МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА**

**М. В. Новожилова,
О. О. Петрова**

ВИКОРИСТАННЯ МОВИ ЛОГІЧНОГО ПРОГРАМУВАННЯ VISUAL PROLOG ДЛЯ РОЗРОБКИ ЕКСПЕРТНИХ СИСТЕМ

НАВЧАЛЬНИЙ ПОСІБНИК

**Харків
ХНУМГ ім. О. М. Бекетова
2019**

УДК 510.755:004.825
Н74

Автори:

Новожилова Марина Володимирівна, доктор фізико-математичних наук, професор, завідувач кафедри прикладної математики і інформаційних технологій Харківського національного університету міського господарства імені О. М. Бекетова;

Петрова Олена Олександрівна, кандидат технічних наук, доцент кафедри прикладної математики і інформаційних технологій Харківського національного університету міського господарства імені О. М. Бекетова

Рецензенти:

Яковлев Сергій Всеволодович, доктор фізико-математичних наук, професор кафедри інформатики Харківського національного аерокосмічного університету ім. М. Є. Жуковського «Харківський авіційний інститут»;

Сізова Наталя Дмитріївна, доктор фізико-математичних наук, професор кафедри комп'ютерних наук та інформаційних технологій Харківського національного університету будівництва та архітектури

Рекомендовано до друку Вченою радою ХНУМГ ім. О. М. Бекетова, протокол № 6 від 28.12.2018.

Новожилова М. В.

Н74 Використання мови логічного програмування Visual Prolog для розробки експертних систем : навч. посібник / М. В. Новожилова, О. О. Петрова ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2019. – 89 с.

Навчальний посібник містить теоретичні основи та прикладні засоби логічного програмування у вирішенні завдань штучного інтелекту, розглянуто теоретичні та практичні питання створення експертних систем. Викладено основні можливості мови логічного програмування Prolog та використання вбудованих механізмів логічного виводу для побудови простих експертних систем.

Призначено для студентів студентів спеціальностей 122 – Комп'ютерні науки, 126 – Інформаційні системи та технології, 151 – Автоматизація та комп'ютерно-інтегровані технології, а також усіх тих, хто цікавиться цими питаннями.

УДК 510.755:004.825

ЗМІСТ

ВСТУП

Логічне програмування використовують для автоматичного доказу теорем, реалізації реляційних систем управління базами даних, обробки природної мови, організації сервера даних або знань, до якого може звертатися клієнтський додаток, що розроблений на будь-якій мові програмування, для автоматизованого управління виробничими процесами, у завданнях штучного інтелекту.

Штучний інтелект передбачає простий структурний підхід до розроблення складних програмних систем прийняття рішення. Комплекс таких програм, кожна з яких є експертом у початковій предметній галузі, отримав назву експертних систем. Експертна система (ЕС) – це програмний засіб, що використовує експертні знання людини та процедури логічного виводу для забезпечення високоефективного вирішення неформалізованих завдань у вузькій предметній області. Розроблення ЕС – це галузь інформатики, що активно розвивається та спрямована на використання комп'ютерів для оброблення інформації у тих галузях науки та техніки, де традиційне математичне моделювання малопридатне, де важливі смислове та логічне оброблення інформації, досвід експертів. Розробка систем, заснованих на знаннях, є складовою досліджень штучного інтелекту, і має на меті створення комп'ютерних методів вирішення проблем, що зазвичай вимагають залучення експертів. У наш час терміни «система, заснована на знанях», і «експертна система» часто використовуються як синоніми. Фактично останнім часом ЕС стали розглядатися як модель програмування або підхід до програмування, альтернативний відносно до звичайного алгоритмічного програмування [1].

Перспективною для розробки експертних систем виявилася реалізація мови логічного програмування Prolog, який використовує спрощений варіант синтаксису предикатної логіки, що забезпечує більш зрозумілий синтаксис, який є наближеним до природної мови. Математична модель Prolog основана на теорії логіки предикатів, зокрема на процедурній інтерпретації хорновських диз'юнктів, що забезпечує розробку систем, заснованих на знаннях.

Однією з найважливіших особливостей Prolog є те, що, на додаток до логічного пошуку відповідей на поставлені питання, він може мати справу з альтернативами і знаходити всі можливі рішення. Замість звичайної роботи від початку програми до її кінця, Prolog може повертатися назад і переглядати більше одного «шляху» під час вирішення всіх складових завдання.

Мова логічного програмування Prolog має високий рівень абстракції, легкість та простоту в поданні складових структур даних, можливість моделювати логічні відношення та процеси, що істотно полегшує створення програми.

Ефективність програм, які розробляються, залежить від системи знань, якими програма оперує, а не лише від схем виводу, які вона використовує,

тобто, щоб зробити програму інтелектуальною, її потрібно наповнити множиною високоякісних спеціальних знань про певну предметну галузь.

Під час програмування мовою Visual Prolog зусилля програміста спрямовані не на деталі програмної реалізації, а на опис логічної моделі фрагмента предметної області завдання, що розв'язується, у термінах об'єктів предметної області, їхніх властивостей і відносин між ними. Фактично Prolog становить не стільки мову програмування, скільки мову для опису даних і логіки їхньої обробки. Розв'язок завдання записується не в термінах комп'ютера, а в термінах предметної області завдання, що розв'язується, з використанням об'єктно-орієнтованого програмування.

Такі мови, як Prolog не призначені для оброблення числових величин, а використовуються для вираження відносин, завдяки чому вони найбільш придатні для програмної реалізації механізмів отримання виводу, які широко використовуються в експертних системах для маніпулювання знаннями.

Prolog притаманна низка механізмів, які відсутні в арсеналі традиційних мов програмування: зіставлення зі зразком, виведення з пошуком і поверненням. Ще одна істотна відмінність Prolog полягає в тому, що для зберігання даних використовуються списки, а не масиви. У мові Prolog добавлено механізм відсікання як з метою керування обходом варіантів пошуку, так і для підвищення ефективності програм. Можливості Prolog розширені нелогічними засобами: вбудованими процедурами у вигляді підпрограм, процедурами для виконання арифметичних операцій, операціями перевірки входження символів, вводу/виводу, управління ходом доведення тощо.

Принципова особливість Prolog полягає в тому, що програма подається як множина об'єктів і множина відносин між ними, що традиційно для математичного стилю опису задач. Джерелами створення Prolog стали логіка предикатів 1-го порядку, теорія рекурсивних функцій, методи логічного виводу (метод резолюцій).

Опис системи та мови програмування Visual Prolog базується на технічній документації, яка знаходиться у відкритому доступі на офіційному сайті фірми Prolog Development Center (PDC) – фірми–розробника Prolog і є для некомерційного приватного використання безкоштовною [2–3].

Метою цього навчального посібника є ознайомлення студентів з основними можливостями Prolog як мови програмування та використання вбудованих механізмів логічного виводу для побудови експертних систем.

Навчальний посібник для спеціалістів у галузі інформаційних технологій містить:

– основи логічного програмування, поняття логічної програми, компоненти логічної програми: факти, правила, запити до логічної програми;

– основні елементи мови: константи, змінні, анонімні змінні, поняття «зв'язаної» змінної, підстановки, зіставлення, перебирання з поверненнями, поняття уніфікації, вбудовані предикати;

– розділи програми, які призначені для запису фактів, правил і цілей,

реалізації складних способів вираження цільових тверджень;

– доказ цільових тверджень із використанням механізму пошуку з поверненням;

– роботу зі структурами довільного вигляду;

– список як приватний вид структури, форми запису списків, робота зі списками;

– рекурсивне подання даних і програм: побудова рекурсивних програм, граничні умови і способи використання рекурсії;

– відсікання і способи його використання, причини використання відсікання та інші можливості мови.

Навчальний посібник допоможе студентам спеціальностей «Комп'ютерні науки», «Інформаційні системи та технології» та «Автоматизація та комп'ютерно-інтегровані технології» ознайомитися із загальною інформацією, необхідною для розуміння ідей логічного програмування на базі наведених прикладів програм, та розробити самостійно експертні системи. Для забезпечення практичного засвоєння мови та системи Visual Prolog наведені лістинги програм з поясненнями та детальними коментарями.

Наведені в навчальному посібнику практичні заняття ілюструють можливості використання Visual Prolog у системах штучного інтелекту. Кожне практичне заняття містить: загальні положення, приклади розв'язання, контрольні питання.

У результаті виконання практичних занять студенти ознайомляться з теоретичними основами та прикладними засобами логічного програмування в розв'язанні задач штучного інтелекту, отримають досвід використання мови Visual Prolog у процесі розв'язання практичних задач та матимуть уявлення про тенденції розвитку інструментальних засобів логічного програмування, оскільки логічне мислення є основним інструментом розумової діяльності людини [4–6].

Автори ставили перед собою завдання не тільки надати студентам знання щодо теоретичних основ дисципліни, що вивчається, але й уможливити набуття певних практичних навичок та вмінь щодо використання сучасних інтелектуальних інформаційних технологій для розв'язання навчальних задач за обраною спеціальністю.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Логічна основа програми в Prolog

Джон Алан Робінсон писав: «В основі ідеї логічного програмування лежить опис задачі сукупністю тверджень на деякій формальній логічній мові та отримання рішення за допомогою виводу в деякій формальній (дедуктивній) системі» [7].

Ідея логічного програмування полягає у використанні комп'ютера для отримання виведення з декларативного опису предметної галузі. У вузькому сенсі термін «логічне програмування» означає використання логіки предикатів 1-го порядку як основи для опису предметної галузі та виконання логічного виводу. Логічне програмування пов'язують із системами програмування, які базуються на використанні спеціальних класів логічних формул, так званих клаузів Хорна або хорновських диз'юнктів, та спеціальних методів логічного виводу (варіантів методу резолюцій) як логічної моделі обчислень та способу виконання логічної програми [8].

Програма на Prolog задається хорновськими диз'юнктами (у дещо зміненому записі), а виконання програми становить у певний спосіб упорядкований процес побудови дерева виведення шляхом використання методу резолюції. Метод резолюцій є узагальненням методу «доказу від протилежного». Замість виведення деякої формули-гіпотези з наявної несуперечливої множини аксіом до множини аксіом додається заперечення формули і виконується спроба вивести з неї протиріччя. У випадку успішної спроби доходять до висновку, що вихідна формула є такою, що виводиться з множини аксіом.

1.2 Компоненти логічної програми

Логічна програма складається з набору речень, які називаються фактами, правилами та запитами. Prolog–програма – це набір фактів разом із правилами для здійснення висновків із цих фактів, а не порядок дій [1–3].

Факт – це формула: $P(t_1,t_2,\ldots t_n)$,

де P – символ предиката;

t_n – терми, що складаються зі змінних, констант та функціональних символів.

У Prolog зв'язок між об'єктами називається предикатом. У математичній логиці предикат (n-арний або n-місний) – це функція з областю значень {0,1}, визначена на n-му декартовому ступені множини M.

Факти у Prolog – це відносини між об'єктами предметної галузі. Факт подається у вигляді:

ПРЕДИКАТ (об'єкт, значення).

Українською мовою зв'язок виражається реченням. Розглянемо факт,

який використовує зв'язок *подобається* українською мовою: Колі подобається Ганна. Той самий факт на Prolog має вигляд: *подобається (коля, ганна)*. У цьому прикладі власні імена Коля та Ганна записані з маленької літери, тому що Prolog розглядає імена як символи, що мають постійне значення.

Правило – це формула: $A_1 \wedge A_2 \wedge ... A_{n_0} \rightarrow A_0$,

де A_i – атомарні формули.

Мовою Prolog формули: $A_0 :- A_1, A_2, ... A_n$ – це безквантворні імплікації вигляду: *кон'юнкція атомарних формул → атомарна формула*. Якщо у цьому правилі зустрічаються змінні $X_1, X_2, ..., X_m$, то говорять, що «для всіх об'єктів $X_1, X_2, ..., X_m$, якщо є правильними ствердження $A_1, A_2, ..., A_n$ то є правильним і ствердження A_0».

У Prolog усі правила мають дві частини: голову (Head) і тіло (Body), які розділені спеціальною лексемою «`:-`», аналогічною слову «якщо», і яка призначена для того, щоб відділити ці дві частини правила.

Голова (Head) (відома так само, як висновок або залежне відношення) є факт, який буде істинним, якщо певний набір умов є істинним. Тіло (Body) є набором умов, який має бути істинним, щоб механізм логічного виводу Prolog міг довести, що голова правила є істинною.

Правила дають можливість виконати виведення одних фактів з інших.

Факти та правила містяться в розділі диз'юнктів `CLAUSES`.

Запит до логічної програми – це кон'юнкція атомарних формул, тобто формула $A_1 \wedge A_2 \wedge ... A_n$. Якщо у запиті немає змінних, то його читають як «Дійсно є правильними A_1 та A_2 та …A_n?» Якщо запит містить у собі змінні $X_1, X_2, ..., X_m$, то його інтерпретують як «Для яких об'єктів $X_1, X_2, ...X_m$ є правильними A_1 та A_2 та …A_n?»

