МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ
УНІВЕРСИТЕТ

Інженерно-енергетичний факультет
Кафедра вищої та прикладної математики

ОСНОВИ ШТУЧНОГО ІНТЕЛЕКТУ

контрольні завдання та методичні рекомендації для
виконання самостійної роботи
здобувачами початкового рівня (короткий цикл) вищої освіти
ОПП "Комп'ютерні науки"
спеціальності 122 " Комп'ютерні науки "
денної форми здобуття вищої освіти

Миколаїв
2023

УДК 004.89
О-75

Друкується за рішенням науково-методичної комісії інженерно-енергетичного факультету Миколаївського національного аграрного університету (протокол № 7 від 27.02. 2023 р.)

Укладачі:

Є. Ю. Борчик – канд. фіз.-мат. наук, доцент кафедри вищої та прикладної математики, МНАУ.

.

Рецензенти:
В.В. Зосімов – д.т.н., професор, завідувач кафедри інформаційних технологій, Миколаївський національний університет ім. В. О. Сухомлинського;
Ю.В. Волосюк – к.т.н, доцент, завідувач кафедри інформаційних систем і технологій , Миколаївський національний аграрний університет.

ВСТУП

Знання загальних концепцій та методів логічного програмування, зокрема мови Пролог, дозволяє ефективно вирішувати завдання штучного інтелекту, пов'язані з обробкою символьної інформації, побудовою систем підтримки прийняття рішень, експертних систем.

У методичних рекомендаціях запропоновано завдання для самостійної роботи студентів за програмою з основ штучного інтелекту для здобувачів початкового рівня (короткий цикл) спеціальності 122 “Комп’ютерні накуки”.

Матеріал даного методичного посібника складається з основних теоретичних відомостей, прикладів вирішення завдань та завдання з програмування мовою Пролог. У посібнику розглядаються такі теми:

Тема 1. Структура програми на Visual Prolog.

Тема 2. Управління пошуком рішень на Пролозі.

Тема 3. Організація циклічних операцій на Пролозі.

Тема 4. Організація циклічних операцій на Пролозі.

Тема 5. Внутрішні бази даних.

Тема 6. Створення програми в Пролозі.

Тема 7. Робота з файлами у Пролозі.

Тема 1. Структура програми на Visual Prolog

1.1 Основні теоретичні відомості

Програма на Пролозі складається із тверджень (clauses). Є два види тверджень: факти та правила.

Факт – це твердження, яке свідомо істинне. Наприклад:

likes(elena, tennis). /*Ліні подобається теніс*/

likes(tom,swimming). /*Тому подобається плавати*/

student(dima). /*Дима – студент*/

Кожен факт є відношенням, що пов'язує об'єкти. Символічне ім'я відношення називається ім'ям предикату чи предикатом. Об'єкти відносини називаються аргументами. У наведених прикладах likes – ім'я предикату, elena та tennis – аргументи предикату.

Правило в Пролозі використовується для отримання висновку на основі фактів, що є в програмі. Наприклад:

ikes(bill,X) if likes(tom,X). /* Біллу подобається щось, що подобається Тому */

Правила у Пролозі складаються з трьох частин:

–заголовка;

– слова if або символів ":-";

– тіла затвердження.

Заголовок – це факт, який має бути істинним, якщо істинно певна кількість умов.

Слово if або послідовність символів ":-" відокремлюють заголовок від тіла затвердження.

Тіло твердження – це безліч умов, які мають бути істинними, якщо потрібно довести істинність заголовка. Умови можуть бути об'єднані словом and (і) або or (або). Замість слова and часто використовують кому, а замість слова or – точку з комою. Заголовок правила також називають лівою частиною правила, а умови, що йдуть за символом ":-" - правою частиною правила.

Змінні в Пролозі використовуються для позначення невизначених об'єктів. Ім'я змінної повинно починатися з великої літери латинського алфавіту або символу підкреслення "_", після яких можуть йти будь-яке число букв, цифр і символів підкреслення. Змінні з такою позначенням називаються іменованими. Якщо змінна позначена одним символом

підкреслення "_", вона є анонімною. Анонімна змінна використовується, коли її значення не становить інтересу. Змінні у Пролозі локальні. Наприклад, якщо два твердження містять змінну X, це дві різні змінні.

Програма на Пролозі складається із розділів. Основні розділи:

domains /* розділ опису доменів*/

predicates /*розділ опису предикатів */

goal /* цільове затвердження */

clauses /* розділ тверджень */

constants /* розділ констант */

Інші розділи будуть розглянуті пізніше. У програмі можуть бути відсутні розділи. З іншого боку програма може містити декілька розділів constants, domains, predicates, або clauses. Константи, домени та предикати мають бути визначені до їх використання. Коментарі у програмі обрамляються символами /* та */. Коментарі можна розміщувати у будь-якому місці програми.

Головною частиною програми є **розділ тверджень**. У розділі clauses перераховуються всі факти та правила, що є у програмі. Кожне твердження закінчується крапкою. Усі твердження, які описують той самий предикат, повинні йти одне за одним.

У **розділі опису предикатів** (predicates) визначаються імена предикатів і типи його аргументів. Якщо в програмі використовується вбудований предикат, його описувати не потрібно.

Ім'я предикату повинне починатися з літери. Далі можуть слідувати літери (латинські, великі та малі), цифри і знак підкреслення в довільній послідовності. Не рекомендується використовувати великі літери як першу літеру імені предикату (інші версії Прологу це не дозволяють). Назва може мати довжину до 250 символів.

Типи аргументів перераховуються через кому і обрамляються круглими дужками. Типами аргументів предикату можуть бути або стандартні домени або домени, оголошені в секції domains. На відміну від пропозицій у секції clauses опис предикату не закінчується крапкою.

Наприклад, можливий наступний опис предикату likes:

predicates

likes(symbol,symbol)

Для одного предикату може бути кілька описів. Наприклад,

predicates

likes(name,activity)

likes(name,thing)

У разі домени name, activity, thing визначаються користувачем.

Арністю предикату називається кількість аргументів, які має. Можна оголосити предикати з однаковими іменами, але різними арнощами:

predicates

student(name)

student(name, university)

Коли для предикату є кілька описів, вони повинні йти один за одним.

Якщо аргументи, що використовуються вами, не належать до стандартних типів даних, їх тип повинен бути описаний в секції доменів. Домени в Пролозі подібні до описів типів у Паскалі. Ви можете давати власні імена різним типам даних. Наприклад:

domains

name,activity=symbol /* ім'я, рід діяльності */

predicates

likes(name,activity)

Таке оголошення зазвичай використовується для синтаксично схожих об'єктів, але семантично розрізняються. Використання спеціальних доменів покращує розуміння програми та дозволяє Прологу контролювати домени, але при цьому аргументи з типами спеціальних доменів не можуть змішуватися. Незважаючи на те, що name та activity – symbol, ці аргументи належать різним типам даних.

У наведеній нижче таблиці наведено основні стандартні домени Прологу.

Домен	**Можливі значення**
Short	-32768 .. 32767
Ushort	0 .. 65535
long	-2147483648 .. 2147483647
ulong	0 .. 4294967295
integer	-2147483648 .. 2147483647
unsigned	0 .. 4294967295
byte	0 .. 255

word	0 .. 65535
dword	0 .. 4294967295
char	Символ, укладений у поодинокі лапки: 'a'.
Real	Речові числа записуються у вигляді знака, мантиси, десяткової точки, дробової частини, символу e і показника – без пробілів. Знак, дробова частина та показник необов'язкові. Приклади чисел: 2.5 -32769 4e-23 Якщо потрібно, цілі значення автоматично перетворюються на речові.
string	Послідовність латинських літер, цифр і знаків підкреслення, що починається з символу в нижньому регістрі, або послідовність будь-яких символів, укладених у подвійні лапки. Наприклад: my_name1 "Олена" "123000 Minsk" Рядки, які записуються в тексті програми, не повинні бути довшими за 255 символів. Рядки, які вводяться з файлу або обчислюються, практично мають необмежену довжину (до 4G).
symbol	Синтаксис такий самий, як у рядків.

Домени string та symbol по-різному зберігаються та обробляються, хоча зовні практично не відрізняються. Наприклад, значення типу symbol дуже швидко порівнюються. При виборі слід керуватися міркуваннями ефективності програми.

Про інші можливості опису доменів буде розказано пізніше.

У розділі Goal формулюється призначення програми, тобто. запит, що дозволяє отримати відповідь системи. Якщо ціль є фактом, то Турбо-Пролог відповідає True або False; якщо мета містить змінні, то Турбо-Пролог видає або їх значення, які призводять до рішення, якщо воно існує, або повідомлення No Solution (Немає рішень). Наприклад, на запит

Goal:

likes(X,tennis) /* Кому подобається теніс?*/

система дасть відповідь наступного виду: X = elena, а на запит

Goal:

likes(_,tennis) /* Чи є хтось, кому подобається теніс ?*/

система дасть відповідь Yes.

Цільове твердження може складатися з кількох частин. У цьому випадку ціль є складовою, і кожна частина такої мети називається підціллю. Підцілі можуть бути сформовані як кон'юнкції або диз'юнкції.

