить факти - набір назв (базу даних) деяких мов і правило, за яким користувач опитується на предмет знання тієї чи іншої мови.

```
domains                                                    /*pr_2. pro*/
   mov = symbol
predicates
   знае_мову (mov)
goal
   знае_мову(L),
   write ("Чи знаете Ви ", L ," мову?"), nl ,
   readln(A), A = так ,
   write ("Непогано знати ", L, " мову."), nl, fail.
clauses
   знае_мову ( англійську ).
   знае_мову ( німецьку ).
   знае_мову ( російську ).
   знае_мову ( китайську ).
```

Коментар. Так як програма містить розділ goal, то після виконання команди Run виконується запит, що складається з твердження, записаного у цьому розділі. Новий запит можна ввести, записавши його у дану секцію. Діалогове вікно для введення нового запиту можна використати лише вилучивши секцію goal з тексту програми. У запиті предикат знае_мову (L) співставляється з першим фактом-предикатом бази даних і змінна L набуває значення англійську. Перший предикат write трьома аргументами (рядок, змінна, рядок) виводить на екран повідомлення "Чи знаєте Ви англійську мову?", після чого предикат nl переводить курсор на наступний рядок екрана. Предикат readln призупиняє роботу програми та переводить її у режим очікування вводу. Якщо користувач з клавіатури вводить певний набір символів, то змінна A набуває значення введеного рядка символів. У випадку, коли введено рядок "так" другий предикат write виводить повідомлення "Непогано знати англійську мову."; якщо ж змінна A набуває іншого значення, то другий предикат write не виконується і повідомлення не виводиться. Предикат fail у випадку, коли змінній A присвоєне значення "так" ініціює наступне виконання запиту з переглядом фактів бази даних, що не розглянуті. Без цього предикату побічні ефекти, які породжуються предикатами readln та write, зупинили б процес погодження нерозглянутих фактів бази даних у випадку, коли при роботі з деяким фактом змінна A набуває значення "так". Таким чином, механізм повернення разом з предикатом fail забезпечують у даній програмі опитування користувача на предмет знання всіх вказаних мов, незалежно від того, позитивна чи негативна відповідь введена користувачем.

2.4. Семантика програми

Термін семантика (смисл) формули символьної логіки відноситься до істиносного значення цієї формули. Коли говорять про семантику константи у формулі, то розуміють її істиносне значення з урахуванням області інтерпретації. Коли ж мова йде про обчислення, то під семантикою деякої конструкції мови програмування розуміється поведінка комп'ютера при виконанні цієї конструкції. Оскільки Пролог одночасно є і логічною мовою, і мовою програмування, то для нього використовуються всі ці тлумачення терміну *"семантика"*.

Для пояснення змісту програми на Пролозі використовуються три семантичні моделі: декларативна модель, процедурна модель і модель у вигляді абстрактної машини. Існування трьох семантичних моделей надає мові Пролог більшої виразності.

Декларативна модель

В декларативній моделі специфікуються (встановлюються) істиносні значення конкретних випадків відношень. Слово декларативний використовується тому, що у фразах мови Пролог декларується, тобто оголошується, що будуть зберігатися деякі відношення між аргументами, якщо виконані всі умови, що входять до складу цих фраз.

Наприклад, фраза

```
знае (X, Y):-
        вчиться(X, Клас),  вчиться(Y, Клас),  X <> Y.
```

читатиметься так:

„Особи X та Y знатимуть одна одну, якщо
вони навчаються в одному класі і якщо це не одна й та ж особа".

Для цієї моделі порядок слідування умов не суттєвий, так як вважається, що всі умови повинні дотримуватись одночасно.

Процедурна модель

Для процедурної моделі умови, що входять до складу фрази специфікують *процес* визначення істиносного значення цієї фрази. Таким чином, умови трактуються як послідовність кроків, які необхідно успішно виконати для того, щоб зберігалось відношення, що міститься у висновку цієї фрази. Множина фраз з одним і тим же іменем терму і однаковою кількістю аргументів визначає предикат (див. на стор.15) і трактується як *процедура*. Запит з певним предикатом трактується як *виклик процедури* з таким же іменем і такою ж кількість аргументів, що і в предикаті запиту. Для того, щоб запит був погоджений успішно, необхідно, щоб процедура, що викликається у ньому, була виконана успішно. Кожна умова у фразі також розуміється як виклик процедури.

Процедурний смисл наведеної вище фрази буде таким:

„Один зі способів відшукати осіб, що знають одна одну - це:
по-перше, відшукати особу X, що вчиться у певному класі,
по-друге, відшукати особу Y, що вчиться у тому ж класі, де вчиться особа X,
і, по-третє, перевірити, чи знайдені X та Y є різними особами".

Умови певної фрази програми виконуються послідовно зверху вниз та зліва направо щодо того, як вони записані у тексті програми. Тому для процедурної моделі є суттєвим порядок виконання умов.

Модель у вигляді абстрактної машини

З позиції декларативної моделі Пролог-програма являє собою опис логічної структури предметної області задачі. Компілятор мови Пролог є активною силою, тобто процедурою, яка здатна зробити висновок з цього опису.

Під час виконання запиту компілятор застосовує по відношенню до множини фраз Пролог-програми деяку стратегію розв'язування задачі. З точки зору обчислення ця стратегія може бути описана за допомогою деякої абстрактної машини. В мові Пролог запит і множина фраз програми мають обчислювальний смисл. Це виявляється у тому, що вони викликають певну поведінку компілятора мови Пролог.

Сфери використання моделей

Декларативна модель найбільш близька до семантики логіки предикатів, що робить Пролог ефективною мовою для подання знань.

Однак у декларативній моделі неможливо адекватно подати ті фрази, в яких важливий порядок слідування умов–підцілей, наприклад, виведення повідомлень на екран дисплею у певній послідовності. У цьому випадку користуються процедурною моделлю.

Проте навіть процедурна модель не зможе пояснити смисл фраз, що викликають побічні ефекти при виконанні програми; наприклад, зупинка виконання запиту або вилучення факту з тексту програми. Зміст таких фраз пояснюється тільки за допомогою моделі у вигляді абстрактної машини.

ЛАБОРАТОРНА РОБОТА №1

Введення, редагування, компіляція та виконання програм

Мета: Одержати уміння та закріпити навички роботи з інтегрованим середовищем Турбо-Пролог 2.0. Освоїти технологію введення, редагування, компіляції та виконання програм, організації запитів, введення та виведення даних.

Теоретична частина: завдання та контрольні питання

І рівень

1. Призначення команд головного меню Турбо-Пролог.
2. Призначення команд редактора.
3. Які вікна містить система Турбо-Пролог 2.0?
4. Об'єкти даних Турбо-Прологу. Поняття структури та списку.
5. Поняття терму. Функтор та арність терму.
6. З яких програмних секцій (директив) складається програма?
7. Яка програма є найпростішою у мові Пролог?
8. З чого складається тіло програми (секція clauses)?
9. Як дається команда компілятору на виконання програми?

ІІ рівень

1. Види запитів до програми.
2. Як, використовуючи запити, передати у програму необхідні вхідні дані та одержати результати роботи?
3. Описати процес розв'язування задач на Пролозі.

ІІІ рівень

1. У чому полягає різниця між процедурним та логічним програмуванням? У чому суть декларативного програмування?
2. Порівняйте етапи розв'язку задачі на процедурній мові та на мові логічного програмування.
3. Дайте порівняльну характеристику поняття змінної у процедурних мовах та на Пролозі.

Практична частина

І рівень

1-13 варіанти. Ввести, відредагувати та скомпілювати текст програми прикладу №1. Зберегти програму у файлі "LNN1_1. pro", де NN – номер групи користувача. Виконати запити до програми:

- чи вчиться Ольга у 10-му класі?
- чи вчиться Хома у 10-му класі?
- чи знає Леся Хому?
- кого знає Ольга?
- хто знає Ольгу?
- відшукати тих, хто знає один одного.

II рівень

1-3 варіанти. Доповнити базу даних програми прикладу №1 фактами: Олег вчиться у 9 класі, Олег знає Тамару. Програму зберегти у файлі "LNN1_23. pro". Виконати запити до модифікованої програми:

- чи є така особа, яка знає і Хому і Петра?
- чи є така особа, яка знає Хому або Петра?
- хто вчиться у 9-му класі і знає Тамару?
- хто вчиться у 10-му класі і знає Тамару?
- відшукати тих, хто вчиться у 10-му класі або знає Тамару.

4-6 варіанти. Доповнити базу даних програми прикладу №1 фактами: Тамара вчиться у 10 класі, Леся знає Тамару. Програму зберегти у файлі "LNN1_26. pro". Виконати запити до модифікованої програми:

- чи є така особа, яка знає і Ольгу і Лесю?
- чи є така особа, яка знає Ольгу або Лесю?
- хто вчиться у 9-му класі і знає Лесю?
- хто вчиться у 10-му класі і знає Лесю?
- відшукати тих, хто вчиться у 9-му класі або знає Лесю.

7-9 варіанти. Доповнити базу даних програми прикладу №1 фактами: Олег вчиться у 10 класі, Валя вчиться у 8 класі, Валя знає Олега. Програму зберегти у файлі "LNN1_29. pro". Виконати запити до модифікованої програми:

- чи є така особа, яка знає і Хому і Петра?
- чи є така особа, яка знає Хому або Петра?
- хто вчиться у 10-му класі і знає Валю?
- хто вчиться у 9-му класі і знає Валю?
- відшукати тих, хто вчиться у 9-му класі або знає Олега.

10-13 варіанти. Доповнити базу даних програми прикладу №1 фактами: Олег вчиться у 9-му класі, Валя знає Петра, Валя знає Олега. Програму зберегти у файлі "LNN1_20. pro".

Виконати запити до модифікованої програми:

- чи є така особа, яка знає і Олега і Петра?
- чи є така особа, яка знає Олега або Петра?
- хто вчиться у 9-му класі і знає Валю?
- хто вчиться у 10-му класі і знає Валю?
- відшукати тих, хто вчиться у 10-му класі або знає Валю.

III рівень

Зберегти модифіковану програму (вказану у варіанті для III-го рівня), у файлі під новим іменем (вказано у варіанті). Виконати запит (вказа-но у варіанті) у покроковому режимі з трасуванням; домогтися візуалі-зації всього зображення вікна трасування під час такого виконання запиту. Зкомпілювати нову програму у EXE-файл. Якщо виникають помилки компіляції, то (змінивши розмір відповідного вікна) домогти-ся того, щоб вікно повідомлень містило код і інформацію про помилку у повному обсязі; виконати дії по усуненню помилки. EXE-

програма повинна успішно погоджувати вказаний запит. Вийти з компілятора у DOS. Виконати EXE-файл. У разі необхідності модифікувати програму так, щоб результатом її роботи був повний перелік осіб, виведений на екран.

Вказівка: у запиті використати предикати write і fail.

1 варіант

Програма: модифікована програма II рівня, 1-3 варіант.
Нове ім'я: "L1_3_1. pro".
Запит: "Відшукати тих осіб, які знають Олега, вказавши для кожної особи клас, у якому вона навчається ".

2 варіант

Програма: модифікована програма II рівня, 1-3 варіант.
Нове ім'я: "L1_3_2. pro".
Запит: "Відшукати тих, хто вчиться у 9-му класі".

3 варіант

Програма: модифікована програма II рівня, 4-6 варіант.
Нове ім'я: " L1_3_3. pro".
Запит: "Відшукати тих, кого знає Леся".

4 варіант

Програма: модифікована програма II рівня, 4-6 варіант.
Нове ім'я: " L1_3_4. pro".
Запит: "Відшукати тих, хто вчиться у 10-му класі".

5 варіант

Програма: модифікована програма II рівня, 7-9 варіант.
Нове ім'я: " L_3_5. pro".
Запит: "Відшукати тих, кого знає Олег".

6 варіант

Програма: модифікована програма II рівня, 10-13 варіант.
Нове ім'я: " L1_3_6. pro".
Запит: "Відшукати тих осіб, які знають Валю, вказавши для кожної особи клас, у якому вона навчається".

Вимоги до захисту лабораторної роботи

Звіт виконаної роботи повинен містити назву теми, тексти запитів мовою Пролог та відповіді компілятора на запити; текст секції goal для завдання III-го рівня.

ЛАБОРАТОРНА РОБОТА №2

Описання предметної області за допомогою фактів і правил

Мета: Закріпити уміння та навички запису фактів та правил Пролог-програми, організації запитів, модифікації програм. Одержати уміння та навички з технології розв'язування найпростіших задач на Турбо-Пролог версії 2.0.

Теоретична частина: завдання та контрольні питання

І рівень

1. Якими стандартними типами даних оперує Пролог?
2. З яких фраз складається Пролог-програма?
3. Що розуміють під фактом програми? Синтаксис фактів.
4. Що являють собою правила програми? Синтаксис правил.
5. Описати директиви компілятора.

ІІ рівень

1. Як визначається операція співставлення двох структур?
2. Описати роботу механізму повернення Турбо-Прологу.
3. З якими припущеннями про предметну область працює Пролог?
4. Призначення предиката fail. Для чого у програмах використовують комбінацію "відсікання" - fail. Навести приклад.

ІІІ рівень

1. Який предикат припиняє роботу механізму повернення? З якою метою примусово припиняють таку роботу? Навести приклад.
2. Дайте порівняльну характеристику фразам Хорна та фразам мови ПРОЛОГ.
3. Яка різниця між поняттям предикату у логіці предикатів першого порядку та у ПРОЛОГ-програмі?
4. Опишіть семантичні моделі ПРОЛОГ-програми.

Практична частина

І рівень

1-13 варіанти. Відомо, що студенти у складі групи осіб відправляються у туристичну подорож. Дані про студентів подано у таблиці:

Ім'я	Факультет	Курс	№ гуртожитку
Петро	фізико-математичний	IV	3
Хома	філологічний	III	4
Тамара	філологічний	II	4
Ольга	фізико-математичний	IV	5
Тарас	фізико-математичний	I	3
Леся	філологічний	II	5

Записати програму, що містить факти "вчиться/3" і "проживає/2" на основі наведених даних. Програму доповнити процедурою "знає/2", за якою певний студент знає іншого, якщо вони навчаються на одному курсі і одному й тому ж факультеті або проживають в одному гуртожитку. Врахувати й те, що певна особа не може знати сама себе.

Зберегти програму у файлі "LNN2_1. pro", де NN – номер варіанту користувача.

Організувати запити до створеної множини фраз програми:

- чи вчиться конкретна особа на певному факультеті, курсі?
- хто вчиться на певному факультеті, курсі?
- чи проживає конкретна особа у певному гуртожитку?
- студенти яких факультетів проживають у певному гуртожитку?
- чи знає одна конкретна особа іншу конкретну особу?
- які особи знають одна одну?
- чи можна сказати про певну особу, що її знають?
- які особи знають одна одну і вчаться на різних факультетах?
- які особи вчаться на філфаці або проживають у гуртожитку №3?

II рівень

1-3 варіанти. Доповнити базу даних програми фактом про те, що до туристичної групи включено викладача Олега, який проводить заняття на IV-му курсі фізмату і на II-му курсі філфаку.

Модифікувати базу даних так, щоб можна було отримати відповідь на питання про те, чи є особа студентом або викладачем.

Модифікувати правило "знає/2" так, щоб ураховувалося, що студент і викладач знають один одного, якщо викладач проводить заняття на тому курсі, де навчається студент.

Зберегти програму у файлі "LNN22_13. pro".

Виконати запити до модифікованої програми:

- конкретна особа є викладачем чи студентом?
- чи знає викладач конкретного студента?
- кого із студентів знає викладач?
- кого з тих, хто проживає у гуртожитку №3, знає Олег?

4-6 варіанти. Доповнити базу даних програми фактом про те, що до туристичної групи включено викладача Віктора, який проводить заняття на I-му курсі фізмат факультету і проживає у гуртожитку №3.

Модифікувати базу даних так, щоб можна було отримати відповідь на питання про те, чи є особа студентом або викладачем.

Модифікувати правило "знає/2" так, щоб враховувалось, що студент і викладач знають один одного, якщо викладач проводить заняття на тому курсі, де навчається студент, або вони проживають в одному й тому ж гуртожитку.

Зберегти програму у файлі "LNN22_46. pro".

Виконати запити до модифікованої програми:

- конкретна особа є викладачем чи студентом?
- кого знає конкретна особа?
- хто проживає у конкретному гуртожитку?
- кого з тих, хто вчиться на IV курсі, знає Віктор?

7-9 варіанти. Доповнити базу даних програми фактом про те, що до туристичної групи включено викладача Віктора, який проводить заняття на І-му курсі фізико-математичного факультету і проживає у гуртожитку №5.

Записати нове правило "турист/4", за яким можна було б отримати такі дані про туриста: ім'я, професія (викладач або студент), факультет (на якому вчиться або викладає особа), номер гуртожитку.

Зберегти програму у файлі "LNN22_79. pro".

Виконати запити до модифікованої програми:

- яку професію має конкретний турист?
- хто з туристів проживає у конкретному гуртожитку?
- який турист має відношення до фізико-математичного факультету: його ім'я та професія?
- хто з туристів філологічного факультету знає викладача Віктора?

10-13 варіанти. Доповнити базу даних програми фактом про те, що до туристичної групи включено викладача Олега, який проводить заняття на І-му курсі фізико-математичного факультету і на ІІІ-му курсі філологічного факультету.

Записати нове правило "профіль/3", за яким можна було б отримати інформацію про профіль спеціальності, на якій навчається студент (проводить заняття викладач) - гуманітарний чи природничий і, окрім того, певні дані: ім'я та гуртожиток, де проживає особа.

Зберегти програму у файлі "LNN22_01. pro".

Виконати запити до модифікованої програми:

- який профіль у конкретної особи?
- студенти якого профілю проживають у певному гуртожитку?
- хто з гуманітаріїв знає викладача Олега?
- чи є серед тих, хто навчається на певному курсі, особи конкретного профілю?

ІІІ рівень

Організувати діалог користувача з програмою (вказана у варіанті). Діалогом передбачити постановку від програми користувачу питання і виведення програмою на екран потрібних відповідей у разі позитивної реакції користувача на дане питання (введення користу-

вачем з клавіатури необхідного набору символів). Програму зберегти у файлі (вказаний у варіанті) у каталог...\PROLOG\WORK.

Вказівка: у запиті використати предикати write і readln.

1 варіант

Програма: II рівень, варіант 1-3.

Питання: "Чи потрібна Вам інформація про те, кого із студентів знає викладач Олег?"

Файл: “l23_1. pro”.

2 варіант

Програма: II рівень, варіант 1-3.

Питання: "Чи потрібна Вам інформація про те, на яких факультетах проводить заняття викладач Олег?"

Файл: “l23_2. pro”.

3 варіант

Програма: II рівень, варіант 4-6.

Питання: "Чи хотіли б Ви дізнатися, кого серед тих, хто проживає у гуртожитку №3, знає викладач Віктор?"

Файл: “l23_3. pro”.

4 варіант

Програма: II рівень, варіант 4-6.

Питання: "Чи хотіли б Ви дізнатися, кого знає Тамара?"

Файл: “l23_4. pro”.

5 варіант

Програма: II рівень, варіант 7-9.

Питання: "Ви хотіли б дізнатися імена туристів, які мають відношення до фізико-математичного факультету і на якому курсі вони навчаються або викладають?"

Файл: “l23_5. pro”.

6 варіант

Програма: II рівень, варіант 10-13.

Питання: "Вас цікавить інформація про те, студенти якого профілю проживають у гуртожитку №5 і кого з них знає викладач Олег?"

Файл: “l23_6. pro”.

Вимоги до захисту лабораторної роботи

Звіт виконаної роботи повинен містити назву теми, текст програми, текст запитів мовою Пролог та відповідей на них для завдань І-го рівня; текст модифікованої бази даних і модифікованого правила, запити для завдань II-го рівня; текст секції goal для завдання III-го рівня.

§3. ОПЕРАЦІЇ НАД ТЕРМАМИ

Арифметичні операції

За допомогою арифметичних операторів + (додавання), - (віднімання), * (множення), / (ділення), mod (остача від ділення), div (цілочисельне ділення) конструюються арифметичні вирази. Стандартний предикат = обчислює їх.

Приклад 3. Обчислити середнє арифметичне двох чисел.

```
domains                                                    /*pr_3. pro*/
   число = real
predicates
   сер_ар ( число, число, число)
clauses
   сер_ар (X, Y, R) :-
       R = (X + Y) / 2.
```

Операції порівняння

Операції порівняння реалізуються предикатами: > (більше), < (менше), = (дорівнює), >= (більше або дорівнює), <= (менше або дорівнює), < >, >< (не дорівнює).

Приклад 4. Визначити більше з двох чисел.

```
domains                                                    /*pr_4. pro*/
   число = real
predicates
   макс (число, число, число)
clauses
   макс (X, Y, Max) :-
     X>=Y, Max=X ;
     X < Y, Max=Y.
```

Коментар. Не можна не врахувати умову X < Y, так як програма буде некоректною: твердження "чи серед чисел 7 і 5 число 5 є максимальним?" за першою частиною правила не погодиться, але за другою - буде істинним, так як без умови X < Y максимальним буде число, яке представлено другим аргументом.

Операції перевірки типу

Можна передбачити деякі помилки виконання програм, перевіривши, чи відносяться аргументи запиту до потрібного типу.

Стандартні предикати:

not (Атом)- заперечення. Виконується успішно, якщо аргумент Атом є ціль, яка не досягається (твердження Атом є хибним).
Прототип[1]*: (o).*

free (Змінна) - перевіряє, чи аргумент Змінна є неконкретизована змінна. Виконується успішно, якщо Змінна не зв'язана певним значенням.
Прототип: (i).

bound (Змінна) - перевіряє, чи аргумент Змінна є конкретизована змінна. Виконується успішно, якщо Змінна зв'язана.
Прототип: (o).

Приклад 5. Є дані про студентів вузу: їх імена та курс, на якому вони навчаються. Для будь-якої особи необхідно визначити, чи вона не є студент. Такою буде та особа, яка не вчиться на жодному з курсів.

```
domains                                                    /*pr_5. pro*/
  номер = integer
  особа = symbol
predicates
  курс (особа, номер)
  не_студент (особа)
clauses
  курс (хома, 1).   курс (петро, 2).   курс (олег, 3).
  курс (ольга, 4).   курс (леся, 5).   курс (василь, 2).
  не_студент (X) :-
    bound (X), not ( курс (X,1) ), not ( курс (X,2) ),
    not ( курс (X,3) ), not (курс (X,4) ), not( курс (X,5) ).
```

Коментар. За правилом спочатку перевіряється, чи аргумент запиту є константою. Особа не студент, якщо вона не навчається ні на 1-му і т.д., ні на 5-му курсі. Тому на запит не_студент(олег) відповідь No, а на запит не_студент(христя) відповідь Yes.

Приклад 6. Знайдемо одне з чисел, якщо є друге число та значення їх середнього арифметичного за програмою на стор. 32. Під час компіляції виникає помилка ("змінна не зв'язана у твердженні"). Модифікуємо програму так, щоб уникнути помилки.

```
domains                                                    /* pr_6. pro*/
  число = real
predicates
  сер_ар (число, число, число)
```

[1] Вид аргументів:
(i) input - параметр відомий, тобто аргумент повинен мати певне значення;
(o) output - параметр невідомий - аргументові передається певне значення.

```
clauses
  cep_ap (X, Y, R):-   /*(1)*/
       bound(X), bound(Y),
       R = (X + Y) / 2.
  cep_ap (X, Y, R):-                                    /*(2)*/
       free (X), bound (Y), bound (R),
       X = 2 * R - Y.
  cep_ap (X, Y, R):-                                    /*(3)*/
       free (Y), bound (X), bound (R),
       Y = 2 * R - X.
```

Коментар. Процедура cep_ap/3 реалізується трьома правилами. За правилом (1) забезпечується знаходження середнього арифметичного двох даних чисел (R- змінна у запиті) та перевірка факту того, що у трійці аргументів запиту cep_ap/3 останній аргумент є числом - середнім арифметичним перших двох чисел (R - константа у запиті). За правилом (2) знаходиться перше з трьох чисел, якщо дано друге число та значення середнього арифметичного двох чисел (X-змінна у запиті). За правилом (3) запит буде виконано успішно, якщо у запиті X-число, Y-змінна, R-число..

Операції перетворення

Операції перетворення реалізуються стандартними предикатами:

char_int (Символ, Число) - перетворює символ у ціле число - ASCII код символу або навпаки.

Символ - тип char.

Число - тип integer.

Прототип: (i, o), (o, i), (i, i).

str_int (Рядок, Число) - перетворює рядок символів у ціле число або навпаки. Рядок символів не може містити інших символів, крім символів-цифр; символ "пробіл" може міститися тільки на початку або у кінці рядка.

Рядок - тип string.

Число - тип integer.

Прототип: (i, o), (o, i), (i, i).

str_char (Рядок, Символ) - перетворює рядок символів, що складається з єдиного символу, у символ або навпаки.

Рядок - тип string.

Символ - тип char.

Прототип: (i, o), (o, i), (i, i).

str_real (Рядок, Число) - перетворює рядок символів у дійсне число або навпаки. Рядок символів не може містити інших символів, крім символів-цифр, символу "десяткова крапка" та символу E (символ, що вказує на порядок числа при його представленні у плаваючій формі); символ "пропуск" може міститися

тільки на початку або в кінці рядка.

Рядок - тип string.

Число - тип real.

Прототип: (i, o), (o, i), (i, i).

Обчислення значень числових функцій

У Турбо-Пролозі можна обчислювати значення таких функцій:

Предикат	Функція	Предикат	Функція
abs(x)	$y=\|x\|$	sin(x)	$y=sin(x)$
exp(x)	$y=e^x$	cos(x)	$y=cos(x)$
log(x)	$y=lg(x)$	tan(x)	$y=tg(x)$
ln(x)	$y=ln(x)$	arctan(x)	$y=arctg(x)$
sqrt(x)	$y=\sqrt{x}$	round(x)	Округлює число до цілого

Зауваження 5.

Значення інших функцій можна обчислити, використовуючи вказані функції та відповідні математичні формули, наприклад:

$$\log_a b = \frac{lg\, b}{lg\, a} = \frac{ln\, b}{ln\, a},\ \pi=4*\text{arctg}(1).$$

§4. РЕКУРСІЯ

Рекурсія - потужний алгоритмічний метод Прологу, що забезпечує той же ефект, що і цикли у процедурних мовах.

Рекурсія - це процес розв'язування задачі, при якому дана задача розбивається на більш дрібні підзадачі і кожна з підзадач є зменшеним варіантом даної задачі за *способом розбиття і розв'язку*. Рекурсія реалізується конструкцією, яка включає в себе *граничну умову* та *рекурсивне правило*.

Приклад 7. Використовуючи рекурсивне означення натурального числа, забезпечити перевірку чисел на предмет їх належності до натуральних.

```
domains                                              /*pr_7. pro*/
    число = real
predicates
    натуральне (число)
clauses
  натуральне (1).                          /* гранична умова */
  натуральне (N) :-                        /* рекурсивне правило */
    N > 0,                                 /* умова виходу з рекурсії */
    N1 = N -1,
    натуральне (N1).
```

Коментар. Рекурсивно означити природною мовою натуральне число можна так: "Число є натуральним, якщо воно є одиницею або попереднє йому число (менше на одиницю) є натуральним".

У програмі *гранична умова*: число 1 є натуральне число; *рекурсивне правило*: число є натуральним, якщо воно додатне і якщо натуральним буде число, що менше від даного числа на одиницю. Отже, для того, щоб перевірити чи є певне число натуральним, необхідно перевірити, чи є натуральним попереднє йому число, а для попереднього – чи є натуральним наступне попереднє число і т.д. доти, доки процес не дійде до граничної умови або умови виходу.

Можливі запити до програми:

Запит	Інтерпретація	Відповідь
натуральне (5)	Чи є натуральним число 5?	Yes
натуральне (5.7)	Чи є натуральним число 5,7?	No
натуральне (a)	Чи є натуральним атом a?	Помилка типу

У прикладі наведеної програми рекурсивне правило містить предикати з аргументами-змінними, всі поточні значення яких у процесі виконання рекурсії заносяться у стек[1].

[1] Стек - область оперативної пам'яті, доступ до комірок якої здійснюється за принципом LIFO (Last Input First Output) - "останнім увійшов - першим вийшов".

Якщо програма у рекурсивному правилі не має умови виходу із рекурсії, то це може зумовити виникнення "нескінченної" рекурсії, і число елементів даних, які використовуються рекурсією, постійно зростатиме, що може призвести до переповнення стекової пам'яті.

Якщо рекурсивне правило не містить предикатів, аргументами яких є змінні, і остання підціль правила є рекурсивним викликом самої себе, то стекова пам'ять не використовується. Такий процес називається *процедурою з усуненням хвостової рекурсії.*

Приклад 8. Класичний приклад рекурсивного означення у Пролозі - програма "Предок", що складається з двох правил.

```
domains                                                    /*pr_8. pro*/
  имя = symbol
predicates
батько_мати ( имя, имя )
  предок ( имя, имя )
clauses
  батько_мати ( петро, ольга ).
  батько_мати ( петро, олег ).
  батько_мати ( ольга, василь ).
  предок (A, B) :-                                         /*(1)*/
      батько_мати (A, B).
  предок (A, B) :-                                         /*(2)*/
      батько_мати (C, B),
      предок (A, C).
```

Коментар. У базі даних подані факти про те, що Петро є батьком Ольги та Олега, а Ольга є матір'ю Василя. Пошук предків відбувається за двома правилами: за правилом (1) A є предком B, якщо A - один з батьків B; за правилом (2) A є предком B, якщо A є предком одного з батьків B, тобто предком C.

Можливі запити до програми:

Запит	Інтерпретація	Відповідь	Примітка
предок (петро, ольга)	Чи є Петро предком Ольги?	Yes	За правилом 1
предок (петро, X)	Хто є нащадком Петра?	X=ольга X=олег X=василь 3 Solution	За правилом 1 За правилом 1 За правилом 2

Якщо у правилі (2) переставити місцями умови, то одержимо версію процедури предок/2, яка вважатиметься не діючою. У цьому випадку в новій процедурі змінна C буде неконкретизована в момент обробки рекурсивної підзадачі предок (A, C), що на практиці призведе до знаходження компілятором спочатку правильних відповідей, а потім до виконання рекурсивних дій доти, доки не буде вичерпано

доступний об'єм стекової пам'яті. Така процедура називається *процедурою з лівою рекурсією*, так як у модифікованому правилі (2) рекурсивна підзадача буде першою порівняно з іншими підзадачами правила.

Вихідна рекурсія

Розглянуті програми містять рекурсивні правила, що використовують *метод вихідної рекурсії*.

У вихідній рекурсії будується така рекурсивна структура, за якою, щоб довести деякий факт, необхідно довести попередньо-рекурсивний факт, а щоб довести попередньо-рекурсивний факт, необхідно довести наступний попередньо-рекурсивний факт і т.д. доти, доки цей процес не дійде до перевірки граничної умови. Далі, виходячи з істинності чи хибності граничної умови формуються висновки (передаються потрібні значення) у зворотному (вихідному) порядку щодо наступно-рекурсивних фактів - від граничної умови аж до факту, що рекурсивно перевіряється.

Вхідна рекурсія

На відміну від попереднього методу, *метод вхідної рекурсії* дозволяє будувати рекурсивну структуру, починаючи одразу з граничної умови.

Враховуючи те, що у Турбо-Пролозі відсутні локальні змінні для збереження проміжних результатів і зміни результатів у процесі обчислення, то для реалізації вхідної рекурсії процедури доповнюються проміжними аргументами - *накопичувачами*.

Приклад 9. Внесемо зміни у програму, що визначає належність чисел до множини натуральних чисел. Вихідну рекурсію замінимо вхідною.

```
domains                                                    /*pr_9. pro*/
    число = real
predicates
    натуральне(число)
    натуральне (число, число)
clauses
  натуральне (N):-
    натуральне (1, N).
  натуральне (L, N) :-
      L< N, L1 = L + 1, натуральне (L1, N).
  натуральне(N, N).
```

Коментар. Накопичувач L (початкове значення 1) є логічною змінною, а не коміркою пам'яті. У процесі рекурсії передається не адреса, а значення. Так як логічні змінні мають властивість "одноразового запису", то змінене значення - нова логічна змінна - передається

кожен раз. Цей факт і виражається суфіксом 1 в імені змінної. Слід врахувати, що логічна змінна N, значення якої перевіряється, повинна слідувати по всім правилам та фактам програми, щоб одержати потрібний висновок при заключному виклику процедури натуральне(N, N).

Правило повтору та метод повтору

Правило повтору є найпростішим випадком рекурсії і визначається користувачем. Його конструкція:

```
повтор.                                    /* (1)  - факт */
повтор :-  повтор.                         /* (2) – правило */
```

Коментар. Факт повтор є завжди успішним твердженням. Оскільки існує правило (2), яке теж визначає предикат повтор, то точка повернення встановлюється на перший повтор. Другий повтор - це правило, яке використовує само себе як компоненту (третій повтор). Другий повтор викликає третій повтор і цей виклик виконується успішно, оскільки перший повтор задовольняє підціль повтор. Отже, правило (2) також завжди успішне. Тому визначений фразами (1) і (2) предикат повтор *ніколи не буває неуспішним*.

Метод повтору, що базується на правилі повтору, використовує відкіт (fail), причому відкіт у даному методі можна виконувати завжди. Цей метод є дуже гнучким засобом програмування, яке будучи однією з компонент деякого правила, забезпечує циклічне виконання функцій даного правила.

Приклад 10. Наведемо приклад програми, яка при виконанні запиту і введенні символів працює циклічно доти, доки користувачем не введено потрібного набору символів для зупинки роботи програми.

```
predicates                                          /*pr_10 pro*/
  повтор
  цикл_зупинка
  пароль(symbol)
goal
  цикл_зупинка.
clauses
  повтор.
  повтор :- повтор.
  цикл_зупинка :-
       повтор,
       write ("Введіть пароль"), nl,
       readln (S),
       пароль (S) ,  !.
  пароль ("стоп").
  пароль (_) :- fail.
```

Коментар. Правило цикл_зупинка першою підціллю має повтор, яка викликає повторне виконання всіх компонент правила цикл_зупинка. Дії предикатів write, nl, readln викликають відповідно появу повідомлення на екрані, перехід на новий екранний рядок та зчитування символів, що вводяться з клавіатури. Запропонований підхід можна використовувати при створенні діалогових програм, у яких аналізується відповідь користувача, що введена з клавіатури.

Для організації роботи різних меню, доступу до даних у базі даних або файлах на диску зручно використовувати більш ніж одне правило повтору. Наведемо загальну схему методу повтору з n правилами повтору:

```
n_цикл_зупинка:-
  повтор_1,
      <процедури, що повторюються>,
      <умова виходу_1>, !,
  повтор_2,
      <процедури, що повторюються>,
      <умова виходу_2>, !,
  ...
  повтор_n,
      <процедури, що повторюються>,
      <умова виходу_n>, !.

  повтор_1.
  повтор_1:- повтор_1.
      <правило для умови виходу_1>.
  повтор_2.
  повтор_2:- повтор_2.
      <правило для умови виходу_2>.
  ...
  повтор_n.
  повтор_n:- повтор_n.
      <правило для умови виходу_n>.
```

ЛАБОРАТОРНА РОБОТА №3

Арифметичні вирази у програмах. Використання рекурсії

Мета: Одержати уміння та навички використання у програмах арифметичних операцій, операцій порівняння, логічних операцій, операцій перевірки типу. Розглянути та відпрацювати технологію використання рекурсії для розв'язування циклічних обчислювальних задач.

Теоретична частина: завдання та контрольні питання

І рівень

1. За допомогою яких операндів конструюються арифметичні вирази? Який стандартний предикат обчислює їх?
2. Які стандартні предикати реалізують операції порівняння?
3. Описати стандартні предикати перевірки типу.
4. Значення яких числових функцій дозволяють обчислити стандартні предикати?
5. Означення рекурсії. Яка конструкція рекурсії?

ІІ рівень

1. Поняття вхідної та вихідної рекурсії.
2. У якому випадку виникає "нескінченна" рекурсія?
3. Який процес називається "процедурою з усуненням хвостової рекурсії"?
4. Який процес називається "процедурою з лівою рекурсією"?