Виконати програму означає знайти всі значення вільних змінних, за яких запит логічно виходить із фактів та правил. Це відомо як `querying the Prolog system`. Prolog завжди починає пошук розв'язку з початкових фактів та правил розділу `CLAUSES` і зберігає знайдені розв'язки до того як досягне кінця програми. Механізм виводу в Prolog відбирає умови перевірки правила (тіло правила) і, переглядаючи список відомих фактів і правил, виконує спробу задовольнити умовам. Якщо всі умови були відібрано, то голова правила – істинна. Якщо всі умови перевірки не можуть бути співставленими з відомими фактами, правило не має жодного виводу.

Запит до Prolog–програми розміщується в розділі цілей `GOAL`.

1.3 Змінні

У Prolog змінні дають можливість описати загальні факти, правила і задавати загальні питання. Першим символом ідентифікатора змінної в Prolog має бути символ підкреслення «_» або велика літера.

Prolog не має оператора присвоювання. У Prolog змінні отримують значення шляхом порівняння з константами у фактах і правилах (зіставлення, `matching`). Перш ніж змінна отримує значення, вона є вільною, після отримання значення вона стає зв'язаною. Змінна залишається зв'язаною, доки не знайдено чергову відповідь на запит (не досягнена ціль). Після досягнення цілі Prolog звільняє змінну, виконує відкат і шукає альтернативні розв'язки. Наступний приклад продемонструє, як і коли змінні отримують значення.

```
PREDICATES
   nondeterm likes (symbol,symbol)

CLAUSES
    likes (ellen, reading).
    likes (john, computers).
    likes (jonn, badminton).
    likes (leonard, badminton).
    likes (eric, swimming).
    likes (eric, reading).
```

Prolog під час формулювання в розділі `GOAL` питання «Чи існує особистість, якій подобається і читати, і плавати?» *(likes (Person, reading), likes (Person, swimming))* відповідає на обидві частини цього питання пошуком диз'юнктів у програмі від початку до кінця програми. У першій частині питання *likes (Person, reading)* змінна *Person* вільна і її значення залишається невідомим доти, доки Prolog виконає спробу знайти розв'язок. Перший факт у програмі *likes (ellen, reading)* зіставляється з першою частиною запиту, для цього факт *reading* збігається з ціллю *reading*. Отже, Prolog привласнює вільній змінній *Person* значення *ellen*, яке відповідає значенню у факті. У той самий час Prolog розміщує покажчик у списку фактів, який показує точку, до якої дійшов процес пошуку. Далі, щоб задовольнити ціль повністю (знайти особистість, якій подобається і читання, і плавання), друга частина цілі також має бути виконана. Оскільки змінна *Person* зараз зіставлена з *ellen*, то Prolog має шукати факт *likes (ellen, swimming)* від початку програми. Проте Prolog не знаходить збігу, тому що відсутній такий факт у програмі. Друга частина цілі – хибна, при значенні змінної *Person,* яке уніфікується з *ellen*. Надалі Prolog «звільняє» змінну *Person* і намагається знайти інший розв'язок для першої частини запитання, водночас змінна *Person* знову вільна. Пошук іншого факту, який є порівняним

із першою частиною запиту, починається від покажчика в списку фактів. Це повернення до останнього позначеного місця відоме як перебирання із поверненнями (backtrcking). Prolog шукає наступну особистість, якій подобається читання, і знаходить факт *likes (eric, reading)*. Змінна *Person* набуває значення *eric*, отже, Prolog пробує ще раз задовольнити другу частину цілі, відшукуючи в програмі факт *likes (eric, swimming)*. У разі знаходження збігу (останні диз'юнкти у програмі) ціль повністю буде задоволена. Відповіддю Prolog буде значення змінної *Person*, яке уніфікується з *eric*.

Анонімні змінні дозволяють упорядковувати програми. Якщо необхідна певна інформація із запиту, можна використовувати анонімну змінну, проігнорувавши значення, у яких немає необхідності. У Prolog анонімна змінна подана нижнім підкресленням «_» і зіставляється з усіма фактами і правилами. Анонімна змінна може бути використана замість будь-якої іншої змінної. Різниця полягає в тому, що анонімна змінна ніколи не отримує значення.

1.4 Програмні розділи Prolog–програми

Зазвичай Prolog–програма містить такі основні програмні розділи.

1.4.1 Розділ CONSTANTS – розділ опису констант. Оголошення константи має вигляд:

<ім'я константи> = <значення>.

Ім'я константи має бути ідентифікатором, тобто воно може складатися з англійських літер, цифр і знака підкреслення, до того ж не може починатися з цифри. Кожне визначення константи повинно розміщуватись в окремому рядку.

Система не розрізняє великих і малих літер в описах константи. Отже, коли ідентифікатор константи використовується розділом диз'юнктів програми, перша літера має бути рядковою, щоб відрізнити константу від змінної:

```
CONSTANTS
    two = 2

GOAL
    A=two, write (A).
```

Може бути декілька описів розділів констант у програмі, але константи мають бути оголошені перед їхнім використанням.

1.4.2 Розділ доменів (DOMAINS) – розділ опису доменів, який є аналогом розділу опису типів у звичайних імперативних мовах програмування. У розділі описуються області даних для предикатних змінних, задається область інтерпретації предметних змінних. У розділі доменів користувач оголошує домени, які він використовує (стандартні домени оголошувати не потрібно) (табл. 1.1).

Таблиця 1.1 – Стандартні домени

Назва	Характеристика домену і скорочена реалізація
integer	ціле зі знаком –32 768..+32 767
unsigned	16 або 32 біти без знака
byte	байт, беззнакове число 0.. 255
word	беззнакове число 0.. 65 535
long	довге знакове число –2 147 483 647.... 2 147 483 648, 32 біти без знака
ulong	довге беззнакове число 0..4294967295, 32 біти без знака
char	символ, подається як unsigned byte. Синтаксично він записується як символ, узятий в одинарні лапки: ‘a’
short	коротке знакове число, 16 бітів зі знаком
ushort	16 бітів без знака
real	число з плаваючою точкою $-10^{307} \ldots 10^{308}$
symbol	символи, довільні текстові рядки
string	символи, довільні текстові рядки, у яких текст взятий у подвійні лапки

У Prolog–програмах об’єкти у відношеннях (аргументи предиката) належать до доменів. Це можуть бути заздалегідь визначені домени або спеціальні домени, які задає користувач. Розділ доменів має два призначення. По-перше, у цьому розділі можна присвоювати значущі імена доменам навіть, якщо вони становлять існуючі домени. По-друге, спеціально оголошені домени використовуються для оголошення структури даних, які не є стандартними. Іноді зручно оголошувати домен, якщо бажано роз’яснити частини розділу предикатів.

Наприклад, дано речення: Ганна – дівчина 20 років. Для розуміння того, які аргументи присутні в оголошенні предиката, бажано оформити розділи доменів та предикатів у такий спосіб:

```
DOMAINS
   name, sex = symbol
   age = integer
PREDICATES
   person(name, sex, age).
```

1.4.3 Розділ PREDICATES – розділ опису предикатів, у якому представлені всі предикати, що використовуються. У результаті оголошення предиката повідомляється, до яких доменів (типів) належать аргументи цього предиката. Предикати задають факти і правила. У розділі PREDICATES усі предикати просто перераховуються із зазначенням типів (доменів) їхніх аргументів. Оголошення предиката починається з імені цього предиката, за яким йде дужка, що відкривається, після чого йде нуль або більше доменів (типів) аргументу предиката. Декларація предиката, на відміну від речень у розділі CLAUSES, не завершується крапкою. Домени аргументів предиката можуть бути або стандартними доменами, або доменами, які оголошені в розділі DOMAINS. Ім'я предиката повинно починатися з літери та містити послідовність літер, цифр і символів підкреслення (максимум 250 символів). В імені предиката не можна використовувати пробіл, мінус, * та інші алфавітно-цифрові символи. Аргументи предикатів повинні належати доменам, які відомі Prolog. Це або стандартні, або оголошені користувачем домени. Prolog має декілька вбудованих предикатів, які не потрібно оголошувати в програмі (табл. 1.2).

Таблиця 1.2 – Вбудовані предикати

Вбудований предикат	Дія предикату
1	2
nl	перехід на новий рядок
write	виведення повідомлення
readchar	читання символу
asserta	додавання нового факту в базу даних перед існуючим фактом для певного предиката
assertz	додавання нового факту в базу даних після існуючих фактів для певного предиката
consult	завантаження з файлу фактів бази даних
retract	знищення факту з бази даних
retractall	знищення всіх фактів із бази даних
save	збереження динамічної бази на диск
free(Var)	перевірка, чи є змінна Var вільною
bound(Var)	перевірка, чи є змінна Var зв'язаною

Продовження таблиці 1.2

1	2
`findall`	формування списку із всіх можливих рішень, які забезпечують істинність деякого неоднозначного предиката (іншими словами це пошук всіх рішень для мети одночасно)
`length`	обчислення довжини списку
`member`	приналежність елемента списку
`append`	приєднання одного списку до іншого
`fail`	визначення того, що поточний пошук рішення для деякої підцілі не має успіху та необхідно почати новий пошук
`cut (!)`	примусове завершення всіх пошуків
`not`	визначення того, що поточний пошук рішення для деякої підмети дає успіх, коли пов'язана з ним підцілі не може бути доведена

Наприклад, якщо оголошується предикат *my_predicate* (`symbol, integer`) у розділі предикатів:

```
PREDICATES my_predicate (symbol, integer)
```

то немає необхідності в тому, щоб оголошувати його аргументи-домени в розділі доменів, тому що символ і ціле число – стандартні домени. Проте, якщо оголошується предикат *my_predicate (name, number)* у розділі предикатів:

```
PREDICATES
   my_predicate  (name, number)
```

то необхідно оголосити відповідні домени для *name* і *number*. Припустимо, що це має бути символ і ціле число відповідно. Оголошення домену виглядає так:

```
DOMAINS
   name = symbol
   number = integer

PREDICATES
   my_predicate (name, number).
```

Під час опису предикатів може бути задано однозначність або

неоднозначність результату – `determ` (детермінований (однозначний), або `nondeterm` (недетермінований). Детермінований предикат (`determ`), встановлений за замовчуванням, повертає лише одне рішення, недетермінований (неоднозначний, `nondeterm`) предикат за допомогою пошуку з поверненням може давати багато рішень.

1.4.4 Розділ диз'юнктів (`CLAUSES`) – розділ опису речень, у якому описуються факти і правила, що утворюють програму. Диз'юнкти для певного предиката повинні розміщуватися разом у розділі диз'юнктів. Порядок диз'юнктів визначає як буде викликано предикат у процесі виконання програми. Коли Prolog робить спробу задовольнити ціль, він починає перегляд з початкових диз'юнктів розділу, оглядаючи кожен факт і правило в пошуках збігу. Як тільки Prolog доходить до кінця розділу диз'юнктів, він розміщує внутрішні покажчики на наступні диз'юнкти за тими, які були зіставлені з поточною підціллю. Якщо диз'юнкт не є частиною логічного шляху, що веде до розв'язку, то Prolog повертається на встановлений покажчик і відшукує інший збіг (це і є перебирання з поверненнями, `backtracking`).

1.4.5 Розділ цілей (`GOAL`). У розділі цілей розміщується початкова ціль для Prolog–програми. Розділ цілей записується так само, як і тіло правила: у вигляді списку підцілей. Існує дві відмінності між ціллю і правилом:

1) ключове слово завдання не слідує за лексемою «`:-`»;

2) Prolog автоматично виконує ціль, коли програма запускається. Це відбувається так, ніби Prolog виконує виклик цілі, після чого програма запускається, намагаючись задовольнити тіло правила в цілі. Якщо всі підцілі в розділі `GOAL` мають успіх, програма закінчується успішно. Інакше програма повертає для цілі – неуспіх.

Програма на Prolog може містити питання в програмі (так звана внутрішня ціль). Якщо програма містить внутрішню ціль, то після запуску програми на виконання система перевіряє досяжність заданої цілі. Якщо внутрішньої цілі в програмі немає, то після запуску програми система видає запрошення вводити питання в діалоговому режимі (зовнішня ціль).

Розміщення фактів, правил і запитів в одній програмі розглянемо на прикладі лістингу Prolog–програми, до якої сформульовано такий запит: «Чи подобається Білу баскетбол?»

```
PREDICATES
     nondeterm likes (symbol, symbol)

CLAUSES
    likes (ellen, tennis).
    likes (john, football).
    likes (tom, baseball).
```

likes (eric, swimming).
likes (mark, tennis).
likes (bill,Activity) : –likes(tom, Activity).

```
GOAL
```
likes (bill, baseball).

Під час запуску програми буде отримано відповідь «Так», яка означає, що у системи є інформація, яка дозволяє зробити висновок про істинність запитання.

Приклад демонструє використання Prolog–правил для відповіді на питання. У цій програмі існує одне правило: голова правила *likes (bill, Activity)* і тіло *likes (tom, Activity).* Необхідно зауважити, що в цьому прикладі відсутній факт, який говорить про те, що Білу подобається бейсбол. Для знаходження відповіді на це питання Prolog використовує правило *likes (bill, Activity): – likes (tom, Activity)* і комбінує це правило з фактом *likes (tom, baseball),* щоб довести, що *likes (bill, baseball).*

У Prolog можна формулювати складні цілі з використанням кон'юнкції та диз'юнкції. Цілі можуть бути простими, як, наприклад, наступні дві:

likes (Who, swimming).
likes (bill, What).