У розділі constants можна дати імена символічним константам. Визначення константи має вигляд:

<ім'я> = <визначення костанти>

де <ім'я> - ім'я символічної константи, <визначення константи> - значення константи. На рядку може бути лише один опис константи. Перед компіляцією Пролог замінить у програмі імена констант на відповідні рядки. Приклад розділу констант:

```
constants
pi = 3.141592653
path = "c:\tprolog\bgi"
hundred = (10 * (10-1) + 10)
```

Тоді фрагмент програми на Пролозі

… A = hundred * pi …

буде оброблений компілятором так:

… A = (10 * (10-1) + 10) * 3.141592653 …

Унифікація в Турбо-Паскалі

Змінні у Пролозі можуть бути вільними чи пов'язаними. Змінна є вільною, доки вона не набула якогось значення. Змінна, яка має значення, називається пов'язаною.

Основою процедури логічного висновку є процес зіставлення двох термів і привласнення вільним змінним, що входять до них, таких значень, які зроблять ці терми ідентичними. Цей процес називається уніфікацією. Алгоритм уніфікації знаходить необхідні значення змінних, якщо вони існують, або повідомляє про відмову, якщо таких значень немає. Наприклад, likes(X,tennis) і likes(elena,tennis) уніфікуються при X=elena, а likes(X,tennis) і likes(tom,swimming) не уніфіковані.

При порівнянні двох термів Турбо-Пролог дотримується наступних правил:

* однакові константи можна порівняти один з одним;

* якщо змінна вже пов'язана, вона діє також, як нормальна константа;

* вільна змінна зіставляється з константою або з раніше пов'язаною змінною і стає пов'язаною з відповідним значенням;

Дві вільні змінні зв'язуються один з одним, з цього моменту вони трактуються як одна змінна: якщо одна з них набуває будь-якого значення, то друга теж набуває цього ж значення;

* вільна змінна то, можливо пов'язані з складовим об'єктом;

* два складових терма можна порівняти, коли вони мають один і той же функтор і однакову кількість аргументів; у разі попарно зіставляються аргументи термів.

Як приклад уніфікації складових об'єктів розглянемо уніфікацію об'єктів birthday(Name,date(_, _, Y)) і birthday(person(“Іван”,”Петров”), date(“Серп.”,2,1980)). У разі змінна Name отримає значення person(“Іван”,”Петров”), а змінна Y - значення 1980.

Турбо-Пролог виконує уніфікацію у двох випадках:

коли мета зіставляється із заголовком речення;

коли використовується знак рівності, який є інфіксним предикатом (предикатом, розташованим між своїми аргументами, а чи не перед ними).

Наприклад, розглянемо мету

goal

P1 = birthday(person("Іван","Петров"), date("Авг.",2,1980)),

P1 = birthday(Name,date(_, _, 1980)), write(Name).

При узгодженні першої підцілі змінна P1 отримає значення, вказане праворуч від символу “=“. За узгодження другої підцілі P1 вже пов'язана. Так як терми, що знаходяться по обидва боки знака "=" співставні, змінна Name буде пов'язана зі значенням person("Іван", "Петрів"). При узгодженні третьої підцілі, стандартного предиката write, буде надруковано значення пов'язаної змінної Name.

Пошук із поверненням

На відміну від процедурних мов, таких як Бейсік або Паскаль, у яких програміст повинен точно описати процес вирішення завдання, Турбо-Пролог є описовою (декларативною) мовою. Програма на Пролозі містить опис проблеми, для цього використовуються правила і факти та мета. Виконання програми полягає у спробі довести цільове затвердження, використовуючи припущення, задані у програмі. Механізм пошуку рішень “вбудований” у Пролог, він називається “пошук із поверненням”.

Нагадаємо, що кожен факт і кожне правило у програмі називається твердженням. Правило - це твердження, що має заголовок та тіло. Факт - це

твердження, що має заголовок та порожнє тіло. Сукупність тверджень, заголовки яких мають однаковий функтор, називається процедурою.

Основні принципи механізму пошуку рішень у Пролозі:

- Підцілі узгоджуються по порядку, зверху донизу, зліва направо.
- Для кожної мети або підцілі Пролог переглядає затвердження в порядку, в якому вони з'являються в програмі.
- Мета вважається узгодженою, якщо вона зіставляється із заголовком будь-якого затвердження процедури. При узгодженні мети з процедурою, має порожнє тіло (факт), мета узгоджується негайно. Якщо узгодження мети відбувається із заголовком затвердження, то мета узгоджується лише тоді, коли кожна підціль у тілі цього правила буде узгоджена після виклику її як поточну мету.
- Присвоювання значень змінним виконується внутрішніми програмами уніфікації щоразу коли вільна змінна зіставляється з константою. У цьому випадку змінна набуває значення константи. Змінна знову стає вільною, коли зіставлення виявляється неуспішним.
- Якщо спроба узгодження мети виявилася невдалою, Пролог виконує відкат для визначення альтернативних шляхів обчислення мети або підцілі. При знайденні рішень Пролог завбачливо встановлює невидимі покажчики – точки відкату у місцях, де можливий більш як шлях узгодження мети.

Розглянемо програму Prim0403.pro, яка містить відомості про імена та віки кількох гравців у тенісному клубі:

```
domains
   child = symbol
   age = integer
predicates
   player(child, age) - nondeterm (o,i), nondeterm (i,i)
/*гравець(дитина,вік)*/
clauses
   player(peter, 9).
   player (Paul, 10).
   player(chris, 9).
   player(susan, 9).
```

Ціль полягає в організації двох ігор для кожної пари гравців, вік яких не перевищує 9 років.

Це цільове твердження можна сформулювати так:

Goal:

player(Person1,9),player(Person2,9),Person1<>Person2.

На першому етапі Пролог намагається знайти рішення для першої підцілі player(Person1,9). Ця підціль узгоджується шляхом зіставлення Person1 з peter, оскільки Пролог переглядає затвердження для узгодження зверху донизу.

Після досягнення першої підцілі Пролог переходить до другої підцілі: player (Person2, 9). Ця подцель знову узгоджується зі зіставленні Person2 з peter.

На наступному етапі Пролог переходить до узгодження третьої підцілі: Person1<>Person2. Оскільки Person1 і Person2 мають значення peter, то дана підціль не узгоджується, і Пролог здійснює відкат до попередньої підцілі player(Person2,9). Ця подцель зіставляється з фактом player(chris,9), і Person2 узгоджується з crhis .Тепер Пролог знову як поточної підцілі виконує мету Person1<>Person2 . Ця підціль успішна, тому що peter та chris відмінні. Таким чином, Пролог погодив усі підцілі цільового затвердження, і ціль вважається узгодженою. Отримано першу пару гравців: Peter-Chris.

Проте, Пролог під час виконання зовнішнього цільового твердження має знайти всі можливі рішення цільового твердження. І тому він повертається до попередньої мети, звільняючи змінну Person2. Оскільки player(Person2,9) узгоджується з player(susan,9), то Пролог знову перетворюється на третьої подцели . Узгодження підцілі завершується успішно, отже друге рішення цільового затвердження знайдено (пара Peter-Susan).

Для пошуку наступного рішення Пролог повертається до другої підцілі, але оскільки для цієї підцілі всі варіанти вичерпані, Пролог виконує відкат до першої підцілі. Змінна Person1 стає вільною і може бути узгоджена з Chris. Для узгодження другої підцілі Пролог виконує пошук зверху вниз і зв'язує змінну Person2 з peter. Третя підціль завершується успіхом, оскільки chris і peter різні. Цільове твердження вважається узгодженим і наступна пара гравців Chris-Peter.

У пошуках ще одного рішення цільового затвердження Турбо-Пролог повертається до другої підцілі. При виконанні відкату змінна Person2 стає вільною і при уніфікації зв'язується з chris. Проте, Person1 і Person2 у разі

еквівалентні, і третя підціль завершується неуспіхом. Пролог знову виконує пошук із поверненням, і змінна Person2 зіставляється зі susan. Третя підціль виконується, отже отримано четверте рішення для цільового затвердження (пара Chris-Susan).

І знову для пошуку всіх рішень Пролог повертається до другої підцілі, але цього разу безуспішно. Отже, Пролог знову виконує відкат до першої підцілі, і вільна змінна Person1 зіставляється зі susan. Далі виконується узгодження другої підцілі, та змінна Person2 зіставляється з peter. Третя підціль успішна, отже, отримано п'яте рішення (Susan-Peter).

На наступному кроці Пролог виконує повернення до другої підцілі, і Person2 зіставляється з Chris. Таким чином отримано шосту пару: Susan і Chris.

Останнє досліджуване рішення завершується неуспіхом, оскільки Person1 та Person2 ідентичні.

1.2 Завдання для самостійного розв'язання

Завдання

Запустіть Visual Prolog.

Тема 2. Управління пошуком рішень на Пролозі

2.1 Основні теоретичні відомості

Відсікання

Для переривання пошуку з поверненням у Пролозі використовується предикат cut або відсікання (позначається !). Цей предикат, обчислення якого завжди завершується успішно, міститься в тіло правила як будь-яка інша мета. Після того, як процес пройшов через відсікання, вже не можна виконати пошук із поверненням у підцілях, розташованих в предикаті, що обробляється перед відсіканням, а також не можна повернутися до інших тверджень, що визначають предикат, що містить відсікання.