ІІІ рівень

1. Описати рекурсивне виконання правила повтору.
2. У чому полягає різниця між правилом та методом повтору?
3. Проілюструйте у вигляді схеми (послідовного набору рекурсивних фактів від факту, що перевіряється, до граничної умови, і далі, від граничної умови до відповіді) дії компілятора при доведенні запитів, які вимагають виконання рекурсивного правила при роботі класичної програми "Предок" (див. приклад на стор. 37)

Практична частина

І рівень

1-13 варіанти.

а) Скласти програму для визначення N-го числа ряду Фібоначі: $F_1, F_2,..., F_n$, де $F_1=F_2=1$, $F_i=F_{i-1}+F_{i-2}$.

Протестувати програму.

Зберегти програму у файлі "LNN3_1a. pro", де NN – номер варіанту користувача.

б) Знайти факторіал числа N. Протестувати програму.
Зберегти програму у файлі "LNN3_1b. pro".

II рівень

Скласти програму, що дозволяє обчислити значення функції $z=f(a, b)$, де $a, b \in R$. У програмі передбачити можливість обчислення одного з аргументів функції, якщо дано значення функції та другий аргумент. У програмі врахувати область визначення функції.

Програму протестувати та зберегти у файлі (ім'я файлу вказане у варіанті).

Вказівка: використати предикати free, bound.

1-3 варіанти

Функція: z - значення середнього арифметичного для a та b.
Файл: "LNN3_21. pro".

4-6 варіанти

Функція: z - значення середнього гармонічного для a та b.
Файл: "LNN3_24. pro"

7-9 варіанти

Функція: z - значення середнього геометричного для a та b.
Файл: "LNN3_27. pro"

10-13 варіанти

Функція: $f(a,b) = \sqrt{a + a \ln b}$
Файл: "LNN3_20. pro"

III рівень

Дано дійсне число x і натуральне n. Скласти програму, яка, використовуючи вихідну рекурсію, обчислює суму членів ряду (ряд указаний у варіанті).

Програма повинна містити директиву include, за якою підключатиметься текст програми "LNN3_1b. pro" (див. коментар до прикладу на стор. 48).

Протестувати та зберегти текст програми у файлі (ім'я файла програми вказано у варіанті).

1 варіант

$$S(n) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}.$$

Файл "l3_2_1. pro"

2 варіант

$$S(n) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}.$$

Файл "l3_2_2. pro".

3 варіант

$$S(n)=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\ldots+(-1)^n\frac{x^{2n}}{(2n)!}.$$

Файл "l3_3_3. pro".

4 варіант

$$S(n)=1+\frac{x\ln a}{1!}+\frac{(x\ln a)^2}{2!}+\ldots+\frac{(x\ln a)^n}{n!}.$$

Файл "l3_3_4. pro".

5 варіант

$$S(n)=1-\frac{x}{1!}+\frac{x^2}{2!}+\ldots+(-1)^n\frac{x^n}{n!}.$$

Файл "l3_3_5. pro".

6 варіант

$$S(n)=1-\frac{x\ln a}{1!}+\frac{(x\ln a)^2}{2!}+\ldots+(-1)^n\frac{(x\ln a)^n}{n!}.$$

Файл "l3_3_6. pro".

Вимоги до захисту лабораторної роботи

Звіт виконаної роботи повинен містити назву теми, текст програм, текст запитів мовою Пролог та відповіді для завдань І-ІІІ-го рівнів. Для рекурсії, використаної у програмах, необхідно вміти описувати природною мовою граничну умову та рекурсивне правило.

§5. РОБОТА ЗІ СТРУКТУРАМИ ДАНИХ

5.1. Типи даних користувача

Нові типи області значень можна означити через стандартні типи і використати їх в оголошеннях предикатів. У прикладі №1, наприклад, міститься вказівка на те, що учень стає новим типом області значень, який співпадає з областю symbol, а тип клас - з областю цілих чисел.

Структури теж можна означити як нові типи області значень.

Приклад 11. Програма містить деяку базу даних осіб-студентів з відомостями: ім'я студента, номер академічної групи, дата народження.

```
domains                                                    /*pr_11. pro*/
  імя = symbol
  група = integer
  структура_студент = студент (імя, група)
  структура_дата = дата (integer, integer, integer)
predicates
  особа ( структура_студент, структура_дата)
clauses
  особа ( студент (петро, 43),  дата (1, 11, 76) ).
  особа (студент (ольга,12),  дата (15, 6, 75) ).
```

Коментар. Предикат особа першим аргументом має структуру студент/2, а другим - структуру дата/3.

Якщо одним із аргументів предикату *є список*, то область значень цього аргументу слід оголосити у секції domains як об'єкт із символом "зірочка", де об'єкт - це область значень кожного елемента списку. Нагадаємо, що на Турбо-Пролозі елементами списку можуть бути числа, атоми, структури та списки, але певний список може мати аргументи лише одного типу. Можна оголосити список, аргументами якого є структури.

Приклад 12. Опишемо многочлени як списки одночленів.

```
domains                                                    /*pr_12. pro*/
  коеф_однчл = integer
  степ_однчл = integer
  структура_одночлен = x (коеф_однчл, степ_однчл)
  многочл = структура_одночлен*
predicates
  сума_многочл (многочл, многочл, многочл)
clauses
  сума_многочл ( [ ], Q, Q ).
  сума_многочл ( P, [ ], P ).
```

Коментар. Розділ domains містить описання типів області значень: многочл - це список, елементами якого є структури з функтором x та арністю 2; перший аргумент структури структура_одночлен/2 є цілим числом - коефіцієнтом одночлена, другий аргумент є цілим числом - показником степеня одночлена. Якщо дано многочлен $Q(x) = x^3 - 7x^2 + 5$, то його можна подати як список: [x (1, 3), x (-7, 2), x (0, 1), x (5, 0)]. У наступному розділі програми описано предикат сума_многочл/3, перші два аргументи якого - це многочлени, що додаються, а третій аргумент - многочлен, що є результатом додавання перших двох. Програма містить два факти, за якими сумою двох многочленів у випадку, коли один з них є нульовим (порожній список), є многочлен, що рівний ненульовому многочлену.

Дописавши у даний фрагмент необхідні факти та правила, одержимо програму, яка може виконувати процедуру додавання двох довільних многочленів, забезпечуючи при цьому виведення у символьній формі третього многочлена - результата додавання двох даних.

5.2. Списки

При програмуванні на Пролозі, серед іншого, найчастіше використовується подання даних у вигляді списків.

Список є бінарна структура: перший аргумент списку - голова списку, другий аргумент списку - хвіст. Зважаючи на рекурсію, список формально можна означити так: “Списком є або порожній список, або пара елементів, з яких другий (хвіст списку) є списком”.

Списки на Пролозі часто використовуються для подання знань (даних). Варто зауважити, що характерною ознакою мов програмування штучного інтелекту є ефективна робота зі списками: від того, як мова програмування працює зі списками, залежить придатність такої мови до розв’язування задач зі штучного інтелекту.

Позначається список так: [H | T], де H - голова списку, T- хвіст списку.

Приклади списків:

- [ ] - порожній список;
- [a] - має точно один елемент a;
- [1,2,3] - має три елементи;
- [[1,2], [3], [4,5,6]] - має три елементи - списки цілих чисел.

Процедура встановлення членства у списку

Природною мовою дану процедуру можна описати так: “деякий елемент міститься у списку елементів, якщо він є або головою списку, або міститься у хвості списку”.

Приклад 13. Опишемо процедуру встановлення членства у списку цілих чисел.

```
domains                                                    /*pr_13. pro*/
    list = integer*
predicates
    member_int (integer, list)
clauses
    member_int ( R, [ R | T ] ).                           /*(1)*/
    member_int ( R, [ H | T ] ) :-                         /*(2)*/
      member_int ( R, T ).
```

Коментар. Гранична умова (1): число належить списку цілих чисел, якщо воно є головою списку. Рекурсивне правило (2): число, яке не являється головою списку, належить йому, якщо воно належить хвосту списку.

Процедура об'єднання двох списків

Цю процедуру можна описати так: "результатом об'єднання двох списків буде список, голова якого - голова першого списку, а хвіст - список, що є результатом об'єднання хвоста першого списку і всього другого списку".

Приклад 14. Об'єднати списки, елементами яких є структури типу вчиться/2.

```
domains                                                    /*pr_14. pro*/
   учень = symbol
   клас = integer
   структура = вчиться (учень, клас)
   сп = структура*
predicates
   вчиться (учень, клас)
   append (сп, сп, сп)
clauses
   append ( [ ], Q, Q).
   append (P, Q, R):-
      P = [ HP | TP ],
      append ( TP, Q, TR ),
      R = [ HP | TR ].
```

Коментар. Гранична умова: об'єднання порожнього списку зі списком Q є список Q. Рекурсивне правило: у першому списку P визначаємо голову HP та хвіст TP (P = [HP|TP]); у список TR рекурсивно об'єднуємо хвіст TP першого списку з другим списком Q (такі операції виконуємо доти, доки перший список не стане порожнім); результатом об'єднання буде список R, з головою HP і хвостом TR.

Процедура визначення кількості елементів списку

Природною мовою дану процедуру можна описати так: “для того, щоб визначити кількість елементів (довжину) списку, необхідно визначити довжину його хвоста і до неї додати одиницю”.

Приклад 15. Визначити довжину списку, елементи якого – дійсні числа.

```
domains                                                    /*pr_15. pro*/
   list = real*
predicates
   довжина ( list, integer )
clauses
    довжина ( [ ], 0 ).
    довжина ( [ H |T ], N) :-
       довжина (T, N1),
       N = N1 + 1.
```

Коментар. Гранична умова: довжина порожнього списку рівна нулю. Правило: визначається довжина N1 хвоста T і довжина списку: N=N1+1.

Процедура індексування елементів списку

Процедура розуміється як дві окремі процедури: визначення номера деякого елемента списку та, навпаки, одержання елементу списку за його номером.

Природною мовою перша процедура описується так: "Визначити порядковий номер елемента у списку можна як номер, менший на одиницю від попередньо визначеного порядкового номера у хвості списку". Аналогічно можна описати другу процедуру.

Приклад 16. Процедура індексування списку, елементи якого - символи.

```
domains                                                    /*pr_16. pro*/
   element = symbol
   list = symbol*
   num = integer
predicates
   indexlist (list, num, element)
clauses
   indexlist ( [H | T],1, H).                                  /*(1)*/
   indexlist ( [H | T], N, Y):-                                /*(2)*/
      M=N-1,
      indexlist (T, M, Y).
```

Коментар. Гранична умова: елемент, що співпадає з головою списку має індекс 1 – фраза (1), фраза (2) описує рекурсивне правило, за яким для того, щоб визначити елемент списку, який має індекс N і не

співпадає з головою списку, необхідно визначити його індекс у хвості списку.

Для роботи зі списками можна реалізувати й інші процедури, наприклад, процедуру вилучення елементу зі списку, друкування елементів списку тощо.

Предикат findall

Стандартний предикат findall/3, проводить оцінку всіх можливих відповідей на конкретне питання та об'єднує їх у список:

findall (Змінна, Структура, Список змінних).

Предикат використовує все, пов'язане зі "Структурою" як ціль. "Змінна" повинна бути одним з аргументів "Структури". Предикат шукає всі можливі варіанти погодження "Змінної", що можуть виникнути при досягненні цілі різними шляхами, і дані про кожне погодження заносяться у список, визначений "Списком змінних".

Приклад 17. Є відомості про учнів 9, 10 класів: ім'я, клас, середня оцінка. Визначити середній бал для учнів, що навчаються у певному класі.

```
include "pr_15. pro"                                    /*pr_17. pro*/
domains
  число = real
  клас = integer
predicates
  учень ( symbol, клас, число )
  сер_оц ( клас, число )
  сума_ел ( list, число )
goal
  сер_оц ( 9, L), сер_оц ( 10, K), write ("L=", L, " K=", K).
clauses
  учень (хома,  10, 5 ).
  учень (леся,  10, 4.5 ).
  учень (петро, 10, 4.3 ).
  учень (олег, 9, 4.5 ).
  учень (ольга, 9, 3 ).
  сер_оц (X, Y) :-
      findall ( O, учень ( _ , X, O), L ),
      довжина ( L, N ), сума_ел (L, S), Y = S / N.
  сума_ел ( [ ] ,0 ).
  сума_ел ( [ H | T ], S ) :-
      сума_ел (T,S1), S=S1+H.
```

Коментар. Директива include "ім'я файла" додатково підключає весь текст програми ("pr_15. pro"). Тому можна використовувати предикат довжина/2, не описуючи його та не записуючи для нього відповідні факти і правила. Програма у правилі сер_оц використовує предикат сума_ел/2, за яким обчислюється сума елементів списку: "щоб ви-

значити суму елементів списку, необхідно визначити суму елементів хвоста списку і додати значення голови списку". Предикат сер_оц/2 першим аргументом має клас, а другим - середню оцінку для всього класу. За правилом сер_оц формується список L оцінок класу X, визначається їх кількість N, сума S та середня оцінка Y.

Програма містить директиву goal. Тому під час запуску компілятора на запит, що міститься у директиві, автоматично сформується відповідь. Новий запит можна ввести, записавши його до даної директиви (секції). Діалогове вікно для введення нових запитів можна використовувати лише вилучивши секцію goal з тексту програми.

Можливі запити до наведених програм роботи зі списками.

Приклад	Запити	Результат
Приклад 13.	1) member_int (5, [5,2,1,7,333]) 2) X=2, member_int (X, [5,2,1,7,333]) 3) member_int (6, [1,2,5,7,66])	Yes X=2 1 Solution No
Приклад 14.	append ([вчиться(а,8)], [вчиться(к,9)], L)	L=[вчиться(“а”,8), вчиться(“к”,9)] 1 Solution
Приклад 15.	1) довжина ([1,2,3,4,5,7], L) 2) довжина ([1,2,3,4,5,7], 6) 3) довжина ([1.2, 3.4, 5.7], L)	L=6 1 Solution Yes L=3 1 Solution
Приклад 16.	Indexlist ([a,b,c,d], 3, L)	L="c" 1 Solution
	Indexlist ([a,b,[c,d]], 3, L)	Помилка типу[1].
Приклад 17.	реалізується директивою goal	L=3.75 K=4.6

[1] Список не може у даному випадку включати як елемент інший список..

ЛАБОРАТОРНА РОБОТА №4

Структури даних на Турбо-Пролозі. Використання списків. Введення та виведення даних

Мета: Закріпити поняття про списки. Одержати уміння та навички щодо використання процедур для роботи зі списками; використання процедур для роботи зі списками під час розв'язування задач.

Теоретична частина: завдання та контрольні питання

І рівень

1. Які стандартні предикати використовуються у програмах для введення даних? Синтаксис таких предикатів.
2. Які стандартні предикати використовуються у програмах для організації виведення даних? Синтаксис таких предикатів.
3. Означення списку як бінарної структури.
4. Як рекурсивно означається список? Приклади списків.
5. Які процедури найчастіше використовуються для роботи зі списками?
6. Описати процедуру встановлення членства у списку.

ІІ рівень

1. Чому стандартні предикати вводу-виводу називають позалогічними? У чому полягає суть поняття "побічний ефект"?
2. Описати процедуру об'єднання двох списків.
3. Описати процедуру визначення довжини списку.
4. Пояснити роботу предикату findall. Навести приклад.

ІІІ рівень

1. Описати процедуру індексування списку.
2. Означення графа. Опишіть способи подання графів на Турбо-Пролозі. Які типові операції виконують над графами?
3. Дайте рекурсивне означення бінарного дерева.
4. У чому переваги та недоліки різних способів подання структур даних на Пролозі?

Практична частина

І рівень

1-13 варіанти.

Скласти програму, що дозволяє у списку цілих чисел:

- відшукати мінімальний елемент;
- знайти максимальний елемент;
- знайти кількість нульових елементів;

- порахувати кількість додатних елементів;
- впорядкувати елементи списку у порядку зростання.

Записати програму у файл "LNN4_1. pro", де NN – номер варіанту користувача. Сформулювати запити та протестувати програму.

II рівень

Є інформація про ділянки доріг певного регіону, що сполучають міста між собою. Ця інформація подана у вигляді таблиці:

№ ділянки	Міста, які між собою сполучає ділянка		Відстань між містами
	1-ше місто	2-ге місто	
1	Бердичів	Чуднів	40
2	Бердичів	Житомир	45
3	Житомир	Чуднів	55
4	Житомир	Червоноармійськ	40
5	Житомир	Черняхів	26
6	Житомир	Коростишів	26
7	Коростишів	Черняхів	26
8	Коростишів	Кочерів	20
9	Кочерів	Радомишль	20
10	Червоноармійськ	Новоград	42
11	Червоноармійськ	Черняхів	30
12	Черняхів	Коростень	91
13	Черняхів	Потієвка	29
14	Радомишль	Потієвка	24
15	Радомишль	Малин	32
16	Потієвка	Малин	38

Скласти програму, що містить факти "ділянка/4" на основі наведених даних. Використовуючи стандартний предикат findall/3, та підключивши директивою include текст програми I-го рівня, забезпечити виконання процедури (вказана у варіанті).

Програму зберегти у файлі (ім'я файлу вказане у варіанті).

Протестувати програму.

1-3 варіанти. Процедура: відшукати номер найкоротшої ділянки.
Файл: "LNN4_21. pro".

4-6 варіанти. Процедура: відшукати міста, які зв'язані ділянкою найбільшої довжини.
Файл: "LNN4_24. pro".

7-9 варіанти. Процедура: відшукати номера ділянок, які мають довжину, меншу за середнє арифметичне довжини всіх ділянок.

Файл: "LNN4_27. pro".

10-11 варіанти. Процедура: для кожної з тих ділянок, довжини яких перевищують середнє арифметичне довжини всіх ділянок, відшукати назву другого міста з пари міст.

Файл: "LNN4_20. pro".

12-13 варіанти. Процедура: для кожної з тих ділянок, довжини яких рівні середньому арифметичному довжини всіх ділянок, відшукати назву першого міста з пари міст.

Файл: "LNN4_22. pro".

III рівень

Модифікувати програму II-го рівня (вказана у варіанті) так, щоб програма проводила діалог за яким:

- дані вводились користувачем з клавіатури і під час введення на екрані з'являлися допоміжні повідомлення-підказки (вказані у варіанті), на які, у разі потреби, користувач має відповісти „так";

- результати роботи програми мають виводитися у вікні з назвою „Результат пошуку (ім'я процедури згідно варіанту)".

Програму зберегти у файлі, ім'я якого вказане у варіанті завдань.

1 варіант

Програма: II рівень, варіант 1-3.

Повідомлення: "Процедура здійснює пошук номера найкоротшої ділянки дороги між містами. Здійснити пошук?"

Ім'я процедури: „номера найкоротшої ділянки".

Файл: "L43_1. pro".

2 варіант

Програма: II рівень, варіант 4-6.

Повідомлення: "Процедура здійснює пошук назв міст, які сполучені ділянкою дороги найбільшої довжини. Здійснити пошук?"

Ім'я процедури: „міст, що сполучені ділянкою найбільшої довжини".

Файл: "L43_2. pro".

3 варіант

Програма: II рівень, варіант 7-9.

Повідомлення: "Процедура здійснює пошук номерів ділянок, які мають довжину, меншу за середнє арифметичне довжин всіх ділянок. Здійснити пошук?"

Ім'я процедури: „ділянок, які мають довжину, меншу за середнє арифметичне довжин всіх ділянок".

Файл: "L43_3. pro".

4 варіант

Програма: II рівень, варіант 10-11.

Повідомлення: "Процедура здійснює пошук міста, з пари міст, ділянки між якими мають довжину, більшу за середнє арифметичне довжин всіх ділянок. Здійснити пошук?"

Ім'я процедури: „пошук міста, з пари міст, ділянки між якими мають довжину, більшу за середнє арифметичне довжин всіх ділянок".

Файл: "L43_4. pro".

5 варіант

Програма: II рівень, варіант 12-13.

Повідомлення: "Процедура здійснює пошук міста, з пари міст, ділянки між якими мають довжину, рівну середньому арифметичному довжин всіх ділянок. Здійснити пошук?"

Ім'я процедури: „пошук міста, з пари міст, ділянки між якими мають довжину, рівну середньому арифметичному довжин всіх ділянок".

Файл: "L43_5. pro".

6 варіант

Програма: II рівень, варіант 1-3.

Повідомлення: "Процедура здійснює пошук номера найкоротшої ділянки дороги між містами та назви міст, що утворюють таку ділянку. Здійснити пошук?"

Ім'я процедури: „номера найкоротшої ділянки між містами та імен міст ділянки".

Файл: "L43_6. pro".

Вимоги до захисту лабораторної роботи

Звіт виконаної роботи повинен містити назву теми; текст програми, тестові запити мовою Пролог та відповіді компілятора для завдань I-го рівня; текст правил, що описують процедуру II-го рівня; текст правил, що описують процедуру III-го рівня. Для правил, що використовують процедури роботи зі списками, вміти описувати правила та вказані процедури природною мовою.

§6. ВНУТРІШНЯ БАЗА ДАНИХ

6.1. Робота з внутрішньою базою даних

Під час виконання програма на Пролозі може модифікувати сама себе, виключаючи з процесу доведення логічних цілей певні фрази та дописуючи нові.

Для Турбо-Прологу версії 2.0 вказані операції, на жаль, допустимі лише для фактів програми. Такі факти, що подаються предикатами, оголошеними у секції database, складають *внутрішню базу даних (ВБД)* програми на Турбо-Пролозі. Зауважимо, що предикати внутрішньої бази даних неможливо одночасно оголосити у секції predicates, і ці предикати не можна використовувати як заголовки правил програми.

Існує ряд стандартних предикатів для роботи з ВБД (їх повний перелік містить допомога (Help) оболонки Турбо-Пролог):

consult (Ім'яФайла) – доповнити поточну ВБД іншою базою даних, що міститься у файлі з іменем Ім'яФайла;
Прототип: (i).

save (Ім'яФайла) – зберегти поточну ВБД у файлі з іменем Ім'яФайла;
Прототип: (i).

assert (Факт) – доповнити поточну ВБД новим фактом Факт;
Прототип: (i).

retract (Факт) – вилучити факт Факт з поточної бази даних;
Прототип: (i).

Приклад 18. Відомо те, що у групі учнів є власники речей: Хома має приймач, Олег - ЕОМ, Петро - пенал і гумку, Ольга - кульку. Леся та Юрій не мають у власності ніяких речей. Вважається, що учень може користуватися річчю, якщо він її має у власності, або якщо її має у власності старші брат або сестра. Хома є старшим братом Лесі, а Ольга - старшою сестрою Юрія. Учням можуть купувати нові речі і ті учні, які мають певні речі у власності, можуть обмінюватися ними.

```
domains                                                    /*pr_18. pro*/
    учень, p = symbol
database
    має ( учень, p )
predicates
    користується ( учень, p )
    старш ( учень, учень )
    обмін ( учень, p, учень, p )
    купити ( учень, p )
```

```
clauses
   має ( хома, приймач ).        має ( олег, еом ).
   має ( петро, пенал ).         має ( петро, гумка ).
   має ( ольга, кулька ).
   старш ( хома, леся ).         старш ( ольга, юра ).
   користується (D, R) :-
        має (D, R).
   користується (D, R) :-
        старш (SD, D), має (SD, R).
   купити (D, R) :-
        bound (D), bound (R),
        not ( має (D, R) ), assert (має (D, R) ).
    обмін (D1, R1, D2, R2) :-
        bound (D1), bound (R1), bound (D2), bound (R2),
        має (D1, R1), має (D2, R2),
          retract ( має (D1, R1) ), retract (має (D2, R2) ),
          assert ( має (D1, R2) ), assert ( має (D2, R1) ).
```

Коментар. Предикат має/2 оголошений як предикат ВБД. Предикат користується/2 описує правило, за яким визначається право користування річчю. Предикат купити/2 описує правило, за яким учню купують річ, якщо її у нього немає (використовується предикат not). Стандартний предикат assert доповнює ВБД відповідним фактом. Предикат обмін/4 описує правило, за яким учні, що мають у власності деякі речі, можуть обмінятися ними. При виконанні правила обміну стандартний предикат retract вилучає з бази даних відповідні факти, а assert вносить до неї факти про нових власників речей. Предикат bound використаний у деяких правилах для того, щоб аргументами були константи (імена учнів, назви речей).

Виконаємо серію послідовних запитів до програми.

Запит	Пояснення	Відповідь
користується (Д,Р)	Хто і яку річ має у користуванні?	Д=хома, Р=приймач Д=олег, Р=еом Д=петро, Р=пенал Д=петро, Р=гумка Д=ольга, Р=кулька Д=леся, Р=приймач Д=юра, Р=кулька
обмін (хома, приймач, олег, еом)	Провести обмін між Хомою та Олегом - приймач на ЕОМ.	Yes
купити (хома, еом)	Купити для Хоми ЕОМ.	No
купити (хома, ручка)	Купити для Хоми ручку.	Yes
користується (леся, Р)	Якими речами користується Леся?	Р=еом Р=ручка
користується (олег, еом)	Чи користується Олег ЕОМ?	No
має (олег, Р)	Що має Олег у власності?	Р=приймач
save (“newbase.txt”)	Записати ВБД у файл DOS.	Yes

Зауваження. Зміни у ВБД, виконані предикатами retract та assert, можуть бути втрачені при виході з режиму діалогу і подальшому виконанні деяких пунктів меню компілятора (Edit, Compile, підпункт Load пункту File тощо). Вихід з діалогового вікна (після зміни програмою бази даних) у головне меню компілятора та наступне повернення до діалогового вікна через пункт меню Run не приводить до втрат ВБД.

Ураховуючи зауваження необхідно виконувати наведені запити послідовно і неперервно. За останнім запитом, який виконується успішно, створюється текстовий файл newbase.txt, що містить факти модифікованої бази даних, тобто стандартний предикат save/1 записує у файл всі факти поточної програми, що відносяться до предикатів бази даних (факти, описані предикатом старш (учень, учень) у файл не записуються).

6.2. Подання баз даних

Різні види подання баз даних проілюструємо на прикладі задачі, за якою необхідно подати деякі відомості про студентів: ім'я студента, курс, факультет, група, де навчається студент.

Приклад 19. Подання бази даних у вигляді множини фактів, кожен з яких відповідає цілісному інформаційному елементу (запису) бази даних.

```
domains                                                    /*pr_19. pro*/
predicates
  студент (symbol, integer, symbol, integer)
clauses
  студент (ярема, 1, фізмат, 13).
  студент (олег, 5, філфак, 52).
  студент (хома, 1, філфак, 15).
```

Приклад 20. Подання бази даних атрибутами у вигляді фактів, тобто фактами записуються окремі атрибути (властивості) інформаційних елементів. У разі потреби атрибути можна зібрати в одне ціле за допомогою правила, при цьому один із атрибутів виступає у ролі ключа.

```
domains                                                    /*pr_20. pro*/
  курс, група = integer
  имя, факультет = symbol
predicates
  курс (имя, курс)
  факультет (имя, факультет)
  група (имя, група)
  студент (имя, курс, факультет, група)
clauses
  курс (ярема,1).   факультет (ярема, фізмат).   група (ярема,13).
```

```
курс (олег,5).   факультет (олег, філфак).   група (олег, 52).
курс (хома,1).   факультет (хома, філфак).   група (хома,15).
студент (Имя, Курс, Факультет, Група) :-                    /*(1)*/
   курс (Имя, Курс),
   факультет (Имя, Факультет),
   група (Имя, Група).
```

Коментар. У ролі ключа програма використовує атрибут имя. Кожен запис бази даних подано у вигляді набору із трьох фактів-відношень між певним іменем студента та тим, на якому курсі, факультеті, у якій групі він навчається. Правило (1), яке реалізує предикат студент/4, об'єднує всі атрибути деякого студента у запис бази даних. Хоча програма значно більша за обсягом, ніж попередня, але такий спосіб подання баз даних більш гнучкий: нові атрибути можна добавляти до окремих елементів, не змінюючи всю базу даних.

Приклад 21. Подання бази даних у вигляді списку структур.

```
domains                                                    /*pr_21. pro*/
  имя, факультет = symbol
  курс, група = integer
  структура_студент = ст (имя, курс, факультет, група)
  список_ст = структура_студент*
predicates
  студенти (список_ст)
  запис (структура_студент)
  member (список_ст, структура_студент)
clauses
  студенти ( [ ст(ярема,1,фізмат,13),
               ст(олег,5,філфак,52),
               ст(хома,1,філфак,15) ] ).
  запис(L) :-
      студенти(C), member (C, L).
  member ( [H | T], H).
  member ( [H | T], Y) :-
      member (T, Y).
```

Коментар. Аргумент список_ст як список складається зі структур типу ст/4. Предикат студенти/1 дозволяє записати відповідні факти у вигляді списку структур. Правило запис/1 успішно виконується, якщо його аргумент L міститься як елемент (процедура memeber/1) даного списку студентів; при цьому список студентів одержується і передається змінній C після успішного погодження предикату студенти/1. Виконуючи запити з підціллю запис/1 можна отримати різноманітну інформацію на основі записаної бази даних. Перевагою даного підходу є можливість використання процедур для роботи зі списками.

Крім того, бази даних на Пролозі можуть подаватися у вигляді *рекурсивної структури, бінарного дерева* тощо.

ЛАБОРАТОРНА РОБОТА №5

Робота з базами даних на Турбо-Пролозі

Мета: Отримати навички роботи зі стандартними предикатами, що забезпечують операції з ВБД. Оволодіти методикою внесення змін до ВБД з використанням побічних ефектів.

Теоретична частина: завдання та контрольні питання

І рівень

1. Яка база даних Турбо-Прологу називається внутрішньою?
2. Які стандартні предикати використовуються для доповнення внутрішньої бази даних новими фразами під час виконання програми?
3. Які стандартні предикати використовуються для вилучення фраз із бази даних під час виконання програми?
4. Який предикат зчитує додаткові нові факти у внутрішню базу даних зі стороннього текстового файла?

ІІ рівень

1. З якими фразами неможливо виконати операції доповнення та вилучення фраз у внутрішній базі даних?
2. Які факти поточної програми можуть бути збережені у вигляді текстового файла DOS?
3. Що відбувається з текстом програми під час її роботи, якщо у програмі використані предикати для роботи з ВБД?

ІІІ рівень

1. Яка база даних Турбо-Прологу називається зовнішньою?
2. У яких місцях (place) може розміщуватися зовнішня база даних?
3. Які предикати використовуються для створення зовнішньої бази даних, її відкриття та закриття, знищення БД?
4. Які стандартні предикати використовуються для доповнення зовнішньої бази даних новими записами під час виконання програми?
5. Які стандартні предикати використовуються для вилучення записів з бази даних під час виконання програми?
6. Який предикат дозволяє відшукати запис у ланцюжку і визначити посилання запису?
7. З якими фразами Пролог-програми неможливо виконати операції доповнення та вилучення фраз у зовнішній базі даних?
8. Які предикати і в якій послідовності використовуються для того, щоб у існуючій зовнішній базі даних вилучити певний запис-факт та замінити його на новий такого ж типу, що і вилучений?

Практична частина

І рівень

1-13 варіанти.

Модифікувати у програмі "LNN2_1. pro" правило "знае" так, щоб при формулюванні запитів програма запитувала у користувача відомості про студентів (факультет, курс, номер гуртожитку) - учасників туристичної подорожі, якщо інформація про певну особу не внесена у базу даних програми. Крім того, для таких випадків програма повинна заносити у ВБД відповідні факти "вчиться/3" та "проживае/2". При модифікації програми використати предикати free, bound, !, щоб не було повернення на повторне погодження предикату "знае" тоді, коли аргументом – іменем особи – є константа (певне ім'я особи може міститися у ВБД тільки один раз); якщо ж аргументом є змінна, то повторне погодження предикату "знае" дозволяється.

Зберегти модифіковану ВБД у файлі "vbdNN_51.txt", де NN – номер варіанту користувача.

Записати програму у файл "LNN5_1. pro".

ІІ рівень

Скласти програму, яка у секції clauses не містить фактів, а лише правила, що дозволяють виконати певні процедури (вказані у варіанті). Записати програму у файл (указаний у варіанті). Використовуючи програму, виконати операції (вказані у варіанті). Зберегти створену внутрішню базу даних у файлі "vbdNN_52.txt".

1-3 варіанти. Процедури: запис у базу даних відомостей про студента (ім'я, факультет і курс навчання) із забезпеченням перевірки наявності імені студента у базі даних та дозволу на перезапис даних у разі потреби (з вилученням попередніх даних); доповнення ВБД інформацією про результати сесії з 2-х екзаменів для кожного студента.

Файл: "LNN5_21. pro".

Операції: створити ВБД з відомостями про 4-х студентів; доповнити її інформацією так, щоб дві особи мали незадовільні оцінки.

4-6 варіанти. Процедури: запис у базу даних відомостей про студента та викладача (ім'я особи та номер гуртожитку, у якому вона проживає) із забезпеченням перевірки наявності імені особи у базі даних та дозволу на перезапис даних у разі потреби (з вилученням попередніх даних); доповнення ВБД інформацією про суму сплати за проживання у гуртожитку для кожної особи.

Файл: "LNN5_24. pro".

Операції: створити внутрішню базу даних з відомостями про 2-х студентів та 2-х викладачів; доповнити ВБД інформацією про сплату за проживання у гуртожитку так, щоб дві особи мали відмітки про несплату.

7-9 варіанти. Процедури: запис у базу даних відомостей про сімейний стан студента (ім'я студента, одружений чи ні) із забезпеченням перевірки наявності імені студента у базі даних та дозволу на перезапис даних у разі потреби (з вилученням попередніх даних); доповнення ВБД інформацією про кількість дітей у кожного з одружених студентів.

Файл: "LNN5_27. pro".

Операції: створити внутрішню базу даних з відомостями про 4-х студентів; доповнити ВБД інформацією про кількість дітей в одружених студентів так, щоб дві особи дітей не мали.

10-11 варіанти. Процедури: запис у базу даних відомостей про факультети вузу (назва факультету, кількість студентів) із забезпеченням перевірки наявності назви факультету в базі даних та дозволу на перезапис даних у разі потреби (з вилученням попередніх даних); доповнення ВБД інформацією про те, які гуртожитки закріплені за факультетами.

Файл: "LNN5_20. pro".

Операції: створити ВБД з відомостями про 4 факультети; доповнити ВБД інформацією про те, який гуртожиток (вказати номер гуртожитку) закріплено за певним факультетом так, щоб деякі гуртожитки були закріплені не менш як за двома факультетами.

12-13 варіанти. Процедури: запис у ВБД відомостей про тролейбусні маршрути міста (номер маршруту, початкова і кінцева зупинки) із забезпеченням перевірки наявності номера маршруту у базі даних та дозволу на перезапис даних у разі потреби (з вилученням попередніх даних); доповнення ВБД інформацією про кількість проміжних зупинок на маршруті (виключаючи початкову та кінцеву).

Файл: "LNN5_22pro".

Операції: створити внутрішню базу даних з відомостями про 4 маршрути; доповнити ВБД інформацією про кількість зупинок на кожному маршруті так, щоб два маршрути мали не більше 6 проміжних зупинок.

III рівень

а) Скласти програму, яка дозволяє перетворити внутрішню базу даних програми II-го рівня (вказана у варіанті) у зовнішню. Розмістити зовнішню базу даних (ЗБД) у файлі (вказаний у варіанті). Програму зберегти у файлі (ім'я файлу вказане у варіанті).

б) Скласти програму, яка при роботі з зовнішньою базою даних, збереженою у файлі за п. а), виконує процедуру (вказана у варіанті), що дозволяє модифікувати зовнішню базу даних, дописуючи факт (вказаний у варіанті). Записати програму у файлі (ім'я файлу вказане у варіанті). Виконати запита до програми. Модифіковану зовнішню базу даних записати у файл (вказаний у варіанті).

1 варіант

а) Програма: II рівень, варіант 1-3.
Файл ЗБД: „db53_1.dbp”
Файл: „L53a_1. pro”

б) Процедура: призначити стипендії студентам у залежності від середнього балу за сесію і вилучити з бази даних тих студентів, що мають незадовільні екзаменаційні оцінки хоча б з одного предмету. Вивести список студентів, які продовжуватимуть навчання.
Факт: Конкретному студенту призначена стипендія у певному розмірі.
Файл: "L53b_1. pro".
Файл модифікованої ЗБД: "db53_1m.dbp".