або більш складними:

likes (Person, reading), likes (Person, swimming).

Ціль, яка складається з двох або більше частин, відома як складна ціль, а кожна частина складної цілі називається підціллю (`subgoal`). Під час формулювання цілі:

```
GOAL
```
parent (Person, _), male (Person).

Prolog спочатку спробує задовольнити першу підціль *(parent (Person,))* шляхом пошуку співпадаючих диз'юнктів, а потім зв'яже змінну *Person* зі значенням, що повертається диз'юнктом *parent*. Це значення буде використовуватися під час пошуку збігів для другої підцілі *male (Person).*

Під час знаходження розв'язку в складній цілі, де підцілі *A* і *B* поєднані за допомогою кон'юнкції, необхідно відокремлювати підцілі комою. Під час знаходження розв'язку в складній цілі, де підцілі *A* і *B* поєднані за допомогою диз'юнкції, необхідно відокремлювати підцілі крапкою з комою.

1.4.5 У розділ `FACTS` (у ранніх версіях `DATABASE`) можна додавати і

видаляти факти для предиката під час роботи програми. Для цього необхідно оголосити предикати, які описують внутрішню базу даних, у розділі бази даних програми та використовувати ці предикати так само, як використовуються предикати, які оголошені в розділі предикатів.

Ключове слово DATABASE вказує на початок оголошення предикатів, що описують внутрішню базу даних. Можна додати факти до внутрішньої бази даних із клавіатури під час виконання програми за допомогою предикатів asserta і assertz. У Prolog предикати retract і retractall використовуються, щоб видалити існуючі факти. Можна модифікувати вміст бази даних спочатку видаленням факту, а потім додаванням нової версії цього факту (або взагалі інший факт). Предикат consult читає факти з файлу і вставляє їх у внутрішню базу даних, а предикат save зберігає зміст розділу внутрішньої бази даних у файлі.

Приклад бази даних:

```
DOMAINS
    name, address = string
    age = integer
    gender = male; female

DATABASE
    person(name, address, age, gender)

PREDICATES
    male(name, address, age)
    female(name, address, age)
    child(name, age, gender)

CLAUSES
    male(Name, Address, Age) :-person(Name, Address, Age, male).
```

У цьому прикладі використовується предикат *person* так само, як використовуються інші предикати: *male, female, child.* Єдина відмінність у тому, що можна вставляти і видаляти факти для особи, доки функціонує ця програма.

Існує два обмеження для використання предикатів бази даних:

1) можна додати до бази даних лише факти, але не правила;

2) факти бази даних не можуть мати вільну змінну.

У програмі можна мати декілька розділів бази даних, кожному з яких необхідно явно задати ім’я:

```
DATABASE - mydatabase
    myFirstRelation (integer)
```

mySecondRelation (real, string)
myThirdRelation (string)

Це оголошення створює базу даних з ім'ям *mydatabase*. Якщо не задано ім'я для внутрішньої бази даних, то за замовчуванням буде привласнене стандартне ім'я – dbasedom. Імена предикатів бази даних повинні бути унікальними в модулі (початковому файлі). Не можна використовувати однакове ім'я предиката бази даних у двох різних розділах бази даних.

1.5 Складові об'єкти даних

Як прості об'єкти даних у Prolog використовуються константи та змінні. Складові об'єкти даних дозволяють інтерпретувати деякі частини інформації як єдине ціле так, щоб потім легко було їх розділити знову. До складових об'єктів даних у Prolog належать структури та списки.

1.5.1 Структура – це об'єкт, який має ім'я (функтор), що починається з букви, і список аргументів, записаних у круглих дужках і розділених комами. Регістр символів в імені структури не має значення, але рекомендується не використовувати прописні символи. Спецсимволи «$», «–», «?» та інші, за винятком символу підкреслення, не можна використовувати в іменах структур.

Аргументами структур можуть бути числа, атоми, змінні, структури та інші складові об'єкти. Проте кількість компонент структури фіксована і не може змінюватися в процесі виконання програми.

Наприклад, дата: грудень 5, 2018, можна описати у такий спосіб:

```
DOMAINS
    date_omp = date(string, unsigned, unsigned)
    D=date("December", 5, 2018)
```

Цей запис – об'єкт даних, який можна обробляти разом зі списками та числами. Функтор визначає вигляд складового об'єкта даних та поєднує разом його аргументи.

Кожна компонента структури, зі свого боку, може бути структурою.

Відносини в Prolog називаються предикатами і записуються вони так само, як структури. Для того щоб Prolog міг відрізнити, де об'єкт даних, а де відносини, необхідно їх явно описати. Відповідно, усі структури мають бути описані в розділі DOMAINS разом із користувацькими типами даних, а всі предикати, які використані в програмі, (крім вбудованих) – у розділі PREDICATES.

1.5.2 У Prolog основним складовим типом даних є список. Список – це об'єднання довільної кількості об'єктів, розділених комами й обмежених квадратними дужками. Списковий домен задається у такий спосіб:

```
DOMAINS
    <ім'я спискового домену> = <ім'я домену елементів списку> *.
```

Символ «*» після імені домену вказує на те, що описується список, який складається з об'єктів відповідного типу (символ «*» означає «список будь-чого»). Нижче наведені приклади списків:

1) [«понеділок», «вівторок», «середа», «четвер», «п'ятниця», «субота», «неділя»] – список, елементами якого є українські назви днів тижня;

2) [1, 2, 3, 4, 5, 6, 7] – список, елементами якого є номери днів тижня;

3) [] – порожній список, тобто список, який не містить елементів.

Елементи списку можуть бути і складовими об'єктами. Зокрема, елементи списку самі можуть бути списками, наприклад:

listI = `integer *` / * список, елементи якого є цілі числа * /;

listR = `real *` / * список, що складається з дійсних чисел * /;

listC = `char *` / * список символів * /;

listS = `string *` / * список, що складається з рядків * / ;

listL = listI * / * список, елементами якого є списки цілих чисел * /.

Останньому прикладу будуть відповідати списки вигляду:

[[1, 3, 7], [], [5, 2, 94], [–5, 13]].

Список, відповідно до рекурсивного визначення списку, – це структура даних, що визначається так:

1) порожній список [] є списком;

2) структура вигляду [H | T] є списком, якщо H – перший елемент списку (або декілька перших елементів списку, перерахованих через кому), а T – список, який складається з елементів вихідного списку, що залишилися. Голову списку H (Head) та хвіст списку T (Tail) розділяють операцією «|».

Таке визначення списку дозволяє організовувати рекурсивне оброблення списків, розділяючи непорожній список на голову і хвіст. Хвіст, зі свого боку, також є списком, що містить меншу кількість елементів, ніж вихідний список. Якщо хвіст не порожній, його також можна розбити на голову і хвіст. І так доти, доки не вийде порожній список, що не має голови.

Наприклад, у списку [1, 2, 3] елемент 1 є головою, а список [2, 3] – хвостом, тобто [1, 2, 3] = [1 | [2, 3]].

Для організації оброблення списку, згідно з наведеним вище рекурсивним визначенням, достатньо задати речення (правило або факт, що визначає послідовність роботи з порожнім списком), яке буде базисом рекурсії, а також рекурсивне правило, яке встановлює порядок переходу від оброблення всього непорожнього списку до оброблення його хвоста.

Для роботи зі списками використовують такі вбудовані предикати

Prolog:

1) предикат `length` (довжина), що дозволяє обчислити довжину списку, тобто кількість елементів у списку;

2) предикат `member` (приналежність), що дозволяє перевірити приналежність елемента списку;

3) предикат `append` (об'єднання), що дозволяє приєднати один список до іншого;

4) предикат `findall` (пошук усіх розв'язків для цілі одночасно).

Предикат `findall` має три аргументи:

1) `VarName` (ім'я змінної) – визначає параметр, який необхідно зібрати в список;

2) `myPredicate` (мій предикат) – визначає предикат, з якого необхідно зібрати значення;

3) `ListParam` (список параметрів) – вміщує список значень, які зібрані методом пошуку з поверненням. Це має бути визначений користувачем тип, якому належить значення `ListParam`.

На відміну від традиційних мов програмування, в яких основним засобом організації повторюваних дій, є цикли, у Prolog для цього використовуються механізм пошуку з поверненням (відкат) і рекурсію. Відкат дає можливість отримати велику кількість рішень щодо одного питання програми, а рекурсія дозволяє використовувати в процесі визначення предиката сам предикат. Будь-яка рекурсивна процедура має містити базис і крок рекурсії. Базис рекурсії – це речення, що визначає початкову ситуацію або ситуацію в момент припинення. У цьому реченні записується простий випадок, за якого відповідь отримується відразу, навіть без використання рекурсії. Крок рекурсії – це правило, у тілі якого обов'язково міститься як підцілі виклик предиката, що визначається. Для того щоб уникнути зациклення, предикат, що визначається, повинен викликатися не від тих параметрів, які вказані в заголовку правила [1–3, 9–12].

2 ПРАКТИЧНІ ЗАНЯТТЯ

ПРАКТИЧНЕ ЗАНЯТТЯ 1
УНІФІКАЦІЯ І ПОШУК ІЗ ПОВЕРНЕННЯМ (`backtracking`)

Мета заняття – ознайомлення з поняттям уніфікації та дією механізму перебирання з поверненням.

Завдання до виконання практичного заняття

Виконати наведені програми, доповнити базу даних запропонованих програм своїми фактами і правилами в розділі диз'юнктів, перефразувати цілі щодо програм, проаналізувати отримані результати, відповісти на контрольні питання.

Загальні положення

У розглянутих вище програмах було показано, як Prolog «шукає збіги для відповіді на запитання», «виявляє збіг», «шукає збіг умови з фактом», «шукає збіг змінних із константами». Існує кілька шляхів зіставлення одного об'єкта з іншим у Prolog:

1) явний, тотожний збіг;

2) механізм перебирання з поверненням (`backtracking`). Змінні не зберігають значення, ці значення їм присвоюються лише на деякий час, який потрібний для того, щоб знайти один розв'язок до однієї цілі. Для роботи зі змінними Prolog використовує спосіб, що має назву «механізм перебирання з поверненням». Під час пошуку збігу з ціллю Prolog намагається обрати один із декількох варіантів, встановлює маркер на місці розгалуження (`backtracking point`) і обирає першу підціль. Якщо цю підціль не можна задовольнити, то Prolog повертається до встановленого маркера і намагається задовольнити альтернативні підцілі. Розглянемо механізм перебирання з поверненням на простому прикладі:

```
PREDICATES
   nondeterm likes (symbol, symbol)
   tastes (symbol, symbol)
   nondeterm food (symbol)

CLAUSES
   likes (bill, X):- food (X), tastes (X, good).
   tastes (pizza, good).
   tastes (brussel_sprouts, bad).
   food (brussels_sprouts).
   food (pizza).
```

```
GOAL
    likes (bill, What).
```

Ця програма складена з наборів фактів і одного правила *likes*, яке стверджує, що Білу подобається смачна їжа (*good-tasting food*). Для розуміння спосібу дії механізму перебирання з поверненням програмі необхідно задати питання:

likes (bill, What).

Під час спроби задовольнити ціль Prolog рухається від початкових диз'юнктів програми в пошуках збігу для цілі *likes (bill, What).* Prolog знаходить збіг із першим диз'юнктом у програмі, водночас змінна *What* уніфікується зі змінною *X.* Ціль збігається з головою правила, тому Prolog намагається задовольнити це правило, перефразовуючи першу підціль: *food (X)*. Для задовільнення цієї підцілі Prolog почне пощук збігу до цього виклику також із початкових диз'юнктів програми та спробує знайти збіг із кожним фактом або головою правила, які зустрічаються в програмі. Prolog знаходить збіг із підціллю в першому факті *food (brussels_sprouts).* При цьому змінна *X* набуває значення *brussels_sprouts.* Оскільки існує більш ніж одна можлива відповідь до виклику *food (X),* Prolog установлює маркер на факті *food (brussels sprouts)*. Цей маркер показує, де Prolog почне пошук наступного збігу для *food (X).*

Після знаходження успішного збігу з підцілю Prolog повертає відповідне знайдене значення з подальшим пошуком наступної підцілі. Для змінної *X*, що має значення *brussels_sprouts*, наступною підціллю буде: *tastes (brussels_sprouts, good).* Prolog знову починає пошук для задовільнення цієї підцілі з початкових диз'юнктів програми. Оскільки в програмі немає диз'юнкта, який можна зіставити з цією підціллю, то Prolog включає механізм перебирання з поверненням і повертається до останнього встановленого маркера. На цьому етапі уніфікація *brussels_sprouts* для змінної *X* знищується, тобто Prolog звільняє весь набір змінних і виконує спробу знайти інше рішення до початкового виклику *food (X)*, починаючи з встановленого маркера. Prolog знаходить збіг із фактом *food (pizza),* і змінній *X* присвоюється значення *pizza*. Prolog переміщується до наступної підцілі в правилі та формулює новий виклик: *tastes (pizza, good*). Пошук знову починається з початкових диз'юнктів програми. Під час уніфікації знаходиться збіг, і змінній *X* присвоюється значення *pizza*. Оскільки змінна *What* у цілі уніфікована зі змінною *X* у правилі *likes*, то змінній *What* також присвоюється значення *pizza,* і Prolog виводить відповідь:

What=pizza `1 Solution.`

Порядок виконання практичного заняття

1. Ознайомитися із запропонованим прикладом 1 написання програм на Prolog, який демонструє, як змінні набувають своїх значень.