Відсікання застосовується у двох випадках:

• Коли ви знаєте, що пошук альтернативних рішень не приведе до розумного результату. Це марна трата часу та пам'яті. Якщо ви використовуєте відсікання в цій ситуації, програма буде працювати швидше і використовувати менше пам'яті. Такі відсікання називаються зеленими.

• Коли логіка програми потребує відсікань, щоб запобігти розгляду альтернативних подцелей. Такі відсікання називаються червоними.

Приклад:

Нехай у програмі є відомості про марки автомобілів, їх ціну та дату випуску. Визначимо предикат pr_inf, які друкує "підходить", якщо машина випущена в 2003 році, і "не підходить" - інакше.

```
domains
    name=string /*марка автомобіля*/
   price,year=integer /*ціна, рік випуску */
predicates
   nondeterm car(name,price,year)-(i,o,o),(o,o,o),(o,o,i)
   pr_inf(year) - multi (i)
clauses
   car("Шевролє Ланос", 2500,2005).
   car("Шевролє Авео",2000,2005).
   car("ВАЗ 2108",1400,2004).
   car("ВАЗ 2108",1300,2003).
   car("ВАЗ 2109",1600,2003).
   car("Таврія", 1000,2002).
   pr_inf(Y):-Y=2003,write(" підходить"),nl.
   pr_inf(_):-write("не підходить"),nl.
```

Тут nl – стандартний предикат Прологу. Під час друку він посилає на екран комбінацію спецсимволів "повернення каретки" та "переведення рядка", тобто переводить курсор на початок наступного рядка.

Якщо як мета задати car("ВАЗ 2109",_,Y),pr_inf(Y), то вийде дивне рішення: буде надруковано "підходить" і "не підходить". Це сталося через те, що знайшовши потрібну машину і надрукувавши "підходить", Пролог намагається знайти всі можливі рішення і переглядає альтернативи pr_inf, що залишилися. Щоб зупинити подальший пошук, коли рішення знайдено, потрібно використати відсікання. Замінимо перше правило для pr_inf на:

```
pr_inf(Y):-Y=2003,!,nl,write(" підходить").
```

У цьому випадку програма працюватиме правильно. Можна було б поставити відсікання і після nl або write, оскільки ці підцілі не мають альтернатив. Програма працюватиме так само. Якщо ж поставити відсікання перед перевіркою Y=2003, то ні для яких значень Y не може бути надруковано "не підходить", тому що відсікання забороняє повернення до інших тверджень для pr_inf після того, як з'ясується, що рік випуску не дорівнює 2003 року.

Програмування операцій, що повторюються

Дуже часто в програмі необхідно виконати те саме завдання кілька разів. У Пролозі операції, що повторюються, зазвичай виконуються за допомогою певного способу побудови правил.

Існують два способи реалізації правил: повторення та рекурсія. Правила, що виконують повторення, використовують відкат, а правила, що виконують рекурсію, використовують самовиклик.

При реалізації відкату у програмах застосовуються різні методи. Зокрема, Пролог має спеціальний предикат fail, який завжди викликає аварійне завершення і, отже, призводить до відкату. Дія предикату fail дорівнює ефекту від порівняння 2=3 або іншої неможливої підцілі. Зверніть увагу, що поміщати підціль після fail у тілі правила безглуздо, оскільки fail завжди закінчується невдало.

Покажемо, наприклад, як за допомогою предикату fail можна знайти всі можливі розв'язки задачі. Розглянемо попередню програму, додавши до неї предикат pr_car та мету, вказану нижче.

```
predicates
     pr_car
clauses
    pr_car:-car(Name,_,_),write(Name),nl,fail.
    pr_car.
goal
    pr_car,write("Інших автомобілів немає"),nl.
```

Зверніть увагу, що для визначення предикату pr_car використовуються два твердження. Їхній порядок у цьому випадку дуже важливий! Перше правило закінчується предикатом fail, який завжди спричиняє відкат. Вперше змінна Name отримає значення " Шевролє Ланос ", яке буде надруковано. Потім станеться відкат, і за відсутності альтернатив у nl і write буде знаходитися інше рішення для предикату car. При цьому спочатку звільняється змінна Name, потім є альтернативне рішення для предикату car, і змінна Name зв'язується з новим значенням. Зрештою, будуть переглянуті всі альтернативи. Тоді і буде виконано друге твердження для предикату pr_car. Якби його не було, то pr_car завжди завершувався б невдало, і отже слова "Інших автомобілів немає" ніколи б не надрукувалися.

Заперечення

Стандартний предикат Турбо-Прологу not (заперечення) дозволяє задати як підціль заперечення деякого твердження. Його вигляд

not(<затвердження>)

Цей предикат успішний, якщо <затвердження> є метою, що виявилася за доказом неуспішною.

Твердження всередині не може не містити вільних змінних, але може містити анонімні змінні.

2.2 Завдання для самостійного розв'язання

Завдання

1. Виконайте наведену вище програму. Розберіться, як вона працює.

Змініть предикат pr_car так, щоб він виводив на екран машини із заданим роком випуску. У розділі опису предикатів у разі його потрібно описати, наприклад, так:

multi pr_car(year) - (i), (o)

Випробуйте різні цілі: надрукувати назви машин, випущених 2011 року; надрукувати назви машин, випущених 2005 року; надрукувати назви всіх машин. Поясніть отриманий результат.

2. Визначте у наведеній нижче програмі предикат "звичайний студент". Студент звичайний, якщо має середній бал <=4 і любить лекції.

Використовуючи цей предикат, виведіть на екран прізвища незвичайних студентів.

Вказівка: Використовуйте предикат not.

```
domains
   name = symbol
   ball = real
predicates
   nondeterm ordinary_student(name)
   nondeterm student(name,ball) - (o,o),(i,o)
   nondeterm like_lecture(name)
clauses
student("Іванов Іван", 3.5).
student("Сидорова Марія", 4.5).
student("Петров Петро", 3.9).
like_lecture("Сидорова Марія").
like_lecture("Іванов Іван").
```

3. Виконайте наведені програми. Поясніть, чим вони відрізняються

Варіант 1	Варіант 2

predicates
  action(integer) - nondeterm (i)
 	
clauses
  action(1):-
  nl,
  write("Вы ввели 1."),nl.
  action(2):-
  nl,
  write("Вы ввели два."),nl.
  action(3):-
  nl,
  write("Была введена цифра 3."),nl.
  action(N):-
  nl,
  N<>1, N<>2, N<>3,
  write("Я не знаю такого числа!").
goal
  write("Введите число от 1 до 3: "),
  readint(Num),
  action(Num).
	predicates
	  action(integer) - procedure (i)
	
	clauses
	  action(1):-!,
	  nl,
	  write("Вы ввели 1.").
	  action(2):-!,
	  nl,
	  write("Вы ввели два.").
	  action(3):-!,
	  nl,
	  write("Была введена цифра 3.").
	  action(_):-nl,
	  write("Я не знаю такого числа!").
	
	goal
	  write("Введите число от 1 до 3: "),
	  readint(Num),
	  action(Num),nl.

Поясніть:

яка програма ефективніша і чому;

чи важливий порядок правил для предикату action;

може предикат action закінчитися fail;

скільки рішень має предикат action;

який тип відсікань ми маємо в даному випадку: червоні чи зелені.

Тема 3. Організація циклічних операцій на Пролозі

3.1 Основні теоретичні відомості

Використання предикату repeat

Ще один метод, що реалізує циклічні операції, заснований на використанні предикату наступного виду:

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

Такий предикат забезпечує безліч різних рішень (оскільки визначений рекурсивно), і дозволяє застосовувати пошук із поверненням у тих випадках, коли в задачі немає інших альтернативних правил. Можна замість repeat написати інше ім'я. Для пояснення використання предикату розглянемо такий приклад.

Приклад:

Нижче наведено програму, в якій символи, що вводяться з клавіатури, виводяться на екран. Ознакою кінця введення є натискання клавіші Enter.

```
predicates
  repeat - nondeterm()
  typewriter - nondeterm()
clauses
  repeat.
  repeat:- repeat.
typewriter:-
   repeat,
   readchar(C),  /* предикат зчитує символ з клавіатури */
   write(C),
   C = '\r'.          /* порівнюємо значення з клавішею Enter */
goal
   write ("Вводьте символи, в кінці натисніть Enter:"),nl,
   typewriter,nl.
```

Розглянемо докладніше, як працює typewriter. Спочатку виконується repeat, який нічого не робить і завжди завершується успішно. Потім предикат readchar(C) зчитує черговий символ і присвоює його змінною, а предикат write(C) виводить цей символ на екран. Якщо введений символ не є клавішею Enter, то підціль C = 'r завершується неуспішно, і відбувається відкат до останньої точки повернення. Так як readchar і write не мають альтернатив, відбувається повернення до repeat, який завжди має альтернативні рішення. Якщо натиснуто клавішу Enter, підціль успішна, і предикат завершує роботу.

Методи організації рекурсії

Правило, що містить саме себе як компонент, називається *правилом рекурсії*. Рекурсивні правила одна із способів організації циклічних процесів (опрацій, що повторюються).

Прикладом рекурсивної процедури може бути завдання зчитування символів з клавіатури. Ознакою завершення введення є символ #.