2 варіант

а) Програма: II рівень, варіант 4-6.
Файл ЗБД: „db53_2.dbp”
Файл: „L53a_2. pro”

б) Процедура: Підрахувати суму коштів, сплачених особами певного гуртожитку і вважати відселеними з гуртожитку (вилучити з бази даних) тих осіб, що не сплатили за проживання. Вивести список осіб, що продовжуватимуть проживати у гуртожитках.
Факт: За проживання у конкретному гуртожитку (вказати номер) сплачена певна сума.
Файл: "L53b_2. pro".
Файл модифікованої ЗБД: "db53_2m.dbp".

3 варіант

а) Програма: II рівень, варіант 7-9.
Файл ЗБД: „db53_3.dbp”
Файл: „L53a_3. pro”

б) Процедура: Призначити допомогу студентам, які одружені і мають дітей: на 1 дитину - у розмірі 1-єї умовної одиниці, на 2 дитини і більше - у розмірі двох умовних одиниць. Для студентів, які одружені, але не мають дітей, вказати на те, що допомога не призначена (призначити допомогу у розмірі 0 ум. одиниць). Вивести список осіб, яким призначена допомога.
Факт: Конкретному студенту призначена допомога у певному розмірі.
Файл: "L53b_3. pro".
Файл модифікованої ЗБД: "db53_3m.dbp".

4 варіант

а) Програма: II рівень, варіант 10-11.
Файл ЗБД: „db53_4.dbp”
Файл: „L53a_4. pro”

б) Процедура: Підрахувати загальну кількість студентів, що навчаються у вузі. Вивести список факультетів, вказавши, які гуртожитки закріплені за певним факультетом.
Факт: Студенти конкретного факультету (вказати назву) проживають у певному гуртожитку (вказати номер).
Файл: "L53b_4. pro".
Файл модифікованої ЗБД: "db53_4m.dbp".

5 варіант

а) Програма: II рівень, варіант 12-13.
Файл ЗБД: „db53_5.dbp”
Файл: „L53a_5. pro”

б) Процедура: Визначити для маршрутів кількість пасажирів, що перевозяться, якщо відомо, що на кожній зупинці у тролейбус сідає 10 пасажирів. Вилучити маршрут, якщо за цим маршрутом від початкової до кінцевої зупинки користується не більше 75 пасажирів. Вивести список маршрутів, що продовжуватимуть функціонувати.
Факт: На конкретному тролейбусному маршруті перевозиться певна кількість пасажирів.
Файл: "L53b_5. pro".
Файл модифікованої ЗБД: "db53_5m.dbp".

Вимоги до захисту лабораторної роботи

Звіт виконаної роботи повинен містити назву теми; текст модифікованого правила для програми I-го рівня; текст програми та текст файла ВБД для II-го рівня; текст процедури та текст модифікованої зовнішньої бази даних для III-го рівня.

§7. РОБОТА З ТЕКСТОМ

7.1. Рядкові величини

Для роботи з рядками символів є такі стандартні предикати:

frontchar (Рядок, ПершийСимвол, Остача) - розбиває рядок на дві частини: перший символ і залишок рядка.

Рядок, Остача - тип string.

ПершийСимвол - тип char.

Прототип: (i, o, o), (i, i, o), (i, o, i), (i, i, i), (o, i, i).

Приклади:

Запит	Відповідь
frontchar("мир",X,Y)	X='м' Y="ир"
frontchar("мир",'м',Y)	Y='ир'
frontchar("мир",'м','ир')	Yes
frontchar(X,'м','ир')	X="мир"

fronttoken (Рядок, Лексема, Остача) - розбиває рядок на лексему[1] та остачу.

Рядок, Лексема, Остача - тип string.

Прототип: (i, o, o), (i, i, o), (i, o, i), (i, i, i), (o, i, i).

Приклади:

Запит	Відповідь
fronttoken ("Цей студент вчиться", X, Y)	X= "Цей" Y=" студент вчиться"
fronttoken ("Цей студент вчиться", "Цей", Y)	Y= " студент вчиться"
fronttoken ("Цей студент вчиться", X, " студент вчиться")	X= "Цей"
fronttoken (X, "Цей", " студент вчиться")	X= "Цей студент вчиться"

frontstr (ЧислоСимволів, ВхРядок, ВихРядок, Остача) - розбиває рядок на дві частини.

ЧислоСимволів - тип integer.

ВхРядок, ВихРядок, Остача - тип string.

Прототип: (i, i, o, o).

[1] Лексема - це послідовність символів, що мають смисл. Вона означається як:
- ім'я відповідно до синтаксису Турбо-Прологу або
- рядкове подання числа, або
- окремий символ (може бути порожнім).

Приклад:

Запит	Відповідь
frontstr (6, "фізико-математичний", X, Y)	X = "фізико" Y = "-математичний"

concat (Рядок1, Рядок2, РядокРез) - склеюється Рядок1 з Рядок2 і результат передається у РядокРез.

Рядок1,Рядок2,РядокРез - тип string.

Прототип: (i, i, o), (i, o, i), (o, i, i), (i, i, i).

Запит	Відповідь
1. concat("фіз","мат",X)	X="фізмат"
2. concat(X,"мат","фізмат")	X="фіз"
3. concat("фіз","мат","фізмат")	Yes

str_len (Рядок, Довжина) - визначає довжину рядка символів.

Рядок - тип string.

Довжина - тип integer.

Прототип: (i, i), (i, o), (o, i).

isname (Рядок) - перевіряє, чи ім'я, вказане в аргументі Рядок допустиме у Турбо-Пролозі.

Рядок - тип string.

Прототип: (i).

upper_lower (РядокВерхнРегістр, РядокНижнРегістр) - перетворює рядок, зводячи його або до символів верхнього регістра (прописні) або символів нижнього регістра (рядкові).

РядокВерхнРегістр, РядокНижнРегістр - тип string.

Прототип: (i, i), (i, o), (o, i).

upper_lower (СимволВерхнРегістр, СимволНижнРегістр) - перетворити символ, звівши його або до символа верхнього регістра, або символа нижнього регістра.

СимволВерхнРегістр, СимволНижнРегістр - тип char.

Прототип: (i, i), (i, o), (o, i).

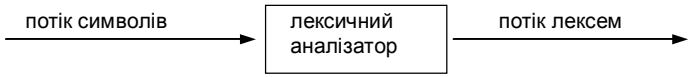
7.2. Обробка тексту і системи граматичного розбору

Пролог має великі можливості для співставлення об'єктів з еталоном, тому ця мова програмування досить добре придатна для обробки текстів. Дії, що виконуються програмою обробки тексту, проходять у двох фазах:

1. *лексичний аналіз* - вхідний текст перетворюється з зовнішньої форми у деяке внутрішнє подання;

2. *граматичний розбір* – виконується аналіз або той чи інший вид обробки внутрішнього подання тексту системою граматичного розбору. Така система - це процедура, яка розпізнає синтаксичні структури високого рівня (об'єкти) у внутрішньому поданні тексту.

Лексичний аналіз виконується лексичним аналізатором (програмою, написаною мовою програмування, наприклад на Пролозі), який розпізнає поєднання символів, що поступають із вхідного потоку, і виробляє потік лексем. Кожна лексема подає один з рядків символів. Множина лексем, генерованих лексичним аналізатором, утворює внутрішнє подання вхідного потоку. Схематично це можна подати:



Приклад 22. Найпростіший лексичний аналізатор.

```
domains                                                    /*pr_22. pro*/
  rechen = symbol
  sp = symbol*
predicates
  lexanaliz (rechen, sp)
  append (sp, sp, sp)
clauses
  lexanaliz ( "", [ ]  ).                                  /*(1)*/
  lexanaliz ( "  ", [ ] ).                                 /*(2)*/
  lexanaliz ( R, LList ) :-
    fronttoken (R, Lex, Ost),
    lexanaliz (Ost, List1),
    append ( [Lex], List1, LList ).
  append ( [ ], Q, Q).
  append (P, Q, R) :-
    P = [ HP | TP ],
    append (TP, Q, TR),
    R = [ HP | TR ].
```

Коментар. Аргумент rechen описується як рядок символів, а аргумент sp - як список рядків символів. Процедура lexanaliz/2 має правило, за яким вхідний текст передається у перший аргумент R та розбивається предикатом fronttoken/3 на лексему Lex (з початку тексту) та остачу тексту Ost; остача тексту, у свою чергу, рекурсивно розбивається на список лексем List1; при цьому процедурою append формується результуючий список лексем LList як об'єднання списків (з однієї лексеми Lex та списку лексем List1). Цей процес проходить доти (гранична умова 1), доки остача тексту не буде містити жодного символу, або (гранична умова 2) буде містити символ "пропуск". Якщо у тексті зустрічається символ "пропуск" (або декілька підряд), то як окрема лексема цей символ не виділяється і у список лексем він не включається.

Наведемо приклади запитів до програми:

Запит	Відповідь
lexanaliz ("Доброго дня, панове!", X)	X = ["Доброго" , "дня" , "," , "панове", "!"]
lexanaliz ("Народ мій є. Народ мій завжди буде...", X)	X = ["Народ" , "мій" , "є" , "." , "Народ" , "мій" , "завжди" , "буде" , "." , "." , "."]
lexanaliz ("Доброго дня! ", X)	X = ["Доброго"," дня", "!"]

Системи граматичного розбору

Переважна більшість систем граматичного розбору на Турбо-Пролозі будується з використанням стратегії, аналогічної до технології розв'язування задач зі зворотнім ходом розв'язку. Така стратегія розв'язку задач прийнята у Турбо-Пролозі за замовчуванням: розв'язок починається з гіпотези (запиту), яка потім розбивається на підцілі, далі кожна підціль ділиться на ще більш дрібні складові частини і т.д. Висхідна гіпотеза буде підтверджена, якщо компілятор прийде до підцілей (фактів), які вже не можна розділити на складові частини, і такі підцілі будуть підтверджені. Хоча існує ряд задач, для яких надається перевага стратегії з прямим ходом розв'язку, застосування якої на Пролозі більш складне.

Розглянемо окремі види граматик[1]: граматику безпосередніх складових (БС-граматику) та граматику, що визначається твердженнями (Т-граматику).

Граматика безпосередніх складових. Для БС-граматики задача граматичного розбору полягає у тому, щоб установити для даної послідовності слів, чи є вона твердженням (реченням) мови.

Розглянемо як приклад фрагмент БС-граматики для певної частини мови, що обслуговує предметну область - геометрію на площині (її певну частину). Відношення між об'єктами та властивості об'єктів подамо у вигляді фраз Хорна (див. п. 1.2.):

1. Речення ← Іменна група, Дієслівна група
2. Іменна група ← Прикметник, Іменник
3. Іменна група ← Іменник
4. Дієслівна група ← Дієслово, Іменна група
5. Дієслівна група ← Дієслово
6. Прикметник ← "дана"
7. Прикметник ← "рівнобедрений"
8. Іменник ← "пряма"
9. Іменник ← "трикутник"
10. Дієслово ← "перетинає"

[1] Граматика - це множина правил, що визначають спосіб побудови речень мови.

Правило 2, наприклад, можна інтерпретувати як те, що компонент речення Іменна група складається з двох компонентів: першого - Прикметник, та другого Іменник, що безпосередньо слідує за першим. Аналогічно описуються й інші правила.

Компоненти з іменами, що починаються з великої літери є *нетермінальними* символами граматики (це граматичні категорії мови), а компоненти з іменами, що починаються з малої літери є *термінальними* (це слова мови).

Один із підходів до розв'язку задачі граматичного розбору полягає у наступному. Розглянемо початковий нетермінальний символ (Речення), що міститься у лівій частині правила 1. Будемо розкривати його, багаторазово "переписувати" ліві частини правил, замінюючи їх компонентами правих частин правил. У випадку, коли компонентом правої частини виявиться термінальний символ, перевіримо, чи співставлятиметься він з черговим словом потоку лексем, що аналізується. Потік (список) лексем буде реченням мови, якщо початковий символ буде повністю розкритий, а всі слова потоку лексем будуть перевірені та успішно погоджені зі словами поточної бази даних (правила 6-10).

Граматика, що визначається твердженнями. Розглянута БС-граматика частково розв'язує задачу граматичного розбору. Вона лише розпізнає речення мови, відокремлюючи їх від набору слів, які не є реченнями, але граматичну структуру не будує. Побудова такої граматичної структури речення називається *деревом розбору*.

Граматика на Пролозі, що будує дерево розбору, *називається граматикою, що визначається твердженнями.*

Т-граматика являє собою більш потужний механізм для описання природної мови, аніж БС-граматика. За його допомогою можна реалізувати деякі важливі властивості мови.

а) Погодження за родом, числом та відмінком.

Розглянемо приклади:

1. *Дана пряма перетинає*
2. *Дана пряма перетинають**
3. *Дані прямі перетинають*
4. *Дана прямі перетинають**

У кожній парі (1,2), (3,4) обидва ланцюжки лексем мають однакову фразову структуру: вони будуються з однакових компонентів в одній і тій же послідовності. Однак, лише перший ланцюжок кожної пари є реченням української мови. Неправильні відмічені *. Ланцюжок 2 недопустимий з тієї причини, що у ньому порушено правило української мови[1], за яким дієслово у ролі присудка має бути погоджене у роді, числі та особі з підметом даного речення. Відпо-

[1] Таке правило називається *правилом контекстної залежності.*

відь на питання про те, чи буде правильним дієслово у ролі присудка, залежить від того контексту, у якому це дієслово знаходиться. Зокрема, правильність дієслова залежить від відмінка, роду і числа підмета, що йому передує. Друге правило контекстної залежності української мови вимагає, щоб прикметник і іменник в іменній групі були погоджені у роді, числі і відмінку. За цим правилом бракується ланцюжок 4.

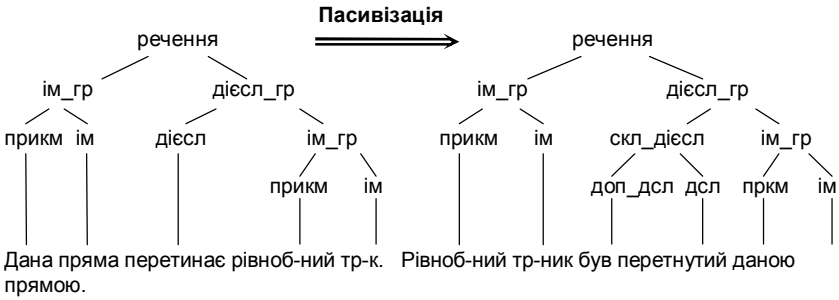
б) Поверхнева та глибинна структура речення.

Розглянемо приклади речень:

5. *Дана пряма перетинає рівнобедрений трикутник.*
6. *Рівнобедрений трикутник був перетнутий даною прямою.*

Ці речення мають однаковий смисл, але їх синтаксичні структури різні. Говорять, що речення 5 подано в активному стані, а речення 6 - у пасивному стані. У реченні 5 підмет виступає у ролі агента, що виконує дію, у реченні 6 - у ролі агента, над яким виконується дія.

З точки зору Прологу речення є ізоморфними і такий ізоморфізм реалізується трансформуванням речення 5 у речення 6 за допомогою правила пасивізації. Подамо вказане правило схематично:



На Турбо-Пролозі можлива реалізація такої граматики, що дозволяє виконувати вказану трансформацію, причому так, щоб дерево розбору подавало глибинну структуру незалежно від того, в активному чи пасивному стані було подано вхідне речення.

ЛАБОРАТОРНА РОБОТА №6

Робота з рядковими величинами. Обробка тексту на Турбо-Пролозі

Мета: Одержати уміння та навички роботи зі стандартними предикатами, що дозволяють виконувати операції з рядками символів. Розглянути роботу з текстом на Турбо-Пролозі.

Теоретична частина: завдання та контрольні питання

І рівень

1. Які величини називаються рядковими, які символьними?
2. Які стандартні предикати використовуються для роботи з рядковими величинами?
3. Що означає у Турбо-Пролозі термін «лексема»?
4. За якими фазами проходять дії, що виконуються програмою обробки тексту?
5. Для чого призначений лексичний аналізатор?

ІІ рівень

1. Поняття системи граматичного розбору.
2. Які можливості Турбо-Прологу як мови програмування дозволяють ефективно використати її для обробки тексту?
3. Чим відрізняються дані типу string від даних типу symbol?
4. Описати роботу найпростішого лексичного аналізатора.

ІІІ рівень

1. У чому полягає суть задачі граматичного розбору для граматики безпосередніх складових?
2. У чому полягає суть задачі граматичного розбору для граматики, що визначається твердженнями?
3. Як розв'язується задача граматичного розбору речення?
4. Описати граматику безпосередніх складових. Навести приклад.
5. Описати граматику, що визначається твердженнями. Які властивості мови можна реалізувати за допомогою граматики, що визначається твердженнями?

Практична частина

І рівень

1-13 варіанти. Скласти програму, за якою:

- у наборі символів, що вводяться з клавіатури, проводиться заміна символу «а» на символ «б» і, навпаки, «б» на «а»;

- лексема, що складається з літер, цифр та вводиться з клавіатури, перетворюється у нову лексему з набором символів у зворотному порядку («паліндром»).

Програму зберегти у файлі "LNN6_1. pro", де NN – номер варіанту користувача.

Вказівка: При написанні програми скористатися рекурсією.

II рівень

Написати програму, яка, використовуючи роботу найпростішого лексичного аналізатора, з текстом, що вводиться з клавіатури, виконує процедуру (вказана у варіанті).

Записати програму у файл (вказаний у варіанті).

1-2 варіанти. Процедура: знайти кількість слів-лексем, що містять літеру «а».

Файл: "LNN6_21. pro".

3-4 варіанти. Процедура: знайти кількість слів-лексем, кожне з яких містить дві літери «а».

Файл: "LNN6_23. pro".

5-6 варіанти. Процедура: знайти кількість «паліндромів» у тексті.

Файл: "LNN6_25. pro".

7-8 варіанти. Процедура: знайти кількість слів-лексем, що складаються з більш як трьох символів.

Файл: "LNN6_27. pro".

9-10 варіанти. Процедура: знайти кількість розділових знаків виду «.», «,», «;», «:», «?», «!».

Файл: "LNN6_29. pro".

11-13 варіанти. Процедура: знайти кількість слів-лексем, що містять хоча б один символ-цифру.

Файл: "LNN6_22. pro".

III рівень

Скласти програму, що дозволяє визначити, чи є речення, що вводиться з клавіатури, реченням мови, з урахуванням наявності у базі даних певних термінальних символів (вказані у варіанті). Побудувати дерево розбору речення. Програму записати у файл (ім'я файлу вказано у варіанті).

1 варіант

Терм. символи: Цей, студент, відвідує, практичні, заняття.

Файл програми: "L6_31. pro".

2 варіант

Терм. символи: Сильна, злива, заважає, пішоходам.
Файл програми: "L6_32. pro".

3 варіант

Терм. символи: Прямокутний, трикутник, має, прямий, кут.
Файл програми: "L6_33. pro".

4 варіант

Терм. символи: Рівносторонній, трикутник, має, рівні, сторони.
Файл програми: "L6_34. pro".

5 варіант

Терм. символи: Непаралельні, прямі, мають, спільну, точку.
Файл програми: "L6_35. pro".

6 варіант

Терм. символи: Молоді, люди, читають, цікавий, журнал.
Файл програми: "L6_36. pro".

Вимоги до захисту лабораторної роботи

Звіт виконаної лабораторної роботи повинен містити назву теми; текст програми І-го та ІІ-го рівнів, запити до програм І-го, ІІ-го та ІІІ-го рівнів.

§8. ДОДАТКОВІ ВІДОМОСТІ

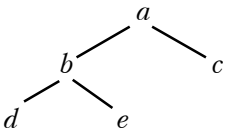
8.1. Бінарні дерева

Бінарне дерево означається рекурсивно як таке, що має ліве піддерево, корінь і праве піддерево. Ліве і праве піддерева є бінарними деревами.

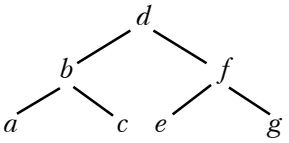
Бінарне дерево називається впорядкованим, якщо довільний елемент лівого піддерева менший, ніж значення кореня, а довільний елемент правого піддерева - більший.

Наведемо відповідні приклади-схеми.

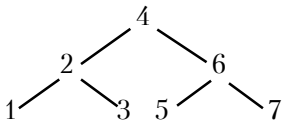
а) невпорядковане бінарне дерево:



б) впорядковані бінарні дерева:



якщо вважати, що $a<b<c<d<e<f<g$



Якщо уявити множину, що має велику кількість елементів у вигляді списку, то робота з такою структурою даних іноді стає неефективною. Наприклад, якщо, маючи список із 1024 перших натуральних чисел, виконати запит `member(3000, L)`, то одержимо відповідь `No`; при цьому компілятор виконує перевірки 1024 чисел для того, щоб дати правильну відповідь. Якщо вказані числа подані у вигляді впорядкованого бінарного дерева, то компілятор для погодження запиту виконуватиме не більш як 11 перевірок.

Турбо-Пролог має ряд стандартних предикатів (які належать до предикатів зовнішньої бази даних) для роботи з бінарними деревами:

bt_create (СелекторБД, ІмяДер, СелекторДер, ДовжнКлюча, Порядок) - створити дерево.

СелекторБД - тип db_selector.
ІмяДер - тип string.
СелекторДер - тип bt_selector.
ДовжнКлюча - тип integer.
Порядок - тип integer.
Прототип: (i ,i, o, i, i).

bt_open (СелекторБД, ИмяДер, СелекторДер) - відкрити дерево.

СелекторБД - тип db_selector.
ИмяДер - тип string.
СелекторДер - тип bt_selector.
Прототип: (i, i, o).

bt_close (СелекторБД, СелекторДер) - закрити дерево.

СелекторБД - тип db_selector.
СелекторДер - тип bt_selector.
Прототип: (i, i).

bt_delete (СелекторБД, ИмяДер) - вилучити дерево

СелекторБД - тип db_selector.
ИмяДер - тип string.
Прототип: (i, i).

bt_statistics (СелекторБД, СелекторДер, ЧислоКлючв, ЧислоСтор, Глибина, ДовжнКлюча, Порядок, РозмрСтор) - статистика доступу до дерева.

СелекторБД - тип db_selector.
СелекторДер - тип bt_selector.
ЧислоКлючв - тип real.
ЧислоСтор - тип real.
Глибина, ДовжнКлюча, Порядок, РозмрСтор - тип integer.
Прототип: (i, i, o, o, o, o, o).

key_insert (СелекторБД, СелекторДер, Ключ, Посилання) - вставити ключ.

СелекторБД - тип db_selector.
СелекторДер - тип bt_selector.
Ключ - тип string.
Посилання - тип ref.
Прототип: (i, i, i, i).

key_delete (СелекторБД, СелекторДер, Ключ, Посилання) - вилучити ключ.

СелекторБД - тип db_selector.
СелекторДер - тип bt_selector.
Ключ - тип string.
Посилання - тип ref.
Прототип: (i, i, i, i).

key_first (СелекторБД, СелекторДер, ПершеПосилання) - перший ключ.

СелекторБД	- тип db_selector.
СелекторДер	- тип bt_selector.
ПершеПосилання	- тип ref.

Прототип: (i, i, o).

key_last (СелекторБД, СелекторДер, ОстаннєПосилання) - останній ключ.

СелекторБД	- тип db_selector.
СелекторДер	- тип bt_selector.
ОстаннєПосилання	- тип ref.

Прототип: (i, i, o).

key_search (СелекторБД, СелекторДер, Ключ, Посилання) - відшукати ключ.

СелекторБД	- тип db_selector.
СелекторДер	- тип bt_selector.
Ключ	- тип string.
Посилання	- тип ref.

Прототип: (i, i, i, o).

key_next (СелекторБД, СелекторДер, НаступнеПосилання) - наступний ключ.

СелекторБД	- тип db_selector.
СелекторДер	- тип bt_selector.
НаступнеПосилання	- тип ref.

Прототип: (i, i, o).

key_prev (СелекторБД, СелекторДер, ПопереднєПосилання) - попередній ключ.

СелекторБД	- тип db_selector.
СелекторДер	- тип bt_selector.
ПопереднєПосилання	- тип ref.

Прототип: (i, i, o).

key_current (СелекторБД, СелекторДер, Ключ, Посилання) - поточний ключ.

СелекторБД	- тип db_selector.
СелекторДер	- тип bt_selector.
Ключ	- тип string.
Посилання	- тип ref.

Прототип: (i, i, o, o).

8.2. Графи

За допомогою графів на Пролозі можна подати відношення між об'єктами, ситуації, структури задач тощо.

Граф - це множина вершин і множина ребер, причому кожне ребро задається парою вершин.

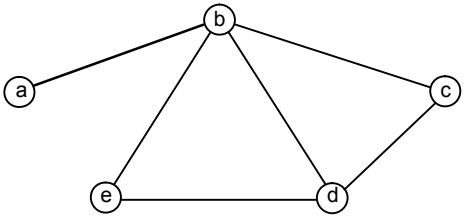
Якщо ребра напрямлені (вказується напрямок зв'язку між вершинами, наприклад, початком ребра "1" є вершина "a", кінцем -

вершина "b"), то їх також називають *дугами*. Кожна дуга задається впорядкованою парою; у цьому випадку граф є *напрямленим*.

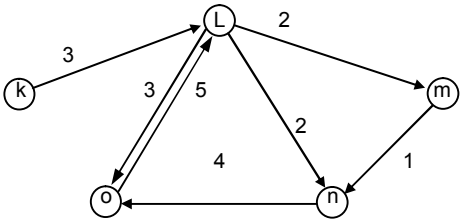
Ребрам графів, як правило, приписуються імена або мітки (довжини, вартості тощо).

Графи подаються схематично. Наведемо приклади графів:

1) ненапрямлений граф



2) напрямлений граф (для кожної дуги встановлено мітку, наприклад, якщо k,l,m,n,o - міста, то числа можуть вказувати на час руху у напрямку від одного міста до іншого):



На Турбо-Пролозі графи можна подавати у різні способи:

а) як множину тверджень: кожне ребро графа подати у вигляді окремого факту.

Наприклад, розглянуті графи записати як базу даних програми (секція clauses) можна так:

```
ребро(a,b).     ребро(b,c).
ребро(b,d).     ребро(b,e).
ребро(c,d).     ребро(d,e).

дуга(k,l,3).    дуга(l,m,2).
дуга(l,n,2).    дуга(l,o,3).
дуга(m,n,1).    дуга(n,o,4).
дуга(o,l,5).
```

б) як один об'єкт: предикат, першим аргументом якого є список вершин, а другим - список ребер. Причому список вершин є списком елементів, як правило, рядкового або числового типу, а список ребер - це список структур типу ребро(x,y). Для графу прикладу 1) об'єкт як факт бази даних програми (секція clauses) можна записати так:

```
граф(    [a,b,c,d,e],
         [ребро(a,b), ребро(b,c), ребро(b,d), ребро(b,e),
          ребро(c,d), ребро(d,e)] ).
```

Аналогічно подається граф прикладу 2):

```
нграф(   [k,l,m,n,o],
         [дуга(k,l,3), дуга(l,m,2), дуга(l,n,2), дуга(l,o,3),
          дуга(m,n,1), дуга(n,o,4), дуга(o,l,5)] ).
```

Якщо кожну вершину графа сполучено ребром хоча б з однією іншою вершиною, то записати у програму вказаний об'єкт можна без списку вершин (першого аргументу відповідного предикату), оскільки множина вершин неявно міститься у списку ребер.

Операції над графами

Над графами виконують, серед інших, такі типові операції:

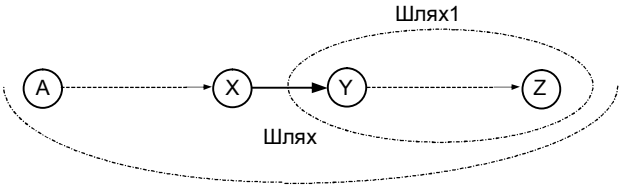
- знайти шлях між двома даними вершинами;
- знайти підграф, що має деякі задані властивості.

Перша операція для багатьох практичних задач вимагає знаходження ациклічного шляху, тобто такого шляху, за яким вершина (з множини даних вершин) проходиться тільки один раз, іншими словами такий шлях - це послідовний перелік вершин від однієї з заданих вершин до другої, і цей перелік повинен містити будь-яку вершину тільки один раз.

Розглянемо один із методів пошуку шляху (граф визначимо за схемою прикладу 1)): для того, щоб знайти ациклічний шлях Шлях між двома вершинами A і Z у графі Граф, необхідно:

- покласти Шлях=[A], якщо A=Z, інакше,
- знайти ациклічний шлях Шлях1 з довільної вершини Y у Z, а далі знайти шлях з A в Y, що не містить вершин із шляху Шлях1.

У свою чергу, Шлях1 можна визначити рекурсивно. Справді, якщо подати операцію знаходження шляху Шлях за схемою



то для правила Шлях1 є гранична умова - початкова вершина Y шляху Шлях1 співпадає з початковою вершиною A шляху Шлях. Рекурсивне правило - якщо початкові вершини вказаних шляхів не співпадають, то повинна існувати така вершина X, що:

- Y - вершина, суміжна з X,
- X не належить шляху Шлях1 і
- наступний рекурсивний виклик предикату Шлях1 включає визначення нового шляху Шлях1 від вершини X через суміжну вершину Y до вершини Z.

Запишемо відповідну програму на Турбо-Пролозі.

Приклад 23. Програма пошуку шляху у графі.

```
domains                                                    /*pr_23. pro*/
  вершина = symbol
  список_вершин = вершина*
  структура_ребро = ребро (вершина, вершина)
  список_ребер=структура_ребро*
predicates
  граф (список_вершин, список_ребер)
  шлях (вершина, вершина, список_вершин)
  шлях1 (вершина, список_вершин, список_вершин)
  суміжні_вершини (вершина, вершина, список_ребер)
  member (вершина, список_вершин)
  member (структура_ребро, список_ребер)

clauses
 граф ([a, b, c, d, e],
          [ребро (a,b), ребро (b,c), ребро (b,d), ребро (b,e), ребро (c,d),
          ребро (d,e)]).
  шлях (A, Z, Шлях):-                                     /* (1)*/
    bound (A), bound (Z),  шлях1 (A, [Z], Шлях).
  шлях1 (A, [A|Шлях1], [A|Шлях1]).                         /* (2)*/
  шлях1 (A, [Y|Шлях1], Шлях):-                             /* (3)*/
      граф (_, Список_ребер),
      суміжні_вершини (X, Y, Список_ребер),
      not (member (X, Шлях1)),
      шлях1 (A, [X, Y|Шлях1], Шлях).
  суміжні_вершини (X, Y, Список_ребер):-                   /* (4)*/
      member (ребро (X, Y), Список_ребер) ;
        member (ребро (Y, X), Список_ребер).
  member (R, [R|T]).                                       /* (5)*/
  member (R, [H|T]): -
      member (R, T).
```

Коментар. Програма предикатом граф/2 встановлює факт про те, що задається граф. Перший аргумент вказаного предикату є списком вершин графа, а другий - списком ребер графа. За правилом (1) у правило (2) передається початкова вершина (перший аргумент); кінцева ж вершина передається як список (другий аргумент), тобто як деякий шлях від деякої вершини, що передує вершині Z до самої вершини Z; шуканий шлях від початкової до кінцевої вершини передається без змін (третій аргумент). Рекурсивне правило (3)

описано раніше. Виклик цим правилом предикату граф/2 дозволяє одержати список ребер, який використовує предикат суміжні_вершини/3 для визначення вершини X, суміжної вершині Y. При цьому правило (4) об'єднує у собі два правила, так як перелік умов розділений символом "крапка з комою": вершина X суміжна вершині Y, якщо існує ребро графа виду ребро (X, Y) *або* виду ребро (Y, X). Правило (3) має граничну умову (2), за якою шлях від A до Z вважається знайденим (третій аргумент), якщо знайдено (під час рекурсії) новий шлях, у якого початкова вершина X співпадає з вершиною A (другий аргумент). Програма містить правило (5), за яким визначається належність деякого елемента списку.

Приклади запитів до програми.

Запит	Відповідь
шлях(a,b,S)	S=["a","b"] 1 Solution
шлях(a,c,S)	S=["a","b","c"] S=["a","b","d","c"] S=["a","b","e","d","c"] 3 Solutions
шлях(A,Z,S)	No Solution

Класичною задачею для графів є пошук цикла (шляху) Гамільтона, тобто такого ациклічного шляху, що проходить через *всі* вершини графа. Наведемо приклад відповідної програми (для графа за схемою 1), модифікувавши попередню програму:

Приклад 24. Пошук шляху Гамільтона.

```
domains                                                    /*pr_24. pro*/
  вершина = symbol
  список_вершин = вершина*
  структура_ребро = ребро (вершина, вершина)
  список_ребер = структура_ребро*
predicates
  шлях_Гамільтона (список_вершин)
  граф (список_вершин, список_ребер)
  шлях (вершина, вершина, список_вершин)
  шлях1 (вершина, список_вершин, список_вершин)
  суміжні_вершини (вершина, вершина, список_ребер)
  member (вершина, список_вершин)
  member (структура_ребро, список_ребер)
  всі_вершини (список_вершин)
  довжина (список_вершин, integer)
  вибір_вершин (вершина, вершина)
clauses
  граф ([a, b, c, d, e],
```

```
        [ребро (a, b), ребро (b, c), ребро (c, d),
        ребро (d, e), ребро (b, e), ребро (b, d)]).
шлях_Гамільтона (Шлях):-
    вибір_вершин (Початкова, Кінцева),
    шлях (Початкова, Кінцева, Шлях),
    всі_вершини (Шлях).
вибір_вершин (Початкова, Кінцева):-
    граф (Список_вершин, Список_ребер),
      member (Початкова, Список_вершин),
      member (Кінцева, Список_вершин).
% суміжні_вершини (Початкова, Кінцева, Список_ребер).    /* (1)*/
шлях (A, Z, Шлях):-
  шлях1 (A, [Z], Шлях).
шлях1 (A, [A|Шлях1], [A|Шлях1]).
шлях1 (A, [Y|Шлях1], Шлях):-
    граф (_, Список_ребер),
      суміжні_вершини (X, Y, Список_ребер),
      not (member (X, Шлях1)),
      шлях1 (A, [X, Y|Шлях1], Шлях).
суміжні_вершини (X, Y, Список_ребер):-
    member (ребро (X, Y), Список_ребер);
  member (ребро (Y, X), Список_ребер).
всі_вершини (Шлях):-
    граф (Список_вершин, _ ),
    довжина (Шлях, N),
    довжина (Список_вершин, N).
member (R, [R|T]).
member (R, [H|T]):-
    member (R, T).
довжина ([ ], 0).
довжина ([H|T], N):-
    довжина (T, N1),
    N=N1+1.
```

Коментар. На відміну від попередньої програми, модифікована програма містить правило шлях_Гамільтона, за яким програма, а не користувач, предикатом вибір_вершин перебирає всі можливі комбінації "початкова вершина"-"кінцева вершина", предикатом шлях/3 визначає ациклічний шлях для кожної комбінації та предикатом всі_вершини перевіряє, чи всі вершини графу увійшли до шляху, що являє собою послідовний список вершин від початкової до кінцевої. Правило вибір_вершин визначає дві потрібні вершини як елементи списку вершин графа. За правилом всі_вершини ациклічний шлях проходить через всі вершини графа, якщо кількість вершин, що містить шлях (кількість елементів списку) рівна кількості вершин у списку вершин графа. При цьому використовується предикат довжина. Якщо у даній програмі вилучити знак коментаря "%", включивши тим самим виконання предикату суміжні_вершини у правило вибір_вершин, то це зумовить вибір початко-

вої і кінцевої вершин як суміжних. Іншими словами, програма буде шукати шлях Гамільтона, при якому можливий обхід вершин від початкової до кінцевої, і далі, до початкової

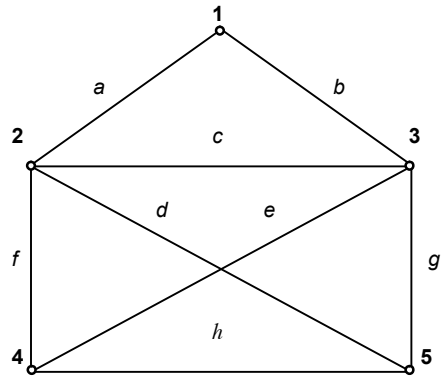
Приклади запитів до програми.

Запит	Відповідь	Примітка
шлях_Гамільтона(S)	S=["a","b","e","d","c"] S=["a","b","c","d","e"] S=["c","d","e","b","a"] S=["e","d","c","b","a"] 4Solutions	
шлях_Гамільтона(S)	No Solution	Якщо у програмі вилучити символ коментаря (%).