Приклад 1.

```
PREDICATES
   nondeterm likes (symbol, symbol)

CLAUSES
    likes (ellen, tennis).
    likes (john, football).
    likes (tom, baseball).
    likes (eric, swimming).
    likes (mark, tennis).
    likes (bill,Activity) :- likes(tom, Activity).

GOAL
    likes (bill, baseball).
```

2. Запустити програму на виконання.

3. Додати до бази даних власні факти, правила та сформулювати нову ціль. Виконати програму, проаналізувати отримані результати.

4. Ознайомитися із запропонованим прикладом 2 написання програм на Prolog. Програма виконує дві функції: додавання і множення.

Приклад 2.

```
DOMAINS
   product, sum = integer

PREDICATES
    add_em_up (sum, sum, sum)
    multiply_em (product, product, product)

CLAUSES
    add_em_up (X, Y, Sum) :- Sum=X+Y.
    multiply_em (X, Y, Product) :- Product=X*Y.

GOAL
    add_em_up (32, 54, Sum).
```

5. Запустити програму на виконання.

6. Перефразувати ціль для знаходження добутку чисел:
multiply_em (31, 13, Product).

7. Ознайомитися із запропонованим прикладом 3 написання програм на Prolog з використанням складних цілей: кон'юнкції та диз'юнкції:

Приклад 3.

```
PREDICATES
    car (symbol, long, integer, symbol, long)
    truck (symbol, long, integer, symbol, long)
    nondeterm vehicle (symbol, long, integer, symbol, long)

CLAUSES
    car (chrysler, 130000, 3, red, 12000).
    car (ford, 90000, 4, gray, 25000).
    car (datsun, 8000, 1, red, 30000).
    truck (ford, 80000, 6, blue, 8000).
    truck (datsun, 50000, 5, orange, 20000).
    truck (toyota, 25000, 2, black, 25000).
    vehicle (Make, Odometer, Age, Color, Price) :-
    car (Make, Odometer, Age, Color, Price);
    truck (Make, Odometer, Age, Color, Price).

GOAL
    car (Make, Odometer, Years_on_road, Body, 25000).
```

Ця ціль шукає автомобіль, описаний у диз'юнктах, який коштує рівно 25 000 грн.

8. Запустити програму на виконання.

9. Перефразувати ціль для знаходження автомобіля за ціною 30 000 грн з червоним кузовом.

10. Ознайомитися із запропонованим прикладом 4 написання програм на Prolog з використанням складних цілей. У цілі цієї програми записані речення:

– що може купити *judy* та *kelly*;

– хто може купити машину *hot_rod.*

Приклад 4.

```
PREDICATES
    nondeterm can_buy (symbol, symbol)
    nondeterm person (symbol)
    nondeterm car (symbol)
    likes (symbol, symbol)
    for_sale (symbol)

CLAUSES
```

```
can_buy (X, Y)  :-  person (X), car (Y), likes (X, Y), for_sale(Y).

person (kelly).
person (judy).
person (ellen).
person (mark).

car (lemon).
car (hot_rod).

likes (kelly, hot_rod).
likes (judy, pizza).
likes (ellen, tennis).
likes (mark, tennis).

for_sale (pizza).
for_sale (lemon).
for_sale (hot_rod).

GOAL
    can_buy (Who, What),
    can_buy (judy, What);
    can_buy (kelly, What);
    can_buy (Who, hot_rod).
```

11. Запустити програму на виконання.

12. Доповнити базу даних фактами, правилами та перефразувати ціль так, щоб у відповіді отримати уподобання та фінансові можливості *ellen*.

13. Ознайомитися із запропонованим прикладом 5, який реалізує пошук порівнянь у Prolog.

Ця програма містить факти про імена і вік гравців тенісного клубу і використовується для влаштування турніру, у якому відбудеться по дві гри для кожної пари дев'ятирічних гравців клуба.

Приклад 5.

```
DOMAINS
   child = symbol
   age = integer


PREDICATES
   nondeterm player (child, age)

CLAUSES
```

```
player (peter, 9).
player (paul, 11).
player (chris, 9).
player (susan, 9).

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

14. Виконати програму і проаналізувати результати. Ця Prolog–програма працює згідно з таким алгоритмом:

14.1 На першому етапі Prolog розпочинає пошук інтерпретації для першої підцілі *player (Person1, 9)* і перейде до наступної підцілі тільки після того, як буде конкретизована змінна в першій підцілі. Для першої підцілі Prolog знаходить факт *player (peter, 9),* і змінній *Person1* присвоюється значення *peter*.

У подальшому Prolog обробляє наступну підціль, у якій змінній *Person2* також присвоюється значення *peter*.

Наприкінці Prolog починає оброблювати третю і кінцеву підцілі:

Person1 <> Person2.

14.2 Оскільки і змінній *Person1,* і змінній *Person2* присвоєно значення *peter*, то ціль не досягнена, і Prolog повертається до попередньої підцілі та шукає інший розв'язок до другої підцілі:

player (Person2, 9).

Ця підціль досягається за збігом змінної *Person2* зі значенням *chris*.

14.3 Далі необхідно задовольнити третю підціль:

Person1 <> Person2.

Підціль досягається, оскільки *peter* і *chris* – однолітки, отже, загальна ціль досягнена, і турнір буде проведено між двома гравцями: *peter* і *chris*.

14.4 Prolog має знайти всі можливі розв'язки до задачі, тому перевіряється попередня підціль *player (Person2, 9),* яка може також бути задоволена присвоєнням змінній *Person2* значення *susan*. Prolog перевіряє третю підціль ще раз, яка успішно досягається, оскільки *peter* і *susan* – це різні значення. Отже, знайдено ще один розв'язок задачі.

14.5 Для пошуку інших розв'язків Prolog ще раз повертається до другої підцілі, але всі можливості для неї вже вичерпано, тому механізм перебирання з поверненням повертає Prolog до першої підцілі, яка

досягається присвоєнням змінній *Person1* значення *chris*. Друга підціль має успіх у разі збігу змінної *Person2* зі значенням *peter*. Отже, третя підціль задоволена, а це означає, що визначилася пара гравців-однолітків: *chris* і *peter*.

14.6 Для знаходження ще одного розв'язку Prolog повертається до другої підцілі, у якій змінна *Person2* також зіставляється зі значенням *chris,* і третя підціль перевіряється за такої інтерпретації: оскільки і змінній *Person1,* і змінній *Person2* присвоєно значення *chris*, то ціль не досягнена, і Prolog повертається до другої підцілі в пошуку іншого розв'язку. Водночас *Person2* уніфікується зі значенням *susan*, після чого досягається третя підціль: влаштовується турнір між *chris* та *susan*.

14.7 Аналогічно відбувається уніфікація змінної *Person1* і значення *susan*. Намагаючись досягти другої підцілі, Prolog конкретизує змінну *Person2* зі значенням *peter*, успішно досягаючи третьої підцілі з цими підстановками. П'ятий турнір організовується для гравців-однолітків: *susan* та *peter*.

14.8 Механізм перебирання з поверненням знову рухається до другої підцілі, де змінна *Person2* зіставлена зі значенням *chris*, що означає влаштування шостого турніру між учасниками *susan* та *chris.*

14.9 Наступна спроба знайти розв'язок веде до того, що змінним *Person1* і *Person2* присвоюється значення *susan*, а це означає, що третя підціль не задовольняється. Prolog повинен повернутися до другої підцілі, але для неї відсутні нові можливості. Аналогічно під час повернення до першої підцілі Prolog немає варіантів для розгляду, оскільки варіанти для *Person1* також вичерпано. Інших рішень немає. Відповідь програми має такий вигляд:

Person1 = peter, Person2 = chris
Person1 = peter Person2 = susan
Person1 = chris, Person2 = peter
Person1 = chris, Person2 = susan
Person1 = susan, Person2 = peter
Person l = susan, Person2 = chris
`6 Solution.`

Потрібно звернути увагу, що механізм перебирання з поверненням може змушувати Prolog генерувати надмірні розв'язки. У цьому прикладі Prolog не розрізняє, що *Person1 = peter* становить те саме, що *Person2 = peter.*

15. Увести іншу ціль для прикладу програми 5 з проведення тенісних турнірів між гравцями клубу і виконати програму з новою ціллю:

player (Person1, 9), player (Person2, 10).

16. Ознайомитися із запропонованим більш ускладненим прикладом 6 написання програм мовою Prolog з використанням механізму перебирання з

поверненням.

Приклад 6.

```
PREDICATES
   nondeterm type(symbol, symbol)
   nondeterm is_a(symbol, symbol)
   lives(symbol, symbol)
   nondeterm can_swim(symbol)

CLAUSES
   type (ungulate, animal).
   type (fish, animal).
   is_a (zebra, ungulate).
   is_a (herring, fish).
   is_a (shark, fish).
   lives (zebra, on_land).
   lives (frog, on_land).
   lives (frog, in_water).
   lives (shark, in_water).
   can_swim (Y) :- type (X, animal),
   is_a (Y, X),
   lives (Y, in_water).
```

17. Виконати програму з ціллю, яка дозволяє визначити *хто* може плавати:

```
GOAL
   can_swim (What),
   write ("A  ", What, "   can swim\n   "),
   readchar (_).
```

Ця Prolog–програма працює за таким алгоритмом.

17.1 На першому етапі Prolog викликає предикат *can_swim* з вільною змінною *What*. У спробі проінтерпретувати цей виклик Prolog переглядає програму в пошуку збігів, знаходячи збіг із диз'юнктом (правилом) *can_swim*, і змінна *What* уніфікується зі змінною *Y*.

17.2 На другому етапі Prolog намагається задовольнити тіло правила. Prolog викликає першу підціль в тілі правила *type (X, animal)* і шукає збіги до цієї підцілі. Ця підціль задовольняється першим фактом *type (ungulate, animal),* і змінній *X* присвоюється значення *ungulate*. Оскільки існують інші можливі варіанти, то Prolog встановлює маркер на факті *type (ungulate, animal)*.

17.3 У подальшому зі змінною *X*, що дорівнює *ungulate*, Prolog робить виклик другої підцілі правила *(is_a (Y, ungulate)),* і знову відшукує збіги.

Другій підцілі задовольняє факт *is_a (zebra, ungulate).* Змінній *Y* присвоюється значення *zebra,* і Prolog установлює маркер на факті *is_a (zebra, ungulate).*

17.4 На наступному етапі зі змінною *X,* що дорівнює *ungulate*, і змінною *Y*, що дорівнює *zebra*, Prolog намагається задовольнити останню підціль: *lives (zebra, in_water)*. Prolog перевіряє кожен із диз'юнктів *lives*, але диз'юнкт *lives (zebra, in_water)* відсутній у програмі.

17.5 Після запуску механізму перебирання з поверненням процес пошуку розв'язку задачі розпочинається з останнього встановленого маркера (`backtracking point`). У цьому випадку – це друга підціль у правилі, факт *is_a (zebra, ungulate).*

17.6 Після повернення до встановленого маркера Prolog звільняє змінні, яким були присвоєні значення, і намагається знайти інший розв'язок щодо поточної підцілі *is_a (Y, ungulate).*

17.7 Prolog досліджує розділ диз'юнктів у пошуку іншого диз'юнкта, який можна уніфікувати з поточним, розпочинаючи від останньої `backtracking point`. Оскільки в програмі відсутні інші диз'юнкти, які можуть бути уніфіковані з підціллю, то підціль – недосяжна. Prolog повертається для досягнення сформульованої цілі до останнього маркера, який установлено на факті *type (ungulate, animal).*

17.8 Prolog звільняє змінні у вихідній цілі та намагається знайти іншу інтерпретацію щодо підцілі: *type (X, animal).* Пошук розпочинається в розділі диз'юнктів після встановленого маркера. Prolog знаходить збіг із наступним фактом *type (fish, animal)*. Водночас змінній *X* присвоюється значення *fish,* і новий маркер установлюється на цьому факті.

17.9 Prolog переміщується до наступної підцілі у правилі і, оскільки це новий виклик, пошук розпочинається з початкових диз'юнктів програми для факту *is_a (Y, fish).*

17.10 Prolog знаходить збіг для цієї підцілі, і змінній *Y* присвоюється значення *herring*.

17.11 Оскільки змінній *Y* на цьому етапі присвоєно значення *herring*, то поточною підціллю є *lives (herring, in_water).* Це є новий виклик, і пошук розпочинається з початкових диз'юнктів програми. Prolog перевіряє кожен із фактів *lives*, але не знаходить збігів.

17.12 Prolog повертається до останнього маркера на факт *is_a (herring, fish).* Змінні, яким були присвоєні значення після цієї уніфікації, звільнені. Починаючи від останнього маркера, Prolog шукає нові інтерпретації для підцілі *is_a (Y, fish).*

17.13 Prolog знаходить збіг із наступним диз'юнктом *is_a,* і змінна *Y* уніфікується зі значенням *shark.*

17.14 Prolog намагається досягти останньої підцілі зі змінною *Y*, зіставленою з *shark*. Поточною підціллю є *lives (shark, in_water).* Пошук розпочинається з початкових диз'юнктів програми, оскільки виклик є новим. Prolog знаходить збіг, і остання підціль у правилі успішно досягається. У цій

точці тіло правила *can swim (Y)* задоволено, і змінна *Y* отримує значення *shark*. Оскільки змінна *What* уніфікована зі змінною *Y*, то змінна *What* також отримує значення *shark*.

17.15 Prolog продовжує обробляти останню підціль в тілі цілі і викликає другу підціль в цілі.

17.16 Prolog завершує роботу повідомленням:

A shark can swim,

і програма закінчується успішно.

18. Сформулювати для програми 6 іншу ціль:

```
GOAL
   can_swim (Y).
```

і проаналізувати отримані результати.

Контрольні питання

1. Охарактеризувати ідею логічного програмування.
2. Які компоненти логічної програми Вам відомі?
3. Які розділи Prolog–програм Вам відомі?
4. Охарактеризувати розділ `DOMAINS`.
5. Охарактеризувати розділ `PREDICATES`.
6. Охарактеризувати розділ `CLAUSES`.
7. Охарактеризувати розділ `GOAL`.
8. Охарактеризувати типи зміних, які використовуються в Prolog–програмах.
9. Пояснити на прикладі поняття уніфікації.
10. Пояснити дію механізму пошуку з поверненням.
11. Пояснити різницю використання в розділі цілей символу коми та коми з крапкою. Які логічні операції вони позначають?

ПРАКТИЧНЕ ЗАНЯТТЯ 2
ВИКОРИСТАННЯ АНОНІМНОЇ ЗМІННОЇ

Мета заняття – ознайомлення з можливістю використання анонімної змінної в програмах.

Завдання до виконання практичного заняття

Виконати наведені програми, використати анонімну змінну в запиті програми, доповнити базу даних програм своїми фактами і правилами, перефразувати цілі програм, проаналізувати отримані результати, відповісти на контрольні питання.

Загальні положення

Під час використання анонімної змінної в запиті певна інформація ігнорується. Це та інформація, яка не потрібна програмісту для подальшої роботи. У Prolog анонімна змінна подана нижнім підкресленням « _». Анонімна змінна може бути використана замість будь-якої іншої змінної. Різниця полягає в тому, що анонімна змінна ніколи не набуває значення.

Порядок виконання практичного заняття

1. Ознайомитися із запропонованим прикладом 1 написання програм мовою Prolog з використанням анонімних змінних. У програмі поставлено питання: потрібно дізнатися, які особи є батьками, але разом з тим немає потреби знати, хто є їхніми дітьми. Prolog вважає, що кожен раз, коли використовується символ анонімної змінної «_» у запиті, програміст не має потреби в інформації про значення, яке присвоєно цій змінній.

Приклад 1.

```
PREDICATES
    male(symbol)
    female(symbol)
    nondeterm parent(symbol, symbol)

CLAUSES
    male (bill).
    male (joe).
    female (sue).
    female (tammy).
    parent (bill, joe).
    parent (sue, joe).
    parent (joe, tammy).
GOAL
    parent (Parent, _).
```

2. Запустити програму на виконання.

3. Перефразувати питання в розділі цілей: які особи є дітьми, але не потрібно знати, хто є їхніми батьками.

4. Ознайомитися із запропонованим прикладом 2 написання програм мовою Prolog з використанням анонімної змінної.

Приклад 2.

```
DOMAINS
    title, author=symbol
    pages=unsigned
```

```
PREDICATES
    book (title, pages)
    nondeterm written_by (author ,title)
    nondeterm long_novel (title)

CLAUSES
    written_by (pushkin, "Poltava").
    written_by (tolstoj, "War and peace").
    book ("Poltava", 250).
    book ("War and peace",4200).
    long_novel (Title):- written_by (_, Title),
    book (Title, Length), Length>300.

GOAL
    written_by (X, "Poltava"),
    long_novel (Y).
```

5. Запустити програму на виконання.
6. Перефразувати ціль так, щоб виводився автор Л. М. Толстой.
7. Запустити програму на виконання з новою ціллю.
8. Ознайомитися із запропонованим прикладом 3 написання програм на Prolog: ілюстрація перебирання з поверненням з використанням анонімних змінних.

Приклад 3.

```
DOMAINS
    name, sex, occupation, object, vice, substance=symbol
    age=integer

PREDICATES
    nondeterm person (name, age, sex, occupation)
    nondeterm had_affair (name, name)
    killed_with (name, object)
    killed (name)
    nondeterm killer (name)
    motive (vice)
    smeared_in (name, substance)
    owns (name, object)
    nondeterm operates_identically (object, object)
    nondeterm owns_probably (name, object)
    nondeterm suspect (name)
```

```
   /*  Facts about the murder  */
CLAUSES
    person (bert, 55, m, carpenter).
    person (allan, 25, m, football_player).
    person (allan, 25, m, butcher).
    person (john, 25, m, pickpocket).

    had_affair (barbara, john).
    had_affair (barbara, bert).
    had_affair (susan, john).

    killed_with (susan, club).
    killed (susan).

    motive (money).
    motive (jealousy).
    motive (righteousness).

    smeared_in (bert, blood).
    smeared_in (susan, blood).
    smeared_in (allan, mud).
    smeared_in (john, chocolate).
    smeared_in (barbara, chocolate).

    owns (bert, wooden_leg).
    owns (john, pistol).

   /*  Background knowledge  */
    operates_identically (wooden_leg, club).
    operates_identically (bar, club).
    operates_identically (pair_of_scissors, knife).
    operates_identically (football_boot, club).

    owns_probably (X, football_boot) :-  person (X, _ , _,  football_player).
    owns_probably (X, pair_of_scissors)  :- person (X, _, _, hairdresser).
    owns_probably (X, Object) :-  owns (X, Object).

   /*************************************************
   *   Suspect all those who own a weapon with
   *   which Susan could have been killed.
   *************************************************/
    suspect (X) :-  killed_with (susan, Weapon),
    operates_identically (Object, Weapon),
```

```
owns_probably (X, Object).

/**************************************************
*   Suspect men who have had an affair with Susan.
**************************************************/
suspect (X):- motive (jealousy),
person (X, _, m, _),
had_affair (susan, X).

/**************************************************
*   Suspect females who have had an
*   affair with someone that Susan knew.
**************************************************/
suspect (X):- motive (jealousy),
person (X, _,  f,  _),
had_affair (X, Man),
had_affair (susan, Man).

/**************************************************
    Suspect pickpockets whose motive could be money.
**************************************************/
suspect (X):- motive (money),
person (X, _, _, pickpocket).
killer (Killer):-
person(Killer, _, _, _),
killed (Killed),
Killed <> Killer,
suspect (Killer),
smeared_in (Killer, Goo),
smeared_in (Killed, Goo).

GOAL
    killer(X).
```

9. Запустити програму на виконання, проаналізувати отримані результати.

10. Перефразувати ціль виконання програми, проаналізувати отримані результати.

Контрольні питання

1. Описати використання змінної в Prolog–програмі.
2. Проаналізувати переваги використання анонімної змінної.
3. Як у Prolog–програмі використовуються коментарі?

4. Навести приклади використання коментарів у Prolog–програмі та сформулювати мету використання коментарів у програмі.

5. Пояснити механізм перебирання з поверненням із використанням анонімних змінних.

ПРАКТИЧНЕ ЗАНЯТТЯ 3
КЕРОВАНИЙ ПОШУК РІШЕНЬ. ВИКОРИСТАННЯ ПРЕДИКАТА `FAIL`

Мета заняття – ознайомлення з можливістю ініціалізації виконання пошуку з поверненням із використанням вбудованого предиката `fail`, який викликає безрезультатне завершення пошуку.

Завдання до виконання практичного заняття

Виконати наведені програми, уміти керувати механізмом перебирання з поверненням за допомогою вбудованого предиката `fail`, доповнити базу даних програм своїми фактами і правилами, перефразувати цілі програм, отримані результати проаналізувати, відповісти на контрольні питання.

Загальні положення

Убудований в Prolog механізм перебирання з поверненням може спричинити зайвий пошук, а це може призвести до неефективності виконання програми. Наприклад, можна отримати різні відповіді за необхідності знаходження однозначного вирішення питання. В інших випадках необхідно примусити Prolog продовжити шукати додаткові розв’язки під час досягнення поставленої мети. У таких випадках необхідно керувати процесом перебирання з поверненням. Prolog пропонує дві можливості керування перебирання з поверненням: предикат неуспіху `fail`, який використовується для продовження перебирання з поверненням, і предикат відсікання `cut` (позначається знаком оклику «`!`»), який використовується для запобігання перебиранню з поверненням. Ефект предиката `fail` відповідає результату порівняння 2 = 3 або будь-якій іншій неможливій підцілі.

Порядок виконання практичного заняття

1. Ознайомитися із запропонованим прикладом 1 написання програми мовою Prolog, яка пояснює, як використовується пошук з поверненням для виконання повторюваних операцій. Ця програма виводить на друк усі можливі розв’язки запиту.

Приклад 1.

```
PREDICATES
   nondeterm country(symbol)
   nondeterm print_countries

CLAUSES
    country ("Ukraine").
    country ("Poland").
    country ("Franse").
    print_countries :- country (X), write (X), nl, fail.
    print_countries.

GOAL
    print_countries.
```

Предикат *print_countries* віддруковує всі розв'язки. У цій програмі правило з предикатом *print_countries* означає: «Щоб віддрукувати країни, потрібно знайти розв'язки *country (X)*, написати значення *X,* почати новий рядок та ініціювати невдале завершення».

У даному випадку «невдале завершення (`fail`)» означає: «Прийняти, що розв'язок поставленого цільового твердження не було досягнено, тому потрібно повернутися назад та шукати альтернативний розв'язок».

2. Запустити програму на виконання.

3. Запустити програму на виконання без предиката `fail`.

```
PREDICATES
   nondeterm country(symbol)
   nondeterm print_countries

CLAUSES
    country ("Ukraine").
    country ("Poland").
    country ("Franse").
    print_countries :- country (X), write (X), nl.
    print_countries.

GOAL
    print_countries.
```

4. Проаналізувати отримані результати та пояснити, чому замість

списку з трьох країн буде виводитися тільки одна країна – Україна.

5. Ознайомитися із запропонованим прикладом 2 написання програми на Prolog: ілюстрація використання вбудованого предиката `fail`.

Приклад 2.

```
DOMAINS
   name = symbol

PREDICATES
   nondeterm father (name, name)
   everybody

   CLAUSES
   father (leonard, katherine).
   father (carl, jason).
   father (carl, marilyn).
   everybody :- father (X, Y), write (X, " is ", Y, " 's father\n"), fail.
   everybody.

GOAL father (X, Y).
```

6. Запустити програму на виконання.

7. Перефразувати ціль і запустити на виконання програму:

```
GOAL everybody.
```

8. Проаналізувати отримані результати.

Контрольні питання

1. Описати керування механізмом перебирання з поверненням.
2. Які можливості керування перебирання з поверненням Вам відомі?
3. Які вбудовані предикати Вам відомі?
4. Який предикат використовується для продовження перебирання з поверненням?
5. Який сенс роботи предиката `fail`?
6. За допомогою якого предиката програма виводить на друк усі можливі розв'язки запиту в практичному занятті?
7. Пояснити різницю використання в розділі цілей символу коми та коми з крапкою. Які логічні операції вони позначають?

ПРАКТИЧНЕ ЗАНЯТТЯ 4
ПЕРЕРИВАННЯ ПОШУКУ З ПОВЕРНЕННЯМ: ВІДСІКАННЯ

Мета заняття – ознайомлення з можливістю відсікання, яка використовується для переривання пошуку з поверненням, та використання вбудованого предиката `cut(!)`, крізь який неможливо виконати відкат (пошук з поверненням, `backtracking`).

Завдання до виконання практичного заняття

Виконати наведені програми, вміти обмежувати простір пошуку за допомогою вбудованого предиката `cut`, доповнити базу даних програми своїми фактами і правилами, перефразувати цілі до програм, відповісти на контрольні питання.

Загальні положення

Вбудований предикат, який англійською мовою називається `cut`, українською – відсікання, у програмі на Prolog позначається знаком оклику «`!`» та призначений для обмеження простору пошуку з метою підвищення ефективності роботи програм. Цей предикат завжди завершується успішно. Після того, як до нього дійшла черга, він встановлює «паркан», який не дає змогу «відкотитися назад», щоб обрати альтернативні розв'язки для підцілей, що вже «спрацювали», тобто для тих підцілей, які розташовані лівіше від відсікання. На цілі, розташовані правіше, відсікання не впливає. Дія відсікання є простою: не можна повертатися крізь відсікання. Після проходження процесу пошуку розв'язку крізь відсікання, підціль – предикат-відсікання негайно має успіх і наступна підціль (за наявності такої) стає поточною. Якщо програма один раз проходить крізь відсікання, то вже неможливо повернутися до підцілей, які розташовані перед відсіканням. Існує два випадки використання відсікання:

1) зелене відсікання, за якого відомо заздалегідь, що в такій ситуації не може бути альтернативних розв'язків. Іншими словами, зеленими називаються ті з відсікань, у разі відкидання яких програма продовжує видавати ті самі рішення, що й за наявності відсікання;

2) червоне відсікання, за якого логіка програми вимагає відсікання, щоб запобігти розгляду альтернативних підцілей.

Відсікання – це потужна, але й небезпечна можливість Prolog. Із цього погляду воно відповідає оператору `goto` в інших мовах програмування, який багато чого дозволяє, але робить програму більш важкою для розуміння.

Порядок виконання практичного заняття

1. Ознайомитися із запропонованим прикладом 1 написання програм мовою Prolog, який ілюструє запобігання перебиранню з поверненням:

відсікання. Мета цього прикладу – знайти машину марки *Corvette* з кольором *sexy color* і допустимою ціною. Відсікання в правилі *buy_car* показує, що, оскільки є тільки один *Corvette* з кольором *sexy color* в базі даних і, якщо його ціна занадто висока, то пошук іншого автомобіля не потрібен.

Приклад 1.