Приклад:

```
predicates
    write_ch - nondeterm()
    read_ch - nondeterm()
clauses
    write_ch:-write("Введіть символи" ),nl,
    write("Для завершення введіть символ #"),nl,nl.
    read_ch:-readchar(C),
    C<>'#',!,
    write(C),
    read_ch.
read_ch.
goal
write_ch, read_ch, nl.
```

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

Правило рекурсії обов'язково має містити умову виходу. Інакше рекурсія нескінченна.

У рекурсії є один недолік - щоразу, коли одна процедура викликає іншу, інформація про процедуру, що викликає, повинна бути збережена для того, щоб відновити виконання процедури на тому ж місці, де вона зупинилася. Інформація зберігається в стеку, який може переповнитися при великій кількості дзвінків. Тому особливий інтерес викликає випадок, коли після завершення роботи викликаної процедури, яка викликає процедуру, нічого не треба робити. У цьому випадку інформацію про процедуру, що викликає, можна не зберігати. При кожному новому виклику достатньо надати аргументам нові значення. Ця операція називається оптимізацією

хвостової рекурсії чи оптимізацією останнього виклику. Вона можлива, якщо виконуються такі умови:

• рекурсивний виклик є останньою підціллю пропозиції;

• раніше у реченні не було зворотних точок.

3.2 Завдання для самостійного розв'язання

Завдання

1. Виконайте наведену вище програму. Розберіться, як вона працює. Чи можна у предикаті repeat поміняти місцями правила? Поясніть, чому.

Змініть предикат typewriter так, щоб при натисканні клавіші друкувався не тільки символ, але і його код ASCII. Використовуйте стандартний предикат char_int().

Переробіть програму так, щоб вона закінчувала роботу при натисканні клавіші Esc (код клавіші 27).

2. Інтерпретація Турбо-Прологом оператора "=" залежить від того, чи відомі обидва значення чи ні. Якщо обидва значення відомі, то оператор інтерпретується як оператор порівняння, навіть якщо обидва змінні терма. Якщо відоме лише одне значення, воно стає значенням вільної змінної, незалежно від цього з якого боку від знака “=“ вона (аналог оператора присвоєння іншими мовами). Наприклад, вираз Z = B*B - 4*A*C ставить у відповідність Z (якщо Z вільна) обчислене значення, чи поверне Yes/No, якщо Z вже пов'язана. Змінні A, B, C повинні мати значення (обов'язково!).

Розглянемо інший приклад. У мові Паскаль після виконання операторів Z: = 1; Z:=Z+1; Змінна Z отримає значення 2. У Турбо-Пролозі виконання підцілей Z = 1, Z = Z +1 закінчиться невдало, в результаті відбудеться відкат і звільнення змінної Z. Справа в тому, що в Пролозі не можна змінити значення змінної. При необхідності потрібно створити нову змінну, і присвоїти їй нове значення (див. приклади нижче).

Розгляньте різні варіанти програми обчислення факторіалу, наведені нижче. Розберіться, як працює програма.

Варіант 1	Варіант 2	Варіант 3
predicates factorial(unsigned,real) - procedure (i,o) clauses factorial(1,1.0):-!. factorial(X,FactX):-	predicates factorial(unsigned,real) - determ (i,o) factorial_aux(unsigned,real,unsigned, real) - determ (i,o,i,i) clauses	predicates factorial (unsigned,real) - procedure (i,o) Factorial (unsigned,real,unsigned, real) - procedure (i,o,i,i) clauses factorial(N,FactN):-

```
    Y=X-1,
   factorial(Y,FactY),
   FactX = X*FactY.

goal
  X=3,
  factorial(X,Y).
```

```
     factorial(N,FactN):-
      factorial_aux(N,FactN,1,1.0).
  factorial_aux(N,FactN,I,P):-
       I <= N,!,
       NewP = P * I,
       NewI = I + 1,

factorial_aux(N,FactN,NewI,NewP).
  factorial_aux(N,FactN,I,P) :-
         I > N,
         FactN = P.
goal
  factorial(3,X).
```

```
    factorial(N,FactN,1,1.0).

  factorial(N,FactN,N,FactN):-!.
  factorial(N,FactN,I,P):-
    NewI = I+1,
    NewP = P*NewI,
    factorial(N,FactN,NewI,NewP).
goal
  factorial(3,X).
```

Поясніть:

• чому другий аргумент предикату factorial має тип real;

• у яких випадках рекурсія є хвостовою;

• навіщо у другому та третьому варіанті потрібен ще один предикат;

• який варіант працює ефективніше.

3. Напишіть предикат, який:

a) для заданого n друкує значення n, n-1, … , 2, 1 (у спадному порядку);

b) для заданого n друкує значення 1, 2, …, n-1, n (у зростаючому порядку).

Тема 4. Використання списків на Пролозі

4.1 Основні теоретичні відомості

Використання списків у Пролозі

Рекурсивними в Пролозі можуть бути не лише затвердження, а й структури даних. Приклад такого типу даних є список.

Список – це рекурсивний складовий об'єкт. Він складається з двох частин: голови, якою є перший елемент, та хвоста, яким є список, що включає всі наступні елементи. Отже, хвіст списку завжди список (можливо порожній), голова списку завжди елемент.

У Пролозі список полягає у квадратних дужках. Для відокремлення голови списку від хвоста використовується знак вертикальної межі "|". Порожній список позначається [ ]. Порожній список не можна розділити на голову та хвіст.

Можна використовувати спрощене уявлення, у якому рекурсивна природа списків виявляється прихованою. У такому записі список

зображується у вигляді послідовності елементів, обмеженої квадратними дужками та розділеною комами. Порожній список, що застосовується для завершення рекурсивного опису, не вказується. Нижче наведено приклади списків. Записи у різних стовпцях еквівалентні.

[a|[]][a]

[a|[b|[ ]]][a|[b]][a,b]

[a|[b|[c|[ ]]]][a|[b|c]][a|[b,c]][a,b,c]

Елементами списку мають бути об'єкти того самого типу, наприклад, цілі числа, дійсні числа, символи, рядки, структури чи списки. Кількість елементів у списку називається його довжиною. Довжина списку може бути будь-яким, єдиним обмеженням є обсяг оперативної пам'яті.

Приклади списків (у першому стовпці наводиться список, у другому – його голова, у третьому – хвіст):

[1,2,3,4,5,6,7]1[2,3,4,5,6,7]

['a', 'b', 'c'] 'a' ['b', 'c']

[[1,2,3], [2,3,4], [ ]][1,2,3][[2,3,4], [ ]]

["Ліссабон"]"Ліссабон"[ ]

[] не визначено не визначено

Приклади уніфікації списків (у перших двох шпальтах наводяться списки, у третьому – результат зіставлення):

[X, Y, Z]["Іван", "їсть", "морозиво"]X="Іван", Y="їсть", Z="морозиво"

[7] [X | Y] X = 7, Y = []

[1, 2, 3, 4] [X, Y | Z] X = 1, Y = 2, Z = [3,4]

[1, 2] [3 | X]fail

Для опису списку в Пролозі використовується конструкція виду:

domains

objects=.....

objectlist=objects*

Зірочка при описі домену вказує на те, що об'єкт є списком.

Оскільки список – це об'єкт даних, він використовується в Пролозі як аргументи предикатів, наприклад pr_list([1,2,3,4]).

Для виконання більшості операцій над списками, таких як виведення на друк елементів списку, визначення приналежності елемента до списку, поділ списку на два, з'єднання списків і т.д., використовується поділ списку на хвіст і голову. Для відповідного предикату зазвичай є два твердження, одне з

яких є рекурсивним правилом обробки списку, а інше – гранична умова для завершення рекурсивної обробки списків. Для пояснення методів обробки списків розглянемо кілька прикладів.

Приклад 1:

```
/*Виведення на екран елементів списку */
domains
list_int=integer* /* список цілих чисел */
predicates
pr_list(list_int)
clauses
pr_list([]). /*Гранічна умова: якщо список порожній, то нічого не робити*/
pr_list([H|T]): -write(H),nl,pr_list(T). /* Рекурсивне правило: якщо список не порожній,
то розділити список на хвіст та голову та надрукувати голову.*/
goal
pr_list([4,-9,5,3]).
```

При виконанні цільового затвердження Пролог виконує пошук фактів і правил зверху вниз. Перший варіант правила (гранична умова) дає неуспіх, оскільки його об'єкт є порожнім списком. Цільове затвердження уніфікується із головою другого правила pr_list (рекурсивне правило). При цьому змінної H присвоюється значення першого елемента в списку (4), а змінної T ставиться у відповідність частина списку, що залишилася [-9,5,3]. Наступною підціллю стає предикат write(H). Після друку елемента та перекладу рядка (nl), поточною підціллю стає предикат pr_list(T). Для задоволення поточної підцілі Пролог переглядає перше речення, яке знову не підходить, а потім друге. Змінної H надається значення -9, а змінної T - [5,3]. Операція повторяться до тих пір, поки поточна підціль не набуде вигляду: pr_list([]). У цьому випадку підціль узгоджується з першою пропозицією. Цей варіант не має рекурсивних викликів, що призведе до згортання рекурсії.

Приклад 2:

Два варіанти предикату, що обчислює довжину списку.

domains list = integer*	domains list = integer*

predicates
  length_of(list,integer)
clauses
  length_of([], 0).
  length_of([_|T],L):-
  length_of(T,TailLength),
  L = TailLength + 1.
goal
  length_of([1, 2, 3], L).
	predicates
	  length_of(list,integer,integer)
	clauses
	  length_of([], Result, Result).
	  length_of([_|T],Result,Counter):-
	  NewCounter = Counter + 1,
	  length_of(T, Result, NewCounter).
	goal
	  length_of([1, 2, 3], L, 0).