Для графів іноді виникають задачі, у яких необхідно визначити шлях від однієї вершини до іншої за умов:

- необхідно пройти всі ребра графа;
- кожне ребро (дугу) дозволяється пройти тільки один раз (у будь якому напрямку);
- вершини дозволяється проходити більше одного разу;
- шлях подається як послідовний список ребер.

Приклад 25. Запишемо програму, що дозволяє розв'язати відому задачу: "Намалювати модель конверта, не відриваючи олівець від паперу та не проводячи двічі однієї і тієї ж лінії". Спочатку переформулюємо задачу у термінах вершин та ребер: "Обійти всі ребра графа, не проходячи більше одного разу по певному ребру так, щоб у кінці обходу повернутися у ту вершину, з якої починався обхід". Поданмо схематично граф і пронумеруємо його вершини та ребра




```
domains                          /*pr_25. pro*/
    вершина=integer
    дуга=symbol
```

```
    список_дуг=дуга*
    структура_ребро=p (symbol, вершина, вершина)
    список_ребер=структура_ребро*
predicates
    граф (список_ребер)
    шлях (вершина, список_дуг)
    шлях1 (дуга, вершина, список_дуг, список_дуг)
    дуга (дуга, вершина, вершина)
    довжина (список_дуг, integer)
    довжина (список_ребер, integer)
    суміжна_дуга (дуга, дуга, вершина, вершина)
    member (дуга, список_дуг)
    member (структура_ребро, список_ребер)

clauses
  граф ([p (a, 1, 2), p (b, 1, 3), p (c, 2, 3), p (d, 2, 5), p (e, 3, 4), p (f, 2, 4),
         p (g, 3, 5), p (h, 4, 5)]).
  шлях (Початкова_вершина, Шлях):-
    дуга (Початкова_дуга, Початкова_вершина, Поточна_вершина),
    шлях1 (Початкова_дуга, Поточна_вершина, [Кінцева_дуга], Шлях),
    Початкова_дуга<>Кінцева_дуга.

шлях1 (Початкова_дуга, _, [Початкова_дуга|Шлях1],
        [Початкова_дуга|Шлях1]).

шлях1 (Початкова_дуга, Поточна_вершина, [Дуга_Y|Шлях1], Шлях):-
  граф (Список_ребер),
  суміжна_дуга (Дуга_X, Дуга_Y, Поточна_вершина, Нова_вершина),
  not (member (Дуга_X, Шлях1)),
  шлях1 (Початкова_дуга, Нова_вершина, [Дуга_X, Дуга_Y|Шлях1],
Шлях),
  довжина (Список_ребер, L), довжина (Шлях, L).

  суміжна_дуга (X, Y, Поточна_вершина, Нова_вершина):-
      дуга (Y, Поточна_вершина, Нова_вершина),
      дуга (X, Нова_вершина, _),
      X<>Y.
  дуга (X, M, N):-
      граф (Список_ребер), member (p (X, N, M), Список_ребер);
      граф (Список_ребер), member (p (X, M, N), Список_ребер).
  member (R, [R|T]).
  member (R, [H|T]):-
      member (R, T).
  довжина  ([ ], 0).
  довжина  ([H|T], N):-
      довжина  (T, N1),
      N=N1+1.
```

Коментар. У програмі предикат граф/1 подає граф як список ребер. При цьому список вершин окремо не вказується, так як вони неявно міс-

тяться у списку ребер. Шуканий шлях у предикаті шлях/2 визначається як список назв дуг (другий аргумент), а першим аргументом є початкова вершина початкової дуги шляху. За правилом шлях/2 знаходиться початкова дуга, у якій одна з вершин фіксується як початкова, а інша - як поточна; предикат шлях/1 визначає шуканий шлях як деякий шлях1 від поточної вершини початкової дуги до кінцевої дуги, при цьому початкова і кінцева дуги різні.

Гранична умова шлях1 та правило шлях1 описуються аналогічно відповідним фразам попередньої програми (Приклад 23.), але вони містять додатковий аргумент Поточна_вершина, який дозволяє при рекурсивному виклику предиката шлях1 (від кінцевої дуги до початкової дуги) вибирати наступну дугу (Дуга_X) як суміжну, тобто як таку, що своєю початковою вершиною (Нова_вершина) має кінцеву вершину дуги (Дуга_Y), яка була останньою вибрана при рекурсії (і початком має поточну вершину (Поточна_вершина)). Далі за поточну вершину береться Нова_вершина. Цей рекурсивний процес продовжується доти, доки вибір наступної дуги (Дуга_X) не співпаде з початковою дугою шляху.

Крім того, у правилі шлях1 вимагається, щоб кількість дуг шляху (довжина шляху) співпадала з кількістю ребер графа (довжина списка ребер), що забезпечується предикатом довжина/2.

Правило дуга/3 визначає всі дуги графа, кожна з яких має ім'я X та вершини N і M за умови, що у списку ребер графа є ребро з іменем X та вершинами N (початкова), M (кінцева) або ребро з іменем X та вершинами M (початкова), N (кінцева).

Можливі запити до наведеної програми.

Запит	Інтерпретація	Результат
шлях(1,S)	Як можна обійти граф, починаючи з вершини 1?	No Solution
шлях(4,_)	Чи існує шлях,якщо розпочати обхід з вершини 4.	Yes
шлях(N,S)	Визначити всі можливі шляхи обходу, вказавши початкові вершини.	N=5,S=["h","f","d","g","c","a","b","e"] … … N=4,S=["h","g","e","f","c","b","a","d"] … … 24 Solutions
шлях(_,S),!	Визначити будь-який один з можливих шляхів обходу.	S=["h","f","d","g","c","a","b","e"]

До розглянутого класу задач належить і відома задача про кенігсбергські мости[1], що вперше розв'язана великим математиком Ейлером. Якщо задачу подати як пошук шляху у відповідному графі, внести відповідні зміни у текст програми (тільки у факт граф/1) та сформулювати потрібний запит (наприклад шлях (N, S)), то одержимо відповідь No Solution. Отже, програма підтвердить, що розв'язку задачі не існує, що було вперше строго доведено Ейлером у 1736 році.

Графи і штучний інтелект

Графи використовуються для ефективного подання знань про предметну область. Клас задач, пов'язаних з пошуком шляхів у графах (що розв'язується на Турбо-Пролозі перебором всіх варіантів), тісно пов'язаний із задачами, що виникають при побудові систем штучного інтелекту. В теоретичних роботах, пов'язаних з цією галуззю, було показано, що здатність до розв'язування інтелектуальних задач мають так звані символічні системи.

Символічна система являє собою множину символів, які можуть об'єднуватися один з одним у так звані символічні структури. Деякі символічні структури можуть "бути інтерпретовані", тобто з ними можуть бути зв'язані процедури обробки даних. Розв'язування інтелектуальних задач символічними структурами виконується методом генерації різних розв'язків і відкиданням непотрібних (тих, що не задовільняють певним умовам), тобто тим же *перебором*.

Очевидно, що мову Пролог цілком можна розглядати як конкретну реалізацію цих абстрактних символічних систем. Символи - це атоми, символічні структури - структури прологу, інтерпретуючі структури - правила, неінтерпретуючі структури - факти, перебір варіантів - основа логічного виводу на Пролозі. Тому Пролог розглядається як перспективна мова для реалізації систем штучного інтелекту.

8.3. Управління ходом виконання програми

Відсікання

Відсікання, як зазначалося, зупиняє роботу механізму повернення Турбо-Прологу. Тому відсікання використовується для запобігання непотрібних повернень і скорочення альтернативних шляхів у

[1] Через місто Кенігсберг тече річка, яка має два острова. Перший острів з лівим берегом річки сполучено двома мостами, з правим - теж двома. Другий острів сполучено з лівим берегом річки одним мостом і з правим одним мостом. Крім того, є міст, що сполучає острова між собою. Отже, є сім мостів. Необхідно пройти по всім мостам міста, не проходячи двічі по одному й тому ж мості.

програ-мі, якщо наперед відомо, що той або інший шлях не приведе до розв'язку.

Відсікання забезпечується предикатом "!", який дає вказівку компілятору не повертатися назад далі тієї точки, де стоїть даний предикат.

Відсікання нескінченних циклів.

Програма, що перевіряла належність чисел до натуральних (стор. 36) використовувала умову N>0, так як без цієї умови компілятор знаходить правильний розв'язок, а потім "зациклюється" і може дати повідомлення про переповнення стеку пам'яті.

Використовуючи відсікання, вкажемо компілятору, що за даною граничною умовою існує не більше одного роз'язку, тобто при знаходженні однієї правильної відповіді інші роз'язки шукати не потрібно.

Приклад 26. Використовуючи відсікання забезпечити перевірку чисел на предмет їх належності до натуральних

```
domains                                                    /*pr_26. pro*/
   число = real
predicates
   натуральне ( число )
clauses
  натуральне ( 1 ): - !.
  натуральне ( N ):-
     N1=N-1,
     натуральне ( N1 ).
```

Коментар. У програмі з правила вилучено умову N>0. Гранична умова натуральне(1) замінена на правило натуральне(1):-!, що містить предикат "!".

Програмування тверджень, що взаємно виключають одне одного.

Розглянемо задачу про переведення бальної оцінки знань студента (з інтервалу від 0 до 750) у традиційну п'ятибальну систему:

- сума балів від 675 і вище - оцінка "відмінно";
- від 450 до 675 - оцінка "добре";
- від 375 до 450 - оцінка "задовільно";
- нижче 375 - оцінка "незадовільно".

Приклад 27. Визначення традиційної оцінки за шкалою балів

```
domains                                                    /*pr_27. pro*/
   число = integer
predicates
   оц ( число, char )
```

```
clauses
  оц ( M, '5' ) : - M>=675.
  оц ( M, '4' ) : - M>=450.
  оц ( M, '3' ) : - M>=375.
  оц ( M, '2' ).
```

Для запиту оц (455, X) одержимо три відповіді (3 Solutions): X=4, X=3, X=2.

Для усунення альтернативних розв'язків введемо відсікання у правила програми. Секція clauses модифікується так:

```
  оц(M,'5'): - M>=675, !.
  оц(M,'4'): - M>=450, !.
  оц(M,'3'): - M>=375, !.
  оц(M,'2').
```

Для попереднього запиту одержимо одну відповідь: X = 4.

Невдале завершення доведення цілі.

Під час виконання запиту оц(955, X) до модифікованої програми попереднього прикладу (Приклад 27.) одержимо відповідь X=5. Однак за умовою задачі вхідні бали повинні бути числом від 0 до 750. Змінивши першу і останню фрази програми, матимемо:

Приклад 28. Модифікована програма для визначення традиційної оцінки за шкалою балів

```
domains                                                    /*pr_28. pro*/
  число=integer
predicates
  оц ( число, char )
clauses
  оц ( M, '5' ) : - M>=675,  !,  M<=750.
  оц ( M, '4' ) : - M>=450,  !.
  оц ( M, '3' ) : - M>=375,  !.
  оц ( M, '2' ) : - M>=0.
```

Відсікання і відкіт

Інший варіант розв'язку задачі полягає у тому, щоб використати предикат fail, за яким Турбо-Пролог вимушений невдало завершити процес погодження фрази програми (імітувати неуспішне обчислення), що спонукає систему здійснити відкіт для погодження фрази програми ще раз.

Тому введемо у твердження випадок виникнення невдачі, використовуючи комбінацію <"відсікання" - fail>.

Предикат "!" у цій ситуації забезпечує усунення всіх наступних відкітів. Тому не потрібно змінювати першу і останню фрази програми (Приклад 27.).

Приклад 29. Змінена програма для визначення традиційної оцінки за шкалою балів. Програма використовує відсікання і відкіт.

```
domains                                                    /*pr_29. pro*/
  число=integer
predicates
  оц(число,char)
clauses
  оц(M,''):   - M>750, !, fail.
  оц(M,'5'): - M>=675, !.
  оц(M,'4'): - M>=450, !.
  оц(M,'3'): - M>=375, !.
  оц(M,'2').
  оц(M,''):   - M<0, !, fail.
```

Коментар. При запиті оц (955, X) спочатку буде досягнуто ціль вказаного запиту за першим правилом програми, потім доведено ціль "відсікання", а за цим виконано предикат fail (невдале завершення доведення фрази за цим правилом); але повернення до розгляду інших фраз програми буде зупинено відсіканням. Тому одержимо відповідь No.

Відкіт після невдачі

Для управління обчисленням внутрішніх цілей (підцілей) Пролог-програми під час відшукання всіх її розв'язків також використовується предикат fail. Відкіт після невдачі (яка імітується предикатом fail) при погодженні фрази програми зручно використовувати при програ-муванні на Турбо-Пролозі прикладних задач з обробки службових да-них, наприклад:

- розробка розрахункової відомості;
- генерація звітів про заробітну плату;
- обробка різноманітних баз даних тощо.

Приклад використання даного предикату для обробки бази даних природних мов та відповідні пояснення наведено у програмі на стор. 21.

Позалогічні предикати вводу-виводу

У п.2.3. розглядались можливості вводу та виводу даних під час організації запитів; при цьому дані можна було ввести тільки до виконання запиту, а одержати результати - після завершення роботи програми. Перед прикладом на стор. 18 викладена коротка інформація про предикати вводу-виводу. Розширимо її.

До предикатів Турбо-Прологу, що забезпечують ввід-вивід даних *у процесі роботи* програми відносяться:

readchar (Змінна) - читає символ з поточного пристрою вводу (вхідного

потоку) і зв'язує символ із змінною Змінна. Встановлює режим очікування вводу.

Змінна - тип char.

Прототип : (о).

readint (Змінна) - читає ціле число з поточного пристрою вводу і зв'язує число зі змінною Змінна. Встановлює режим очікування вводу.

Змінна - тип integer.

Прототип: (о).

readln (Змінна) - читає рядок з поточного пристрою вводу і зв'язує рядок із змінною Змінна. Встановлює режим очікування вводу. Як кінець зчитування рядка використовується клавіша Enter (позначатимемо символом ↵).

Змінна - тип string.

Прототип: (о).

readreal (Змінна) - читає дійсне число з поточного пристрою вводу і зв'язує число зі змінною Змінна. Встановлює режим очікування вводу.

Змінна - тип real.

Прототип: (о).

readterm (Область, Терм) - читає терм з поточного пристрою вводу і зв'язує терм зі змінною Терм; читає з бази даних терм, який був записаний предикатом write.

Область - специфікація області значень, яка визначає те, які типи термів може зчитувати даний предикат.

Терм - тип, оголошений в області значень (розділ domains).

Прототип: (і, о).

write (Зн1, Зн2, Зн3,...) - записує задані значення $Зн_1$, $Зн_2$, $Зн_3$,... на поточний пристрій виводу.

$Зн_1$, $Зн_2$, $Зн_3$,... - константи або змінні.

Прототип: (і).

writef (Формат, $Зн_1$, $Зн_2$, $Зн_3$,...) - записує задані значення $Зн_1$, $Зн_2$, $Зн_3$,... на поточний пристрій виводу у форматі Формат.

$Зн_1$, $Зн_2$, $Зн_3$,... - константи або змінні.

Формат: % (початковий символ)
- (зв'язка)
m (мінімальна ширина поля)
p (точність)

Прототип: (і).

nl - виконує переведення рядка таким чином, як це відбувається під час відправляння на поточний пристрій виводу символу повернення каретки.

Нагадаємо, що позалогічними ці предикати називають тому, що їх дії у процесі розв'язку логічних цілей породжують побічні ефекти, а побічний ефект - це така дія предикату, яка не анулюється при поверненні у разі роботи механізму повернення Турбо-Прологу. Якщо, наприклад, компілятор виконує повернення через виклик предикату read/1, аргументом якого є змінна, то підстановка терму, що зчитується, на місце цієї змінної виявиться анульованою, але при цьому дія, суть якої полягає у читанні терму із вхідного потоку, не анулюється. Терм не може бути "повернутий назад" у вхідний потік і тому вказане значення для програми буде втрачено.

Розглянемо у режимі діалогу приклади роботи окремих предикатів вводу-виводу (у складі програм предикати працюють аналогічно):

а) Goal: readln (X) ↵ (вводимо предикат у запит)
abc ↵ (вводимо з клавіатури рядок "abc")
X = abc (відповідь компілятора)

б) Goal: readterm (real, X) ↵
5 ↵
X = 5

в) Вкажемо, які оголошення необхідно записати до програми, щоб проілюструвати, як можна зчитати структуру:

```
domains                          /*оголошення типів області значень*/
    p = f (symbol, real )        /*специфікація терму в області значень*/
```

Маючи такі записи у тексті програми, виконаємо:
Goal: readterm (p, X) ↵
f ("a", 3) ↵
X = f ("a", 3)

г) Goal: X=7, write(8,fn,X,2,9), nl, write(8,"go",X,2)□
8fn729
8go72

д) Goal: writef("%3.2",85.5) ↵
85.50

Робота з файлами

Для роботи з файлами використовуються такі стандартні предикати[1]:

openread (N, F) - відкриває файл для читання. Зв'язує задане аргументом N ім'я з іменем файла DOS.

N - тип file.

F - тип string.

Прототип: (i, i).

openwrite (N, F) - відкриває файл для запису. Зв'язує задане аргументом N ім'я з іменем файла DOS.

N - тип file.

F - тип string.

Прототип: (i, i).

openappend (N, F) - відкриває файл для приєднання запису у кінець файла. Зв'язує задане аргументом N ім'я з іменем файла DOS.

N - тип file.

F - тип string.

Прототип: (i, i).

openmodify (N, F) - відкриває файл для читання і для запису. У файл довільного доступу можна внести зміни, якщо він відкритий за допомогою предикату openmodify/2, а для позиціювання покажчика запису використовується стандартний предикат filepos/3 (див.нижче). Зв'язує задане аргументом N ім'я з іменем файла DOS.

N - тип file.

F - тип string.

Прототип: (i, i).

filepos (N, Позиція, Режим) - встановлює покажчик даного файла на задану позицію.

N - тип file.

Позиція - тип real. Позиція запису файла.

Режим - тип integer. Вказує на розміщення покажчика відносно позиції Позиція:

0 - відносно початку файла;

1 - відносно поточної позиції;

---

[1] У описанні предикатів використані позначення:

N - внутрішнє (у програмі) ім'я файла, яке буде використовуватись при зверненні до файла системи MSDOS;

F - ім'я файла в операційній системі MSDOS.

2 - відносно кінця файла.
Прототип: (i, i, i).

eof (N) - перевірка на кінець файла. Виконується успішно, якщо покажчик поточної позиції вказує на кінець файла, і завершується невдачею в інших випадках.

N - тип file.
Прототип: (i).

existfile (F) - перевіряє, чи існує файл. Виконується успішно, якщо файл міститься у поточному каталозі, і завершується невдачею в інших випадках.

F - тип string.
Прототип: (i).

deletefile (F) - знищує файл DOS. Аргумент може містити ідентифікатор накопичувача, але не повинен вказувати шлях.

F - тип string.
Прототип: (i).

renamefile (FG, FN) - переіменовує файл DOS. Аргумент FG визначає старе ім'я файла, аргумент FN - нове ім'я.

FG, FN - тип string.
Прототип: (i, i).

disk (X) - встановлює або зчитує (показує) активний накопичувач та шлях до активного каталогу.

X - тип string.
Прототип: (i) , (o).

writedevice (N) - встановлює або зчитує символічне ім'я файла для пристрою виводу. Наступні команди запису зможуть використовувати встановлене символічне ім'я файла. Файл повинен бути попередньо відкритий предикатом openwrite, openmodify або openappend.

N - тип symbol.
Прототип: (i) , (o).

readdevice (N) - встановлює або зчитує симоволічне ім'я файла для пристрою вводу. Предикати зчитування отримуватимуть інформацію з файла, що має встановлене символічне ім'я.

N - тип symbol.
Прототип: (i) , (o).

closefile (N) - закриває вказаний файл N. Ім'я файла не повинне міститися у лапках. Виконується успішно, якщо навіть даний файл перед цим не був відкритий.

N - тип file.
Прототип: (i).

Зауваження 6.

Стандартні предикати, що виконують дії з файлами DOS, виконуватимуться за умови наявності файлів DOS на активному логічному пристрої (накопичувачі) та у активній директорії (каталозі).

Приклад 30. Наведемо приклад програми, яка зчитує число, записане у DOS-файлі "num.txt", обчислює його факторіал, записує число-результат у DOS-файл "res.txt" та дозволяє з останнього файла прочитати будь-який символ і вивести його на екран дисплею.

```
domains                                                    /*pr_30. pro*/
  file = file_N; file_R
predicates
  factorial ( real, real )
  position
goal
  openread ( file_N, "number.txt" ) ,  readdevice ( file_N ) ,
    readint ( N ) ,
    closefile ( file_N ) ,
      factorial ( N, F ) ,
      openwrite ( file_R, "res.txt" ) ,  writedevice ( file_R ) ,
    write ( F ) , closefile ( file_R ) ,
    position.
clauses
  factorial ( 0, 1 ).              /*гранична умова*/
  factorial ( N, F ) :-            /*рекурсивне правило*/
    N1 = N -1,
    N1 >= 0,
    factorial ( N1, F1 ) ,
    F = N * F1.
  position:-
    readdevice ( keyboard ) , nl,
      write ( "Введіть номер позиції:  " ) ,
      readreal ( X ) ,  openread ( file_R, "res.txt" ) ,
      readdevice ( file_R ) ,
        filepos ( file_R, X, 0 ) ,
        readchar ( Y ) ,
      closefile ( file_R ) ,
      write ( "Записаний символ -  ", Y ) ,
      position.
```

Коментар. До початку роботи програми в робочій директорії було створено файли "res.txt" та"num.txt", до останнього було записано певне число. При виконанні команди Run компілятора автоматично погоджується запит, що міститься в секції goal. Предикатом readint/1 зчитується відповідне значення з файла "num.txt" і передається змінній N. Результат обчислення (N!) - число, яке виводиться предикатом write(F) не на екран, а у файл "res.txt". Програма вико-

ристовує предикат factorial/2, у якого перший аргумент число, факторіал якого обчислюється, а другий - значення факторіалу числа. Гранична умова: факторіал числа 0 рівний 1. Рекурсивне правило: щоб обчислити факторіал F числа N, необхідно обчислити факторіал F1 попереднього числа N1 і результат помножити на N. Правило position зв'язує вхідний потік з клавіатурою (keyboard) звідки приймає число, введене користувачем, відкриває для читання файл res.txt, зв'язує з ним вхідний потік, зчитує з нього у вказаній позиції символ та виводить його на екран. Ця процедура циклічно повторюється доти, доки користувачем не буде введено хибний номер позиції.

Доступ до дискової операційної системи

Турбо-Пролог має ряд стандартних предикатів для доступу в ДОС. Серед інших, часто використовуються такі предикати:

system (КомандниРядокDos) - викликає виконання в MS-DOS команди, яка задається рядком КомандниРядокDos. Рядок, що задається, повинен бути допустимою командою DOS. Дозволяються вбудовані і зовнішні команди DOS, такі, як файли з розширенням ".BAT". Предикат не виконується, якщо команда некоректна для DOS.

КомандниРядокDos - тип string.
Прототип: (i).

Приклади:

1) system ("cls") - проводиться очищення екрану;
2) system ("dir A:") - виводиться каталог логічного пристрою A;

dir (ШляхДоступу, СпецификаціяФайла, Ім'яФайла) - показує імена файлів в активному вікні за їх специфікаціями та шляхами доступу. Користувач за допомогою клавіш управління курсором вибирає одне з імен файлів, що містяться у каталозі, і, натиснувши на клавішу Enter, зв'язує це ім'я з третім параметром, після чого вибраний файл маркується. Предикат не виконується, якщо користувач натиснув клавішу Esc.

ШляхДоступу, СпецификаціяФайла, Ім'яФайла - тип string.
Прототип: (i, i, o).

Приклади:

1) dir ("C:", "*. pro", X) - будуть показані всі файли на логічному пристрої C, що мають розширення PRO. Ім'я, яке вибере користувач, буде зв'язане зі змінною X.
2) dir ("A:", "W*.*", X) - будуть показані всі файли на логічному пристрої A, імена яких починаються з символа W. Ім'я, яке вибере користувач, буде зв'язане зі змінною X.

comline (БуферРядка) - читає параметри командного рядка, які задані у команді виклику скомпільованої програми на Турбо-Пролозі.

БуферРядка - тип string.
Прототип: (o).

Приклад:

TEST abc (виклик в MS-DOS) - якщо у програмі з іменем TEST є команда comline (X) , то рядок abc буде зв'язано зі змінною X.

8.4. Зовнішня база даних

На відміну від внутрішньої, зовнішня база даних (ЗБД) може створюватися, використовуватися та зберігатися незалежно від фактів (що складають внутрішню базу даних) певної програми на Турбо-Пролозі. При цьому розміщується зовнішня БД або у пам'яті (in_memory) , або у розширеній пам'яті (in_ems) , або у файлі (in_file) , на що вказує тип даних place. Записи зовнішньої бази даних оголошуються у секції domains як тип даних dbasedom.

Стандартні предикати для роботи з зовнішньою[1] БД:

db_create (Ім'я_СелекторБД, Ім'я_Файла, Місце) - створити базу даних.
Ім'я_СелекторБД - тип db_selector.
Ім'я_Файла - тип string.
Місце - тип place.
Прототип: (i, i, i).

db_open (Ім'я_СелекторБД, Ім'я_Файла, Місце) - відкрити базу даних.
Ім'я_СелекторБД - тип db_selector.
Ім'я_Файла - тип string.
Місце - тип place.
Прототип: (i, i, i).

db_openinvalid (Ім'я_СелекторБД, Ім'я_Файла, Місце) - помилка відкриття бази даних.
Ім'я_СелекторБД - тип db_selector.
Ім'я_Файла - тип string.
Місце - тип place.
Прототип: (i, i, i).

db_close (Ім'я_СелекторБД) - закрити базу даних.
Ім'я_СелекторБД - тип db_selector.
Прототип: (i).

db_delete (Ім'я_Файла, Місце) - знищити базу даних.
Ім'я_Файла - тип string.
Місце - тип place.
Прототип: (i, i).

db_flush (Ім'я_СелекторБД) - запис буферів бази даних.
Ім'я_СелекторБД - тип db_selector.
Прототип: (i).

[1] До предикатів для роботи із зовнішньою БД відносяться і раніше розглянуті предикати для роботи з бінарними деревами.

db_garbagecollect (Ім'я_СелекторБД) - сміттєзбірник бази даних.
Ім'я_СелекторБД - тип db_selector.
Прототип: (i).

db_copy (Ім'я_СелекторБД, Ім'я_Файла, Місце) - копіювати базу даних.
Ім'я_СелекторБД - тип db_selector.
Ім'я_Файла - тип string.
Місце - тип place.
Прототип: (i, i, i).

db_chains (Ім'я_СелекторБД, Ланцюг) - ланцюжки бази даних.
Ім'я_СелекторБД - тип db_selector.
Ланцюг - тип string
Прототип: (i, o).

db_btrees (Ім'я_СелекторБД, Ім'я_дерева) - бінарні дерева БД.
Ім'я_СелекторБД - тип db_selector.
Ім'я_дерева - тип string.
Прототип: (i, o).

db_statistics (Ім'я_СелекторБД, ЧислоЗаписів, РозмірВпам'яті, РозмірБД, ВільнеМісце) -статистика БД.
Ім'я_СелекторБД - тип db_selector.
ЧислоЗаписів, РозмірВпам'яті, РозмірБД, ВільнеМісце - тип real.
Прототип: (i, o, o, o, o).

chain_inserta (Ім'я_СелекторБД, Ланцюг, Об'єкт, Запис, Посилання) - помістити в ланцюжок перед.
Ім'я_СелекторБД - тип db_selector.
Ланцюг - тип string.
Об'єкт - тип symbol.
Запис - тип, визначений в domains.
Посилання - тип ref.
Прототип: (i, i, i, i, o).

chain_insertz (Ім'я_СелекторБД, Ланцюг, Об'єкт, Запис, Посилання) - помістити в ланцюжок після.
Ім'я_СелекторБД - тип db_selector.
Ланцюг - тип string.
Об'єкт - тип symbol.
Запис - тип, визначений в domains.
Посилання - тип ref.
Прототип: (i, i, i, i, o).

chain_insertafter (Ім'я_СелекторБД, Ланцюг, Об'єкт, Посилання, Запис, НовеПосилання) - помістити в ланцюжок після.
Ім'я_СелекторБД - тип db_selector.
Ланцюг - тип string.
Об'єкт - тип symbol.
Посилання - тип ref.
Запис - тип, визначений в domains.
НовеПосилання - тип ref.

Прототип: (i, i, i, i, o).

chain_delete (Ім'я_СелекторБД, Ланцюг) - вилучити з ланцюжка.
Ім'я_СелекторБД - тип db_selector.
Ланцюг - тип string.
Прототип: (i, i).

chain_terms (Ім'я_СелекторБД, Ланцюг, Об'єкт, Запис, Посилання) - записи ланцюжка.
Ім'я_СелекторБД - тип db_selector.
Ланцюг - тип string.
Об'єкт - тип symbol.
Запис - тип, визначений в domains.
Посилання - тип ref.
Прототип: (i, i, i, _, o).

chain_first (Ім'я_СелекторБД, Ланцюг, ПершеПосилання) - перший член ланцюжка.
Ім'я_СелекторБД - тип db_selector.
Ланцюг - тип string.
ПершеПосилання - тип ref.
Прототип: (i, i, o).

chain_last (Ім'я_СелекторБД, Ланцюг, ОстаннєПосилання) - останній член ланцюжка.
Ім'я_СелекторБД - тип db_selector.
Ланцюг - тип string.
ОстаннєПосилання - тип ref.
Прототип: (i, i, o).

chain_next (Ім'я_СелекторБД, Посилання, НаступнеПосилання) - наступний член ланцюжка.
Ім'я_СелекторБД - тип db_selector.
Посилання - тип ref.
НаступнеПосилання - тип ref.
Прототип: (i, i, o).

chain_prev (Ім'я_СелекторБД, Посилання, ПопереднєПосилання) - попередній член ланцюжка.
Ім'я_СелекторБД - тип db_selector.
Посилання - тип ref.
ПопереднєПосилання - тип ref.
Прототип: (i, i, o).

term_delete (Ім'я_СелекторБД, Ланцюг, Посилання) - вилучити запис з ланцюжка.
Ім'я_СелекторБД - тип db_selector.
Ланцюг - тип string.
Посилання - тип ref.
Прототип: (i, i, i).

term_replace (Ім'я_СелекторБД, Об'єкт, Посилання, НовийЗапис) - замінити запис у ланцюжку.
Ім'я_СелекторБД - тип db_selector.

Об'єкт - тип symbol.
Посилання - тип ref.
НовийЗапис - тип, визначений в domains.
Прототип: (i, i, i, i).

ref_term (Ім'я_СелекторБД, Об'єкт, Посилання, Запис) - відшукати запис у ланцюжку.
Ім'я_СелекторБД - тип db_selector.
Об'єкт - тип symbol.
Посилання - тип ref.
Запис - тип, визначений в domains.
Прототип: (i, i, i, _).

Приклад 31. Запишемо програму, що перетворює внутрішню базу даних попередньої програми (факти мае/2) у зовнішню базу даних, що міститься у файлі "have.dbp".

```
include "pr_18. pro"                                    /*pr_31. pro*/
domains
  db_selector=dbhave
  dbdom=m(учень, р)
predicates
  перетворити_ВБД_у_ЗБД
  записати
goal
  перетворити_ВБД_у_ЗБД.
clauses
  перетворити_ВБД_у_ЗБД:-                               /*(1)*/
    db_create(dbhave, "have.dbp", in_file),
      записати,
      db_close(dbhave).
  записати :-                                           /*(2)*/
    мае(Хто, Що),
    chain_insertz(dbhave, мае, dbdom, m(Хто, Що), _), fail.
    записати.                                           /*(3)*/
```

Коментар. Програма за правилом (1) створює зовнішню базу даних dbhave у файлі "have.dbp", записує (записати) відповідні факти у створену БД та закриває базу даних. При цьому правило (2) переглядає всі факти мае/2 внутрішньої бази даних попередньої програми, доступ до яких виконано опцією include "pr_18. pro". Після зчитування кожного окремого факту мае/2 стандартний предикат chain_insertz записує у ЗБД відповідний факт у вигляді предикату m/2 (оголошеного як об'єкт dbdom) і робота правила (2) завершується помилкою (fail), що спонукає механізм виведення Турбо-Прологу до повторного погодження предикату мае/2 і повторного запису нового факту у ЗБД. Фраза (3) завжди успішно погоджується, що дозволяє успішно завершити виконання правила (1) і закрити створену БД.

Приклад 32. Маючи новостворену зовнішню базу даних (dbhave) запишемо програму, яка розв'язує задачу так, як її розв'язує наведена програма для роботи з внутрішньою базою даних (Приклад 18. на стор. 54)

```
domains                          /*pr_32. pro*/
  учень,  p = symbol
   db_selector = dbhave
  dbdom = m (учень, p)

predicates
     має(учень, p)
     користується(учень, p)
     старш(учень, учень)
     обмин(учень, p, учень, p)
     купити(учень, p)

clauses
     старш(хома, леся).
     старш(ольга, юра).
  має(Хто, Що):-
       db_open(dbhave, "have.dbp", in_file),
       chain_terms(dbhave, має, dbdom, _, Посилання),
       ref_term(dbhave, dbdom, Посилання, m(Хто, Що)).
  користується(SD, R):-
       має(SD, R).
  користується(D, R):-
       старш(SD, D),
       db_close(dbhave),
       має(SD, R).
  купити(D, R):-
       bound(D), bound(R),
         not(має(D, R)),
              chain_insertz(dbhave, має, dbdom, m(D, R), _),
         db_close(dbhave).
  обмин(D1, R1, D2, R2):-
       bound(D1), bound(R1),  bound(D2), bound(R2),
       має(D1, R1),
         db_close(dbhave),
       має(D2, R2),
         chain_terms(dbhave, має, dbdom, _, Посилання1),
              ref_term(dbhave, dbdom, Посилання1, m(D1, R1)),
                term_delete(dbhave, має, Посилання1),
                chain_insertz(dbhave, має, dbdom, m(D1, R2), Посиланння1),
         chain_terms(dbhave, має, dbdom, _, Посилання2),
              ref_term(dbhave, dbdom, Посилання2, m(D2, R2)),
                term_delete(dbhave, має, Посилання2),
                chain_insertz(dbhave, має, dbdom, m(D2, R1), Посиланння2),
         db_close(dbhave).
```

Коментар. У програмі правило mae/2 відкриває існуючу ЗБД, проглядає (chain_terms) всі посилання записів (використовуючи механізм повернення) ланцюжка mae (причому ці записи є об'єктами типу dbdom) та знаходить (ref_term) за визначеним посиланням потрібні записи. Всі інші правила програми реалізують процедури, аналогічні правилам програми 18. Відмінність полягає лише у використанні стандартних предикатів. Для раніше створеної програми 18 це предикати для роботи з ВБД, а для програми даного прикладу – предикати для роботи з ЗБД.

Предикат db_open буде відкривати існуючу базу даних, якщо перед цим вона була закрита (предикат db_close). Однак можливо, що існуюча БД незалежно від користувача може бути помічена компілятором як помилкова і доступ до неї за допомогою стандартних предикатів буде неможливий. У цьому разі необхідно скопіювати ЗБД у нову або ще раз створити таку ж (db_create) та повторно наповнити її початковими даними (для розглядуваної програми такою операцією буде повторний запуск попередньої програми (Приклад 31.)).

8.5. Робота з текстом та використання систем граматичного розбору

Раніше розглянутий предикат upper_lower (стор. 64) дозволяє виконувати перетворення тільки з англійськими символами та з рядками таких символів.