```
PREDICATES
    buy_car (symbol, symbol)
    nondeterm car (symbol, symbol, integer)
    colors (symbol, symbol)

CLAUSES
    buy_car (Model, Color) :- car (Model, Color, Price),
    colors (Color, sexy), ! , Price > 25000.
    car (maserati, green, 25000).
    car (corvette, black, 24000).
    car (corvette, red, 26000).
    car (porsche, red, 24000).
    colors (red, sexy).
    colors (black, mean) .
    colors (green, preppy).
```

2. Запустити програму на виконання з ціллю:

```
buy_car (corvette, Y).
```

Програма працює за таким алгоритмом.

2.1 Prolog викликає факт *car* як першу підціль в правилі *buy_car*.

2.2 Prolog перевіряє перший факт: *car* – *maserati*, але зіставлення з ціллю не відбувається.

2.3 Далі Prolog перевіряє наступні диз'юнкти *car* і знаходить збіг, водночас змінній *Color* присвоюється значення *black*.

2.4 Prolog виконує наступний виклик, щоб перевірити, чи має обрана машина *car* колір *sexy color*. Проте згідно з фактами бази даних *black* не є *sexy color* в програмі, тому підціль не завершується успіхом.

2.5 Prolog повертається до виклику факту *car* і ще раз переглядає програму для пошуку машини *car* марки *corvette*.

2.6 Prolog знаходить збіг і знову перевіряє *color*. На цей раз *color* є *sexy*, і Prolog береться за наступну підціль в правилі – відсікання. Відсікання негайно успішно досягається і ефективно «заморожує на місці» змінні в цьому диз'юнкті, які (змінні) були конкретизовані раніше.

2.7 Далі Prolog береться до наступної підцілі в правилі – порівняння для знаходження допустимої ціни.

Ця підціль не досягається, тому Prolog виконує спробу повернутися,

щоб знайти другу машину (*car*) для перевірки. Але відсікання запобігає перебиранню з поверненням.

3. Перефразувати ціль, виконати програму з новою ціллю.

4. Ознайомитися із запропонованим прикладом 2 написання програм на Prolog, який організовує запобігання пошуку з поверненням до наступного диз'юнкта. За відсутності відсікання було б запропоновано два розв'язки.

Приклад 2.

```
PREDICATES
    friend (symbol, symbol)
    girl (symbol)
    likes (symbol, symbol)

CLAUSES
    friend (bill, jane) :- girl (jane), likes (bill, jane), !.
    friend (bill, jim) :- likes (jim, baseball), !.
    friend (bill, sue) :- girl (sue).
    girl (mary).
    girl (jane).
    girl (sue).
    likes (jim, baseball).
    likes (bill, sue).

GOAL
    friend (bill, Who).
```

5. Запустити програму на виконання.

Відсікання в першому диз'юнкті *friend* використовується для знаходження тільки одного друга Біла і немає необхідності продовжувати пошук інших друзів. Перебирання з поверненням може відбуватися всередині диз'юнктів у спробі задовольнити виклик, але, якщо розв'язок знайдено, Prolog проходить відсікання.

6. Перефразувати ціль, виконати програму з новою ціллю.

7. Ознайомитися із запропонованим прикладом 3 написання програм мовою Prolog. Ця програма дозволяє обрати оптимальний маршрут під час подорожі з одного міста в інше.

Приклад 3.

```
DOMAINS
    town=symbol
    distance=integer

PREDICATES
```

```
    nondeterm road (town, town, distance)
    nondeterm route (town, town, distance)

CLAUSES
    road (tampa, houston, 200).
    road (gordon, tampa, 300).
    road (houston, gordon, 100).
    road (houston, kansas, 120).
    road (gordon, kansas, 130).
    route (Town1, Town2, Distance):- road (Town1, Town2, Distance).
    route (Town1, Town2, Distance):-  road (Town1, X, Dist1),
    route (X, Town2, Dist2),
    Distance = Dist1 + Dist2, !.

GOAL
    route (tampa, kansas, X).
```

8. Запустити програму на виконання. Кожен факт для предиката *road* описує дорогу визначеної довжини з одного міста до іншого. Речення для предиката *route* вказує на те, що дозволяється прокладати маршрут з одного міста до іншого через декілька проміжних пунктів. Слідуючи за маршрутом, можна за допомогою третього параметра *Distance* отримати загальну протяжність маршруту. Предикат *route* описано рекурсивно. Маршрут може складатися з однієї ділянки дороги, і протяжність маршруту дорівнюватиме довжині ділянки дороги. З іншого боку, можна продовжити маршрут із міста *Town1* до міста *Town2*, доїхавши спочатку з міста *Town1* до міста *X*, а потім проїхавши іншим шляхом від міста *X* до міста *Town2*. Загальна протяжність маршруту дорівнюватиме додатку відстаней від *Town1* до *X* та від *X* до *Town2*. Карта-прототип наведена на рисунку 2.1.

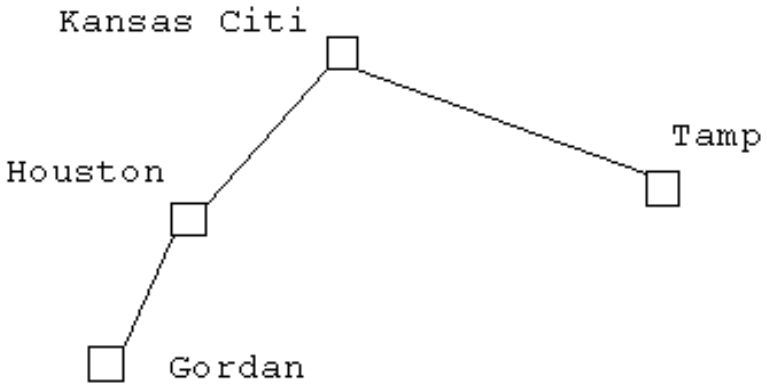

Рисунок 2.1 – Карта для задачі планування маршруту

9. Перефразувати ціль та проаналізувати отриманий результат.

10. Ознайомитися із запропонованим прикладом 4 написання програм на Prolog, який випадково обирає три імені з п’яти.

Приклад 4.

```
PREDICATES
    person(integer, symbol)
    rand_int_1_5(integer)
    rand_person(integer)

CLAUSES
    person (1, fred).
    person (2, tom).
    person (3, mary).
    person (4, dick).
    person (5, george).
    rand_int_1_5(X) :- random(Y), X=Y*4+1.
    rand_person(0):-!.
    rand_person (Count):- rand_int_1_5(N),
    person (N, Name),
    write (Name), nl,
    NewCount = Count – 1,
    rand_person (NewCount).

GOAL
    rand_person(3).
```

11. Запустити двічі програму на виконання та проаналізувати отримані результати.

12. Дописати в базу даних власне ім’я студента та перевірити програму.

13. Ознайомитися з запропонованими прикладами програми 5 знаходження коренів квадратного рівняння:

$$A * X * X + B * X + C = 0.$$

Розв’язок залежить від значення дискримінанти D, що обчислюється у такий спосіб:

$$D = B * B – 4 * A * C.$$

Якщо значення $D > 0$, то обчислюються два нерівних корені, якщо $D = 0$, то існує тільки один розв’язок, якщо $D < 0$, то це означає, що розв’язків не існує (може існувати один або два комплексних розв’язків).

14. Запустити на виконання перший варіант програми 5 для знаходження коренів квадратного рівняння.

Приклад 5, варіант 1.

```
PREDICATES
    zapros
    nondeterm povtor
    nondeterm kvur(real, real, real)

CLAUSES
    povtor.
    povtor:- povtor.
    kvur (A, B, C) :- B * B – 4 * A * C > 0,
    X1 = (–B + sqrt (B * B – 4 * A * C)) / 2 / A,
    X2 = (–B – sqrt (B * B – 4 * A * C)) / 2 / A,
    write (" x1 = "), write (X1),
    write (" x2= "), write (X2), nl.
    kvur (A, B, C) :- B * B – 4 * A * C = 0,
    X = (–B) / 2 / A,
    write (" x= "), write (X), nl.
    kvur (A, B, C) :- B * B – 4 * A * C < 0,
    write (" Korney net ").
    zapros :- povtor,
    write (" a= "), readreal (A), A <> 0, nl,
    write (" b= "), readreal (B), nl,
    write (" c= "), readreal (C), nl,
    kvur (A, B, C),
    write (" for go out press q, continue – anybody klavisha "), nl,
    readchar (S), S='q', !.

GOAL
    zapros.
```

15. Запустити почергово другий варіант програми 5 для знаходження коренів квадратного рівняння на виконання з різними цілями.

Приклад 5, варіант 2.

```
PREDICATES
    solve(real, real, real)
    reply(real, real, real)
    mysqrt(real, real, real)
    equal(real, real)
```

```
CLAUSES
    solve (A, B, C) :- D = B * B – 4 * A * C, reply (A, B, D), nl.
    reply (_, _, D) :- D < 0, write (" No solution "), !.
    reply (A, B, D) :- D = 0, X = –B / (2 * A), write (" x= ", X),!.
    reply (A, B, D) :- mysqrt (D, D, SqrtD),
    X1 = (–B + SqrtD) / (2 * A),
    X2 = (–B – SqrtD) / (2 * A),
    write (" x1 = ", X1," and x2 = ", X2).
    mysqrt (X, Guess, Root) :-
    NewGuess = Guess – (Guess * Guess – X) / 2 / Guess,
    not(equal(NewGuess, Guess)), !,
    mysqrt (X, NewGuess, Root).
    mysqrt (_, Guess, Guess).
    equal (X, Y):- X / Y > 0.99999, X / Y < 1.00001.

GOAL
    % solve (1, 2, 1).
    % solve (1, 1, 4).
    solve (1, –3, 2).
```

Програма обчислює квадратний корінь з ітеративної формули, у якій краще значення *NewGuess* для квадратного кореня *X* може бути отримане з попереднього значення *Guess:*

$$NewGuess = Guess – (Guess * Guess – X) / 2 / Guess.$$

Кожна наступна ітерація дозволяє уточнити корінь. За виконання умови `equal` *(X, Y)* обчислення завершуються і програма розв'язує квадратне рівняння:

$$X1 = (–B + sqrtD) / (2*A)$$
$$X2 = (–B – sqrtD) / (2*A).$$

16. Проаналізувати отримані результати та виконати порівняльну характеристику двох варіантів програми знаходження коренів квадратного рівняння.

Контрольні питання

1. Який предикат призначено для обмеження простору пошуку?
2. Описати випадки використання відсікання.
3. Дати визначення зеленого відсікання.
4. Дати визначення червоного відсікання.
5. Чи полегщує розуміння програми використання відсікання в програмі?

6. Які вбудовані предикати Вам відомі?
7. Описати призначення анонімної змінної в наведеній програмі.

ПРАКТИЧНЕ ЗАНЯТТЯ 5
КЕРУВАННЯ НАПРЯМКОМ ЛОГІЧНОГО ПОШУКУ. ПРЕДИКАТ NOT

Мета заняття – ознайомлення з можливістю використання предиката `not`, який повертає успіх, коли пов'язана з ним підціль не може бути доведеною.

Завдання до виконання практичного заняття

Виконати наведені програми, вміти використовувати предикат `not`, доповнити базу даних програм своїми фактами і правилами, перефразувати цілі програм, отримані результати проаналізувати, відповісти на контрольні питання.

Загальні положення

Під час використання предиката `not` необхідно мати на увазі: предикат `not` буде успішним, якщо не може бути доведена істинність певної підцілі. Правильним способом керування незв'язаними змінними підцілі всередині предиката `not` є використання анонімних змінних.

Порядок виконання практичного заняття

1. Ознайомитися із запропонованим прикладом 1 написання програм мовою Prolog, який демонструє можливості використання предиката `not` для визначення любителів шопінгу.

Приклад 1.