Наведемо ще кілька предикатів, які реалізують різноманітні операції над списками.

```
domains
   list_int=integer* /* список цілих чисел */
predicates
   nondeterm member(integer,list_int) - (i,i), (o,i)
   nondeterm append(list_int,list_int,list_int) - (i,i,o), (o,i,i), (o,o,i)
   sort(list_int,list_int)
   insert(integer,list_int,list_int)
   asc_order(integer,integer)
clauses
        /* Пошук елемента у списку: перший аргумент - що шукаємо,
        другий аргумент - список, в якому шукаємо */
   member(X,[X|_]).
   member(X, [_|T] ):- member(X,T).
        /* Злиття списків: всі три аргументи - списки, третій список -
        результат злиття перших двох списків */
   append([],L,L).
   append ([N | L1], L2, [N | L3]): - append (L1, L2, L3).
        /* Сортування списків: перший аргумент - вихідний список,
        другий аргумент - відсортований за зростанням перелік.
        Реалізоване сортування вставкою.*/
   sort([],[]).
   sort([X|T],L):-sort(T,L1), insert(X,L1,L).
   insert(X,[Y|L],[Y|L1]): -asc_order(X,Y),!, insert(X,L,L1). /* вставляє
```

```
        перший аргумент у список, заданий як другий аргумент,
        результат - третій аргумент */
  insert(X,L,[X|L]).
  asc_order(X,Y):-X>Y. /* допоміжний предикат */
```

Якщо як ціль задати member(2,[1,2,3]), то Пролог відповість Yes (мета доведена). Для мети append([1,2],[3],X) буде знайдено рішення X=[1,2,3], а для мети sort([2,5,1,4],X) - рішення X = [1,2,4,5]. Однак це ще не все. Задамо як мета member(X,[1,2,3]), і Пролог знайде 3 рішення: X=1, X=2 і X=3. Аналогічно, якщо як мета задати append(X,[3],[1,2,3]), буде знайдено рішення X=[1,2].

У Пролозі нерідко предикат, створений на вирішення однієї завдання, може вирішувати інші. Стан аргументів під час виклику предикату називається потоком параметрів. Аргумент, який має значення в момент виклику, називається вхідним та позначається буквою (i), а вільний аргумент – це вихідний аргумент, що позначається буквою (o). Потоки параметрів для стандартних предикатів зазвичай вказуються у довідкових посібниках.

У практичних програмах часто буває необхідно зібрати дані до списку для подальшої обробки. З цією метою використовується вбудований предикат

findall(Var, Predicat, List_name),

де Var (ім'я змінної) означає об'єкт, який необхідно включити до списку, Predicat визначає предикат, з якого потрібно зібрати значення, а List_name – це ім'я змінної вихідного списку. Змінна Var має бути одним із аргументів предикату Predicat. Наприклад, Ви маєте предикат person (name, age) (людина (ім'я, вік)) і факти:

```
person(tom,26).
person(bill,28).
person(sue,20).
```

Припустимо, необхідно визначити список всіх імен, що є в базі даних. Ця операція при використанні вбудованого предикату findall буде виглядати так:

```
findall(Name,person(Name,_),Name_list).
```

Після виконання findall змінна Name_list отримає значення [tom,bill,sue].

4.2 Завдання для самостійного розв'язання

Завдання

1. Проаналізуйте наведені вище предикати. Розберіться, як вони працюють.

Чи можна поміняти місцями правила для предикату member? Спробуйте встановити мета member(X, [1, 2, 3, 4, 5]) в обох випадках.

Для предикату append випробуйте цілі:

append([1, 2, 3], [5, 6], L).

append([1, 2], [3], L), append(L, L, LL).

append(L1, L2, [1, 2, 4]).

append(L1, [3,4], [1,2,3,4]).

2. Напишіть предикати, які виконують такі дії:

- обчислює суму елементів списку;
- додає до всіх елементів списку 1;
- видаляє негативні елементи зі списку.

3. Визначте у своїй програмі предикат person, що містить відомості про прізвище, місце проживання (місто) та вік. Вкажіть кілька фактів. Напишіть програму, яка за фактами визначить середній вік мешканців зазначеного міста.

Тема 5. Внутрішні бази даних

5.1 Основні теоретичні відомості

Бази даних у Пролозі

Кошти, які є в Пролозі для організації баз даних (БД), розраховані на роботу з реляційними БД, оскільки Пролог можна використовувати як потужну мову запитів саме для реляційних БД. Внутрішні уніфікаційні процедури мови автоматично вибирають факти з потрібними значеннями відомих параметрів та надають значення невідомим параметрам. Алгоритм пошуку із поверненням дозволяє знаходити всі рішення для зробленого запиту.

База даних називається динамічною, якщо в процесі роботи в неї можна додавати нові твердження і видаляти будь-які твердження, що містяться в ній. Динамічна БД може бути записана на диск і зчитана з диска на оперативну пам'ять. Статичні БД неможливо знайти змінені у процесі виконання програми, оскільки затвердження такий БД є частиною програми.

Факти та правила, що містяться в розділі clauses програми на Пролозі, можна розглядати як статичну базу даних. Динамічними в Пролозі є внутрішні та зовнішні БД.

Внутрішня БД складається з фактів, які можна додавати та видаляти під час виконання програми.

Створення внутрішніх баз даних

Внутрішня БД Прологу може містити лише факти (не правила!). Факти, що належать до внутрішньої БД, використовуються як звичайні предикати. Як і інші предикати вони повинні бути оголошені до їх використання, але не в розділі predicates, а в спеціальному розділі facts або database. Наприклад:

```
database
   db_person(name,age,addres)
```

Можна при описі вказати ім'я БД, відокремивши його від database знаком "-". Якщо ім'я не задано, БД отримає стандартне ім'я dbasedom. Допускається наявність декількох секцій database, але тоді необхідно вказати ім'я кожної секції (за винятком можливо однієї). Це дозволяє зберігати та завантажувати секції бази даних окремо. У двох різних секціях database не можна використовувати однакові імена предикатів.

```
database - mydatabase
   likes(simbol,symbol)
database - люди
   person(string,string)
```

Факти для предикатів, описаних у розділі database (facts), під час розробки програми можуть бути поміщені у розділ clauses як завжди. Під час виконання програми їх можна видалити так само, як і додані факти пізніше.

За своєю природою предикати БД недетерміновані. Оскільки факти можуть бути додані під час виконання програми, компілятор завжди передбачає наявність альтернативного рішення під час пошуку із поверненням. Якщо у вас є предикат, для якого в БД ніколи не буде більше одного факту, його можна описати як determ. У цьому випадку до існуючого факту ви не зможете додати жодного факту, виникне помилка.

З фактами внутрішньої БД можна працювати як з термами, тому що за описом БД Пролог створює домен з ім'ям, що збігається з ім'ям секції database (або з ім'ям dbasedom, якщо секція БД не мала імені). До кожного предиката БД створюється одна альтернатива. Цей домен можна використовувати як будь-який інший домен. Для оголошеної БД people буде

створено домен people = person(string, string). Домен людей може використовуватися як будь-який інший домен. Наприклад, створимо предикат no_copy, який не дозволить додати до БД факту, якщо там такий факт вже є:

```
predicates
   no_copy (люди)
clauses
   no_copy (person(Name,Address)):-
      person(Name,Address), ! ;
      assert(person(Name,Address)).
```

Щоб додати до БД нові факти під час виконання програми, слід використовувати стандартні предикати.

```
asserta(<факт>)/* (i) */
asserta(<факт>,databaseName)/* (i,i)*/
assertz(<факт>)/* (i) */
assertz(<факт>,databaseName)/* (i,i)*/
assert(<факт>)/* (i) */
assert (<факт>,databaseName)/* (i,i)*/
```

Предикат asserta вставляє новий факт у БД перед наявними фактами для даного предикату, а assertz і assert поміщають нові твердження за наявними в БД твердженнями того ж предикату. Наприклад, assert(person("Михайло","111-11-11")) додасть факт person("Михайло","111-11-11") в кінець БД. Факти, що додаються до БД, що неспроможні містити вільних змінних.

Для видалення предикатів із БД використовуються стандартні предикати

```
retract(<факт>)/* (i) */
retract(<факт>,databaseName)/* (i,i)*/
retractall(<факт>)/* (i) */
retractall(<факт>,databaseName)/* (i,i)*/
```

Предикат retract видаляє перший факт вашої БД, що збігається з фактом <факт>, а retractall видаляє всі твердження, що збігаються із зразком <факт>. Наприклад, предикат retractall(person("Михайло",_)) видалить з БД всі факти person для Михайла, а retract(person("Михайло",_)) видалить перший факт для Михайла. Мета retractall(_,people) видаляє всі факти БД people.

У всіх наведених предикатах другий аргумент (ім'я БД) використовується тільки для перевірки типу, тому його можна опускати.

Для збереження у файлі БД, створеної в оперативній пам'яті, застосовуються предикати

save(fileName)/* (i) */

save(fileName,databaseName)/* (i,i)*/

При виклику предикату з одним аргументом зберігаються факти з БД dbasedom, при виклику з двома аргументами - факти із зазначеної БД. Наприклад, предикат save("d:\prolog\work\bd.dba",mydatabase) зберігає факти БД mydatabase у файлі bd.dba.