Приклад 33. Програма, яка процедурою upper_lower_sym розширює можливості стандартного предикату upper_lower і дозволяє перетворювати окремі російські символи (проте не перетворює рядки символів).

```
domains                                                    /*ULS. pro*/
    симв=char
  predicates
    upper_lower_sym (симв,симв)
    control (симв,симв)
  clauses
    upper_lower_sym(X,Y):-
      free(X), bound(Y), char_int(Y,YI),
            YI>=160,YI<=175,YR=YI-32,char_int(X,YR), ! ;
      free(X),bound(Y), char_int(Y,YI),
            YI>=224,YI<=239,YR=YI-80,char_int(X,YR),! ;
      free(Y), bound(X), char_int(X,XI),
            XI>=128,XI<=143,XR=XI+32,char_int(Y,XR),!;
      free(Y), bound(X), char_int(X,XI),
            XI>=144,XI<=159,XR=XI+80,char_int(Y,XR),!;
      bound(X), bound(Y), control(X,Y), !.
```

```
upper_lower_sym (X, Y):-
  upper_lower (X, Y).
control (X, Y) :-
  char_int(X,XI), char_int(Y,YI), XI>=128, XI<=143, YI=XI+32;
  char_int(X,XI), char_int(Y,YI), XI>=144, XI<=159, YI=XI+80;
  char_int(X,XI), char_int(Y,YI), XI>=160, XI<=175, YI=XI-32;
  char_int(X,XI), char_int(Y,YI), XI>=224, XI<=239, YI=XI-80.
```

Граматика безпосередніх складових

Приклад 34. Програма, що забезпечує граматичний розбір для БС-граматики. Використаємо необхідні факти та правила (див. фрагмент БС-граматики на стор. 66).

```
include "ULS. pro"                                    /*pr_34. pro*/
include "pr_22. pro"
predicates
  чи_речення (rechen)
  речення (rechen, sp)
  именна_група (sp, sp)
  диесл_група (sp, sp)
  именник (sp, sp)
  прикмет (sp, sp)
  диеслово (sp, sp)
clauses
  чи_речення (Rech):-
      frontchar (Rech, L, O),
        upper_lower_sym (L, L1),
            frontchar (R, L1, O),
                речення (R, [ ]).
  речення (R, ROst):-
    lexanaliz (R, RLex),
      именна_група (RLex, DG), диесл_група (DG, ROst), !.
  именна_група (R, Залишок):-                          /* (1)*/
    прикмет (R, R1),  именник (R1, Залишок).
  именна_група (R, Залишок):-                          /* (2)*/
    именник (R, Залишок).
  диесл_група (DG, ROst):-
    диеслово (DG, D1), именна_група (D1, ROst).
  диесл_група (DG, ROst):-
    диеслово (DG, ROst).
  прикмет ([H|T], T) : -      H="дана".                 /* (3)*/
  прикмет ([H|T], T) : -      H="рівнобедрений".
  именник ([H|T], T) : -      H="пряма".
  именник ([H|T], T) : -      H="трикутник".
  диеслово ([H|T], T) : -     H="перетинае".
```

Коментар. Директиви компілятора include "ULS. pro" та include "pr_22. pro" забезпечують виконання у даній програмі предикатів lexanaliz/2, upper_lower_sym/2. Програма не містить секції domains, так як у описі предикатів (секція predicates) всі області зміни аргументів вказані як ті, що використовуються у програмах, підключених до виконання двома директивами include. У правилі чи_речення/1 перший символ речення, що вводиться, переводиться до нижнього регістру та формується нове речення R, що має модифікований перший символ речення; новосформований рядок R лексем предикатом речення/2 перевіряється на можливість розбиття на іменну та дієслівну групу так, щоб у ці групи повністю, у потрібному порядку і коректно ввійшли всі (другий аргумент - порожній список) лексеми рядка. Предикат именна_група/2 за правилом (1) намагається рядок лексем, що передається у перший аргумент, погодити як пару "прикметник-іменник" (за правилом (2) - як "іменник"), передаючи при цьому залишок рядка у другий аргумент. У свою чергу, предикат прикметник/2 успішно погоджується за правилом (3), якщо список лексем, що передається у перший аргумент має головою лексему, яка співпадає з наперед заданою (H="дана").

Граматика, що визначається твердженнями

Нагадаємо, що граматика, що будує дерево розбору, називається граматикою, що визначається твердженнями.

Приклад 35. Модифікуємо програму (Приклад 34.), що реалізує БС-граматику, так, щоб вона будувала дерево розбору. Для цього доповнимо кожне граматичне правило ще одним аргументом.

```
include "ULS. pro"                                    /*pr_35. pro*/
include "pr_22. pro"
domains
  слово,  дерево=symbol
predicates
  речення(rechen, дерево)
  речення(rechen, sp, дерево)
  именна_група(sp, sp, дерево)
  диесл_група(sp, sp, дерево)
  именник(слово, sp, sp, дерево)
  прикмет(слово, sp, sp, дерево)
  диеслово(слово, sp, sp, дерево)
  formuv(symbol, symbol, symbol, symbol)
  formuv(symbol, symbol, symbol)
clauses
  речення(Rech, Дерево_реч):-
      frontchar(Rech, Перший_символ, Остача_речення),
    upper_lower_sym(Перший_символ, Символ_нижн_регістру),
      frontchar(R, Символ_нижн_регістру, Остача_речення),
```

```
        речення(R, [], Дерево_реч).
речення(R, ROst, Дерево_реч):-
  lexanaliz(R, RLex),
  именна_група(RLex, DG, Дерево_им_гр),
  диесл_група(DG, ROst, Дерево_дсл_гр),
        formuv("реч", Дерево_им_гр, Дерево_дсл_гр, Дерево_реч), !.
именна_група(R, DG, Дерево_им_гр):-
  прикмет(_, R, Остача_реч, Прикм),
  именник(_, Остача_реч, DG, Імен),
        formuv("им_гр", Прикм, Імен, Дерево_им_гр).
именна_група(R, DG, Дерево_им_гр):-
  именник(_, R, DG, Імен),
        formuv("им_гр", Імен, Дерево_им_гр).
диесл_група(DG, ROst, Дерево_дсл_гр):-
  диеслово(_, DG, Остача_речення, Диесл),
  именна_група(Остача_речення, ROst, Дерево_им_гр),
        formuv(" дсл_гр", Диесл, Дерево_им_гр, Дерево_дсл_гр).
диесл_група(DG, ROst, Дерево_дсл_групи):-
  диеслово(_, DG, ROst, Диесл),
        formuv(" дсл_гр", Диесл, Дерево_дсл_групи).
formuv(ГрПодв, Частина1, Частина2, Rez):-
     concat(Частина1, ", ", X), concat(X, Частина2, Y),
     concat(ГрПодв, "(", R), concat(R, Y, RR), concat(RR, ")", Rez).
formuv(ГрОдин, Частина, Rez):-
     concat(ГрОдин, "(", R),
     concat(R, Частина, RR),
     concat(RR, ")", Rez).
прикмет(дана, [H|T], T, "пркм<дана>"):-
     H="дана".
прикмет(рівнобедрений, [H|T], T, "пркм<рівнобедрений>"):-
     H="рівнобедрений".
именник(пряма, [H|T], T, "им<пряма>"):-
     H="пряма".
именник(трикутник, [H|T], T, "им<трикутник>"):-
     H="трикутник".
диеслово(перетинати, [H|T], T, "дсл<перетинати>"):-
     H="перетинае".
```

Коментар. Кожний предикат-правило, який визначає групи, доповнено ще одним аргументом, що являє собою рядок символів і формується додатково введеними процедурами formuv/4 (для груп, що складаються з двох частин) та formuv/3 (для груп з однієї складової). Крім того, збільшується число аргументів у предикатах прикмет, именник, диеслово: першим аргументом є атом, що закріплює початковий стан певної частини мови ключовим словом (для іменника - у називному відмінку однини, для дієслова - у неозначеній формі і т.д.). Однак у програмі значення цього аргументу не використовується - атом вводиться для зручності візуального контролю

користувача за базою даних слів мови при її розширенні. Другий та третій аргументи зберігають такі ж значення, як у програмі, що модифікується. Четверті аргументи - рядки символів, що описують слова з бази даних та з яких конструюються дерева відповідних груп, а потім і дерево розбору речення.

Розглянемо приклади запитів до програми:

Запит	Відповідь
речення("Дана пряма перетинае", X)	X=**реч**(**им_гр**(прикм<дана>,им<пряма>), **дсл_гр**(дсл<перетинае>))
речення("Дана пряма перетинае трикутник", X)	X=**реч**(**им_гр**(прикм<дана>,им<пряма>), **дсл_гр**(**дсл**<перетинае>, **им_гр** (им<трикутник>)))
речення("Дана пряма перетинае рівнобедрений трикутник", X)	X=**реч**(**им_гр**(прикм<дана>,им<пряма>), **дсл_гр**(**дсл**<перетинае>, **им_гр** (прикм<рівнобедрений>,им<трикутник>)))

Приклад 36. Для реалізації у граматиці, що визначається твердженнями, однієї з властивостей природної мови – погодження за родом, числом та відмінком (стор. 67), модифікуємо попередню програму, ввівши ще один аргумент, який буде зберігати контекстну інформацію.

```
include "ULS. pro"                                    /*pr_36. pro*/
include "pr_22. pro"
predicates
  речення(rechen,rechen)
  речення(rechen,sp,rechen)
  именна_група(sp,sp,rechen,sp)
  диесл_група(sp,sp,rechen,sp)
  именник(symbol,sp,sp,rechen,sp)
  прикмет(symbol,sp,sp,rechen,sp)
  диеслово(symbol,sp,sp,rechen,sp)
  formuv(rechen,rechen,rechen,rechen)
  formuv(rechen,rechen,rechen)
clauses
  речення(Rech,Дерево_реч):-
      frontchar(Rech,Перший_символ,Остача_речення),
      upper_lower_sym(Перший_символ,Символ_нижн_регістру),
          frontchar(R,Символ_нижн_регістру,Остача_речення),
              речення(R,[],Дерево_реч).
  речення(R,ROst,Дерево_реч):-
      lexanaliz(R,RLex),
      именна_група(RLex,DG,Дерево_им_гр,Форма_групи),
      диесл_група(DG,ROst,Дерево_дсл_гр,Форма_групи),
      formuv("реч",Дерево_им_гр,Дерево_дсл_гр,Дерево_реч),!.
  именна_група(R,DG,Дерево_им_гр,Форма_групи):-
      прикмет(_,R,Залишок_реч,Прикм,[Рид,Число,Видминок]),
      именник(_,Залишок_реч,DG,Імен,[Рид,Число,Видминок]),
```

```
        Форма_групи=[третя,Рид,Число],
            formuv("им_гр",Прикм,Імен,Дерево_им_гр).
именна_група(R,DG,Дерево_им_гр,Форма_групи):-
    именник(_,R,DG,Імен,[Рид,Число,Видминок]),
        Форма_групи=[третя,Рид,Число],
            formuv("им_гр",Імен,Дерево_им_гр).
диесл_група(DG,ROst,Дерево_дсл_гр,Форма_групи):-
    диеслово(_,DG,Залишок_реч,Диесл,[Особа,Рид,Число]),
      именна_група(Залишок_реч,ROst,Дерево_им_гр,_),
            Форма_групи=[Особа,Рид,Число],
            formuv(" дсл_гр",Диесл,Дерево_им_гр,Дерево_дсл_гр).
диесл_група(DG,ROst,Дерево_дсл_гр,Форма_групи):-
    диеслово(_,DG,ROst,Диесл,[Особа,Рид,Число]),
      Форма_групи=[Особа,Рид,Число],
            formuv(" дсл_гр",Диесл,Дерево_дсл_гр).
formuv(ГрПодв,Частина1,Частина2,Rez):-
    concat(Частина1,",",X),concat(X,Частина2,Y),
    concat(ГрПодв,"(",R),concat(R,Y,RR),concat(RR,")",Rez).
formuv(ГрОдинарна,Частина,Rez):-
    concat(ГрОдинарна,"(",R),
    concat(R,Частина,RR),concat(RR,")",Rez).
прикмет(дана,[Н|Т],Т,"пркм<дана>",Форма):-
    Форма=[жн,однина,називний],                    Н="дана";
    Форма=[чол,однина,називний],                   Н="даний";
    Форма=[сер,однина,називний],                   Н="дане";
    Форма=[чол,множина,називний],                  Н="дані";
    Форма=[жн,множина,називний],                   Н="дані";
    Форма=[сер,множина,називний],                  Н="дані".
прикмет(рівнобедрений,[Н|Т],Т,"пркм<рівнобедрений>",Форма):-
    Форма=[жн,однина,називний],                    Н="рівнобедрена";
    Форма=[чол,однина,називний],                   Н="рівнобедрений";
    Форма=[жн,множина,називний],                   Н="рівнобедрені";
    Форма=[чол,множина,називний],                  Н="рівнобедрені".
именник(пряма,[Н|Т],Т,"им<пряма>",Форма):-
    Форма=[жн,однина,називний],                    Н="пряма";
    Форма=[жн,множина,називний],                   Н="прямі".
именник(трикутник,[Н|Т],Т,"им<трикутник>",Форма):-
    Форма=[чол,однина,називний],                   Н="трикутник";
    Форма=[чол,множина,називний],                  Н="трикутники".
диеслово(перетинати,[Н|Т],Т,"дсл<перетинати>",Форма):-
    Форма=[третя,чол,однина],                      Н="перетинає";
    Форма=[третя,жн,однина],                       Н="перетинає";
    Форма=[третя,сер,однина],                      Н="перетинає";
    Форма=[третя,чол,множина],                     Н="перетинають";
    Форма=[третя,жн,множина],                      Н="перетинають";
    Форма=[третя,сер,множина],                     Н="перетинають".
```

Коментар. Предикати правил, що визначають групи, доповнено аргументом Форма_групи. Цей аргумент містить контекстну інформацію, подану як список елементів, що визначають форму слова як частини мови. Відповідно, кожне слово з бази даних подано за допомогою правила, що містить про слово конкретну інформацію (особа, рід, число тощо), яка використовується для перевірки контекстної залежності частин речення.

Приклади запитів до програми:

Запит	Відповідь
речення("Дана пряма перетинае", X)	X=**реч**(**им_гр**(прикм<дана>,им<пряма>), **дсл_гр**(дсл<перетинае>))
речення("Дана пряма перетинають", _)	No
речення("Дані прямі перетинають", _)	Yes
речення("Дана прямі перетинають", _)	No
речення("Дані прямі перетинають рівнобедрені трикутники",_)	Yes

8.6. Графіка

Для виводу графічних зображень у Турбо-Пролозі існують такі стандартні предикати[1]:

arc(X, Y, StAngle, EndAngle, Radius) - креслення дуги кола.
X, Y, StAngle, EndAngle, Radius - тип integer.
Прототип: (i, i, i, i, i).

bar(Left, Top, Right, Bottom) - креслення прямокутника.
Left, Top, Right, Bottom - тип integer.
Прототип: (i, i, i, i).

bar3d(Left, Top, Right, Bottom, Depth, Topflag) - креслення 3-вимірного стовпчика для побудови діаграм.
Left, Top, Right, Bottom, Depth, Topflag - тип integer.
Прототип: (i, i, i, i, i, i).

circle(X, Y, Radius) - креслення кола.
X, Y, Radius - тип integer.
Прототип: (i, i, i).

cleardevice - очистка графічного екрана.

clearviewport - очистка поточного відеопорту.

closegraph - припинення роботи графічної системи.

detectgraph(Graphdriver, Graphmode) - визначення типу драйвера дисплея і режиму його роботи шляхом перевірки апаратних засобів ЕОМ, що використовуються.
Graphdriver, Graphmode - тип integer.
Прототип: (o, o).

drawpoly(PolyPointsList) - креслення многокутника (аргумент - список точок-вершин).

[1] Предикати, що розглядаються призначені для роботи з графічним інтерфейсом фірми Borland International (BGI).

PolyPointsList - тип bgi_ilist.
Прототип: (i).

ellipse(X, Y, StAngle, EndAngle, Xradius, YRadius) - креслення дуги еліпса.
X, Y, StAngle, EndAngle, Xradius, YRadius - тип integer.
Прототип: (i, i, i, i, i, i).

fillellipse(X, Y, Xradius, YRadius) - креслення еліпса і зафарбування його з заданою текстурою і кольором заповнення.
X, Y, Xradius, YRadius - тип integer.
Прототип: (i, i, i, i).

fillpoly(PolyPointsList) - креслення многокутника і зафарбування його з заданою текстурою і кольором заповнення.
PolyPointsList - тип bgi_ilist.
Прототип: (i).

floodfill(X, Y, Border) - зафарбування обмеженої області з заданою текстурою і кольором заповнення.
X, Y, Border - тип integer.
Прототип: (i, i, i).

getarccoords(X, Y, Xstart, Ystart, Xend, Yend) - видача координат, використаних при останньому виклику предикату.
X, Y, Xstart, Ystart, Xend, Yend - тип integer.
Прототип: (o, o, o, o, o, o).

getaspectratio(Xasp, Yasp) - виведення поточного масштабного множника (по X та по Y), що використовується для недеформованого виведення графічних зображень.
Xasp, Yasp - тип integer.
Прототип: (o, o).

getbkcolor(BkColor) - видача поточного кольору фону.
BkColor - тип integer.
Прототип: (o).

getcolor(Color) - видача поточного кольору, яким виконується креслення зображень.
Color - тип integer.
Прототип: (o).

getdrivername(DriverName) - видача імені поточного графічного драйвера.
DriverName - тип string.
Прототип: (o).

getdefaultpalette(DefaultPalette) - видача поточної кольорової палітри.
DefaultPalette - тип bgi_ilist.
Прототип: (o).

getfillpattern(PatternList) - видача текстури зафарбування зображення, яка задається користувачем.
PatternList - тип bgi_ilist.
Прототип: (o).

getfillsettings(FillPattern, FillColor) - видача інформації про поточні колір та текстуру зображення.
FillPattern, FillColor - тип integer.
Прототип: (o, o).

getgraphmode(GraphMode) - видача поточного графічного режиму.
GraphMode - тип integer.
Прототип: (o).

getimage(Left, Top, Right, Bottom, BitMap) - запам'ятовування бітового зображення заданої області екрана.
Left, Top, Right, Bottom - тип integer.
BitMap - тип string.
Прототип: (i, i, i, i, o).

getlinesettings(LineStyle, Upattern, Thickness) - видача поточної товщини та стилю лінії, які використовуються для креслення.
LineStyle, Upattern, Thickness - тип integer.
Прототип: (o, o, o).

getmaxcolor(MaxColor) - видача максимально допустимого значення коду кольору пікселя.
MaxColor - тип integer.
Прототип: (o).

getmaxx(X) - видача максимально допустимої координати екрана по горизонталі.
X - тип integer.
Прототип: (o).

getmaxy(Y) - видача максимально допустимої координати екрана по вертикалі.
Y - тип integer.
Прототип: (o).

getmaxmode(MaxMode) - видача максимально допустимого значення коду графічного режиму.
MaxMode - тип integer.
Прототип: (o).

getmodename(DriverMode, ModeName) - видача назви графічного режиму.
DriverMode - тип integer.
ModeName - тип string.
Прототип: (i, o).

getmoderange(Graphdriver, Lomode, Himode) - видача діапазонів кодів режимів для заданого графічного драйвера.
Graphdriver, Lomode, Himode - тип integer.
Прототип: (i, o, o).

getpalette(PaletteList) -видача інформації про поточну кольорову палітру.
PaletteList - тип bgi_ilist.
Прототип: (o).

getpalettesize(PaletteSize) - видача розміру таблиці перегляду кольорової палітри.
PaletteSize - тип integer.

Прототип: (o).

getpixel(X, Y, Color) - видача кольору заданого пікселя.
X, Y, Color - тип integer.
Прототип: (i, i, o).

gettextsettings(Font, Direction, CharSize, Horiz, Vert) - видача інформації про поточний графічний літерний шрифт.
Font, Direction, CharSize, Horiz, Vert - тип integer.
Прототип: (o, o, o, o, o).

getviewsettings(Left, Top, Right, Bottom, Clip) - видача інформації про поточний відеопорт.
Left, Top, Right, Bottom, Clip - тип integer.
Прототип: (o, o, o, o, o).

getx(X) - видача координати X поточного положення курсора.
X - тип integer.
Прототип: (o).

gety(Y) - видача координати Y поточного положення курсора.
Y - тип integer.
Прототип: (o).

graphdefaults - присвоєння всім графічним параметрам значень за замовчуванням.

imagesize(Left, Top, Right, Bottom, Size) - видача об'єму пам'яті у байтах, необхідного для запам'ятовування даного побітного зображення.
Left, Top, Right, Bottom, Size - тип integer.
Прототип: (i, i, i, i, o).

initgraph(Graphdriver, Graphmode, NewDr, NewMode, Pathtodriver) - ініціалізація графічної системи.
Graphdriver, Graphmode, NewDr, NewMode - тип integer.
Pathtodriver - тип string.
Прототип: (i, i, o, o, i).

line(X0, Y0, X1, Y1) - креслення лінії, що сполучає дві дані точки.
X0, Y0, X1, Y1- тип integer.
Прототип: (i, i, i, i).

linerel(Dx, Dy) - креслення лінії від поточної точки до точки, для якої задається відносне зміщення.
Dx, Dy - тип integer.
Прототип: (i, i).

lineto(X, Y) - креслення лінії від поточної точки до даної.
X, Y - тип integer.
Прототип: (i, i).

moverel(Dx, Dy) - переміщення поточної точки на відстань, що дається у відносних координатах.
Dx, Dy - тип integer.
Прототип: (i, i).

moveto(X, Y) - переміщення з поточної точки у дану.
X, Y - тип integer.
Прототип: (i, i).

outtext(Textstring) - виведення рядка у відеопорт.
Textstring - тип string.
Прототип: (i).

outtextxy(X, Y, Textstring) - виведення рядка у заданому місці екрана.
X, Y - тип integer
Textstring - тип string.
Прототип: (i, i, i).

pieslice(X, Y, Stangle, Endangle, Radius) - креслення і зафарбування кругової секторної діаграми.
X, Y, Stangle, Endangle, Radius - тип integer.
Прототип: (i, i, i, i, i).

putimage(X, Y, Bitmap, Op) - виведення побітного зображення на екран.
X, Y, Op - тип integer
Bitmap - тип string.
Прототип: (i, i, i, i).

putpixel(X, Y, Pixelcolor) - виведення пікселя у заданій точці.
X, Y, Pixelcolor - тип integer.
Прототип: (i, i, i).

rectangle(Left, Top, Right, Bottom) - креслення прямокутника.
Left, Top, Right, Bottom - тип integer.
Прототип: (i, i, i, i).

restorecrtmode - відновлення екранного режиму, що існував до виклику предикату initgraph.

setactivepage(Page) - задання активної сторінки для виведення графіки.
Page - тип integer.
Прототип: (i).

setallpalette(PaletteList) - задання кольорів палітри.
PaletteList - тип bgi_ilist.
Прототип: (i).

setaspectratio(Xasp, Yasp) - зміна прийнятого за замовчуванням коефіцієнта корекції, що використовується для виведення недоформованого зображення на екран.
Xasp, Yasp - тип integer.
Прототип: (i, i).

setbkcolor(Color) - задання кольору фону.
Color - тип integer.
Прототип: (i).

setcolor(Color) - задання поточного кольору зображення, що виводиться.
Color - тип integer.
Прототип: (i).

setfillpattern(UpatternList, Color) - вибір користувачем текстури заповнення зображення.
UpatternList - тип bgi_ilist
Color - тип integer.
Прототип: (i, i).

setfillstyle(Pattern, Color)- задання кольору і текстури заповнення зображення.
Pattern, Color - тип integer.
Прототип: (i, i).

setgraphmode(Mode) - задання активного графічного режиму.
Mode - тип integer.
Прототип: (i).

setgraphbufsize(BufSize) - задання розміру буферу для графічних зображень.
BufSize - тип integer.
Прототип: (i).

setlinestyle(Linestyle, Upattern, Thickness) - задання товщини та стилю лінії, що використовується для креслення графічних зображень.
Linestyle, Upattern, Thickness - тип integer.
Прототип: (i, i, i).

setpalette(Index, Actual_color) - зміна одного з кольорів палітри.
Index, Actual_color - тип integer.
Прототип: (i, i).

settextjustify(Horiz, Vert) - задання виду вирівнювання тексту.
Horiz, Vert - тип integer.
Прототип: (i, i).

settextstyle(Font, Direction, Charsize) - задання поточних характеристик тексту.
Font, Direction, Charsize - тип integer.
Прототип: (i, i, i).

setusercharsize(Multx, Divx, Multy, Divy) - збільшення (зменшення) розмірів шрифтів.
Multx, Divx, Multy, Divy - тип integer.
Прототип: (i, i, i, i).

setviewport(Left, Top, Right, Bottom, Clip) - задання параметрів поточного відеопорту для графічного виведення.
Left, Top, Right, Bottom, Clip - тип integer.
Прототип: (i, i, i, i, i).

setvisualpage(Pagenum) - задання номера графічної сторінки, що виводиться на екран.
Pagenum - тип integer.
Прототип: (i).

setwritemode(WriteMode) - задання режиму виводу для креслення ліній.
WriteMode - тип integer.
Прототип: (i).

textheight(Textstring, Height) - видача висоти рядка у пікселях.
Textstring - тип string.

Height - тип integer.
Прототип: (i, o).

textwidth(Textstring, Width) - видача ширини рядка у пікселях.
Textstring - тип string.
Width - тип integer.
Прототип: (i, o).

Наведемо приклади програм, що виводять графічні зображення.

Приклад 37. Програма, що виводить на екран найпростіші геометричні зображення (фігури) з різними кольорами малюнків, кольорами заповнення малюнку та текстурами.

```
predicates                                                    /*pr_37. pro*/
  граф_фгр
goal
  граф_фгр.
clauses
  граф_фгр:-                                                  /*(1)*/
    initgraph(5, 1,  _,  _, "c:\prolog") , /*вказується шлях доступу до файла
                                                  egavga.bgi (або cga.bgi)*/
    setbkcolor(15),
      setfillstyle(2, 6),
        fillellipse(100, 100, 70, 56),                        /*еліпс*/
      setcolor(3),
        circle(300, 200, 150),                                /*коло*/
      setfillstyle(8, 3),
      setcolor(3),
        fillpoly([450, 200,                                   /*многокутник*/
            375, 100, 225, 100,
              150, 200,
              225, 300, 375, 300]),
    setfillstyle(4, 9),
      setcolor(9),
        pieslice(100, 100, 0, 30, 70),                        /*сектор*/
      setfillstyle(1, 13),
        bar(595, 100, 480, 300),                              /*прямокутник*/
    outtextxy(300, 50, "Геометричні фігури"),  fail.
  граф_фгр:-                                                  /*(2)*/
    readchar (_),  closegraph.
```

Коментар. При виконанні запиту граф_фгр програма погоджує спочатку правило (1), за яким виводяться на екран графічні зображення фігур, а далі за правилом (2) програма очікує введення з клавіатури довільного символа, після чого робота графічної системи припиняється.

Приклад 38. Програма, що виводить на екран зображення прямокутної декартової системи координат з відповідними позначеннями та графік функції $y=sin\ x$ на відрізку від $[-\pi\ ;\ \pi]$.

```
predicates                                                    /*pr_38. pro*/
  graf_sin
  повтор(integer, integer, integer)
  повтор
  точка(integer, integer, symbol)
  умова_виходу(char)
goal
  graf_sin.
clauses
  graf_sin:-                                                  /*(1)*/
    initgraph(5, 1, _, _, ""),
    setbkcolor(8),
    getmaxx(X1), getmaxy(Y1),
    X=X1/2, Y=Y1/2,
    setcolor(1), line(0, Y, X1, Y), line(X, 0, X, Y1),
    setcolor(14),
      точка(X, Y, " O"),
      точка(480, Y, "П/2"),   точка(634, Y, "П"),
      точка(160, Y, "-П/2"),  точка(6, Y, "-П"),
      точка(X, 105, " 1"),   точка(X, 245, " -1"),
      outtextxy(632, 160, "X"),
      outtextxy(330, 6, "Y"),
        повтор(N, -10, X1),                    /*повтор значень N*/
        A=N+X-314, B=(70*sin(N/100))+Y,
        putpixel(A, B, 5),
      fail.
  graf_sin:-                                                  /*(2)*/
    повтор,                                  /*правило повтору*/
    readchar(X),
    умова_виходу(X), closegraph, write("Вихід з графіки."), !.
  точка(X, Y, Назва):-
    circle(X, Y, 1), Q=X-4, W=Y+5, outtextxy(Q, W, Назва).
  повтор(N, N, _).                    /*правило повтору значень N*/
  повтор(N, A, B):-
        B>A, A1=A+1, повтор(N, A1, B).
  умова_виходу('0').
  умова_виходу(_):-fail.
    повтор.                                                   /*повтор*/
    повтор:-повтор.
```

Коментар. Погодження запиту у секції goal розпочинається погодженням правила (1), за яким предикат initgraph встановлює режим екрана 640x350 пікселів для адаптера VGA; предикати getmaxx, getmaxy визначають максимальні значення координат екрана по горизонталі та вертикалі, які використовуються для задання координат

при виводі зображень та тексту на екран[1]; графічні зображення на координатних осях точок та їх імена виводяться на екран за допомогою предикату точка/3, що описується відповідним правилом; предикат повтор/3 разом з предикатом fail забезпечує циклічне обчислення значення N (послідовно від початкової координати X екрана до кінцевої) та циклічне виконання предикату putpixel, за яким на екрані по пікселям (з координатами A,B) будується графік функції $y=sin\ x$. Отже, правило (1) повністю формує потрібне зображення на екрані. За правилом (2) програма переходить у режим очікування вводу символу з клавіатури і виконує повтори (див. приклад на стор.39), зберігаючи при цьому графічне зображення доти, доки користувач не введе з клавіатури символ 0. Після чого робота графічної системи припиняється (closegraph) та на екран предикатом write виводиться відповідне повідомлення.

Таблиця 1. Режими для графічних команд.

Режим	Колонок (пікс.)	Рядків (пікс.)	Результат
1	320	200	Середня розділяюча здатність, 4 кольори
2	640	200	Велика розділяюча здатність, 1 колір
3	320	200	Середня розділяюча здатність, 16 кольорів
4	640	200	Велика розділяюча здатність, 16 кольорів
5	640	350	Найвища розділяюча здатність, 16 кольорів

Таблиця 2. Колір фону і зображення в режимах 3, 4, 5 (EGA,VGA).

Значення	Колір	Значення	Колір
0	Чорний	8	Сірий
1	Синій	9	Блакитний
2	Зелений	10	Світло-зелений
3	Бірюзовий	11	Світло-бірюзовий
4	Червоний	12	Світло-червоний
5	Ліловий	13	Рожевий
6	Коричневий	14	Жовтий
7	Білий	15	Білий підвищеної інтенс.

[1] Деякі предикати для побудови графічних зображень аргументами мають конкретні числа. Замість таких чисел-координат можна записати змінні, значення яких повністю буде залежати від значень X1 та Y1, які видають предикати getmaxx, getmaxy. У такому разі програма коректно працює і для інших режимів екрана та графічних адаптерів, що задаються предикатом initgraph. Але це приведе до збільшення об'єму програми і ускладнить орієнтацію читача у системі координат екрана.

8.7. Робота з екраном

Для роботи з екраном використовуються стандартні предикати:

scr_char(Рядок, Колонка, Символ) - виведення/повернення символу на екран/з екрана.

Рядок, Колонка, Символ - тип integer.
Прототип: (i, i, i), (i, i, o).

scr_attr(Рядок, Колонка, Атрибут) - встановлює/повертає атрибут символу. Номера рядків змінюються від 0 до 24, колонок - від 0 до 79.

Рядок, Колонка, Атрибут - тип integer.
Прототип: (i, i, i), (i, i, o).

field_str(Рядок, Колонка, Довжина, РядокСимволів) - запис/читання рядка символів.

Рядок, Колонка, Довжина - тип integer.

РядокСимволів - тип string.
Прототип: (i, i, i, i), (i, i, i, o).

field_attr(Рядок, Колонка, Довжина, Атрибут) - встановити атрибут поля.

Рядок, Колонка, Довжина, Атрибут - тип integer.
Прототип: (i, i, i, i), (i, i, i, o).

cursor(Рядок, Колонка) - встановити (повернути позицію курсора) курсор в позицію Рядок, Колонка.

Рядок, Колонка - тип integer.
Прототип: (i, i), (o, o).

cursorform(ПочатокРядка, ПочатокКолонки) - Визначення/повернення висоти і вертикальної координати курсора всередині області, що займається під один символ. Аргументи приймають значення від 1 до 13.

ПочатокРядка, ПочатокКолонки - тип integer.
Прототип: (i, i), (o, o).

attribute(Атрибут) - встановлює/повертає значення атрибуту, що визначає колір фону та символів екрана (Таблиця 3).

Атрибут - тип integer.
Прототип: (i), (o).

textmode(Рядок, Колонка) - встановлює/повертає текстовий режим екрана.

Рядок, Колонка - тип integer.
Прототип: (i, i), (o, o).

snowcheck(Перемикач) - включає/виключає перевірку на "сніг". Перемикач приймає одне зі значень on/off.

Перемикач - тип string.
Прототип: (i), (o).

Таблиці значень атрибутів для кольорового графічного адаптера:

Таблиця 3. Колір фону.

Атрибут	Колір	Атрибут	Колір
0	Чорний	72	Світло-червоний
8	Сірий	80	Ліловий
16	Синій	88	Рожевий
24	Блакитний	96	Коричневий
32	Зелений	104	Жовтий
40	Світло-зелений	112	Білий
48	Бірюзовий	120	Білий підвищеної
64	Червоний		інтесивності

Таблиця 4. Колір символу.

Атрибут	Колір	Атрибут	Колір
0	Чорний	4	Червоний
1	Синій	5	Ліловий
2	Зелений	6	Коричневий
3	Бірюзовий	7	Білий

Зауваження 7.

Для визначення числового значення аргументу Атрибут необхідно додати значення атрибутів кольору фону та символу. Додавання до 128 викликає мерехтіння зображення.

8.8. Робота з вікнами

Для введення даних у Пролог-програму та виведення результатів її роботи можуть використовуватися, серед інших, діалогові вікна, що створюються та функціонують за допомогою стандартних предикатів Турбо-Прологу. Візуалізація на екран та робота з такими вікнами можлива в пакетному режимі роботи програм. Наведемо перелік указаних предикатів:

makewindow(НомерВікна, АтрибутЕкрана, АтрибутРамки, РядокЗаголовка, Ряд, Колонка, Висота, Ширина) - на екрані створюється нове вікно, якщо аргументи вхідні (константи) або видається значення параметрів поточного вікна, якщо аргументи вихідні (змінні). Параметри рамки(фрейму) вибираються за замовчуванням.

Прототип: (i, i, i, i, i, i, i, i), (o, o, o, o, o, o, o, o)

makewindow(НомерВікна, АтрибутЕкрана, АтрибутРамки, РядокЗаголовка, Ряд, Колонка, Висота, Ширина, ОчисткаВікна, ЦентрувЗаголовка, СимволиБордюру) -

аналогічно makewindow/8, але необхідно задати 3 додаткових параметри рамки.

НомерВікна, АтрибутЕкрана, АтрибутРамки, Ряд, Колонка, Висота, Ширина, ОчисткаВікна, ЦентрувЗаголовка- тип integer.

РядокЗаголовка, СимволиБордюру - тип string.

Прототип: (i, i, i, i, i, i, i, i, i, i), (o, o, o, o, o, o, o, o, o, o)

Додаткова інформація:

1. ОчисткаВікна =0 - не очищати вікно після його створення.

   ОчисткаВікна =1 - очищати вікно після створення.

2. ЦентрувЗаголовку = 255 - центрувати заголовок вікна.

   ЦентрувЗаголовку ≠ 255 - заголовок вікна помістити у задану позицію.

3. АтрибутРамки ≠ 0 - рамка вікна встановлена.

4. СимволиБордюру - це рядок довжиною 6 символів. Символами формується бордюр вікна: 1-ий символ - верхній лівий кут, 2-ий символ - верхній правий кут, 3-ій символ - нижній лівий кут, 4-ий символ - нижній правий кут, 5-ий символ - горизонтальна лінія, 6-ий символ - вертикальна лінія.

shiftwindow(НомерВікна) - перехід до нового активного вікна (аргумент вхідний), при цьому запам'ятовується зміст поточного активного вікна; якщо аргумент вихідний, то через нього повертається (видається) номер поточного вікна.

НомерВікна - тип integer.

Прототип: (i), (o).

gotowindow(НомерВікна) - швидкий перехід від одного вікна до іншого. Зміст старого вікна не запам'ятовується.

НомерВікна - тип integer.

Прототип: (i)

resizewindow - виклик стандартної[1] процедури для зміни розмірів вікна.

resizewindow(ПочРядка, ЧислоРядків, ПочКолонки, ЧислоКолонок)- зміна розміра, місцезнаходження або разом того і іншого для поточного вікна.

ПочРядка/ПочКолонки - тип integer. Новий початковий рядок/колонка.

ЧислоРядків/ЧислоКолонок - тип integer. Розмір вікна у рядках/колонках.