```
PREDICATES
   nondeterm likes_shopping(symbol)
   nondeterm has_credit_card(symbol, symbol)
   bottomed_out (symbol, symbol)

CLAUSES
   likes_shopping (Who):- has_credit_card (Who, Card),
   not (bottomed_out (Who, Card)),
   write(Who, " can shop with the ", Card, " credit card.\n ").
   has_credit_card (chris, visa).
   has_credit_card (chris, diners).
   has_credit_card (joe, shell).
```

```
    has_credit_card (sam, mastercard).
    has_credit_card (sam, citibank).
    bottomed_out (chris, diners).
    bottomed_out (sam, mastercard).
    bottomed_out (chris, visa).

GOAL
    likes_shopping (Who) .
```

2. Запустити програму на виконання та проаналізувати отримані результати.

3. Перефразувати ціль, виконати програму з новою ціллю.

4. Ознайомитися із запропонованим прикладом 2 написання програми мовою Prolog, що демонструє можливість використання предиката `not` для ідентифікації студентів, у яких середній показник (GPA) є більшим за 3.5 і які не знаходяться на стажуванні.

Приклад 2.

```
DOMAINS
   name = symbol
   gpa = real

PREDICATES
   nondeterm honor_student (name)
   nondeterm student (name, gpa)
   probation (name)

CLAUSES
    honor_student (Name) :- student (Name, GPA),
    GPA >= 3.5, not (probation (Name)).
    student ("Betty Blue", 3.5).
    student ("David Smith", 2.0).
    student ("John Johnson", 3.7).
    probation ("Betty Blue").
    probation ("David Smith").
```

5. Запустити програму на виконання з ціллю:

```
GOAL
    student (Who, GPA),
    GPA > 3.5.
```

6. Увести наступну ціль:

```
GOAL
   honor_student (X).
```

7. Виконати програму 2 з ціллю, яка визначає успішних студентів.

8. Ознайомитися із запропонованим прикладом 3 написання програми мовою Prolog, який класифікує членів альпіністського клубу, якими є Тоні, Майкл та Джон. Кожен член клубу є або гірськолижником, або скелелазом, або і тим і іншим. Нікому із скелелазів не подобається дощ. Усім гірськолижникам подобається сніг. Майклу подобається все, що не подобається Тоні, і не подобається все, що подобається Тоні. Тоні подобається сніг і дощ. Визначити члена клубу, який є скелелазом і не є гірськолижником.

Приклад 3.

```
DOMAINS
    Name = tony; mikle; john
    Weather = rain; snow
    Spec = skalolaz; gornolyshnik

PREDICATES
   nondeterm club (Name)
   nondeterm nlike (Name, Weather)
   nondeterm like (Name, Weather)
   nondeterm is (Name, Spec)
   question

CLAUSES
  /* All memders of club */
   club (tony).
   club (mikle).
   club (john).

  /* What dont likes Tony that likes Mikle */
   like (tony, snow).
   like (tony, rain).
   like (john, snow).
   like (mikle, X) :- not (like (tony, X)).
   nlike (X, Y) :- not(like (X, Y)).
   nlike (mikle,Y) :- like (tony, Y).

  /* Somebody is gornoluchnik and skalolaz */
   is (X, gornolyshnik) :- club (X), like (X, rain).
```

```
is (X, skalolaz) :- club (X), not (like (X, snow)).

/* question Who is skalolaz */
question :- readchar(_), nl, is (Name, skalolaz),
not (is (Name, gornolyshnik)),

/* cursor (10, 20) */
write (" skalolaz, but ne gornolyshnik "),
write (Name), readchar(_), nl, fail.

/* question Who and is skalolaz and gornolyshnik */
question :- nl, /* cursor(12,20) */
write (" and skalolaz, and gornolyshnik "),
is (Name, skalolaz), is (Name, gornolyshnik),
write (Name), readchar(_), nl, fail.

/* Who is Tony */
question :- nl, is (tony, Spec),

/* cursor (14, 20) */
write (" tony - ", Spec),
readchar(_),nl, fail.

/* question ne skalolaz, but gornolyshnik */
question :- nl, is (Name, gornolyshnik), not (is (Name, skalolaz)),
/* cursor (16, 20) */
write (" ne skalolaz, but gornolyshnik "),
write (Name), readchar(_), nl, fail.

/* End of output rezalts */
question :- nl,

/* cursor (20, 1) */ write (" It is all ").

GOAL
   question.
```

9. Запустити програму на виконання.

Визначення *question* вміщує вбудовані предикати `readchar(_), nl, cursor, write`, які організовують запит та друкують результати.

Предикат `fail` завжди закінчується неуспіхом і є механізмом для здійснення перебору. Після спроби доказу першого *question* здійснюється

перехід до доказу другого, потім третього варіанта запиту.

10. Виконати програму з новою ціллю, проаналізувати результати.

Контрольні питання

1. Які вбудовані предикати Вам відомі?
2. У якому випадку предикат `not` буде успішним?
3. Що є правильним способом керування незв'язаними змінними підцілі всередині предиката `not` ?
4. Дати визначення анонімної змінної та пояснити необхідність її застосування.
5. Який предикат доцільно використовувати, якщо не може бути доведена істинність цієї підцілі?

ПРАКТИЧНЕ ЗАНЯТТЯ 6
ПРОЦЕДУРНИЙ СЕНС PROLOG–ПРОГРАМ. ВИКОРИСТАННЯ ПРАВИЛ ДЛЯ УМОВНОГО РОЗГАЛУЖЕННЯ

Мета заняття – розгляд правил Prolog з процедурного погляду і засобів для керування напрямком логічного пошуку в програмі.

Завдання до виконання практичного заняття

Виконати наведені програми, вміти застосовувати засоби керування напрямком логічного пошуку в програмі, відповісти на контрольні питання.

Загальні положення

Prolog – декларативна мова, у якій програміст описує проблему в термінах фактів і правил, на відміну від процедурних мов програмування, в яких необхідно запрограмувати підпрограми і функції, що вкажуть комп'ютеру всі дії, які необхідно виконати для розв'язку завдання. Правила в Prolog можуть бути подані у вигляді опису процедур. З процедурного погляду правила виконують дії оператора `case` та оператора `goto`. Можна сприймати як процедури такі правила:

say_hello :– write («Hello»), `nl`.
greet :– write («Hello, Earthlings!»), `nl`.

які відповідають підпрограмам і функціям в інших мовах програмування. Факти в Prolog можна сприймати як процедури. Наприклад, факт

likes (bill, pasta).

означає: «Щоб довести, що Білу подобається *pasta*, не потрібно виконувати

жодних дій». Якщо аргументи *Who* і *What* у питанні *likes (Who, What)* є вільними змінними, то можна присвоїти їм значення *bill* і *pasta* відповідно.

Механізм логічного виводу Prolog, як і оператор `case`, для отримання різних визначень для кожного значення змінної (або набору значень змінної) буде розглядати одне за одним правила, поки не буде знайдено збіги, і тільки після цього механізм логічного виводу почне виконувати дії, обумовлені правилом.

Порядок виконання практичного заняття

1. Ознайомитися із запропонованим прикладом 1 написання програм мовою Prolog з використанням правил для умовного розгалуження (оператор `case`).

Приклад 1.

```
PREDICATES
   nondeterm action (integer)

CLAUSES
   action(1) :- nl, write (" You typed 1. ") , nl.
   action(2) :- nl, write (" You typed 2. "), nl.
   action(3) :- nl, write(" Three was what you typed. "), nl.
   action(N) :- nl, N<>1, N<>2, N<>3,
   write (" I don't know that number! ").

GOAL
   write (" Type a number from 1 to 3: "),
   readint (Num),
   action (Num).
```

2. Запустити програму на виконання тричі.

Якщо користувач введе цифри 1, 2 або 3, то правило *action* буде викликано з аргументом, якому присвоєно відповідне значення, і водночас механізм логічного виводу знайде збіг тільки з одним із перших трьох правил.

3. Ознайомитися з запропонованим прикладом 2 написання програми на Prolog: відсікання як `goto`.

Приклад 2, варіант 1.

```
PREDICATES
   nondeterm action(integer)

CLAUSES
   action(1) :- nl, write (" You typed 1. "), ! .
   action(2) :- nl, write (" You typed 2. "), ! .
```

```
    action(3) :- nl, write (" Three was what you typed. "), ! .
    action(_) :- write(" I don't know that number! ").

GOAL
    write (" Type a number from 1 to 3: "),
    readint (Num),
    action (Num), nl.
```

4. Запустити програму на виконання.

Програма працює не зовсім коректно через те, що після вибору і виконання потрібного правила Prolog продовжує пошук альтернативи. Можна було б заощадити ресурси і час, якщо б було зазначено, де потрібно припинити пошук альтернатив, використовуючи відсікання. Це означає «Якщо Ви дійшли до цього місця, то не потрібно робити відкати всередині цього правила і не потрібно перевіряти інші альтернативи цього правила». Повернення все ще можливе, але тільки на більш високому рівні. Якщо поточне правило викликається іншими правилами, вищі правила мають альтернативи, то вони можуть бути випробувані. Проте відсікання відкидає альтернативи всередині правила і альтернативи цього правила (предиката).

5. Переписати програму прикладу 2.

Приклад 2, варіант 2.

```
PREDICATES
    nondeterm action(integer)

CLAUSES
    action(1) :- !, nl, write (" You typed 1 ").
    action(2) :- !, nl, write (" You typed 2 ").
    action(3) :- !,nl, write (" Three was what you typed. ").
    action(_) :- write(" I don't know that number! ").

GOAL
    write (" Type a number from 1 to 3: "),
    readint (Num),
    action (Num), nl.
```

6. Запустити програму на виконання. Відсікання не має ніякого ефекту, поки воно не буде виконано реально. У наведеному прикладі для того, щоб виконати відсікання, механізм логічного виводу Prolog повинен увійти в правило, яке містить відсікання, і досягти точки, у якій розташоване відсікання. Відсікання в другому варіанті програми змінило логіку програми, таке відсікання називається червоним. Якби програміст зберіг перевірки X <> 1, X <> 2 і X <> 3, змінивши програму внаслідок вставки відсікання в кожному

диз'юнкті, це б називалося зеленим відсіканням. Зелене відсікання заощаджує час програми, яка буде правильною і без його використання.

Контрольні питання

1. Дати визначення вільних та зв'язаних змінних.
2. Описати призначення анонімної змінної.
3. Яка різниця між правилами в Prolog та процедурами в імперативних мовах програмування?
4. Дати визначення червоного відсікання.
5. Дати визначення зеленого відсікання.
6. Навести синтаксис оператора `case` та оператора `goto` в імперативних мовах програмування.
7. Які парадигми програмування притаманні мові програмування Prolog?
8. Які вбудовані предикати Вам відомі?
9. Охарактеризувати компоненти логічної програми.
10. Охарактеризувати призначення розділів логічної програми.

ПРАКТИЧНЕ ЗАНЯТТЯ 7
РЕКУРСІЯ

Мета заняття – використання рекурсії в Prolog–програмах.

Завдання до виконання практичного заняття

Виконати наведені програми, вміти застосовувати рекурсію та розуміти основний принцип програмування в Prolog, відповісти на контрольні питання.

Загальні положення

Рекурсію використовують не лише в Prolog, але й в імперативних мовах програмування. Проте для Prolog, на відміну від імперативних мов, рекурсія є одним з основних прийомів програмування, яка дозволяє використовувати в процесі визначення предиката сам предикат. Prolog дозволяє визначати рекурсивні структури даних. Рекурсивна процедура – це процедура, яка викликає сама себе. Рекурсивна процедура не має проблеми щодо запам'ятовування результатів її виконання, тому що будь-які обчислені значення можна передавати з одного виклику в інший як аргументи предиката, що рекурсивно викликається.

Будь-яка рекурсивна процедура повинна містити *базис* і *крок рекурсії*. *Базис рекурсії* – це речення, що визначає початкову ситуацію або ситуацію в момент припинення. У цьому реченні зазвичай записується простий випадок, за якого відповідь отримується відразу, навіть без використання рекурсії. *Крок рекурсії* – це правило, у тілі якого обов'язково міститься як підціль виклик предиката, що визначається. Для того щоб уникнути зациклення, предикат, що

визначається, повинен викликатися не від тих параметрів, які вказані в заголовку правила. Параметри повинні змінюватися на кожному кроці так, щоб у результаті або спрацював базис рекурсії, або умова виходу з рекурсії, яка розміщена в самому правилі.

У загальному вигляді правило, яке реалізує крок рекурсії, буде виглядати так:

< ім'я предиката, що визначається> :-
[<підцілі>],
[<умова виходу з рекурсії>],
[<підцілі>],
< ім'я предиката, що визначається, не від тих саме параметрів, які вказані в заголовку правила >,
[<підцілі>].

Для здійснення хвостової або правосторонньої рекурсії рекурсивний виклик предиката, що визначається, має бути останньою підціллю в тілі рекурсивного правила, а до моменту рекурсивного виклику не повинно залишитися точок повернення (неперевірених альтернатив). Це означає, що в підцілях, розташованих лівіше від рекурсивного виклику предиката, що визначається, не повинно залишатися будь-яких неперевірених варіантів, а процедура не повинна мати речень, розташованих нижче від рекурсивного правила.

У лівосторонній рекурсії тіло правила розпочинається з рекурсивного виклику предиката, що визначається. За змогою, потрібно намагатися уникати використання лівосторонньої рекурсії, на відміну від правосторонньої або хвостової рекурсії.

Порядок виконання практичного заняття

1. Ознайомитися із запропонованим прикладом 1 написання програм мовою Prolog для знаходження факторіала заданого числа.

Завдання знаходження значення факторіала *n!* вирішується за допомогою рекурсії, оскільки може бути зведена до вирішення аналогічного підзавдання, яке, зі свого боку, зводиться до вирішення аналогічного підпідзавдання тощо. Рекурсія – це спосіб для вирішення завдань, що містять у собі підзавдання такого саме типу.

Приклад 1.