Для використання бази даних, збереженої у файлі, її можна завантажити в оперативну пам'ять за допомогою предикатів

consult(fileName)/* (i) */

consult(fileName, databaseName)/* (i,i)*/

Предикат consult зчитує з файлу факти і вставляє в кінець відповідної БД. Якщо предикат викликаний з одним параметром, будуть зчитані факти, які були описані в секції без імені. Наприклад, consult("d:\prolog\work\bd.dba", mydatabase) завантажить з файлу bd.dba факти БД mydatabase. Якщо файл містить щось крім фактів зазначеної БД, видається повідомлення про помилку. Факти зчитуються по одному, тому якщо з десяти фактів у шостому була помилка, то перші 5 фактів будуть занесені до БД. Предикат consult зчитує факти лише у тому форматі, у якому їх записує предикат save. Файли не повинні містити:

- великих символів;
- прогалин (крім рядків всередині "");
- коментарів;
- порожніх рядків;
- Символів без подвійних лапок.

5.2 Завдання для самостійного розв'язання

Завдання

1. Виконайте наведену нижче програму. Це приклад найпростішої експертної системи. Розберіться, як вона працює.

```
domains
thing = string
conds = cond *
```

```
cond = string

database /*facts*/
is_a(thing,thing,conds)
type_of(thing,thing,conds)
false(cond)

predicates
run(thing) - nondeterm (i)
ask(conds) - nondeterm (i)
update - procedure ()

clauses
run(Item):-
        is_a(X,Item,List),
        ask(List),
        type_of(ANS,X,List2),
        ask(List2),
        write(Item," який вам потрібен, - це", Ans), nl.

run(_):-
        write("Не вистачає даних"),
        write("для отримання висновків."),
        nl.

ask([]).
ask([H|T]):-
        not(false(H)),
        write("Це використовується для"),
        write(H," (натисніть y/n)"),
        readchar(Ans), nl, Ans='y',
        ask(T).

ask([H|_]):-
        assertz(false(H)), fail.
```

```
is_a("мова","Інструмент",["спілкування"]).
is_a("молоток","Інструмент",["будівлі будинку","ремонту паркану",
                              "колки горіхів"]).
is_a("швейна машина","Інструмент",["виготовлення одягу",
                                   "ремонту вітрил"]).
is_a("плуг","Інструмент",["оранки","обробки землі"]).

type_of("english", "мова", ["спілкування з людьми"]).
type_of("prolog","мова",["спілкування з комп'ютером"]).
type_of("молоток", "молоток", []).
type_of("швейна машина", "швейна машина", []).
type_of("плуг", "плуг", []).

update:-
        retractall(type_of("prolog","мова,["спілкування з комп'ютером"]))),
        asserta(type_of("PDC Prolog","мова",
                        ["Спілкування з персональним комп'ютером"])),
        asserta(type_of("prolog","мова",
                        ["Спілкування з універсальним комп'ютером"])).

goal
run("Інструмент").
```

2. Збережіть базу даних у файлі. Перегляньте отриманий файл, використовуючи, наприклад, редактор WordPad.

3. Змініть ціль, додавши перед предикатом run предикат update. Виконайте програму. Перегляньте отриману в результаті БД.

4. Видаліть із тексту програми факти БД is_a і type_of (або закоментуйте їх). Змініть ціль так, щоб база даних спочатку завантажувалася з файлу, потім виконувався предикат run і база даних знову зберігалася на диску.

5. Поясніть призначення фактів false. Передбачте видалення всіх фактів false з бази даних після її завантаження з диска.

6. Напишіть програму, яка дозволяє працювати з БД «Телефони». Приблизний текст наведено нижче. Допишіть предикати, які будуть додавати

інформацію до БД, видаляти інформацію про абонентів з БД та видавати телефони вказаних абонентів.

```
database
dbperson(name,phon)

predicates
nondeterm repeat
new_db

clauses
repeat.
repeat:-repeat.

new_db:-
repeat,
write("Введіть ім'я => "),
readln(Name) ,
write("Введіть телефон => "),
readln(Phon),
assertz(dbperson(Name,Phon)),
write(Name," доданий до БД"),nl,
write("Будете ще вводити дані? (натисніть y/n)"),
readchar(Ans), nl, Ans='n',!,
save("c:\wwww\my_db").

goal
new_db.
```

Тема 6. Створення програми в Пролозі

6.1 Основні теоретичні відомості

Створення програми

Для створення нових програм у Пролозі використовується Application Expert. Щоб його відкрити, виконайте команду меню Project | New Project. Перевірте, щоб на вкладці Target були встановлені наступні значення: Platform – Windows32, UI Strategy – VPI, Target Type – exe, Main Program – Prolog (вони мають бути встановлені за замовчуванням).

На вкладці General потрібно вказати ім'я проекту (воно одночасно є ім'ям файлу та ім'ям предикату, тому має відповідати відповідним вимогам) та каталог, у якому зберігатиметься проект. Рекомендується зберігати кожен проект у окремому каталозі. Якщо вказано неіснуючий каталог, він буде створено. Щоб переглянути дерево каталогів, натисніть кнопку Browse. Якщо ввімкнути опцію Multiprogrammer Mode, всі компоненти проекту зберігатимуться в окремих файлах, а також буде згенеровано .PRJ–файл та забезпечено відповідну підтримку для роботи з ним. Цю опцію можна змінити на будь-якій стадії розробки проекту. Щоб змінювати установки у існуючому проекті, потрібно знову відкрити Application Expert, виконавши команду Options | Project | Application Expert.

На вкладці VPI Options слід зазначити ті кошти, які потрібно включити до проекту. Всі ці установки крім "toolbar and the help line" пізніше можна змінити. Панель інструментів також можна додати пізніше, але код у цьому випадку доведеться додавати вручну, використовуючи Toolbar Editor та Toolbar Expert.

Після того, як усі опції будуть встановлені, натисніть кнопку Create. Вікно проекту міститиме 2 модулі VPITOOLS.PRO та MYPROJ.PRO (якщо ви вказали ім'я проекту MYPROJ). MYPROJ.PRO – головний модуль. Файл VPITOOLS.PRO включає засоби, вибрані в Application Expert. У головному модулі є такі розділи:

- керування подіями для вікна завдань (Task window – це головне вікно програми, яке створюється під час запуску програми та знищується при завершенні роботи програми);
- головна мета проекту;
- створення панелі інструментів та рядки статусу в проекті;
- створення вікна About.

Щоб відкомпілювати та виконати створений проект, натисніть на кнопку або виконайте команду меню Project | Run.

Усі компоненти вашої програми реєструються у вікні проекту. Кнопки зліва вікна дозволяють вибрати потрібний компонент. Кнопки праворуч призначені для виконання дій над цими компонентами. Використовуючи кнопку Module (ліворуч) та Edit (праворуч), перегляньте вміст створених модулів.

Зміна меню в меню Editor.

У вікні проекту натисніть кнопку Menu на лівій панелі інструментів. Далі виконайте подвійне клацання на Task Menu (головне меню програми), щоб активізувати редактор меню. Щоб додати новий пункт у меню, пересуньте курсор у потрібне місце та натисніть кнопку New. У полі Text вкажіть назву пункту меню, яке відображатиметься на екрані. Можна використати російські літери. Наявність '&' перед символом означає, що цей символ використовується для швидкого доступу до пункту меню. У полі Constant вкажіть ім'я пункту меню, яке буде використовуватись у програмі. За промовчанням це ім'я починається з id_ (не можна використовувати російські літери). Можна також встановити комбінацію гарячих клавіш і початковий стан меню. Щоб створити підменю, спочатку натисніть кнопку Submenu, а потім кнопку New. Натисніть кнопку Back, щоб повернутися до меню вищого рівня. Якщо натиснути кнопку Test, стандартне меню буде замінено на створене вами, і ви можете протестувати його. Повторне натискання кнопки Test поверне стандартне меню.

Додамо код, який потрібно виконати під час вибору доданого пункту меню. Для цього спочатку у вікні проекту виберіть Window, а потім у списку виберіть Task window і натисніть на кнопку Code Expert. На дисплеї з'явиться вікно Dialog and Window Expert. Після цього оберіть у списку Event Type елемент Menu, а у списку Event or Item – ім'я потрібного пункту меню. Натисніть кнопку Add Clause, щоб додати до програми обробку потрібної події. Назва кнопки зміниться на Edit Clause. Натисніть цю кнопку, щоб побачити текст, доданий до програми. Якщо пункт меню називався id_text, ви побачите текст:

```
%BEGIN Task Window, id_text
task_win_eh(_Win,e_Menu(id_text,_ShiftCtlAlt),0):-!,
    !.
%END Task Window, id_text
```

Коментарі %Begin - %End використовуються для локалізації коду, що належить до обраної компоненти інтерфейсу (у разі пункту меню id_text). Між двома відсіками вручну або за допомогою майстра додайте код, який потрібно виконати.