Прототип: (i, i, i, i).

colorsetup(Код) - інтерактивна зміна кольорів у поточному вікні.

Код - тип integer. Якщо Код=0, то змінюється колір вікна; якщо Код=1, то змінюється колір фрейма.

Прототип: (i).

---

[1] як і для вікон системи Турбо-Пролог, виконавши підпункт "*Window size*" пункту "*Setup*" головного меню.

existwindow(НомерВікна) - перевіряється, чи існує вікно із номером НомерВікна. Якщо таке вікно не існує, то предикат не погоджується.

НомерВікна - тип integer.
Прототип: (i).

removewindow - поточне вікно вилучається з екрана, а його місце займають вікна, що знаходились під ним.

removewindow(НомерВікна, Режим) - вилучити вікно з екрана. Якщо Режим=0, то не оновлюється зміст вікон, що містились під даним вікном; якщо ж Режим=1, то зміст вікон оновлюється.

НомерВікна, Режим - тип integer.
Прототип: (i, i).

clearwindow - очистити поточне текстове вікно, заповнивши його фоновим кольором. Курсор встановлюється у точку з координатами (0, 0).

window_str(РядокЕкрана) - читання/запис рядка з активного вікна/у вікно. Якщо аргумент вихідний, то у рядкову змінну зчитується текст, відображений в активному вікні, тому у цій змінній буде стільки ж рядків тексту, скільки і в активному вікні. Якщо аргумент вхідний, то в активне вікно виводиться зміст РядокЕкрана; при цьому, якщо у РядокЕкрана міститься більше рядків тексту, ніж рядків у вікні, то у вікно буде виведено стільки символів, скільки може поміститися у вікні.

РядокЕкрана - тип string.
Прототип: (i), (o).

window_attr(Атрибут) - аргумент Атрибут задає атрибути поточного вікна.

Атрибут - тип integer.
Прототип: (i).

scroll(ЧислоРядків, ЧислоКолонок) - зміст поточного вікна буде "прокручено" (зміщено) на ЧислоРядків і ЧислоКолонок. Позитивне значення першого аргумента означає зміщення вверх, негативне - вниз. Для другого аргумента відповідно вліво і вправо.

ЧислоРядків, ЧислоКолонок - тип integer.
Прототип: (i, i).

framewindow(АтрибутРамки) - зміна атрибутів рамки вікна.

АтрибутРамки - тип integer.
Прототип: (i).

framewindow(АтрибутРамки, ЗаголовокВікна, ПозицЗаголовкаВікна, СимвБордюру) - зміна атрибутів і символів рамки вікна. ПозицЗаголовкаВікна (приймає значення від 0 до ширини вікна) вказує, де розмістити загаловок вікна. СимвБордюру - рядок довжиною 6 символів, які вказують, як зобразити рамку (див. makewindow).

АтрибутРамки, ПозицЗаголовкаВікна - тип integer.
ЗаголовокВікна, СимвБордюру- тип string.
Прототип: (i, i, i, i).

Наведемо приклад програми, якак для організації діалогу з користувачем використовує вікна.

Приклад 39. Програма, що забезпечує знаходження коренів квадратного рівняння за коефіцієнтами, що вводяться користувачем з клавіатури.

```
predicates                                              /*pr_39. pro*/
  розв_кв_рв
  дані (real, real, real)
  відповідь (real, real, real)
  корені (real, real, real)
  повтор
  помилка (real)
  перевірка_дій
  дія (integer)
goal
  розв_кв_рв.
clauses
  розв_кв_рв :-
    clearwindow,
    makewindow (1, 30, 30, " Квадратне рівняння. ", 4, 26, 10, 42),
      дані (A, B, C),
        відповідь (A, B, C),
            readchar (_).
  дані (A, B, C) :-
    повтор, clearwindow,
    nl, write (" Введіть значення коефіцієнта A: "),
    readreal (A),
    помилка (A),
      nl, write (" Введіть значення коефіцієнта B: "), readreal (B),
      nl, write (" Введіть значення коефіцієнта C: "), readreal (C).
  помилка (A) :- A=0,
      makewindow (2, 78, 78, " Увага! ", 15, 16, 5, 62),  gotowindow (2),
        write (" Введіть правильний коефіцієнт A!"), nl,
        write ("<Enter>- повторити ввід, <Esc>-завершити програму:"),
      перевірка_дій.
  помилка (A) :- A<>0.
  перевірка_дій :-
      readchar (Ch), char_int (Ch, X),
      дія (X).
  дія (X) :- X=13, !, removewindow (), gotowindow (1), fail.
  дія (X) :- X=27, exit.
  дія (_) :- перевірка_дій.
```

```
відповідь (A, B, C) :-
    makewindow (3, 78, 78, " Відповідь. ", 15, 26, 5, 42),
    gotowindow (3), nl,
      D = - 4*A*C+B*B,
      корені (A, B, D).
  корені (A, B, D) :- D>0,                                  /* (1)*/
    X1= (-B+sqrt (D))/ (2*A),
    X2= (-B-sqrt (D))/ (2*A),
      write (" Два дійсних корені:"), nl, nl,
      write (" X1= ", X1), nl, write (" X2= ", X2).
  корені (A, B, D) :- D=0,                                  /* (2)*/
    X= (-B+sqrt (D))/ (2*A),
      write ("Два дійсних корені, що співпадають:"), nl, nl,
      write ("X1=X2= ", X).
  корені (_, _, D) :- D<0,                                  /* (3)*/
      write (" Дійсних розв'язків немає").
повтор.
повтор :- повтор.
```

Коментар. Виконання програми розпочинається погодженням фраз правила розв_кв_рв, за яким очищається інформація у активному вікні (clearwindow), виводиться зображення вікна1 для вводу даних (makewindow), вводяться коефіцієнти квадратного рівняння (дані/3), формується відповідь (відповідь/3) та очікується введення довільного символу (readchar) для завершення виконання правила та роботи програми.

Правило дані/3 забезпечує зчитування значень коефіцієнтів з клавіатури та перевірку на помилку (помилка/1) при введенні коефіцієнта A, рівного 0; у такому разі предикат помилка/1 генерує появу вікна2 (заголовок "Увага!") з відповідним повідомленням. Фразою перевірка_дій аналізується те, які клавіші натискає користувач: правило дія/1 при натисканні клавіші Enter (код символу 13) вилучає вікно2 з екрана (removewindow), активізує вікно1 (gotowindow(1)) та завершує виконання фрази невдачею (fail), що дає змогу виконати відкіт до предикату повтор, який спонукає до повторного виконання введення значень коефіцієнтів; якщо ж користувач натиснув клавішу Esc (код 27), то виконання програми завершується (exit); у випадку натискання інших клавіш предикат дія/1 повертає виконання програми на очікування вводу з клавіатури нової команди доти, доки користувач не натисне Enter або Esc.

Правило відповідь генерує зображення вікна3, активізує його та фразою корені/3 обчислює корені квадратного рівняння (правила 1,2,3), про що виводяться відповідні повідомлення у активне вікно (вікно3).

8.9. Звуковий супровід

Для звукового супроводу програм, що написані на Турбо-Пролозі, використовуються такі стандартні предикати:

beep - звуковий сигнал. При виконанні у програмі за цим предикатом буде подано звуковий сигнал.

sound (Тривалість, Частота) - виконується звуковий сигнал заданої тривалості та частоти. Час звучання задається у сотих долях секунди.

Тривалість, Частота - тип integer.
Прототип: (i, i).

Таблиця 5.
Значення частот (у Гц) для предикату sound.

Нота		Частота
C	Нижнє До	131
Cis	До#	139
D	Ре	147
Dis	Ре#	156
E	Мі	165
F	Фа	175
Fis	Фа#	185

Нота		Частота
G	Соль	196
Gis	Соль#	208
A	Ля	220
Ais	Ля#	233
B	Сі	247
C	Верхнє До	262

Наведемо приклад програми, яка використовує предикат sound для подання звукових сигналів.

Приклад 40. Програма, яка після введення символу з клавіатури генерує подачу звукового сигналу певної частоти з тривалістю 1 секунда.

```
predicates                                              /*pr_40. pro*/
  сигнал
  звук (char)
  повтор
goal
  сигнал.
clauses
  сигнал:-
    повтор,
    makewindow (1, 14, 7, "*** ЗВУКОВІ СИГНАЛИ ***",  0,  0,  25,  80),
      write ("Введіть символ з клавіатури
                        (e, a, d, g, або h для верхнього e)\n",
                        "і програма подасть звуковий сигнал"), nl,
      readchar (Символ),
      звук (Символ).
  звук ('e'):- sound (100,  165).
  звук ('a'):- sound (100,  220).
```

```
звук ('d'):- sound (100,  294).
звук ('g'):- sound (100,  392).
звук ('b'):- sound (100,  494).
звук ('h'):- sound (100,  660).
звук ( _ ):-
     write ("\n\nПомилковий символ\n",
     "Натисніть будь-яку клавішу для продовження.\n"),
        beep,
        readchar (_),
     clearwindow, fail.
повтор.
повтор:-повтор.
```

Коментар. У процесі погодження запиту сигнал на екран виводиться вікно, що має заголовок "Звукові сигнали". У вікно предикатом write виводиться відповідна інформація та виконання програми переводиться у режим вводу символа з клавіатури (readchar). Предикатом звук/1 аналізується символ, що вводиться: якщо він такий, що міститься у базі даних, то програма генерує звуковий сигнал (sound) певної тривалості та частоти і програма завершує роботу. У випадку набору символа, відмінного від вказаних, на екран виводиться повідомлення про введення помилкового символу, подається звуковий сигнал (beep), очікується нажаття довільної клавіші та, у разі нажаття, очищається активне вікно і предикатом fail здійснюється відкіт до предикату повтор, який викликає повторний вивід зображення вікна і виконання процедур по введенню символа для генерації звукового сигналу. Очевидно, що програма циклічно працює доти, доки не буде введено правильний символ.

8.10. Модульне програмування

Використання технології модульного програмування дозволяє розбити складну програму на сукупність більш простих модулів, що взаємодіють між собою. Такі модулі можуть бути написані, відредаговані та скомпільовані окремо один від одного. Для цього у Турбо-Пролозі використовуються засоби створення проектів (project) і глобальні оголошення (global). Якщо програма складатиметься з декількох модулів, то необхідно підготувати файл, який містить список їх імен.

Цей файл має такий формат:

```
ім'я файлу з першим модулем
ім'я файлу з другим модулем
...
ім'я файлу з n-ним модулем
```

Вказаний файл (і інші подібні файли) повинен мати розширення prj. Він створюється за допомогою за допомогою підпункту *Edit PRJ file* в підменю *Options*. Поряд із цим, ім'я проекту необхідно записати

у кожен модуль за допомогою відповідної директиви компілятора, наприклад:

```
project "inform".
```

За замовчуванням, всі об'єкти, що використовуються в модулі, є локальними. Взаємодія декількох модулів здійснюється через спільні (глобальні) області, які оголошуються за допомогою частини (ключового слова) global відповідних директив. Вони повинні бути відомі всім модулям, що взаємодіють.

Найбільш просто це досягається шляхом створення файла глобальних описів і включення його у всі модулі за допомогою директиви компілятора include.

Оголошення глобальних предикатів (global predicates) має особливості і виконується за схемою:

ім'я_предикату (x_1,..., x_n) - (послідовність_1) (послідовність_2)...

де x_1,..., x_n - аргументи предикату (об'єкти); послідовність_1, послідовність_2,... - послідовності символів *i* (значення об'єкту відомо на вході (вхідний параметр)) та *o* (значення об'єкту відомо на виході (вихідний параметр)) між якими ставиться символ "кома", і при цьому кількість символів *i* та *o* у послідовності рівна кількості аргументів предикату.

Кожна зі вказаних послідовностей задає окремий шаблон для аргументів предикату, а сукупність таких шаблонів утворює *прототип* вказаного предикату.

Наприклад, для предикату upper_lower_sym (стор. 98) для оголошення його як глобального необхідно записати:

```
upper_lower_sym (X, Y) - (i, o), (o, i), (i, i)
```

Якщо глобальні описи помістити у файл globals. pro, а файл проекту має ім'я inform.prj, то кожен модуль проекту обов'язково повинен містити такі дві директиви:

```
project "inform"
include "globals. pro"
```

Секція goal може міститися тільки в одному (головному) модулі.

Приклад 41. Розробимо конкретний проект inform.prj з декількома модулями: програму, що включає модулі main. pro і fun. pro. Проектом передбачається використання двох модулів для того, щоб з клавіатури ввести і зчитати ім'я користувача, а потім вивести на екран повідомлення, що містить введене ім'я

Для реалізації проекту необхідно виконати такі дії:

1. Створити файл проекту inform.prj, у який як текст записати імена файлів-модулів:

```
main  fun
```

2. Створити файл globals. pro глобальних оголошень. Файл глобальних оголошень має бути записаний та збережений як Пролог-програма без секцій goal та clauses у вигляді:

```
global domains                                    /*globals. pro*/
    name = string
global predicates
    modul (name) - ( l )
```

3. Записати та зберегти тексти програм головного (main. pro) та додаткового (fun. pro) модулів проекту.

3.1. Головний модуль проекту: програма, яка зчитує з клавіатури ім'я користувача та для виведення повідомлення з введеним іменем використовує додатковий модуль.

```
project "inform"                                  /*main. pro*/
include "globals. pro"
predicates
  information
goal
  information.
clauses
  information:-
    system ("cls"),
    write ("Як Вас звати?"), nl,
    readln (Name),
    modul (Name).
```

Коментар. Програма main. pro є головним модулем, а виведення повідомлення на екран має забезпечується модулем fun. pro. Правило information предикатом system("cls") очищає екран, предикатом write формулює питання, зчитує з клавіатури (readln) ім'я, введене користувачем, та передає його як аргумент у предикат modul для виконання.

3.2. Додатковий модуль проекту: програма fun. pro, що виводить на екран інформацію з іменем, введеним під час виконання модуля main. pro.

```
project "inform"                                  /*fun. pro*/
include "globals. pro"
clauses
  modul(Name):-
    write ("Добрий день, ", Name, "!"), nl, nl,
    write ("Натисніть будь-яку клавішу."),
    readchar ( _ ).
```

Коментар. Програма за правилом modul спочатку виводить потрібні повідомлення (write), очікує натискання користувачем будь-якої клавіші, після чого завершує роботу.

4. Згенерувати файли main.obj, fun.obj (перед тим, як генерувати вказані файли, необхідно мати файли модулів main. pro, fun. pro).

5. Створити виконувану програму inform.exe, виконавши опцію *Project* підменю *Compile*.

Коментар. Після запуску inform.exe на виконання на екран виведеться питання: "Як Вас звати?". Якщо вказати ім'я, наприклад, Олег, то програма дасть відповідь: "Добрий день, Олег!", і через рядок з'явиться: "Натисніть будь-яку клавішу". Після виконання вказаних дій програма-проект припинить свою роботу.

ШТУЧНИЙ ІНТЕЛЕКТ

§1. ПОНЯТТЯ ШТУЧНОГО ІНТЕЛЕКТУ

Під штучним інтелектом (ШІ) розуміють:

- *науковий напрямок*, у рамках якого ставляться і розв'язуються задачі апаратного або програмного моделювання тих видів людської діяльності, які традиційно вважаються інтелектуальними;
- *властивість штучних (інтелектуальних) систем* виконувати функції, які імітують інтелектуальну діяльність людини і традиційно вважаються людською прерогативою.

1.1. Штучний інтелект як наука

Завданням штучного інтелекту як науки є відтворення за допомогою штучних пристроїв (в основному за допомогою ЕОМ) інтелектуальних міркувань та дій. При цьому виникають труднощі двох типів:

- у більшості випадків, виконуючи певні дії, людина не усвідомлює, як вона це робить – їй невідомий точний спосіб виконання розумних міркувань та дій. Наприклад, не існує алгоритмів розуміння тексту, впізнавання обличчя, доведення теореми, розробки плану дій тощо;
- ЕОМ апріорі далекі від людського рівня компетентності: для того, щоб вони почали виконувати роботу, необхідно скласти відповідні програми. Однак у дійсності мови програмування дають можливість виражати поняття, які є досить примітивними порівняно з людськими.

Тому методи штучного інтелекту являють собою *експериментальну наукову дисципліну*. Під експериментом у даному випадку розуміється перевірка та уточнення моделей на чисельних прикладах-спостереженнях над людиною з метою розкрити ці моделі та краще зрозуміти функціонування людського розуму.

Розрізняють два напрями досліджень у галузі ШІ: *нейробіологічний* - імітація структури і функціонування біологічних клітин живого інтелекту; *прагматичний* - відтворення у ШІ тих інформаційних процесів, що відбуваються при розв'язуванні інтелектуальних задач людиною.

Прагматичний напрямок виявився на даний момент більш ефективним, ніж нейробіологічний. Він був спрямований на практику і не робив спроби моделювати функції мозку. Творчі процеси тут намагались відтворити своїм, відмінним від людського, машинним, тобто комп'ютерним способом.

Області використання штучного інтелекту

Задача, для якої невідомий алгоритм розв'язку, апріорно відноситься до штучного інтелекту. Тут під алгоритмом розуміється вся послідовність заданих дій, які чітко визначені, та можуть бути виконані на сучасних ЕОМ, при цьому розв'язок задачі необхідно одержати протягом прийнятного часу (порядку хвилини або години).

До сфери штучного інтелекту відносяться ті досить різноманітні області, для яких характерні такі особливості:

- у вказаних областях інформація використовується у символьній формі: літери, слова, знаки, малюнки;
- передбачається наявність вибору. Дійсно, сказати, що не існує алгоритму, означає, по суті, тільки те, що необхідно зробити вибір між багатьма варіантами в умовах невизначеності, і ця невизначеність є суттєвою складовою інтелекту.

Деякі напрями досліджень з проблем штучного інтелекту:

- автоматичне доведення теорем і логічне програмування;
- робота зі знаннями (одержання знань з різних джерел, набуття знань від професіоналів, подання знань, маніпулювання знаннями, пояснення на основі знань);
- природно-мовна область (машинний переклад, генерація текстів, автоматичне реферування, взаємодія людини з ЕОМ засобами природної мови);
- експертні системи (системи для широкого загалу користувачів, системи для спеціалістів);
- сприйняття та розпізнання образів;
- робототехніка (обробка зображень, сенсори, розв'язування завдань у певному середовищі);
- навчання (консультації, інтелектуальний тренінг, системи загальноосвітнього та спеціального навчання);
- ігри (комп'ютерні ігри, людські ігри);
- музика (розробка та аналіз музичних творів);
- графіка та живопис.

На даний час в Україні важливими напрямками досліджень, викликаними проблемами створення, вивчення та розвитку національних інформаційних ресурсів, серед інших, є такі:

- інтеграція інтелектуальної інформації;
- просторове і часове мислення;
- моделювання мислення в інтелектуальних навчаючих системах;
- емпіричний штучний інтелект;
- наукові відкриття за допомогою штучного інтелекту;
- багатозначна логіка у системах штучного інтелекту;
- теорія прийняття рішень і штучний інтелект;
- багатомовні машинні лексикони.

1.2. Інтелектуальні системи

Під інтелектуальною системою (ІС) в практичному плані розуміють, як правило, комп'ютерну програму, здатну "думати" і розв'язувати так звані "творчі задачі".

Для реалізації ШІ у такій системі необхідно попередньо вивчити, як мислять люди, коли їм потрібно прийняти будь-яке рішення або вирішити відповідну проблему. Слід виділити у такому процесі основні стадії, що дозволить потім розробити комп'ютерну програму, здатну розв'язувати різноманітні задачі, використовуючи ті ж стадії процесу мислення. Штучний інтелект, таким чином, забезпечує, щонайменше в принципі, простий структурний підхід до розробки досить складних проблем, що дозволяють розв'язувати творчі задачі різного рівня.

Штучний інтелект імітує основний людський процес навчання, за яким відбувається прийом інформації для подальшого використання. Людський мозок здатен сприймати все нові та нові знання без зміни процесів життєдіяльності і без функціональних порушень різних відділів головного мозку. Система ШІ діє майже у такий спосіб.

Тому *характерною рисою програми штучного інтелекту*, яка ототожнюється з рисою людського інтелекту, є те, що довільна ланка у програмі може бути змінена і модифікована без шкідливого впливу на структуру всієї програми. Елементи програми ототожнюються зі "складовими частинами" людського мозку, тому можна закласти у програму ШІ сам спосіб мислення висококваліфікованого спеціаліста довільної галузі знань. Якщо можна визначити, що "відбувається" в мозку на кожній стадії цього процесу, то можна підшукати еквівалентну ділянку програми, яка функціонально відповідає такій же ділянці у людському мисленні.

Системи ШІ можна поділити на такі групи:

- *інформаційно-пошукові системи*, що працюють в інтерактивному режимі на професійних мовах так званої "ділової прози" користувачів-непрограмістів;
- *розрахунково-логічні системи*, що дозволяють користувачам, які не є спеціалістами у галузі прикладної математики, розв'язувати у діалоговому режимі свої задачі з використанням складних математичних методів і відповідних прикладних програм;
- *системи проектування і наукових досліджень*;
- *навчаючі системи* (системи шкільної та вузівської освіти, інтелектуальні тренажери, консультуючі системи);
- *експертні системи* як для широкого загалу користувачів, так і для спеціалістів, що дозволяють здійснити ефективну комп'ютеризацію тих галузей людського знання, в яких використання математичних моделей, характерних для точних наук, ускладнене, а іноді і неможливе.

Необхідність створення систем штучного інтелекту та попит на відповідні технології на національному та закордонному ринках стрімко росте. Цей процес насамперед пов'язаний з розвитком глобальної інформаційної мережі і підвищенням рівня комп'ютеризації управління всіх сфер людського життя. Тому закономірною є співпраця України у міжнародному проекті "Комп'ютеризація природних мов" (1999р., м. Варна) за участю Болгарії, Росії та Татарстану. У рамках цього проекту проводились ряд міжнародних конференцій "Штучний інтелект" (2000р., 2002р., 2004р. в Україні, 2001р., 2003р. в Росії).

Вказаний проект включає такі проблеми:

1) *введення тексту, переведення його у машинний код*. Задачі, що ставляться у рамках цієї проблеми:

- сканування, переведення у машинний код і розпізнання друкованого, рукописного або іншим способом створеного тексту, а також довільного друкованого зображення, включеного у текст;
- введення, переведення у машинний код, розпізнання усної мови.

2) *основні перетворення комп'ютерного тексту*. Задачі:

- автоматична або автоматизована обробка тексту;
- автоматичний пошук (індексування) тексту;
- автоматичний пошук та одержання знань з комп'ютерного тексту;
- автоматичний переклад з однієї мови на іншу;
- автоматичне перетворення комп'ютерного тексту в усну мовлення;
- синтез тексту за зображенням;
- синтез зображення за текстом;
- синтез тексту за сценарієм.

3) *створення універсальних комп'ютерних словників національних та інших мов у державі*.

Слід зазначити, що найбільш актуальною залишається задача *машинного перекладу з однієї мови на іншу*. Існуючі системи машинного перекладу дозволяють користувачу, спеціалісту у предметній галузі, зрозуміти, про що йдеться у тексті, який перекладається. Але якість перекладу буде неприйнятна без застосування складного апарату аналізу зв'язаного тексту, в якому граматичний розбір та семантичний аналіз – лише одні із багатьох обов'язкових компонентів. Тільки для того, щоб розібратися з семантикою контексту, необхідно використати мільйони підпрограм, ув'язаних зі словниковими статтями вхідної і вихідної мов: такі підпрограми потрібно не тільки написати, а й супроводжувати, так як мови – явище динамічне, і система має враховувати реальність, що змінюється.

Розпізнання усного мовлення, серед іншого, включає машинний переклад як обов'язкову складову частину, а саме: переклад з письмової мови (з багатьох діалектів і вимов) у канонічну, літературну форму, з подальшим аналізом смислу того, що було сказано.

1.3. Програмування штучного інтелекту

Програмування ШІ часто визначається "від супротивного", як все те, що не вкладається у рамки процедурного програмування. Однак більшість програм ШІ мають подібні елементи. Для цих програм характерне те, що *вони мають справу зі складними проблемами, які недостатньо добре розуміються, для яких не існує чітко визначених алгоритмічних розв'язків і які можуть бути досліджені за допомогою того чи іншого механізму символічних міркувань.*

Програми ШІ, як правило, пишуться на мовах типу Лісп або Пролог. Змінні такої програми знаходяться швидше віртуально у стеку комп'ютера, а не у фіксованих місцях у пам'яті. Управління даними здійснюється шляхом порівняння зі зразком і побудови списків. Технологія побудови списків досить проста, і на її основі може бути організована майже будь-яка структура даних. Програми ШІ, крім того, використовують відмінний від традиційних програм набір операторів: *виклики процедури, послідовне виконання, рекурсія.*

Якщо порівнювати мови Лісп та Пролог, то слід зауважити, що Пролог реалізує філософію цільового програмування (програмування у термінах цілей) - так зване програмування типу *"що"*, тоді як Лісп, маючи великі виражальні властивості, дозволяє програмісту описати найбільш "виразно" саме те, *як* що-небудь слід робити, і тому Лісп, будучи мовою функціонального програмування, ще залишається мовою типу "*як"*, до якого слід віднести і процедурні мови. Лісп відображає ту точку зору, що основою більшості інтелектуальних задач є добре організовані перебір та пошук; Пролог переносить акцент на область задач логічного виводу. Пролог пориває з традиціями мов типу "як", через те, що він відповідним чином направляє програмістське мислення, примушуючи програміста давати означення ситуацій і формулювати задачі замість того, щоб у всіх деталях описувати спосіб розв'язку цих задач. Слід зауважити, що переформулювати умову задачі на Пролозі, як і на іншій мові логічного програмування, - процес не тривіальний, при цьому можуть бути допущені помилки (без яких, як відомо, не проходить написання будь-якої програми будь-якою мовою). Знаходження помилок може вимагати багаторазових запусків програми на виконання, тобто налагодження програми. Для полегшення налагодження у Пролозі передбачені спеціальні засоби, які дозволяють програмісту прослідковувати виконання програми крок за кроком. Отже, діяльність з розв'язування складних задач за допомогою логічного програмування вимагає від програміста значних інтелектуальних зусиль.

Логічне програмування - не панацея від усіх бід, а все ж таки саме методика програмування. Інша справа, що зусилля, які прикладаються на розробку програми в рамках логічного програмування, можуть бути в декілька разів меншими, ніж при процедурному програмуванні.

Пролог все більше використовується для програмування ШІ. Але історично склалися два взаємо протилежні погляди на Пролог. Негативна оцінка Прологу зумовлена тим, що, по-перше, на нього наклався негативний досвід роботи з мовою Мікропленер, яка близька до логічного програмування, але реалізована не ефективно; по-друге, існувала "ортодоксальна школа" логічного програмування, прибічники якої наполягали на використанні чистої логіки без позалогічних (у т.ч. процедурних) засобів; по-третє, у США, де проблемами ШІ займалися на високому науково-практичному рівні, який впливав на роботу дослідників інших країн, довгий час Лісп не мав серйозних конкурентів серед мов програмування, що привело до зрозумілої протидії дослідницьких центрів з міцними лісповськими традиціями. У процесі використання Прологу всі вказані негативні фактори з часом нейтралізувалися його очевидними перевагами, суперництво між Лісп ом та Прологом утратило свою гостроту, і в даний час вважається, що оптимальний підхід полягає у поєднанні ідей, на яких ґрунтуються обидві мови.

На даний час існує велика кількість середовищ розробки і діалектів мови логічного програмування Пролог. Однією з перших реалізацій, яка привернула увагу спеціалістів, стала комерційна версія Arity Prolog 6.1. Далі з'явилася мова Delphina Prolog, що працює на Unix-платформі та має високопродуктивний компілятор та інтерпретатор. Для Windows є версія LPA-Prolog як зі стандартним синтаксисом так і з розширеним набором бібліотек для роботи з Windows-інтерфейсом. Turbo Prolog, що розроблявся раніше компанією Borland тепер іменується як PDC Prolog і реалізований для DOS (наприклад, версія 2.0), Windows, OS/2 та Unix. Існує і спеціальна версія для візуальної розробки Visual Prolog, яка, серед іншого, має багато можливостей для Internet-програмування і, починаючи з версії 5.0, включає експертну оболонку ESTA.

Уявлення про те, що процедури логічного виводу у задачах ШІ повинні бути доповнені новою конструкцією, в основі якої лежить об'єкт зі своїми властивостями і ознаками, зумовило появу об'єктно-орієнтованих мов, серед яких найбільш відомою є Смолток. При цьому розв'язування задач подається як маніпулювання поняттями, що узагальнюють об'єкти, які зв'язані з проблемною областю.

Існують програми ШІ, які написані на процедурних мовах. Як метод програмування область програмування ШІ значно відрізняється від традиційного (процедурного) стилю програмування, за яким необхідно докладно повідомляти комп'ютеру, що і як він повинен робити.

§2. ЗНАННЯ У СИСТЕМАХ ШТУЧНОГО ІНТЕЛЕКТУ

2.1. Дані та знання. Робота зі знаннями

Робота зі знаннями покладена в основу сучасного періоду розвитку штучного інтелекту. Кожна предметна область діяльності може бути описана у вигляді сукупності відомостей про структуру цієї області, що базуються на її характеристиках, процесах, які проходять у ній, а також про способи розв'язку задач, що виникають у області. Всі ці відомості утворюють *знання про предметну область.*

Дані - це відомості про стан будь-якого об'єкта, подані у формалізованому вигляді для обробки або вже опрацьовані. Дані можуть бути не тільки числовими (статистичними), а й подані у іншій формі, наприклад, символьній.

На сьогодні немає загальновизнаного формального означення поняття "знання". Як дані, так і знання мають спільні ознаки (внутрішня інтерпретованість, структурованість тощо), але існує цілий ряд ознак, за якими знання відрізняються від даних, і принциповою відмінністю є те, що дані - первинна "пасивна" інформація, що вводиться у інтелектуальну систему, а знання - активні, і протиріччя, які містять знання зумовлюють цю внутрішню активність знань, що спрямована на усунення протиріч.

Знання, на відміну від даних, неможливо просто "перегнати" як інформацію на магнітні носії ЕОМ з довідників, наукових трактатів тощо. Їх може повідомити системі тільки людина. Разом з тим деякі інтелектуальні системи можуть на основі початкових знань, одержаних від спеціалістів, формувати нові знання, людині не відомі.

Головним із завдань в області ІІІ є створення таких систем, які, з одного боку, можуть використовувати велику кількість знань, що передаються їм спеціалістами, а з другого - здатні вести діалог з користувачем та пояснювати свої власні висновки. Це передбачає наявність ефективного управління великої за об'ємом і достатньо структурованої бази знань, чітке розмежування між різними рівнями знань, наявність множини зручних методів подання для правил, схем предикатів і цілком визначений процес обміну інформацією між різними джерелами.

Разом з тим необхідно, щоб система знала, що саме вона знає. Адже, якщо провести аналогію з людиною, то подібне метазнання означає постійне використання протягом усього життя інформації про кожний прожитий день. Яке б не було людське знання, воно вимагає метазнання, і метазнання визначає, яке місце ми відводимо даному знанню серед іншої інформації, як ми відносимося до нього, для яких цілей воно нам корисне, до якого сімейства належить тощо.

Робота зі знаннями

Роботу зі знаннями можна розбити на такі складові:

· *Здобуття знань*[1] *з різних джерел* (документи, статті, книги, фотографії тощо).

> При цьому важливо наперед визначити, які знання важливі і потрібні для ІС; мати методи, які дозволяють перейти від знань у текстовій формі до їх аналогів, придатних для вводу в пам'ять системи; інтегрувати знання з різних джерел у деяку взаємозв'язану і несуперечливу систему знань про предметну область.

· *Одержання знань від професіоналів.* Спеціалісти переважну частину свого професійного досвіду не можуть виразити словесно (такі знання іноді називають професіональним умінням або інтуїцією).

> Щоб одержати такі знання, необхідні спеціальні прийоми та методи. Крім того, знання різних спеціалістів бувають, на перший погляд, несумісними, тому їх необхідно оцінити з точки зору раніше накопичених знань, узгодити між собою та формалізувати для вводу в ІС.

· *Подання знань у пам'яті інтелектуальної системи.* В ІС використовуються чотири основні моделі знань: семантичні мережі, фрейми, логічні системи, продукції.

> На сучасному етапі розвитку ІС проводяться дослідження по змішаному (з точки зору вказаних моделей) поданню знань у системах ІІІ.

· *Маніпулювання знаннями.* Включає у себе процедури поповнення, класифікації, узагальнення знань, процедури виводу на знаннях та міркувань за допомогою знань.

· *Пояснення на основі знань.* ІС повинна мати засоби, які можуть сформулювати користувачу необхідні пояснення (щодо процесу одержання розв'язків, підстав, які були для цього використані, способів відсікання альтернативних розв'язків тощо).

> Так як інтелектуальна система приймає рішення (у т.ч. робить висновки), базуючись на знаннях, які можуть бути невідомі користувачу, який розв'язує свою задачу за допомогою системи, то він може взяти під сумнів правильність одержаного розв'язку.

---

[1] Спеціалісти, які займаються питаннями, пов'язаними зі знаннями, називаються інженерами знань. Іншими словами, *інженер знань* - це спеціаліст, навчений мистецтву переймати знання у експертів та переводити ці знання у форму, яка сприймається комп'ютером.

Типи знань

Можна виділити основні типи знань за наступними ознаками.

Базові елементи, об'єкти реального світу. Вони зв'язані з безпосереднім срийняттям, не вимагають обговорення і добавляються до ба-зи знань у тому вигляді, у якому вони одержані.

Твердження та означення. Вони грунтуються на базових елементах і наперед вважаються достовірними.

Концепції. Являють собою перегрупування або узагальнення базових об'єктів. Для побудови концепцій можуть бути використані різноманітні прийоми (серед яких приклади, контрприклади, частинні випадки тощо).

Відношення. Виражають як елементарні властивості базових елементів, так і відношення між концепціями. Разом з тим, до властивостей відношень відносяться їх більша або менша вірогідність, більший або менший зв'зок з даною ситуацією.

Теореми та правила перезапису. Явна присутність теорем в ІС є головною відмінністю інтелектуальних систем від класичних систем управління базами даних, у яких вони або відсутні або програмуються. Разом з тим теореми не мають ніякої користі без правил їх використання.

Алгоритми розв'язків. Необхідні для виконання певних задач, пов'язаних зі знаннями особливого типу, де елементи необхідної інформації розміщуються у зв'зку один з одним для виконання строго визначеної послідовності дій. Використання чистих алгоритмів обмежене дуже частинними випадками, які, у більшості, мають справу з оброб-кою числової інформації.

Стратегії та евристики. Цей тип являє собою природжені або набуті правила поведінки, які дозволяють у даній конкретній ситуації прийняти рішення про необхідні дії. Інформація використовується у порядку, оберненому до того, за яким вона була одержана.

Метазнання. Представляє собою знання того, що відомо і визначає міру довіри до цього знання, важливість нової елементарної ін-формації по відношенню до існуючої множини знань; організує кожен тип знань і вказує, коли і як вони можуть бути використані.

2.2. Логічні методи подання знань

Логічні методи подання знань - це методи подання знань про предметну область за допомогою фраз: фактів та правил. При цьому кожна фраза еквівалентна аксіомі, а множина фраз еквівалентна теорії, що описує предметну область.

Логічний вивід - це процес, у ході якого встановлюється істинність фраз-наслідків з даної теорії або на основі теорії виводяться певні фрази-наслідки.

При поданні знань Пролог базується на припущеннях про те, що:
- існують сутності (факти з одним елементом);
- існують відношення між сутностями (факти з більш як одним елементом та правила).

Пролог забезпечує логічний вивід на основі операції співставлення двох структур та механізму повернення. Враховуючи те, що пролог-програми реалізують концепцію логічного програмування, теоретичною основою логічного виводу на Пролозі є *правило резолюції* для фразової форми логіки предикатів.

Сила логічних методів у тому, що вони дозволяють подавати об'єкти і відношення між ними у вигляді символів, якими можна легко оперувати за допомогою добре вивчених методів (метод структурної індукції, метод резолюції тощо), здійснюючи тим самим логічні міркування.

2.3. Семантичні мережі

Семантична мережа - модель подання знань, в основі якої лежить ідея про те, що вся необхідна інформація може бути описана як сукупність трійок виду (перший об'єкт, бінарне відношення, другий об'єкт). Ця модель, можливо, найбільш близька до того, як подаються знання у текстах природною мовою.

Подання знань у *семантичній мережі* (мережі сутностей) реалізується *вузлами* та *дугами*. При цьому з вузлами можна асоціювати сутності та класи сутностей (об'єкти, поняття), а з дугами - відношення між сутностями. Дуга, поєднана з одиничним вузлом, визначає властивість цього вузла. Семантична мережа дозволяє виконати вивід за ланцюжками, що описуються певними типами дуг.

Найважливішою концепцією семантичних мереж є *ієрархія*. Кожен рівень знань подається у вигляді вузла, який з'єднано негоризонтальними лініями з вищими та нижчими рівнями.

Графічні схеми семантичних мереж служать зручним способом для зображення бінарних відношень і властивостей.

Приклад 42. Розглянемо приклад семантичної мережі.

У даній мережі вузол людина має властивість мислить. Вузол учень нижчого рівня зв'язаний з вузлом людина вищого рівня, тому можна зробити висновок про те, що вузол учень теж має властивість мислить. Дуга є подає або відношення включення у клас (наприклад, клас учнів включається у клас людей), або відношення належності до класу (петро - це член класу учнів).

Складемо відповідну програму на Турбо-Пролозі.

```
domains                                                    /*pr_42. pro*/
  c = symbol
predicates
  e ( c, c )
  займається ( c, c )
  вчиться_у ( c, c )
  мислить ( c )
clauses
  e ( учень, людина ).
  e ( петро, учень ).
  займається ( петро, спорт ).
  вчиться_у ( учень, школа ).
  вчиться_у ( X, Y ) :-
    e ( X, Z ), вчиться_у  ( Z, Y ).
  мислить ( людина ).
  мислить ( X ) :-
    e ( X, Y ), мислить ( Y ).
```

Коментар. Програма містить факти, що описують значення вузлів певного рівня знань і вказують на ієрархію вузлів: перший аргумент предикату e/2 є іменем вузла, який на один рівень менший, ніж вузол, ім'я якого записується другим аргументом. Інші факти програми (займається, вчиться_у, мислить) першими аргументами відповідних предикатів описують те, які властивості мають вузли; другі аргументи у разі потреби конкретизують ці властивості. Правила описують те, як вузлам нижчого рівня передаються властивості вузлів вищого рівня. Наприклад, за правилом мислить для визначення того, чи має таку властивість певний вузол (для якого у тексті програми такий факт не вказано) переглядаються вузли вищого рівня і вони, у свою чергу, перевіряються на те, чи мають властивість "мислити". Для такої перевірки погоджуються відповідні факти програми або знову використовується правило мислить. Якщо такий вузол вищого рівня є, то робиться висновок про володіння цією властивістю і даним вузлом з нижчого рівня.

2.4. Фрейми

Фрейми ("фрейм" у перекладі з англ. - "рамка") можна розглядати як узагальнення семантичних мереж.

Під фреймами розуміють описання виду Ім'я фрейма (Множина слотів). Кожен слот є пара виду (Ім'я слота. Значення слота).

Допускається, щоб слот (в букв. перекладі – "щілина") сам був фреймом. Тоді у ролі значень слота виступає множина слотів. Для заповнення слотів можуть бути використані константи, змінні, будь-які допустимі вирази обраної моделі знань, посилання на інші слоти та фрейми тощо. Отже, фрейм являє собою гнучку конструкцію, що дозволяє відображувати у пам'яті інтелектуальної системи різноманітні знання.

Найважливіші концепції фреймів - *ієрархія* (як у семантичних мережах) та *успадкування*. Суть успадкування полягає у тому, що якщо значення слота в одному із заданих фреймів не задається, то фрейм повинен успадковувати значення цього слоту із слоту більш високого рівня. Успадкування можна заборонити, якщо ввести значення слоту у конкретний фрейм, при цьому значення слоту, що успадковується буде перекрито.

Існує поняття про *фрейми* як про один із способів подання знань про ситуації. Кожен фрейм має слоти, які задають або тип ситуації, або параметри конкретної ситуації. Якщо подати певний стан знань як деяку ситуацію, то подання знань за допомогою фреймів можна описати так:

- фрейм відповідає стану знань;
- слот відповідає фразі, що відноситься до стану знань;
- успадкування значень слотів від фрейму до фрейму відповідає успадкуванню фраз від стану до стану, і
- можливість локального значення слоту перекривати значення слоту, що успадковується, відповідає можливості спростувати фразу, що успадковується, і замінити її новою фразою.

Приклад 43. Проводиться чемпіонат з шахів у два тури. Визначено час початку проведення всіх змагань з шахів (для кожного туру о 8.00) та місце проведення - шаховий клуб. У першому турі суддями є Хома, Петро, Олег, Юрій, у другому - Олег та Юрій. З деяких причин час проведення другого туру було перенесено з установленого часу початку змагань о 8.00 на новий час - о 16.00."

```
domains                                                    /*pr_43. pro*/
  фрейм, ім_слоту = symbol
  знач_слоту = symbol*
predicates
  породжений (фрейм, фрейм)
```

```
  слот (фрейм, ім_слоту, знач_слоту)
  визначити (фрейм, ім_слоту, знач_слоту)
  спростувати_слот (фрейм, ім_слоту)
clauses
 /*--------------дерево станів-----------------------------------*/
  породжений (шахи, чемпіонат).
  породжений (тур_1, шахи).
  породжений (тур_2, шахи).
 /*------- фрази стану "Змагання з шахів"------------------*/
  слот (шахи, час, [год_8_00]).
  слот (шахи, місце, [клуб]).
 /*------- фрази стану "Змагання з шахів - 1-й тур"-----*/
  слот ( тур_1, судять, [хома, петро, олег, юра] ).
 /*------- фрази стану "Змагання з шахів - 2-й тур"-----*/
  слот (тур_2, судять, [олег, юра] ).
  слот (тур_2, час, [год_16_00] ).
  спростувати_слот (тур_2, час).
 /*------- правила для значень слотів ------------------------*/
  визначити (Фрейм, Слот, Значення) :-
    слот (Фрейм, Слот, Значення).
  визначити (Фрейм, Слот, Значення):-
    породжений (Фрейм, Предок),
    визначити (Предок, Слот, Значення),
    not (спростувати_слот (Фрейм, Слот)).
```

Коментар. Маємо фрейм найвищого рівня - чемпіонат, що породжує фрейм нижчого рівня - шахи, а останній - два фрейми однакового (найнижчого) рівня: тур_1, тур_2. Фрейм шахи має два слоти, значення яких повинне успадковуватися фреймами тур_1, тур_2. Останні фрейми мають додаткові слоти, яких немає у фреймів вищого рівня. Крім того, фрейм тур_2 має локальне значення год_16_00 слоту час, яке перекриває успадковуване значення год_8_00.

За програмою, виконуючи запити, можна отримати відповіді на різноманітні питання, що стосуються як наведених фактів, так і фактів, які не подані у програмі, але дедуктивно виводяться з множини даних фактів. Розглянемо приклади тверджень природною мовою та їх інтерпретацію у запити до наведеної програми.

Твердження природною мовою	Запит до програми
Хто з суддів обслуговує 1-й тур?	визначити (тур_1, судять, X)
Який час початку змагань 1-го туру?	визначити (тур_1, час, X)
Де проводиться 2-й тур змагань?	визначити (тур_2, місце, X)
Який тур змагань з шахів розпочинається о 8.00?	породжений (F, шахи), визначити (F, час, [год_8_00])
Які змагання проводяться у клубі?	визначити (X, місце, [клуб])

2.5. Правила продукцій

"Продукція" або "правило-продукція" являє собою пару "причина-наслідок". У найпростіших випадках продукція схожа на відому логічну зв'язку "імплікація" ("якщо - то"). Продукції, таким чином, являють собою правила, які іноді називають "евристиками". Ці правила закодовані у вигляді тверджень типу "ЯКЩО (виконується конкретна умова), ТО (зроби відповідний висновок або виконай конкретну дію)".

Правила такого виду дозволяють різко скоротити число варіантів можливого пошуку, що має неабияке значення навіть для суперЕОМ.

Математично правило продукції можна подати формулою:

$$P_1A,\ P_2A,\ P_3A,...,\ P_nA \rightarrow A,$$

де P_iA $(i=1,2,3,...,n)$ - умови застосування, А - висновок, який у загальному випадку трактується як *дія* (що суттєво відрізняє такі продукції від імплікацій).

Приклад 44. Наведемо приклад системи продукцій, яка моделює та ілюструє дії людини при посадці в автобус.

Якщо (не(має грошей)), то (відмовитися від посадки).
Якщо (є гроші і не(має автобуса)), то (чекати автобус).
Якщо (є автобус і не(потрібний маршрут автобуса)), то (чекати автобус).
Якщо (є автобус і потрібний маршрут автобуса), то (здійснити посадку).

Складемо програму на Турбо-Пролозі:

```
predicates                                          /*pr_44. pro*/
  дія (symbol)
  питання
  що_робити
database
  умова (symbol)
goal
  що_робити.
clauses
  що_робити:-
    питання, дія (X), write(X), retractall (умова ( _ ) ).
  питання:-
    write ("Чи е гроші?"), nl,
    readln(X), X = "т", assert (умова (е_гроші) ), fail.
  питання:-
    write ("Чи підійшов автобус до зупинки?"), nl,
    readln(X), X= "т", assert (умова (е_автобус) ), fail.
  питання:-
    write("Чи автобус має потрібний маршрут?"), nl,
    readln(X), X = "т", assert (умова (е_маршрут) ).
  питання.
```

```
дія ("Відмовитися від посадки.") :-
    not (умова (е_гроші) ).
дія ("Чекати автобус.") :-
    умова(е_гроші), not (умова (е_автобус) ).
дія ("Чекати автобус.") :-
    умова (е_автобус), not (умова (е_маршрут) ).
дія ("Здійснити посадку.") :-
    умова (е_автобус), умова (е_маршрут).
```

Коментар. Система продукцій представлена процедурами, реалізованими правилами дія/1. Правило що_робити спочатку виконує процес опитування користувача (предикат питання), за яким формулюються питання та, за позитивними відповідями, до внутрішньої бази даних записуються відповідні факти (умова/1), які описують потрібну ситуацію. На основі таких фактів (умов правил продукції) предикат дія визначає потрібну дію, який у вигляді певного повідомлення, зрозумілого користувачу, видається на екран. Для повторного використання програми при аналізі нової ситуації (нових фактів) попередні факти вилучаються з внутрішньої бази даних предикатом retractall. Предикат питання реалізується трьома відповідними правилами та фактом. Ці правила виконуються послідовно зверху вниз (зліва направо) і кожне з них, крім останнього, завершується невдачею (fail), що дає змогу (у разі виконання певного правила і внесення відповідних фактів до ВБД) продовжити процес опитування користувача. Якщо при виконанні правил питання користувач не ввів жодної позитивної відповіді (літера "т"), то для того, щоб предикат питання був успішно погоджений у правилі що_робити, до програми записано факт питання, за яким предикат питання завжди успішно погоджується.

ЛАБОРАТОРНА РОБОТА №7

Подання знань

Мета: Одержати уміння та навички моделювання подання знань при розв'язуванні задач зі штучного інтелекту на Турбо-Пролозі версії 2.0 з використанням правил продукції, семантичних мереж, фреймів.

Теоретична частина: завдання та контрольні питання

І рівень

1. Який зміст поняття "штучний інтелект"?
2. Які задачі відносяться до задач штучного інтелекту?
3. Які області відносяться до сфери штучного інтелекту?
4. Поняття системи штучного інтелекту. Яка риса, що ототожнюється з рисою людського інтелекту, є характерною для програм ШІ?
5. Які мови використовуються для програмування ШІ?

ІІ рівень

1. Поняття про дані та знання. Описати логічні методи подання знань та логічний вивід.
2. Поняття про семантичні мережі та фрейми, їх концепції.
3. Поняття правила-продукції.
4. Які напрями досліджень з проблем ШІ у світовій та вітчизняній науці?
5. Дайте характеристику напрямкам розвитку штучного інтелекту як науки.
6. Які особливості програмування штучного інтелекту?

ІІІ рівень

1. Характеристика, переваги та недоліки використання функціонального, логічного, процедурного програмування до розв'язання інтелектуальних задач. Реалізації мови логічного програмування Пролог.
2. Що розуміється під терміном "знання"? Як відбувається робота зі знаннями?
3. На яких припущеннях базується Пролог при поданні знань логічними методами?
4. У чому суть концепції успадкування для фреймів? Описати поняття про фрейми як про один із способів подання знань про ситуації.
5. Пояснити на прикладах реалізації семантичних мереж, фреймів, правил продукції.
6. Які проблеми з комп'ютеризації природних мов є актуальними в Україні? Дайте коротку характеристику задачам, які ставляться у рамках зазначених проблем.

Практична частина

І рівень

1-13 варіанти.

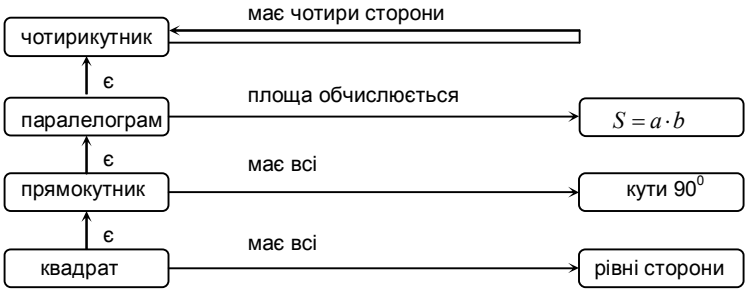
Використовуючи правила продукції, подати знання про відшукання площі трикутника за різними формулами, кожна з яких містить по три елемента метричних даних трикутника. Якщо користувач програми вводитиме метричні дані, що не передбачені у програмі, або їх кількість буде нерівна трьом, то необхідно забезпечити видачу користувачу відповідного повідомлення та повторний ввід таких даних.

Програму зберегти у файлі "LNN7_1a. pro", де NN – номер варіанту користувача. Виконати запити до програми.

ІІ рівень

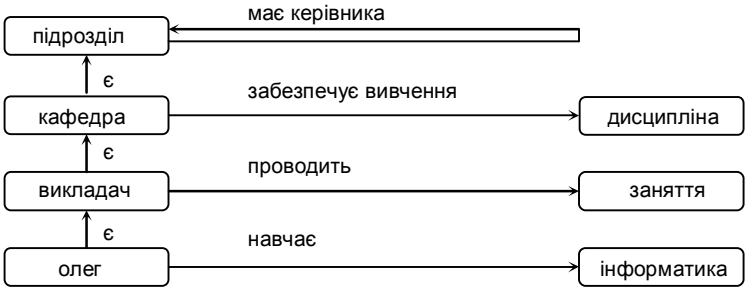
Дано семантичну мережу (вказана у варіанті). Описати її та скласти програму, яка дозволяє подати дану семантичну мережу. Виконати запити до програми. Програму зберегти у файлі (ім'я файлу вказане у варіанті).

1-6 варіанти



Файл: "LNN7_21. pro".

7-12 варіанти



Файл: "LNN7_24. pro".

III рівень

Використовуючи фрейми як методи подання знань описати таку ситуацію:

"Проводиться чемпіонат з футболу у три тури, у кожному турі по 2 гри. Визначено час початку проведення всіх змагань (для кожного туру о 16.00), місце проведення – стадіони "Спартак" (перша гра кожного туру) та "Динамо" (друга гра кожного туру), 4 суддівські бригади: 1-ша, 2-га, 3-тя та 4-та. Перший тур обслуговують на стадіонах "Спартак" та "Динамо" відповідно 1-ша та 3-тя суддівські бригади, другий – 2-га та 4-та, третій – 1-ша та 2-га. З деяких причин внесено зміни (вказані у варіанті) до порядку проведення змагань."

Скласти програму, яка у базі даних містить лише факти, що описують установлений порядок проведення змагань, і має правило внести_зміни/3, що дозволяє шляхом виконання запитів до програми внести зміни до її бази даних – записати нові факти виду слот/3 та спростувати_слот/2. Виконати запити до програми (вказані у варіанті).

Програму зберегти у файлі (ім'я файла вказане у варіанті).

1-2 варіанти.

Зміни: Місця і час проведення першого туру перенесено на стадіон "Динамо" на 17.00 та на 19.00.

Запити:
- де заплановано проведення всіх змагань?;
- на який час був запланований початок усіх змагань?;
- де проводиться 1-й тур змагань?;
- коли розпочинаються змагання 2-го туру?;
- коли розпочинаються змагання 1-го туру?;
- у якому турі змагань з футболу гра розпочинається о 19.00?;
- які суддівські бригади працюють на стадіоні "Спартак" у третьому турі?;
- які суддівські бригади працюють на стадіоні "Динамо" у першому турі?

Файл: "LNN7_21. pro

3-4 варіанти.

Зміни: Час проведення другого туру на стадіоні "Спартак" перенесено на 17.00.

Запити:
- де заплановано проведення всіх змагань?;
- на який час був запланований початок усіх змагань?;
- де проводиться 2-й тур змагань?;
- коли розпочинаються змагання 2-го туру?;
- коли розпочинаються змагання 1-го туру?;
- у якому турі змагань з футболу гра розпочинається о 17.00?;
- які суддівські бригади і відповідно на яких стадіонах працюють у першому турі?;
- на яких стадіонах і о котрій годині працює 2-га суддівська

бригада?

Файл: "LNN7_23. pro

5-6 варіанти.

Зміни: Час проведення третього туру перенесено на 14.00 і на стадіоні "Динамо" 2-гу суддівську бригаду замінено на 3-тю.

Запити:
- де заплановано проведення всіх змагань?;
- на який час був запланований початок усіх змагань?;
- де проводиться 3-й тур змагань?;
- коли розпочинаються змагання 2-го туру?;
- коли розпочинаються змагання 1-го туру?;
- у якому турі змагань з футболу гра розпочинається о 14.00?;
- які суддівські бригади працюють на стадіоні "Спартак" у другому турі?;
- яка суддівська бригада і о котрій годині працює на стадіоні "Динамо" у третьому турі"?

Файл: "LNN7_25. pro

7-8 варіанти.

Зміни: Час проведення першого туру на стадіоні "Спартак" перенесено на 14.00, а на стадіоні "Динамо" – на 15.00.

Запити:
- де заплановано проведення всіх змагань?;
- на який час був запланований початок усіх змагань?;
- де проводиться 1-й тур змагань?;
- коли розпочинаються змагання 3-го туру?;
- коли розпочинаються змагання 1-го туру?;
- у якому турі змагань з футболу гра розпочинається о 14.00?;
- які суддівські бригади і на яких стадіонах працюють у другому турі?

Файл: "LNN7_27. pro

9-10 варіанти.

Зміни: Час проведення всіх змагань перенесено на 17.00. У другому турі 4-та суддівська бригада замінена на 1-шу.

Запити:
- де заплановано проведення всіх змагань?;
- на який час був запланований початок усіх змагань?;
- де проводиться 1-й тур змагань?;
- у якому турі змагань з футболу гра розпочинається о 17.00?;
- які суддівські бригади працюють на стадіоні "Динамо"?;
- які суддівські бригади працюють на стадіоні "Спартак" у другому турі?

Файл: "LNN7_29. pro

11-13 варіанти.

Зміни: Місце і час проведення третього туру перенесено на стадіон "Спартак" на 16.00 та на 19.00.

Запити:
- де заплановано проведення всіх змагань?;
- на який час був запланований початок усіх змагань?;
- де проводиться 3-й тур змагань?;
- коли розпочинаються змагання 2-го туру?;
- яка суддівська бригада і на якому стадіоні працює о 16.00?;
- у якому турі змагань з футболу гра розпочинається о 19.00?;
- які суддівські бригади працюють на стадіоні "Спартак" у першому турі?

Файл: "LNN7_20. pro

Вимоги до захисту лабораторної роботи

Звіт виконаної лабораторної роботи повинен містити назву теми; текст програм І-го та ІІ-го рівнів, запити до програм.

ЕКСПЕРТНІ СИСТЕМИ

§1. ПОНЯТТЯ ЕКСПЕРТНОЇ СИСТЕМИ

Експертна система (ЕС) - комп'ютерна програма, яка використовує знання з порівняно вузької предметної області для розв'язку конкретних задач, що виникають у цій області, і при цьому поводить себе як експерт[1]. Тобто, ЕС містить знання (досвід) спеціалістів про деяку предметну область, у межах даної області здатна давати інтелектуальні поради або приймати інтелектуальні рішення на рівні людини-експерта і за вимогою користувача пояснювати свою лінію міркувань.

За допомогою експертних систем, користувачі, які мають звичайну кваліфікацію, можуть розв'язувати свої поточні задачі настільки ж успішно, як це зробили б самі експерти. Такий ефект досягається завдяки тому, що експертна система у своїй роботі відтворює ту ж схему міркувань, яку застосовує людина-експерт при аналізі проблеми. Таким чином, ЕС дозволяють копіювати і розповсюджувати знання, і, у результаті чого, досвід декількох висококласних професіоналів стає доступним широкому колу рядових спеціалістів.

Так як експертні системи призначені для розв'язку задач, які вимагають експертних знань, то вони повинні володіти цими знаннями. Тому ЕС також називають системами, що базуються на знаннях. Але від системи, що базується на знаннях, не обов'язково вимагати пояснення своєї поведінки та власного рішення користувачу, тоді як для ЕС ця вимога є необхідною. Здатність експертної системи пояснювати потрібна для того, щоб підвищити довіру користувача до порад системи, а також для того, щоб користувач мав змогу виявити можливий дефект у міркуваннях системи.

Для більшості ЕС характерно, що інформація про задачі, які розв'язуються системами, є неповною або ненадійною; відношення між об'єктами предметної області можуть бути наближеними. У цих випадках від системи вимагаються міркування з використанням імовірнісного підходу.

---

[1] Існує альтернативне біхевіористичне означення *експертної системи* - це *будь-яка програма, що застосовується для експертних консультацій*. Дане означення є більш слабким, ніж попереднє, так як воно охоплює всі програми, які використовуються як ЕС, не враховуючи того, що справжні експерти могли не брати участь у розробці таких програм.

ЕС як комп'ютерні програми специфічні, бо використовують механізм автоматичного міркування (виводу) і так звані слабкі методи, до яких належать пошук та евристика. Цим вони відрізняються від точних і добре аргументованих алгоритмів і не схожі на математичні процедури більшості традиційних розробок.

Експертні системи використовуються у таких видах діяльності людини, як проектування, планування, діагностика, переклад з однієї мови на іншу, реферування, ревізія, видача рекомендацій. Сфери застосування експертних систем - бізнес, проектування, дослідження, управління.

Вкажемо деякі галузі використання експертних систем:

- допомога медикам у постановці діагнозу і лікуванні деяких груп захворювань, таких як зараження крові та різні види раку;
- оцінка займів, ризиків страхування і капітальних вкладень для фінансових організацій;
- допомога хімікам у знаходженні правильної послідовності реакцій для створення нових молекул;
- одержання молекулярної структури хімічної речовини на основі дослідів;
- вивчення та сумування великих обсягів даних, що швидко змінюються і які не в змозі (з причини великої швидкості їх зміни) прочитати людина, наприклад, телеметричних даних зі штучних супутників;
- діагностика і виявлення неполадок у телефонній мережі на підставі результатів тестів і повідомлень про несправності;
- розв'язування математичних задач у системі середньої освіти;
- використання як складової частини забезпечення навчального процесу у середній та вищій школі.

Експертні системи та системи ШІ

Експертні системи є інтелектуальними системами і розглядаються як одна із груп систем ШІ. До характеристик, притаманних ЕС як системам штучного інтелекту, можна віднести:

- побудова "міркувань" на основі символьних перетворень;
- використання як загальних, так і окремих схем породження рішень;
- здатність розв'язувати завдання в реальних предметних галузях;
- здатність до інтерпретації формулювання запитів і завдань;
- здатність до аналізу своєї роботи.

Експертні системи відрізняються від інших програм ШІ своєю метою та побудовою. *Мета* ЕС полягає у розв'язуванні задач, які підходять для людини-експерта. *Побудова* ЕС полягає у створенні не просто механічної, а інтелектуальної програми.

Критеріями оцінки ЕС є такі:

- чи відображує внутрішнє функціонування програми підхід до

проблеми з точки зору людини;

- чи може програма пояснювати свої дії у такий спосіб, який зрозумілий людині;
- чи може програма взаємодіяти з користувачем за допомогою гнучкого діалогу, подібного діалогу, що ведеться природною мовою.

Експертні системи використовують усі методи програмування, які застосовуються в інших програмах ШІ.

Вкажемо деякі програми ШІ, які не є експертними системами за такими ознаками:

а) за метою:

- програма для друку з голосу: користувач говорить у мікрофон, а програма видає друкований текст;
- програма, яка може переглянути, а потім видати у перефразованій формі текст розповіді дитини і навіть дати відповіді на деякі питання по тексту. Програма повинна деякою мірою "розуміти" мову та відношення у повсякденному житті. Для розуміння розповіді дитини необхідно мати великий досвід і високий рівень інтелекту, однак з такою задачею впорається будь-яка людина, а не лише експерт.

б) за побудовою:

- прості програми для гри у шахи, дія яких ґрунтується на "грубій силі" (машина просто виробляє і класифікує множину ймовірних "сценаріїв" гри, а потім виконує ходи у відповідності до найбільш сприятливого сценарію).

Такі програми заносять до каталогу всі можливі для комп'ютера ходи і, крім того, враховують усі можливі контрходи гравця, зроблені у відповідь на хід машини, а також усі прийнятні ходи ЕОМ у відповідь на хід людини-гравця. Дана модель передбачає декілька рівнів гри, доки число варіантів не стане настільки великим, що навіть комп'ютер не справиться з ним. Хоча шахи є областю діяльності експертів, але програму в такій формі не можна вважати експертною системою, так як вона не функціонує подібно людині.

Експертні навчаючі системи

Характерною тенденцією розвитку систем штучного інтелекту є кооперація у сфері розробки та впровадження експертних навчаючих систем (ЕНС).

Зарубіжна школа накопичила значний досвід по використанню як інтелектуальних навчаючих систем, так і експертних навчаючих систем. Наприклад, у школах Швеції створюються інтелектуальні навчаючі системи, що базуються на знаннях, та системи допомоги. Важливим напрямом досліджень є вивчення мови Пролог. Ця мова проходить у школах Швеції перевірку як інструмент навчання математики у старшій середній школі і як інструмент підготовки вчителів математики для молодших класів середньої школи.

В англійських школах вивчається досить широкий спектр програмного забезпечення - від традиційного до програм штучного інтелекту.

Розробку ЕНС (інтелектуальних систем-помічників) передбачає відомий європейський проект ESPRIT. До його складу входить дослідницький проект EUROHELP, що планує побудову інтелектуальних помічників з використанням двох основних орієнтирів: по-перше, це орієнтир на методи побудови діалогових довідкових систем; по-друге, на використання методів програмованого навчання. Для створення ЕНС у проекті використовують мову Пролог.

Створення інтелектуальних навчаючих середовищ проводиться у рамках міжнародного проекту Exploring System Earth project. Проект передбачає поширення навчаючих середовищ у навчальних закладах як типу середня школа, так і університет.

У Японії побудовою інтелектуальних навчаючих систем займаються з самого початку робіт над створенням обчислювальних машин п'ятого покоління. Найбільш поширеними інструментальними засобами є: ESHELL, BRAINS, COMEX, EURECA, EXCORE.

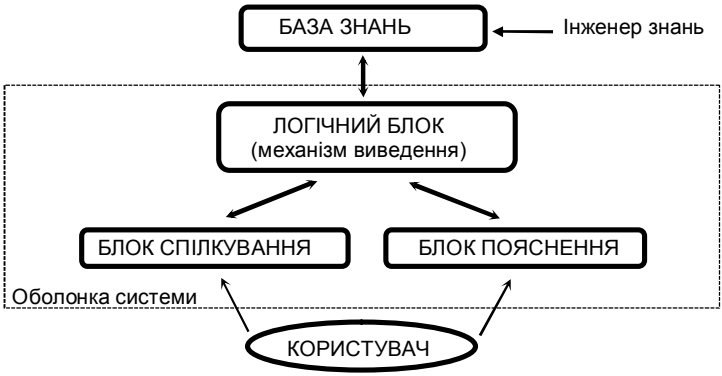
У Росії ЕС розглядаються, в основному, в експериментальних програмах і базуються на вивченні мови Пролог. У центрі інформатики МІЕМ на базі мови Пролог накопичений досвід створення навчальних експертних систем для гуманітарних дисциплін. Разом з тим проводяться дослідження по використанню ЕС для тестування і діагностування знань учнів, для розв'язання проблем профорієнтації та профвідбору, по використанню оболонок експертних систем для створення педагогічних програмних засобів, створюються експертно-навчаючі системи тощо.

Певний досвід по розробці і використанню експертних систем накопичений в Україні. В Інституті кібернетики ім. В. Глушкова створена інструментальна експертна оболонка BESS, що базується на бейєсівському статистичному методі прийняття рішень. У Національному педагогічному університеті (м. Київ) розроблена експертна оболонка INTER, в основі якої лежить фреймова модель подання знань. У Харківському педагогічному університеті створена експертна навчальна система ECOP з хімії. В Українському інституті інженерів водного транспорту створена і експлуатується ЕС "Визначення якості води". У Київській ВШ МВС нагромаджено значний досвід використання експертних систем у процесі вивчення юридичних дисциплін. У Донецькому медичному державному інституті розробляються діагностуючі експертні системи, що використовуються для вивчення курсу професійних захворювань. У Донецькому інституті з проблем штучного інтелекту ведуться дослідження з питань розробки експертних навчаючих систем, призначених для вивчення іноземних мов тощо.

§2. АРХІТЕКТУРА ЕКСПЕРТНОЇ СИСТЕМИ

2.1. Функціональна структура

Функціональну структуру ЕС можна подати у вигляді окремих функціональних блоків, зв'язаних між собою у певному порядку, що забезпечують роботу комп'ютерної програми як експертної системи:



Основу будь-якої експертної системи складають щонайменше два елементи: *база знань* та *механізм виведення*.

База знань

База знань ЕС містить знання з конкретної прикладної області, у тому числі окремі *факти*, *правила*, що описують відношення та явища, *твердження (що у більшості випадків представлені правилами)*, які описують методи, у т. ч. евристики, і різноманітні ідеї, необхідні для розв'язування задач у цій прикладній області. Факти являють собою короткотермінову інформацію і можуть змінюватися, наприклад, у ході експерименту. Правила являють собою довгочасну інформацію про те, як породжувати нові факти або гіпотези з того, що відомо на даний час.

Знання в експертних системах (як в системах штучного інтелекту) подаються щонайменше чотирма формами: семантичними мережами, фреймами, логічними моделями та системами продукцій.

Механізм виведення

Механізмом виведення називається механізм, що застосовує правила у деякій послідовності для маніпулювання знаннями, які зберігаються у базі знань.

Щоб система була здатною розв'язувати проблеми, потрібно, щоб у ній був реалізований хоча б один спосіб виведення і хоча б один метод управління виводом. Управління виводом передбачає, що у процесі роботи даються відповіді щонайменше на два питання:

- звідки слід починати процес міркувань;
- що робити, коли на певному кроці розв'язання можливі декілька різних ліній міркувань.

Для експертних систем існує щонайменше два види механізмів виведення: один ґрунтується на моделі логічного програмування, інший - на моделі системи продукцій.

Механізм виведення, побудований на моделі логічного програмування, можна уявити як звичайний механізм для виконання програми на Пролозі. Робота починається спробою виконати правило високого рівня шляхом перевірки, чи істинна його передумова. Система перевіряє істинність таких передумов, досліджуючи, чи істинні передумови цих передумов і т. д. Іншими словами, процес виводу на Пролозі є процесом пошуку в глибину і перегляду зліва направо всередині кожного логічного твердження. Процес повертається назад, якщо він заходить у тупик. Цей процес називають ще як такий, що працює на основі "*оберненого ланцюжка міркувань*", за яким для доведення гіпотези намагаються відшукати вихідні дані. Механізм логічного виведення може працювати також на основі "*прямого ланцюжка міркувань*", за яким міркування ведуться від вихідних даних до гіпотези.

Для іншого виду ***механізму виведення, що побудований на моделі системи продукцій***, правила скомпоновані у список. Цей механізм діє у два способи:

1) Управління починається з початку списку. Правила перевіряються послідовно доти, доки не знайдеться те, для якого справджується умова. Тоді система виконує дію, яка б вона не була. Далі вона намагається відшукати наступне правило для виконання або з початку списку, або продовжуючи пошук з того місця, де він перервався.

2) Переглядається весь список перед тим, як система почне виконувати яке-небудь правило, і у конфліктний набір збираються всі ті правила, умови яких істинні. Далі, у відповідності до заданих пріоритетів, система вибирає з набору одне з правил і виконує його. Це аналогічно процесу збудження нервових клітин. Далі дії системи циклічно повторюються: знайти збуджене правило, а потім виконати його.

Різниця між моделлю системи продукцій і моделлю логічного програмування полягає в основному в управлінні. Управління у системі продукцій має лінійних характер. Система продукцій діє подібно до оператора вибору у процедурній мові. Управління на Пролозі має структуру дерева. Завжди існує ієрархія дій, які необхідно виконати, що розуміється вже самим способом написання правил. Рух по дере-

ву відбувається суворо встановленим і передбачуваним способом, що визначений стратегією пошуку у глибину.

Блок спілкування

Блок спілкування здійснює спілкування з користувачем ЕС. Діалог будується у зручній для користувача формі, максимально наближеній до людського спілкування. Цей блок використовує всі досягнення у галузі ШІ, що стосуються розуміння текстів і синтезу відповідей природною мовою (система повинна бути здатна розуміти речення на природній мові, що формулюються користувачем у вільній формі, а також формулювати користувачу питання, що відповідають ситуації діалогу).

Блок пояснення

Блок пояснення забезпечує видачу пояснень, що стосуються шляху одержання розв'язків, якщо вони цікавлять користувача. Тобто, коли система видає деякий висновок або формулює питання для користувача, то останній має право запитати: "Як одержано такий розв'язок?" або "Навіщо це потрібно знати?". Система має бути здатною відповісти на подібні питання так, щоб створювалося враження інтелектуального характеру міркувань.

2.2. Операційна структура

Для різних експертних систем можна виділити спільні операційні елементи, з яких складається внутрішня структура ЕС: *пошук, розмежування знань та управління, евристики, природна мова, автоматичне міркування у стилі архітектури, що базується на правилах.*

Пошук

Більшість експертних систем працюють за такою моделлю. ЕС як комп'ютерна програма має початковий стан і здійснює пошук кінцевого стану (тобто цілі). У програмі передбачені можливості вибору, які реалізуються автоматично і дозволяють їй виконувати кроки від початкового стану до нових станів, які в тій чи іншій мірі наближаються до цілі. Таким чином, програма погоджує ціль, "крокуючи" від стану до стану. Вона повинна розпізнавати ситуацію, коли знаходить ціль або заходить у тупик. Як правило, на проміжних стадіях обчислюється деяке число, за допомогою якого програма проводить оцінку свого ходу. Ціль може бути одна або набір прийнятних цілей (кінцевих станів).

При реалізації пошуку ЕС у більшості випадків створює дерево пошуку на підставі запропонованих їй початкових відомостей, до того ж робить це поступово у процесі самого пошуку.

Зауважимо, що деякі ЕС можуть використовувати програми пошуку, що базуються на "грубій силі" та здійснюють пошук цілі, йдучи від стану до стану, і систематично досліджують усі можливі шляхи. При цьому для визначення кожного наступного кроку використовується два різновиди стратегії: пошук у глибину та пошук у ширину.

Стратегія пошуку в глибину базується на дослідженні послідовностей (кроків) одного із варіантів вибору до вивчення інших варіантів. Якщо ж процес пошуку заходить у тупик, він повертається вгору в останній пункт вибору, що містить невивчені альтернативні варіанти, а потім здійснюється наступний варіант вибору.

Стратегія пошуку у ширину полягає у дослідженні всіх можливих варіантів вибору "довжиною" в один крок, далі всіх можливих варіантів вибору "довжиною" у два кроки, потім - у три і так доти, доки не буде знайдено ціль.

Евристики

Евристика - це емпіричне правило, за допомогою якого людина-експерт у разі відсутності формули чи алгоритму намагається здійснити свої наміри.