Як створити нове вікно

Натисніть кнопку Window у лівій частині вікна проекту, а потім на кнопку New у правій частині вікна. Введіть ім'я вікна (в полі Name) та заголовок (у полі Title). Натисніть кнопку OK. У цей момент з'являться панелі, що дозволяють додавати елементи у вікно та змінювати його розмір. Якщо ви не хочете нічого додавати до цього вікна, то просто закрийте його.

Створимо новий модуль, який надалі помістимо код для вікна. У вікні проекту натисніть кнопку Module у лівій частині вікна, а потім на кнопку New у правій частині вікна. Введіть назву файлу (наприклад, MyWinFile). Виберіть тип файлу *.pro зі списку. Натисніть кнопку Відкрити. Відобразиться діалогове вікно File Inclusions for Module. Натисніть OK, не змінюючи параметрів у цьому вікні.

Далі потрібно згенерувати код цього вікна. Цей код виконує дві функції: створює вікно із заданими атрибутами та містить предикати, керовані подіями, тобто забезпечує реакцію вікна на події. Виділіть потрібне вікно у вікні проекту та натисніть кнопку Code Expert. У полі Module виберіть створений модуль (MyWinFile.pro) і натисніть кнопку Default Code. Натиснувши кнопку Edit Code, ви побачите, який код був доданий у вказаний модуль.

6.2 Завдання для самостійного розв'язання

Завдання

1. Створіть нову програму з налаштуваннями за промовчанням. Запустіть його. Проаналізуйте створені файли. Змініть назву головного вікна програми. Для цього натисніть кнопку Window, потім на кнопку Attribute і в полі Window Title введіть нову назву вікна, наприклад, "Додаток на Пролозі".

2. Додайте до існуючого меню програми нове меню Тест, що містить пункт Вікно повідомлення. При виборі цього пункту меню на екран відображається стандартне вікно повідомлення із заданим текстом.

Як додати пункт меню описано вище. Щоб вивести вікно повідомлення, помістіть курсор у точку, в яку ви хочете вставити код, у нашому випадку перед другим відсіканням, та виконайте Edit | Insert | Predicate Call | Windows, Dialog або Toolbar. У вікні виберіть Common Dialog, у списку знайдіть dlg_Note і наберіть текст повідомлення, включаючи лапки, наприклад, "Привіт!". В результаті код матиме вигляд:

```
%BEGIN Task Window, id_text
```

```
task_win_eh(_Win,e_Menu(id_text,_ShiftCtlAlt),0):-!,
    Title="Title",
    dlg_Note(Title,"Привіт!"),
    !.
%END Task Window, id_text
```

Ви можете змінити текст заголовка вікна повідомлення. Перевірте роботу програми.

3. Створіть нове пусте вікно. Як це зробити описано вище.

Щоб перевірити роботу вікна, додайте в меню Тест новий пункт Моє вікно (id_mywin). Щоб вивести вікно на екран, помістіть курсор у точку, в яку потрібно вставити код, та виконайте Edit | Insert | Predicate Call | Windows, Dialog або Toolbar. У вікні позначте User Defined Window і у списку виберіть потрібне вікно. Натисніть кнопку ОК. У текст буде вставлено предикат, що створює вікно. Запустіть програму та перевірте роботу програми.

4. У MyWinFile.pro вставте наступне правило (3 рядки) для предикату win_mywin_eh перед іншими правилами для цього предикату:

```
win_mywin_eh(Win, Event, 0):-
        write(Win, ":", Event),nl,
        fail.
```

В результаті у вікні повідомлень будуть друкуватись всі події, що генеруються для вашого вікна. Запустіть програму та спробуйте виконати різні операції з вікном: створити, закрити, змінити розмір, перемістити, згорнути, розгорнути тощо.

5. Додайте в програму код обробки події e_Update для вікна. Воно виникає при перемальовуванні вікна, наприклад коли вікно або його частина були закриті іншим вікном, а потім вікно стало активним. Для цього виберіть у вікні проекту потрібне вікно та натисніть Code Expert. Повинні бути обрані такі значення: Window – MyWin, Module – MyWinFile.pro, Event Type – Window, Event or Item – e_Update. Натисніть кнопку Add Clause, а потім на кнопку Edit Clause, щоб переглянути отриманий код. Додайте необхідні предикати, щоб правило мало вигляд:

```
%BEGIN MyWin, e_Update
win_mywin_eh(_Win,e_Update(_UpdateRct),0):-!,
  RCT = win_GetClientRect(_Win),
  RCT = rct(_,_,R,B),
```

```
  draw_Line(_Win, pnt(0,0),pnt(R,B)),
  draw_Line(_Win, pnt(0,B),pnt(R,0)),
  !.
%END MyWin, e_Update
```

У цьому випадку предикат GetClientRect дозволяє отримати внутрішню (робочу) область вікна. Предикати draw_Line малюють на вікні дві прямі лінії у вигляді хреста.

Запустіть вашу програму, відкрийте два вікна і поекспериментуйте. Ви помітите, що перемальовується не все вікно, а лише частина, яка була закрита. Тому зображення на екрані не завжди коректне. Щоб усунути цей дефект, потрібно дещо змінити правило (воно вже існує) для обробки події e_Size, що виникає при зміні розмірів вікна, на:

```
%BEGIN MyWin, e_Size
win_mywin_eh(_Win,e_Size(_Width,_Height),0):-!,
  win_Invalidate(_Win), %додали цей рядок
ifdef use_tbar
  toolbar_Resize(_Win),
enddef
  !.
%END MyWin, e_Size
```

6. Щоб надрукувати текст у вікні, потрібно використовувати предикат draw_Text. Наприклад, draw_Text(_Win, 55, 20, "Мій текст"). Додайте у вікно виведення тексту. Щоб вставляти в текст програми стандартні предикати, зручно використовувати комбінацію клавіш Ctrl+Shift+V. Спробуйте ще щось намалювати у вашому вікні, використовуючи різні предикати draw_… Для зміни кольору тексту використовуються предикати:

```
Col=vpi_ComposeRGB(0, 255, 0), %вибираємо колір
win_SetForeColor(_Win, Col), %встановлюємо його як колір тексту
```

Колір тла змінюється аналогічно.

Тема 7. Робота з файлами у Пролозі

7.1 Основні теоретичні відомості

Використання файлів у Пролозі

Використання файлів у Пролозі вимагає опису файлового домену у розділі domains. На відміну від доменів інших типів, тип домену file задається ліворуч від знака рівності, а ім'я домену - праворуч, наприклад:

```
  domains
```

file=myfile;myfile1;myfile2

Логічне або символічне ім'я файлу (у даному випадку myfile) ототожнюється з ім'ям файлу в DOS. При використанні кількох символічних імен файлів, імена їх доменів вказуються через знак (;).

Запис у файл та модифікація файлу

Перед записом інформації у файл його необхідно відкрити за допомогою предикату

openwrite(SymbolicFileName,DosFileName).

У разі дозапису інформації у файл слід відкрити, використовуючи предикат

oppenappend(SymbolicFileName,DosFileName).

При модифікації файлу слід використовувати предикат

opermodify(SymbolicFileName,DosFileName).

Предикати openwrite(), openappend() і openmodify() встановлюють зв'язок між доменом користувача (myfile) і реальним ім'ям файлу в DOS ("File.dat").

Наступна процедура, яку слід виконати, це призначити файл як пристрій запису, використовуючи предикат

writedevice(SymbolicFileName).

Запис інформації у файл виконується за допомогою стандартних предикатів

write() та writef().

Після запису в файл інформації може знадобитися перепризначення пристрою запису, в якості яких можуть виступати екран монітора, принтер або інший файл.

Після завершення роботи з файлом його слід закрити. Для цієї мети служить предикат

closefile(SymbolicFileName).

Читання з файлу

Для читання інформації з файлу необхідно:

- Виконати відкриття файлу для читання, використовуючи предикат

openread(SymbolicFileName);

- призначити файл пристроєм читання за допомогою предикату

readdevice(SymbolicFile);

закити файл після завершення читання з нього інформації, застосувавши предикат

closefile(SymbolicFile).

Розглянемо приклад:

```
 domains
        file=myfile
 predicates
        start
        read_info(char)
 goal
        start
 clauses
        start:- write("Введення символів з клавіатури та запис у файл"),
                write("Для завершення введення натисніть #") ,
                openwrite(myfile, "File.dat"),
                readchar(X),
                read_info(X),
                closes(myfile),
                write(screen),
                write("Запис виконано").
        read_info('#'):-!. /* Перше правило*/
        read_info('\13'):-!, /* Друге правило*/
                write("\13\10"),
                writedevice(screen),
                write("\13\10"),
                readchar(X),
                read_info(X).
        read_info(X):- writedevice(myfile),/* Третє правило*/
                write(X),
                writedevice(screen),
                write(X),
                readchar(Y),
                read_info(Y).
```

Ця програма послідовно виконує такі підцілі:

- Виведення на екран повідомлення;
- Відкриття файлу для запису;
- Зчитування символу з клавіатури;
- Виклик правила read_info(X);
- Закриття файлу;

- Призначення екрана пристроєм виведення;

- на екрані текстового повідомлення.

Правило read_info(X) має три варіанти. Перший варіант :

read_info('#'):-!

виконується успішно, якщо було введено символ '#'. Відсікання з тілі цієї пропозиції дозволяє виключити пошук інших рішень.

Друге правило аналізується Прологом, якщо введений символ - "повернення каретки", код якого дорівнює 13. Після уніфікації цільового затвердження read_info(X) з головою цього правила виконуються такі підцілі:

- призначається пристрій виведення інформації (файл);

- у файл записується послідовність символів "\13\10", що відповідає операції "повернення каретки та переведення на нову сторінку";

- призначається пристрій виведення інформації (екран);

- на екран виводиться операція " повернення каретки та переведення на нову

рядок";

- Виконується зчитування з клавіатури чергового символу;

- Виконується рекурсивний виклик процедури read_info(X) з новим значенням

символ;

третє правило read_info(X) виконується кожного разу, коли введений символ не є символом '#' або '\13'. Це

7.2 Завдання для самостійного розв'язання

Завдання

Створіть вікно з наступними властивостями:

- вікно повинне мати своє меню;
- вікно повинне мати панель інструментів з правого боку вікна, за допомогою якої вікно може бути очищене;
- вікно повинно мати контекстне меню, за допомогою якого його можна згорнути, максимізувати, повернути до нормального стану, а також очистити та змінити колір;
- для створення зображення використовується миша, малювання виконується при натиснутій кнопці.

1. Створіть нову програму. Помістимо вікно, що створюється, в окремий модуль. Виберіть Module, натисніть кнопку New та введіть ім'я модуля Mywin. Буде створено модуль MYWIN.PRO, до якого буде вміщено потрібний код.

2. Щоб створити нове меню, потрібно вибрати Menu у вікні проекту та натиснути на кнопку New. Дайте нове меню ім'я 'Меню вікна', ідентифікаційна константа id_menu. Натисніть кнопку OK, щоб закрити діалог. Використовуючи Menu Editor (викликається кнопкою Edit), додайте в меню 'Очистити'. Змініть константу для нього на id_clear.

3. Щоб створити нове зображення, натисніть у вікні проекту на кнопку Bitmap, а потім на кнопку New. Введіть ім'я (mybmp_norm) та розмір зображення (залишіть за замовчуванням). Ідентифікаційна константа та ім'я файлу будуть згенеровані за замовчуванням, якщо ви їх не поставите. Намалюйте невелике зображення, яке буде використано як малюнок для кнопки, і закрийте графічний редактор. Потім створіть ще одне зображення (mybmp_down), яке буде використано для натиснутої кнопки. Скопіюйте малюнок (наприклад, за допомогою контекстного меню) та змініть у ньому кольори, наприклад, використовуючи кнопку  .

4. Створіть панель інструментів. Виберіть у вікні проекту Toolbar і натисніть кнопку New. Дайте новій панелі інструментів ім'я 'Нова'. Після того, як ви натиснете OK, ім'я нової панелі інструментів з'явиться у списку та з'явиться вікно Toolbar Editor з його панеллю інструментів. Додайте кнопку панелі інструментів. Вкажіть ім'я константи id_clear, Released Bitmap Name – mybmp_norm, решта двох малюнків – mybmp_down. Ім'я константи збігається з константою для пункту меню, тому не потрібно створювати окремий код. Використовуючи кнопку Attribute, змініть значення в полі Toolbar Style так, щоб панель була розташована праворуч.

5. Створимо вікно. Виберіть у вікні проекту Window і натисніть кнопку New. Вкажіть ім'я вікна (new_win), меню (Меню вікна) та натисніть OK.

6. Панель інструментів (Нова) можна додати лише після того, як буде створено вікно. Використовуйте кнопку Attribute, а потім кнопку Toolbars... . У списку Remaining Toolbars виберіть Нова та натисніть кнопку Add. У списку Toolbar також виберіть Нова, а в полі Module вкажіть Mywin.pro. Натисніть кнопку Default code, щоб згенерувати необхідний код.

7. Далі потрібно створити код вікна. Відкрийте Dialog and Window Expert (наприклад, за допомогою кнопки Code Expert). Вкажіть потрібне

вікно (new_win), модуль, що містить код Mywin.pro. Потім натисніть кнопку Default Code. Щоб вивести вікно на екран, додайте новий пункт у головне меню. Перевірте, як відображається вікно на екрані. Зауважте, що меню вікна виводиться замість головного меню.

8. Реалізуйте малювання у вікні, натиснувши кнопку миші. При обробці подій миші знадобиться БД, в якій зберігатиметься інформація про позицію миші, стан кнопки миші та покажчика на вікно. Помістіть наведені нижче рядки у файл Mywin.pro після операторів include:

```
DATABASE - mouse
point(WINDOW,PNT)
mouse_isdown(WINDOW)
```

Далі потрібно обробити три події миші: e_MouseDown, e_MouseMove, e_MouseUp. Default code для них потрібно додати за допомогою Dialog and Window Expert. Остаточний код наведено нижче:

```
win_new_win_eh(_Win,e_MouseUp(_PNT,_ShiftCtlAlt,_Button),0):-!,
    retractall(mouse_isdown(_Win)),
    retractall(point(_Win,_)),
    !.
win_new_win_eh(_Win,e_MouseMove(_PNT,_ShiftCtlAlt,_Button),0):-!,
    mouse_isdown(_Win),
    point(_Win,_PNT1),
    draw_Line(_Win,_PNT1,_PNT),
    retract(point(_Win,_PNT1)),
    assert(point(_Win,_PNT)),
    !.

win_new_win_eh(_Win,e_MouseDown(_PNT,_ShiftCtlAlt,_Button),0):-!,
    assert(mouse_isdown(_Win)),
    assert(point(_Win,_PNT)),
    !.
```

9. При натисканні на кнопку панелі інструментів виникає подія e_Menu для id_clear. Додайте до програми код, що реалізує очищення вікна:

```
win_new_win_eh(_Win,e_Menu(id_clear,_ShiftCtlAlt),0):-!,
    win_Clear(_Win, color_White),
    win_Invalidate(_Win),
    !.
```

10. Створіть спливаюче меню. Для цього створіть у редакторі меню як звичайно меню, що має наступні 4 пункти: Очистити(id_clear), Згорнути(id_close), Максимум(id_max), Нормальне(id_restore). Для пункту Очистити використовуйте ту ж константу (id_clear), що в меню вікна та кнопки панелі інструментів.

Додайте ще одне правило для події e_MouseDown, помістивши його перед старим або змінивши перший рядок старого правила так, щоб воно виконувалося лише для лівої кнопки миші:

```
win_new_win_eh(_Win,e_MouseDown(PNT, _,mouse_button_right),0):-!,
   menu_PopUp(_Win,res_menu(id_popup),PNT, align_Right).

win_new_win_eh(_Win,e_MouseDown(_PNT,_, mouse_button_left),0):-!,
   …
```

Тут id_popup – ідентифікаційна константа спливаючого меню. У вас вона може бути іншою.

При створенні коду для останніх трьох пунктів меню, ви не знайдете їх серед об'єктів вікна у вікні Dialog and Window Expert, так як це контекстне меню. Код слід додавати вручну. Помістіть курсор, наприклад, після предикату обробки події e_Menu для меню id_clear. Введіть наведені нижче предикати:

```
win_new_win_eh(_Win,e_Menu(id_close,_ShiftCtlAlt),0):-!,
   win_Destroy(_Win).
win_new_win_eh(_Win,e_Menu(id_max,_ShiftCtlAlt),0):-!,
   win_SetState(_Win, [wsf_Maximized]).
win_new_win_eh(_Win,e_Menu(id_restore,_ShiftCtlAlt),0):-!,
   win_SetState(_Win, [wsf_Restored]).
```

Додамо до спливаючого меню ще один пункт, що дозволяє змінювати колір ліній. Для вибору кольору використовується стандартне діалогове вікно. Якщо ідентифікаційна константа цього пункту id_color, то відповідне правило має вид:

```
win_new_win_eh(_Win,e_Menu(id_color,_ShiftCtlAlt),0):-!,
   Pen = win_GetPen (_Win),
   Pen = pen (Penwidth, PenStyle, Color),
   NewColor = dlg_ChooseColor(Color),
   win_SetPen(_Win,pen(Penwidth,PenStyle,NewColor)).
```

Список рекомендованої літератури

Основна:

1. Різник О. Я. Логічне програмування : навчальний посібник. Львів : Львівська політехніка, 2008. 332 с.

2. Тарасенко О. П., Волков С. Г. Практикум з логічного програмування. Харьков, 2009. 92 с.

3. Трушевський В. М. Технології та мови програмування для штучного інтелекту. Частина 1: Основи програмування мовою Prolog. Львів : Львівська політехніка, 2006. 120 с.

4. Шумейко О. О., Кнуренко В. М. VISUAL PROLOG. Опануй на прикладах : навчальний посібник. Дніпро : Біла К.О, 2014. 404 с.

Додаткова:

1. Eduardo Costa. Visual Prolog 7.0 for Tyros. URL : www. ProgBook. Net.

2. Visual Prolog 7.4 Language Reference Prolog Development Center. URL : www.visual-prolog.com .

Зміст

Навчальне видання

Основи штучного інтелекта

Методичні рекомендації

Укладач: **Борчик** Євген Юрійович

Формат 60х84/16 Ум. друк. арк. 2
Тираж 25 прим. Зам. №

Надруковано у видавничому відділі
Миколаївського національного аграрного університету
54020, м. Миколаїв, вул. Г. Гонгадзе, 9

Свідоцтво суб'єкта видавничої справи ДК № 4490 від 20.02.2013 р.