Якщо алгоритм є, то експерт скористається ним. Але існує велика кількість задач з предметних областей, де алгоритму немає. І тому більшість ЕС, що використовують евристики, намагаються просто змоделювати процеси міркувань людини-експерта. Для написання такої програми спочатку досліджується діяльність людини-експерта (спостереження, інтерв'ювання, збір статистики), формується набір стратегій людини-експерта для певних ситуацій. Указаний набір складає типову евристичну інформацію, яка може бути використана ЕС.

Природна мова

Експертні системи, як правило, працюють у діалоговому режимі, тобто обмінюються інформацією і висновками з користувачем у формі діалогу. Якщо програма може приймати дані, що вводяться у вигляді простих речень, то вважається, що програма веде діалог природною мовою.

Автоматичне міркування

ЕС, як і людина-експерт, повинні вміти проводити міркування, щоб на основі знань про предметну область розв'язувати задачі з цієї області. При цьому для ЕС є необхідною вимога автоматично проводити процес міркування від початкового стану до одержання виснов-

ку. Такі міркування забезпечуються механізмом виведення ЕС, який, як зазначалося, базується на моделі логічного програмування або на моделі системи продукцій.

Якщо проаналізувати механізми, які застосовуються ЕС для своєї роботи, то виявляється, що ці механізми в основному побудовані на так званій "архітектурі, що базується на правилах".

Розмежування знань та управління

Одним з головних принципів при розробці ЕС є те, що правила, які несуть знання про предметну область повинні бути відокремлені від правил, що забезпечують управління виводом.

Приклад 44. є прикладом того, як послідовне виконання правил (поповнення знань (питання), розв'язку задачі (погодження правила дія), видачі повідомлення (write), вилучення використаних фактів з внутрішньої бази даних (retractall)), які забезпечують управління процесом виводу, є відокремлене від виконання правил (описаних як множина правил дія), що містять знання про ситуацію, яка аналізується.

§3. КЛАСИФІКАЦІЯ ЕКСПЕРТНИХ СИСТЕМ

3.1. Експертні системи різних поколінь

Комп'ютерні системи, які можуть лише повторити виведення експерта відносяться до експертних *систем першого покоління*. Такі експертні системи були призначені для розв'язання добре структурованих завдань, що потребують порівняно невеликого обсягу емпіричних знань.

Для того, щоб ЕС була у ролі повноцінного помічника, недостатньо можливостей системи, яка лише імітує діяльність людини. Необхідно, щоб ЕС була здатна проводити аналіз нечислових даних, висувати і відкидати гіпотези, оцінювати достовірність фактів, самостійно поповнювати свої знання, контролювати їх несуперечливість тощо. Наявність таких властивостей є характерною для ЕС *другого покоління*, концепція яких почала розроблятися на початку 90-х років. ЕС цього покоління – це так звані оболонки. Основою оболонки є готовий механізм виводу і порожня база знань. Процес наповнення знаннями ЕС підтримується спеціальними сервісними програмами – редактор бази знань, засоби налагодження, трасування, які призначені для інженера по знаннях.

Порівняльні можливості ЕС першого та другого покоління:

ЕС першого покоління	ЕС другого покоління
Подання знань	
· знання одержуються тільки від експерта. Досвід самої системи не накопичується і не застосовується; · використовуються поверхневі знання у вигляді евристичних правил і певна модель подання знань; · моделі подання знань орієнтовані на статичні, відносно прості і добре структуровані предметні області; · відсутні знання про межі області компетентності системи, поза якими система не працездатна.	· використовуються глибинні знання, що являють собою теорії предметних областей і загальні стратегії розв'язку проблем; · знання подаються складеними моделями, що можуть включати одночасно мережі фреймів, продукції і логічні моделі; · система має модель не тільки предметної області, але і самої себе, що дозволяє їй визначати межі своєї компетентності; · ЕС може розв'язувати задачі з динамічних предметних областей, знання про які можуть змінюватися у процесі виведення.
Механізм виведення	
· одержання нових висновків за допомогою виводу на знаннях. Вико-	· заміна виводу на обґрунтування; · поєднання достовірного (дедукти-

ристання дедуктивного виведення; · реалізація виводу тільки за умови повноти знань і даних; · невміння проводити виведення з урахуванням зв'язку об'єктів або фактів у просторі і часі; · нездатність відшукати наближену відповідь, якщо для точної недостатньо даних або правил; · невідповідність схеми виводу в ЕС міркуванням експерта; · відсутність засобів імітації процесів предметної області, що не дозволяє системі ставити питання типу "Що буде, коли...?".	вного) і правдоподібного виведення, тобто такого, за яким кожному одержуваному висновку присвоюється вага, що характеризує ступінь його достовірності; · здатність системи послаблювати (посилювати) прийняті припущення; · наявність індуктивного (від конкретного до загального) і абдуктивного (від конкретного до конкретного) виведень, а також проведення немонотонних міркувань, у процесі яких нові факти іноді змінюють істинності раніше зроблених висновків.
Інтерфейс користувача	
· жорсткість діалогу - відповіді користувача повинні бути представлені у цілком визначеному виді та форматі; · відомості від користувача ЕС у процесі експертизи не запам'ятовуються, тому, у разі потреби, на наступному сеансі роботи вони вводяться спочатку; · неузгодженість питань, що ставляться ЕС користувачу, хоча кожне окреме питання цілком логічне, у їх послідовності може бути відсутня цілеспрямованість; · надмірність сукупності питань ЕС, адресованих користувачу, зумовлена тим, що при її розробці не були виявлені взаємозв'язки між даними, що потрібні для одержання розв'язків і дозволяють побудувати для кожної експертизи свою серію питань.	· ЕС включає модель користувача, яка дає можливість організувати взаємодію з ним в оптимальній формі, тобто формі, за якою розв'язок задачі буде одержано за мінімальний час. У такій моделі враховуються особливості роботи конкретного користувача, специфіка розв'язуваних ним задач та типові для нього сценарії діалогу.
Пояснення одержаних результатів	
· механістичність побудови пояснень, що формуються шляхом об'єднання аргументації, яка міститься у кожному правилі, що застосо-	· регулювання ступеня узагальнення пояснень, що видаються ЕС у відповідності до побажань користувача;

вується; · недосконаліть механізму пояснення одержаного розв'язку, яка полягає у тому, що користувач отримує або тривіальну інформацію, або вона не покриває всіх потреб.	· інформація, яку отримує користувач, задовольняє його потреби у поясненні одержаного розв'язку.
Поповнення знань та навчання	
· поповнення знань системи і контроль їх несуперечливості користувачем; · обов'язковість подання знань експерта у формі, яка вимагається моделлю подання знань ЕС; · неспівпадання структури знань про предметну область в ЕС зі структурою знань експерта. Тому розв'язки ЕС можуть дещо відрізнятися від розв'язків експерта; · нездатність ЕС навчатися.	· наявність засобів автоматичного виявлення закономірностей. Система може одержувати знання; · самостійно із бази даних шляхом висування гіпотез і побудови їх обґрунтувань, тобто навчатися на прикладах; · наявність засобів, що дозволяють обрати модель подання знань, яка досить добре відповідає структурі знань експерта; · здатність системи аналізувати свої знання і виявляти протиріччя між старими знаннями, тими, що одержані від експерта, або тими, що одержані емпірично; встановлювати факт їх неповноти та помилковості. При цьому система проявляє активність: може самостійно звернутися до користувача, щоб усунути вказані протиріччя.

Експертні системи *третього покоління* або третього типу – це "гібридні" ЕС і ЕС у реальному масштабі часу. Метою цього напряму є надання системності даним, наведеним у великих системах обробки інформації. У даному випадку ЕС – це методологічний засіб, свого роду "інтелектуальний інтерфейс", який дозволяє користувачеві виходити оперативно на знання, що знаходяться в базах даних і пакетах прикладних програм. Особливістю застосування таких систем може бути розв'язання завдань з використанням методів системного аналізу, дослідження операцій, математичної статистики, прикладної математики, обробки інформації.

До *четвертого покоління* ЕС належать мережі експертних систем – таке об'єднання кількох ЕС, в якому результатами розв'язання однієї із них є умови для функціонування інших. Системи такого типу ефективні для завдань з інтегрованою структурою розв'язків в умовах розподільної обробки інформації.

3.2. Категорії експертних систем

Класифікувати експертні системи можна і за функціями, які вони виконують:

- *системи інтерпретації*, що означає одержання розумних висновків на основі початкових даних;

- *системи діагностики і ремонту*. ЕС цієї категорії іноді називають програмами класифікації, так як діагностика і класифікація є одне і те ж;

- *системи планування і проектування*. Вони виконують компонування послідовності дій або наборів об'єктів для розв'язку тієї чи іншої проблеми;

- *системи управління і контролю*. Такі системи зв'язані з великими масивами елементів, що вводяться і виводяться, які, у свою чергу, утворені деяким складним пристроєм (системою). Мета управління і контролю полягає в автоматичній підтримці оптимальної роботи пристрою. ЕС застосовуються у тих випадках, коли проблемна область управління непроста і мінлива для того, щоб виявити потрібні заходи контролю. Вказані ЕС застосовуються для контролю на атомних і великих електростанціях, на нових реактивних літаках для забезпечення інтерфейсу між пілотом і системою управління польотом тощо;

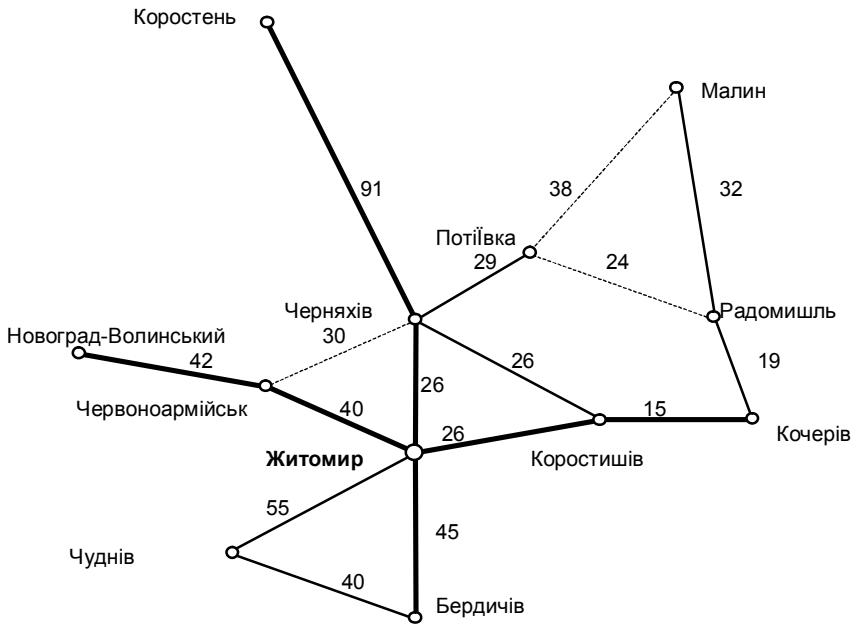
- *прогнозуючі системи*. Такі системи здатні, наприклад, передбачити новий урожай на базі супутникових даних тощо;

- *експертні системи навчання*. Ці системи можуть здійснювати навчання у оптимальному режимі на основі побудованої ними моделі того, що знає і чого не знає учень на кожному етапі навчання.

§4. ЕС З ПРОСТИМ ПОШУКОМ І РОЗПІЗНАННЯМ ОБРАЗІВ

Найбільш вагомо достоїнства мови Пролог виявляються у двох аспектах – пошуку і співставленні зі зразком. Тому існує клас програм, які, незважаючи на свою досить просту внутрішню структуру, є експертними системами за родом розв'язуваних задач.

Приклад 45. Дано карту доріг Житомирської області де вказано довжину кожної ділянки дороги. Розробити ЕС "Транспорт", що знаходить маршрути, їх відстані та найкоротший маршрут між двома містами.



Складемо програму на Турбо-Пролозі, яка є моделлю пропонованої ЕС:

```
include "pr_22. pro"                                    /*pr_45. pro*/
include "ULS. pro"
domains
  км = integer
  місто,  имя = symbol
  список_км = км*
  список_міст = symbol*
  структура_ділянка = ділянка (місто, місто, км)
  список_ділянок = структура_ділянка*
```

```
predicates
  маршрут (місто, місто, список_міст, км)
  мін_маршрут (місто, місто, список_міст, км)
  шлях (місто, список_міст, км, список_міст, км)
  суміжні (місто, місто, список_ділянок, км)
  місто (місто, місто)
  member (місто, список_міст)
  member (структура_ділянка, список_ділянок)
  member (місто, sp)
  min (integer, integer, integer)
  minelement (км, список_км)
  повтор
  діалог
  інф
  имя_користувача
  small_sym (symbol, symbol)
  питання (sp)
  відповідь (sp)
  визначити_міста (sp, місто, місто)
  ключові_слова (имя, sp, integer)
  member_list (sp, sp)
  member_list (список_міст, список_міст)

database
  карта (список_міст, список_ділянок)
  користувач (имя)
  е_маршрут (місто, місто, список_міст)

 goal
  діалог.

clauses

% = = == = = = = база знань = = = = = = = = = = = = = =
=
% ------------------------- факти ----------------------------------------------------
-
 карта ( [бердичів, чуднів, житомир, коростишів, кочерів, червоноармійськ,
          новоград, черняхів, радомишль, потіевка, малин, коростень ] ,
[ ділянка (бердичів, чуднів, 40),          ділянка (бердичів, житомир, 45),
  ділянка (житомир, чуднів, 55),    ділянка (житомир, червоноармійськ, 40),
  ділянка (житомир, черняхів, 26),       ділянка (житомир, коростишів, 26),
  ділянка (черняхів, коростишів, 26),      ділянка (коростишів, кочерів, 20),
  ділянка (кочерів, радомишль, 20), ділянка (червоноармійськ, новоград, 42),
  ділянка (червоноармійськ, черняхів, 30), ділянка (черняхів, коростень, 91),
  ділянка (черняхів, потіівка, 29),        ділянка (потіевка, радомишль, 24),
  ділянка (радомишль, малин, 32),          ділянка (потіівка, малин, 38) ] ).
```

```
% ------------------------- правила бази знань ------------------------------------
--

мін_маршрут (A, Z, МінМаршрут, МінВідстань):-
        findall (M, маршрут (A, Z, _, M), List),  minelement (МінВідстань, List),
        маршрут (A, Z, МінМаршрут, МінВідстань).

маршрут (A, Z, Маршрут, Відстань_AZ):-
        шлях (A, [Z], 0, Маршрут, Відстань_AZ).

шлях (A, [A|Шлях], Відстань_AZ, [A|Шлях], Відстань_AZ).
шлях (A, [Y|Шлях], Відстань_AX, Маршрут, Відстань_AZ):-
        карта (_, СписокДілянок),
        суміжні (X, Y, СписокДілянок, Відстань_XY),
        not (member (X, Шлях)),  Відстань_AY = Відстань_AX+Відстань_XY,
        шлях (A, [X, Y|Шлях], Відстань_AY, Маршрут, Відстань_AZ).

  суміжні (X, Y, Список, S):-
     member (ділянка (X, Y, S), Список);
     member (ділянка (Y, X, S), Список).

% ---------------------- загальні правила-процедури -----------------------------
  minelement (El, [El]).
  minelement (El, [H|T]):-
     minelement (El1, T),  min (H, El1, El).

  min (X, Y, Z):-
     X< = Y, Z = X ; X>Y, Z = Y.

  member (R, [R|T]).
  member (R, [H|T]) :-
     member (R, T).

  повтор.
  повтор :- повтор.

  small_sym ("", "").
  small_sym (RechX, RechY) :-
     frontchar (RechX, L, O), upper_lower_sym (L, L1),
     small_sym (O, Rech1),  frontchar (RechY, L1, Rech1).

 member_list ( [ ], _).
 member_list ( [H | T], L) :-
   member (H, L), member_list (T, L).

% = = = = = = = = блок спілкування = = = = = = = = = = = = = =

діалог :-
     clearwindow, інф, имя_користувача, повтор,
     питання (R),  відповідь (R),  fail.

інф :-
   write ("\tЕкспертна система <ТРАНСПОРТ> дае можливість визначити"),
   nl, карта (L, _),   write ("\t\tмаршрут, відстань між містами:\n\n", L), nl, nl.
```

```
имя_користувача :-
    write ("Як Вас звати?  "),
    readln (Имя), nl, retractall (користувач (_)), asserta (користувач (Имя)).

питання (R):-
    користувач (Имя), write ("Запитуйте,  ", Имя, ":  "),
    readln (Rech), small_sym (Rech, ModRechen),  lexanaliz (ModRechen, R).

відповідь (R):-
    ключові_слова (L, Слово, 1), member_list (Слово, R),
    e_маршрут (A, B, M1), маршрут (A, B, M, S),
    not (member_list (M, M1)), not (e_маршрут (A, B, M)),
        write ("\nВідповідаю: ", L, " <", A, "-", B,  "> -\n", M),
        write ("; відстань - ", S, "км."), nl, nl,
        assertz (e_маршрут (A, B, M)), !.

  відповідь (R) :-
      ключові_слова (L, Слово, 1), member_list (Слово, R),  e_маршрут(A,B,_),
      write ("\nПроглянуто всі можливі маршрути <", A, "-", B,  ">!"), nl, nl, !.

  відповідь (R) :-
      ключові_слова (L, Слово, 2), member_list (Слово, R),
      визначити_міста (R, A, B), мін_маршрут (A, B, K, S),
        write ("\nВідповідаю:  ",  L , " <",  A ,  "-",  B ,  "> -\n", K ),
        write ("; відстань - ", S, "км."), nl, nl,
        retractall (e_маршрут (_, _, _)),
        assertz (e_маршрут (A, B, K)), !.

  відповідь (R) :-
      ключові_слова (L, Слово, 3), member_list (Слово, R),
      визначити_міста (R, A, B), мін_маршрут (A, B, K, S),
        write ("\nВідповідаю: Найкоротша ", L, " - ", S, "км."),
        retractall (e_маршрут (_, _, _)),
        assertz (e_маршрут (A, B, K)), nl, nl, !.

  відповідь (R) :-
      ключові_слова (L, Слово, 4),
      member_list (Слово, R),
      карта (List, _),
        write ("\t\tСписок міст:\n", List), nl, nl, !.

  відповідь (_) :-
        write ("\nПомилка формулювання питання! Повторіть питання."),
        write ("\n (Завершити роботу з ЕС - <Ctrl+Break>)"), nl, !.

ключові_слова ( "Інший маршрут", W, 1 ) :-
      W = [інша]; W = [інший]; W = [по, іншому] ; W  =  [як,  ще];  W  =  [яка,
ще];
      W = [якось, ще]; W = [який, ще]; W = [по, другому].
  ключові_слова ( маршрут, W, 2 ) :-
      W = [шлях]; W = [маршрут]; W = [добратися]; W = [доіхати].
  ключові_слова ( відстань, W, 3 ) :- W = [відстань].
  ключові_слова ( міста, W, 4) :-
      W = [які, міста]; W = [список, міст]; W = [які, населені, пункти].
```

```
визначити_міста (R, A, B) :-
    member (X, R), member (Y, R),
    місто (A, X), місто (B, Y),
    карта (L, _),
    member (A, L), member (B, L),
    A<>B.

місто (бердичів, Н):-        Н = бердичів; Н = бердичева; Н =
бердичевом.
місто (чуднів, Н):-          Н = чуднів; Н = чуднова; Н = чудновом.
місто (житомир, Н):-         Н = житомир; Н = житомира; Н = житомиром.
місто (коростишів, Н):-      Н = коростишів; Н = коростишева;
                             Н = коростишевом.
місто (кочерів, Н):-         Н = кочерів; Н = кочерова; Н = кочеровом.
місто (червоноармійськ,Н) :- Н = чевоноармійськ; Н = червоноармійська;
                             Н = червоноармійськом.
місто (новоград, Н):-        Н = новоград; Н = новограда; Н =
новоградом.
місто (черняхів, Н):-        Н = черняхів; Н = черняхова; Н = черняховом.
місто (радомишль, Н):-       Н = радомишль; Н = радомишля;
                             Н = радомишлем.
місто (потіївка, Н):-        Н = потіївка; Н = потіївки; Н = потіївкою.
місто (малин, Н):-           Н = малин; Н = малина; Н = малином.
місто (коростень, Н):-       Н = коростень; Н = коростеня; Н = коростенем.
```

У прикладі транспортна система подана у вигляді графа і для роботи з нею використані ідеї, що реалізують операції над графами[1]. В базі знань указується факт розташування міст та ділянки доріг між містами як граф - предикат карта/2, першим аргументом якого є список міст, а другим - список ділянок доріг між містами.

Кожна ділянка дороги визначається як предикат ділянка/3, де перший та другий аргументи - міста, які сполучає ділянка, а третій - довжина ділянки.

Маршрут між початковим та кінцевим містом знаходиться як список назв міст від початкового до кінцевого (за правилом маршрут/4). Правила маршрут/4 та шлях/5 базуються на певних процедурах для роботи з графами і мають аргументи Відстань_AZ, які дозволяють визначати довжину знайденого маршруту.

Серед правил бази знань є правило мін_маршрут, за яким визначається мінімальний серед всіх можливих маршрутів від одного міста до іншого: спочатку предикатом findall формується список List довжин всіх можливих маршрутів від міста A до міста Z, потім предикатом minelement визначається найменше значення довжини маршруту серед усіх можливих, і знайдене значення Мін_відстань (разом з назвами міст) пе-

[1] Докладніше про реалізацію операцій з графами див. *Братко И. Программирование на языке Пролог для искусственного интеллекта: Пер.с англ. – М.: Мир,1990.*

редається у предикат маршрут для погодження та визначення найкоротшого маршруту.

Серед загальних правил-процедур записані інші правила: member - визначення належності елемента списку, повтор - для повторного виконання частини правила діалог при організації діалогу, member_list - для визначення належності всіх елементів деякого списку іншому списку, small_sym - для переведення всіх літер речення, що вводиться користувачем з клавіатури, у літери нижнього регістру.

Зауважимо, що для виконання у правилі small_sym предикату upper_lower_sym/2 використовується директива include "ULS. pro", яка підключає текст програми ULS. pro і дозволяє перевести символ російської абетки з верхнього регістру у нижній та навпаки.

Правило діалог/0 забезпечує організацію діалогу у даній моделі експертної системи.

Предикатом clearwindow проводиться очистка діалогового вікна перед початком сеансу роботи, предикатом інф/0 на екран виводиться початкова інформація про можливості системи, предикатом имя_користувача очікується введення користувачем власного імені та заноситься відповідний факт (користувач (имя)) у внутрішню базу даних, зв'язкою предикатів <повтор - fail> забезпечується циклічне виконання предикатів питання та відповідь.

Правило питання дозволяє користувачу ввести з клавіатури питання природною мовою: readln аргументу Rech надає значення рядка символів, введених користувачем з клавіатури; small_sym перетворює Rech у рядок, що містить тільки малі літери; lexanaliz (підключається директивою include "pr_22. pro") розбиває Rech на потік лексем - список окремих слів та символів.

Предикатом відповідь формується відповідь на введене питання і процес генерації відповідей відбувається на основі роботи з правилами продукцій: яке б не було питання, послідовно зверху вниз проглядаються всі правила відповідь/1, у кожному з яких предикатом ключові_слова/2 відшукується окреме ключове слово (або певні набори ключових слів) і, якщо таке слово знайдено, то потім формулюється відповідь і процес перегляду припиняється (предикат "!"). У такому випадку погодження правила діалог завершується невдачею (fail), що зумовлює (предикат повтор) повторне виконання предикатів питання та відповідь, тобто після відповіді на питання експертна система очікує на введення наступного питання.

У разі введення питання, яке не містить визначених ключових слів, система реагує (відповідь (_)) повідомленням користувачу про помилку у формулюванні питання та знову переходить у режим очікування на ввід нового питання.

У блоці спілкування для сприйняття системою природної мови передбачені правила місто/2, які дозволяють реагувати на назви міст, введені користувачем у непрямих відмінках.

Правило визначити_міста виконує такі операції: спочатку (у реченні як у списку лексем) переглядається кожна лексема та вибираються, як мінімум, перші дві лексеми, які є назвами міст у визначених відмінках (місто/2) і при цьому назви міст подаються у називному відмінку (значення першого аргументу предикату місто/2); далі виконується перевірка на предмет того, чи входять визначені міста у список міст (перший аргумент предикату карта/2) - цим забезпечується несуперечливість даних, що містяться у базі знань та допоміжних даних з блоку спілкування.

У процесі виконання правил відповідь, які знаходять потрібні маршрути, до внутрішньої бази даних заносяться (assert) або вилучаються (retractall) знайдені маршрути (факт е_маршрут/3).

§5. КОРОТКИЙ ОГЛЯД ПІДХОДІВ ДО РОЗРОБКИ ЕС

Програмування ЕС з евристиками

Модель ЕС "Транспорт" використовує стратегію "грубої сили". Зокрема, для знаходження найкоротшого маршруту між двома містами спочатку визначаються всі можливі маршрути від міста до міста, формується їх список, а потім серед таких маршрутів відшукується той, що має найменшу довжину. Тобто хід розв'язку задачі (пошук) співпадає з ходом, що використовується системою виводу, а не з тим ходом, який продиктований логікою задачі.

Якщо область пошуку досить мала (наприклад, до бази знань ЕС включені вибіркові міста невеликого регіону), то така стратегія виконується успішно. Але для більшості випадків (наприклад, до бази знань включено тисячі населених пунктів та ділянок доріг) вимагається більш направлене дослідження області пошуку. Тоді використовують ЕС з евристиками для того, щоб обмежити область пошуку і найбільш важливі її частини переглянути раніше від інших.

Існує ряд евристичних підходів. Найпростіший можна описати так: якщо у вас є прийнятні варіанти вибору, виберіть найкращий із них; якщо ж ви попали у тупик, поверніться до останнього місця, де є альтернативні недосліджені варіанти вибору, і здійсніть наступний найкращий вибір.

Модель ЕС "Транспорт" можна модифікувати, включивши евристику: при виборі для певного міста суміжного потрібно проглядати не всі міста, а спочатку найкраще - те місто, яке визначає азимут, близький до азимуту, що визначається початковим та кінцевим містом маршруту. Очевидно, що базу даних ЕС необхідно, крім такої евристики, насамперед доповнити координатами міст та правилом, яке дозволяє визначити азимут маршруту.

Існує два типових способи включення евристичної інформації у пошукову структуру:

- за допомогою евристичної функції, яка "визначає вагу" тверджень конкретної задачі і визначає їх відносну значущість;
- безпосередньо розмістивши евристичну інформацію у правила ЕС.

Є декілька варіантів використання евристичної функції.

Одна із стратегій полягає у тому, що за допомогою такої функції вибирається найкращий крок і перевіряються у той же спосіб (використовуючи функцію) всі його наслідки, перед тим як здійснити інші кроки. У випадку тупика потрібно повернутися до найближчого розгалуження та спробувати здійснити інший найкращий крок. Тобто спочатку перевіряються всі наслідки одного вибору на всю можливу глибину, і тільки потім відбувається черговий вибір. Суттєвим недо-

ліком такої стратегії є те, що для здійснення вибору використовується тільки локальна інформація.

Інший варіант евристичного пошуку, що об'єднує у собі позитивні риси пошуку у глибину та ширину (див. стор. 151), є пошук під назвою *Кращий перший крок*. На кожній стадії пошуку досліджується деяка кількість варіантів вибору, і оціночна функція обчислюється для їх кінцевого стану. У *Кращому першому кроці* пошук продовжується від досягнутого кінцевого стану, що має найкращу оціночну функцію уверх до цього місця. При такій стратегії часто відбуваються стрибки від кінця одного варіанту вибору до іншого, намагаючись завжди працювати з тим, що є найбільш перспективним. Цей метод досить ефективний, але тривалі обчислення і великий обсяг обліку (всі частинні варіанти вибору (шляхи) повинні постійно підтримуватися) примушують часто використовувати більш просту процедуру, яка швидко перевіряє багато розв'язків. Ефективна стратегія пошуку жорстко залежить від предметної області, ефективність евристичного пошуку визначається оціночною функцією, що використовується.

ЕС та вивід в умовах невизначеності

Для багатьох реальних задач потрібно провести оцінку гіпотез, для яких інформація є неповною або недостатньою. Експертні системи, що призначені для розв'язування таких задач, повинні, незважаючи на невизначеність, приймати необхідні рішення.

Міркування, що базуються на невизначеності, використовуються у всіх ЕС медичної діагностики і консультацій по способам лікування. Те ж саме маємо і у системах спостереження за наявності одночасно декількох конкуруючих гіпотез та їх постійної переоцінки при надходженні нових даних: у кінцевому підсумку визначається одна гіпотеза, яка дозволяє здійснити відповідний висновок. У системах розпізнання природної мови теж мають бути конкуруючі гіпотези, наприклад, про те, яке конкретне слово використовується у реченні.

Окрім того, багато програм, що використовують евристики, теж повинні працювати в умовах невизначеності, так як евристика – наближений метод, що вказує напрям пошуку. Якщо існує тільки одна евристика, то проблем немає. А якщо два евристичних методи орієнтують програму у двох різних напрямках? Або якщо дві евристики вказують один і той же напрямок пошуку, чи має це викликати більшу довіру, ніж би була тільки одна із них?

Тому при виводі в умовах невизначеності природним є використання понять теорії ймовірності: умовної ймовірності, правила Байєса, правила і/або, правила композиції тощо.

ЛАБОРАТОРНА РОБОТА №8

Робота з демонстраційною версією експертної системи, що використовує метод співставлення зі зразком

Мета: Одержати уміння та навички роботи з експертною системою з: модифікації та введення нових знань у базу знань ЕС, модифікації блоку спілкування; організації та здійснення діалогу у системі

Теоретична частина: завдання та контрольні питання

І рівень

1. Поняття експертної системи.
2. У яких видах діяльності людини використовуються експертні системи? Які сфери та галузі застосування експертних систем?
3. Навести приклади розроблених експертних систем в Україні та за кордоном.
4. Навести приклади інтелектуальних, експертних систем, які використовуються в операційних системах та офісних пакетах.
5. Що розуміють під оболонкою експертної системи.
6. Загальна характеристика поколінь експертних систем.
7. Кого називають інженером по знаннях?

ІІ рівень

1. Описати функціональну структуру експертної системи. Що розуміється під базою знань, механізмом виведення, блоками спілкування та пояснення в експертній системі?
2. Описати операційну структуру експертної системи?
3. У чому полягає різниця між інтелектуальними системами та експертними системами?
4. Форми подання знань в експертних системах.
5. Описати модель системи продукцій та навести приклад такої системи при поданні знань.

ІІІ рівень

1. Порівняйте механізм виведення, побудований на моделі логічного програмування та механізм виведення, побудований на моделі системи продукцій.
2. Порівняйте стратегії пошуку у глибину та пошуку у ширину.
3. Поняття евристики. За яких умов використовуються експертні системи з евристиками? Які евристичні підходи використовуються у процесі розробки таких експертних систем?
4. Експертні системи та вивід в умовах невизначеності.

Практична частина

І рівень

1-13 варіанти. Виконати запуск демонстраційної версії експертної системи (Приклад 45.). Провести діалог з експертною системою, відшукавши відповіді на питання:

- які міста включені до карти доріг Житомирської області?
- яка найкоротша відстань від Житомира до Коростеня?
- який найкоротший маршрут від Коростеня до Житомира? Яка відстань цього маршруту?
- який можливий маршрут руху від Чуднова до Черняхова? Які інші маршрути є між указаними містами?

ІІ рівень

Доповнити базу знань відомостями про нове місто, нові ділянки доріг, які зв'язують нове місто з існуючими в базі знань містами згідно додатку №1. Забезпечити можливість проведення діалогу експертної системи з користувачем, використовуючи у фразах діалогу ім'я нового міста у 3-х відмінках.

Провести діалог з ЕС, відшукавши відповіді на питання:

- яка найкоротша відстань від нового міста до Житомира?
- яка найкоротша відстань від нового міста до Малина?
- який можливий маршрут руху від нового міста до Коростеня? Які є інші маршрути між указаними містами?

Програму зберегти у файлі "LNN8_2. pro", де NN – номер варіанту користувача.

ІІІ рівень

Передбачити у базі знань експертної системи можливість обчислення відомостей про умовну відстань ділянки дороги між містами (ум. відстань = коефіцієнт покриття * довжину ділянки, де коефіцієнт покриття для доріг, позначених на схемі стор. 157 подвійною лінією – 1, суцільною – 1.2, пунктирною – 1.5).

Записати програму у файл "l8_3NN. pro", де NN – номер варіанту.

Внести зміни до експертної системи, забезпечивши можливість одержання користувачем необхідних відповідей на питання (вказані у варіанті). Провести з експертною системою відповідний діалог. Модифіковану програму зберегти у файлі (ім'я файлу вказано у варіанті).

1 варіант

Яке число можливих маршрутів від одного міста до іншого? Яка умовна відстань для маршруту від одного міста до іншого?

Файл програми: "l8_31. pro".

2 варіант

Визначити такий маршрут від одного міста до іншого, який проходить через місто Житомир. Вказати умовну відстань для визначеного маршруту.

Файл програми: "l8_32. pro".

3 варіант

Який з маршрутів від одного міста до іншого має найменшу умовну відстань?

Файл програми: "l8_33. pro".

4 варіант

Який з маршрутів від одного міста до іншого має найбільшу умовну відстань?

Файл програми: "l8_34. pro".

5 варіант

Визначити такий маршрут від одного міста до іншого, який не проходить через місто Черняхів. Яка кількість таких маршрутів?

Файл програми: "l8_35. pro".

6 варіант

Який з маршрутів від одного міста до іншого має найбільшу різницю між умовною та реальною відстанями?

Файл програми: "l8_36. pro".

Вимоги до оформлення звіту лабораторної роботи

Звіт лабораторної роботи повинен містити назву теми; відповіді експертної системи до питань І-го та ІІ-го рівнів; тексти нових правил до завдань ІІІ-го рівня.

Додаток №1.

Варіант	Назва нового міста	Ділянка 1: нове місто - місто №1		Ділянка 2: нове місто - місто №2	
		Місто №1	км	Місто №2	км
1-2	Андрушівка	Житомир	35	Бердичів	40
3-4	Попільня	Бердичів	25	Коростишів	50
5-6	Ружин	Бердичів	40	Кочерів	55
7-8	Ємільчино	Коростень	60	Новоград-Вол.	45
9-10	Баранівка	Чуднів	45	Новоград-Вол.	35
11-12	Народичі	Коростень	50	Малин	30
13-14	Київ	Кочерів	70	Малин	90

ВИКОРИСТАНА ТА РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Аверкин А.Н. и др.. Толковый словарь по искусственному интеллекту. – М.: Радио и связь, 1992.

2. Братко И. Программирование на языке Пролог для искусственного интеллекта: Пер.с англ. – М.: Мир,1990.

3. Вивчення експертних систем у курсі основи інформатики і обчислювальної техніки: Методичні рекомендації/ Укл. Ю.С.Рамський, Н.Р.Балик.- К.: УДПУ, 1995.

4. Жалдак М. І., Рамський Ю.С. Інформатика: Навчальний посібник / За ред. М.І. Шкіля. – К.: Вища школа, 1991.

5. Лорьер Ж. Системы искусственного интеллекта. – М.: Мир, 1991.

6. Малпас Дж. Реляционный язык Пролог и его применение: Пер.с англ. /Под редакцией В.Н.Соболева. - М.: Наука. Гл.ред.физ.-мат.лит., 1990.

7. Марселлус Д. Программирование экспертных систем на Турбо-Прологе: Пер. с англ./ Предисл. С.В.Трубицына. - М.: Финансы и статистика, 1994.

8. Рамський Ю.С., Балик Н.Р. Методичні основи вивчення експертних систем у школі.- Київ: Логос, 1997.

9. Янсон А. Турбо-Пролог в сжатом изложении: Пер. с нем. /Пер. с нем. / Под ред.канд.физ.-мат.наук Ю.А.Бухштаба. - М.: Мир, 1991.

ЗМІСТ

Навчальне видання

Спірін Олег Михайлович

ПОЧАТКИ ШТУЧНОГО ІНТЕЛЕКТУ

Рекомендовано Міністерством освіти і науки України як навчальний посібник

Житомир,
Вид-во Житомирського державного університету імені Івана Франка

Надруковано з оригінал-макета автора.

Підписано до друку 11.10.2004. Формат 60x90/16. Ум.друк.арк.10.25. Обл.вид.арк. 10.75.
Гарнітура тип Petersburg, Arial. Зам. 227. Наклад 300.

Видавництво Житомирського державного університету імені Івана Франка,
м. Житомир, вул. Велика Бердичівська, 40