```
PREDICATES
   factorial (integer, integer)

CLAUSES
   % факторіал 0! дорівнює 1
   factorial (0, 1) :- !.
```

```
% факторіал n! дорівнює факторіалу (n–1)!, помноженому на n
factorial (N, Factorial_N) :- M = N – 1, factorial (M, Factorial_M),
Factorial_N=Factorial_M * N.

GOAL
   write ("Для якого числа необхідно знайти факторіал? "),
   readint (Number),
   factorial (Number, Result), write (Number, " != ", Result).
```

2. Ознайомитися із запропонованим прикладом 2 написання програм мовою Prolog для знаходження числа Фібоначчі, меньше заданого.

Послідовність Фібоначчі – це числова послідовність, що задана рекурентним співвідношенням другого порядку. Рекурентним співвідношенням називається формула вигляду $a_{n+1} = F(a_n, a_{n-1}, ..., a_{n-k+1})$, де F – деяка функція від k аргументів, яка дозволяє обчислити наступні члени числової послідовності через значення попередніх членів. Рекурентне співвідношення однозначно визначає послідовність a_n, якщо вказано k перших членів послідовності. Рекурентне співвідношення є прикладом рекурсивного визначення послідовності.

Приклад 2.

```
PREDICATES
   nondeterm fib (integer, integer, integer, integer)
   nondeterm fibless (integer)

CLAUSES
   fib (_, F1, _, Max) :- F1 >= Max, !.
   fib (N, F1, F2, Max) :- write (F1), nl, NN = N + 1, F3 = F1 + F2,
   fib (NN, F2, F3, Max).
   fibless (Max) :- fib(1, 1, 1, Max).

GOAL
    fibless (20).
```

3. Ознайомитися із запропонованим прикладом написання програм на Prolog: Ханойські вежі.

Відгадка до загадки про Ханойські вежі – класичний приклад рекурсії. Ця давня загадка складається з кількох дерев'яних дисків, установлених на трьох стрижнях, які, зі свого боку, прикріплені до основи. Усі диски мають різний діаметр і отвір усередині, достатній для того, щоб стрижні могли пройти крізь них. На початку всі диски знаходяться на лівому стрижні, як зображено на рисунку 2.2.

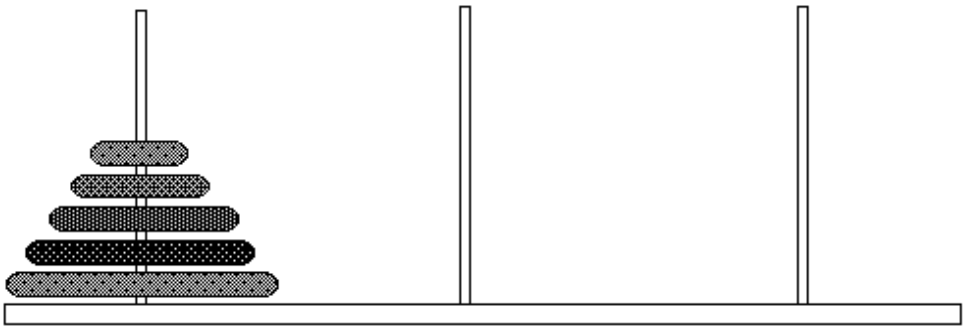

Рисунок 2.2 – Ханойські вежі

Мета загадки полягає в тому, що гравець повинен перемістити всі диски на правий стержень так, щоб вони розмістилися на ньому в такому порядку, в якому були розміщені на початку. Можна використовувати середній стрижень для тимчасового розміщення дисків, але ніколи більший диск не має бути розташованим вище за менший. Легко відгадати загадку про Ханойські вежі з двома або трьома дисками, але процес стає більш важким з чотирма або більше дисками.

Стратегія для вирішення загадки полягає у такому:

– можна безпосередньо перемістити один диск;

– можна перемістити N дисків за три основних кроки:

1) перемістити N – 1 дисків на середній стрижень;

2) перемістити останній (N-й) диск безпосередньо на правий стрижень;

3) перемістити N – 1 дисків із середнього стриженя на правий стрижень.

Програма на Prolog для відгадки загадки про Ханойські вежі використовує три предикати:

1) предикат *hanoi* з одним параметром, який визначає загальну кількість дисків, з якими працюють;

2) предикат *move*, який описує переміщення N дисків від одного стрижня до іншого, використовуючи стрижень, що залишився, для тимчасового зберігання дисків;

3) предикат *inform*, який показує, що сталося з окремим диском.

Приклад 3.

```
DOMAINS
    loc = right; middle; left

PREDICATES
    nondeterm hanoi(integer)
    move(integer, loc, loc, loc)
    inform (loc, loc)

CLAUSES
    hanoi (N) :- move (N, left, middle, right).
```

```
move (1, A, _, C)  :-  inform (A, C),  !.
move (N, A, B, C) :-  N1 = N – 1, move (N1, A, C, B),
inform (A, C), move (N1, B, A, C).
inform (Loc1, Loc2) :- nl,
write(" Move a disk from ", Loc1, "  to  ", Loc2).

GOAL
    hanoi (3).
```

4. Запустити програму на виконання та проаналізувати отримані результати.

Контрольні питання

1. Дати визначення рекурсії.
2. Дати визначення рекурсивної процедури.
3. Дати визначення кроку та базису рекурсії.
4. Охарактеризувати хвостову або правосторонню рекурсію.
5. Охарактеризувати лівосторонню рекурсію.
6. Дати визначення рекурентного співвідношення.
7. Що є прикладом рекурсивного визначення послідовності?
8. Перерахувати мови програмування, у яких використовується рекурсія.

ПРАКТИЧНЕ ЗАНЯТТЯ 8
СКЛАДОВІ ОБ'ЄКТИ

Мета заняття – використання складових об'єктів у Prolog –програмах.

Завдання до виконання практичного заняття

Виконати наведену програму, вміти оперувати складовими об'єктами, доповнити базу даних програми своїми фактами і правилами, перефразувати цілі програми, проаналізувати отримані результати, відповісти на контрольні питання.

Згальні положення

Простий об'єкт – це змінна або константа. Складові об'єкти даних дозволяють інтерпретувати деякі частини інформації як єдине ціле так, щоб потім можна було легко поділити їх знову. Складові об'єкти можуть розглядатися в реченнях Prolog як єдині об'єкти, що спрощують написання програм. Якщо розглянути дату «1 січня 2019 року», то вона складається з трьох частин: день, місяць, рік. Складовий об'єкт *date* можна оголосити у такий спосіб:

```
DOMAINS
    date ("January", 1, 2019).
```

Цей запис виглядає як факт Prolog, але це об'єкт даних, який можна обробляти поряд із символами та числами. Складовий об'єкт починається з імені, яке називається *функтором* (у цьому випадку *date*), за яким йдуть аргументи. Функтор – це не те саме, що функція в інших мовах програмування. Це просто ім'я, яке визначає вигляд складового об'єкта даних та поєднує разом його аргументи. Функтор не означає, що будуть виконані деякі дії. Аргументи складового об'єкта даних можуть самі бути складовими об'єктами.

Складовий об'єкт може бути уніфікований з простою змінною або із складовим об'єктом. Це означає, що складовий об'єкт можна використовувати для того, щоб передавати цілий набір значень як єдиний об'єкт, а потім використати уніфікацію для їхнього розділення.

Prolog виконує уніфікацію в двох випадках. По-перше, коли ціль зіставляється із заголовком речення, по-друге, через знак «=», який є інфіксним предикатом (предикатом, що розташований між своїми аргументами, а не перед ними).

Фактично Prolog виконує операцію присвоювання для уніфікації об'єктів з різних боків від знака «=». Ця властивість корисна для знаходження значень аргументів складового об'єкта.

Порядок виконання практичного заняття

1. Ознайомитися із запропонованим прикладом написання програми на Prolog для створення списку студентів, які народилися в поточному місяці. Програма вирішує це завдання, використовуючи вбудований предикат `date` для отримання дати із внутрішнього календаря комп'ютера.

```
DOMAINS
    name = person(symbol, symbol)
 %  (перший, останній)
    birthday = b_date(symbol, integer, integer)
 % (місяць, день, рік)
    ph_num = symbol
 %  (телефонний номер)

PREDICATES
    nondeterm phone_list (name, ph_num, birthday)
    get_months_birthdays ()
    convert_month(symbol, integer)
    check_birthday_month(integer, birthday)
    write_person (name)
```

```
CLAUSES
   get_months_birthdays :-
   write  (" * This Month's Birthday List * "), nl,
   write  (" First name \t \t Last Name\n "),
   write(" ********************************** "), nl,
   date  (_, This_month, _),

   % отримати місяць із системних часів
   phone_list (Person, _,  Date),
   check_birthday_month (This_month,  Date),
   write_person (Person), fail.
   get_months_birthdays :-
   write  (" \n\n Press any key to continue: "), nl, readchar(_).
   write_person (person (First_name, Last_name)) :-
   write  (" First_name  \t \t Last_name "), nl.
   check_birthday_month (Mon, b_date (Month ,_, _)) :-
   convert_month (Month, Month1),
   Mon = Month1.

   phone_list (person (ed, willis), "767-8463", b_date (jan, 3, 1955)).
   phone_list (person (benjamin, thomas), "438-8400", b_date (feb, 5, 1985)).
   phone_list (person (ray, william), "555-5653", b_date (mar, 3, 1935)).
   phone_list (person (thomas, alfred), "767-2223", b_date (apr, 29, 1951)).
   phone_list (person (chris, gram), "555-1212", b_date (may, 12, 1962)).
   phone_list (person (dustin, robert), "438-8400", b_date (jun, 17, 1980)).
   phone_list (person (anna, friend), "767-8463", b_date (jun, 20, 1986)).
   phone_list (person (brandy, rae), "555-5653", b_date (jul, 16, 1981)).
   phone_list (person (naomi, friend), "767-2223", b_date (aug, 10, 1981)).
   phone_list (person (christina, lynn), "438-8400", b_date (sep, 25, 1981)).
   phone_list (person (kathy, ann), "438-8400", b_date (oct, 20, 1952)).
   phone_list (person (elizabet, ann), "555-1212", b_date (nov, 15, 1987)).
   phone_list (person (jennifer, caitlin), "438-8400", b_date (dec, 31, 1981)).
   phone_list (person (mishenko, kola), "573-2252", b_date (oct, 30, 1962)).

   convert_month (jan, 1).
   convert_month (feb, 2).
   convert_month (mar, 3).
   convert_month (apr, 4).
   convert_month (may, 5).
   convert_month (jun, 6).
   convert_month (jul, 7).
   convert_month (aug, 8).
   convert_month (sep, 9).
```

```
    convert_month (oct, 10).
    convert_month (nov, 11).
    convert_month (dec, 12).

GOAL
    get_months_birthdays().
```

2. Запустити програму на виконання, визначити, хто з поданого списку є іменинником у поточному місяці.

3. Доповнити базу даних інформацією про студента і визначити, чи буде виводитися інформація про нього як про іменинника поточного місяця. Змінити програму так, щоб інформація про дні народження виводилася програмою.

Контрольні питання

1. Які об'єкти є складовими в Prolog–програмах?
2. Які об'єкти є простими в Prolog–програмах?
3. Які об'єкти даних дозволяють інтерпретувати деякі частини інформації як єдине ціле?
4. Навести структуру складового об'єкта.
5. Описати використання уніфікації в складових об'єктах.
6. Описати використання предиката `date` в Prolog.

ПРАКТИЧНЕ ЗАНЯТТЯ 9
СКЛАДОВІ ОБ'ЄКТИ. СПИСКИ

Мета заняття – виконання дій зі складовими об'єктами даних, використання списків в Prolog–програмах.

Завдання до виконання практичного заняття

Виконати наведені програми, вміти оперувати списками, доповнити базу даних програм своїми фактами і правилами, перефразувати цілі до програм, проаналізувати отримані результати, відповісти на контрольні питання.

Загальні положення

В імперативних мовах основною структурою даних зазвичай є масиви. У Prolog основним складовим типом даних є список. *Списком* називають упорядковану послідовність елементів довільної довжини. Список задається перерахуванням елементів списку через кому в квадратних дужках, як показано в наведених нижче прикладах:

[1, 2, 3, 4, 5, 6, 7] – список, елементами якого є цілі числа;

[] – порожній список, тобто список, який не містить елементів.

Елементи списку можуть бути і складовими об'єктами, наприклад, списками. Рекурсивне визначення списку: список – це структура даних, що

визначається у такий спосіб:

1) порожній список [] є списком;

2) структура вигляду [H | T] є списком, якщо H – перший елемент списку (або кілька перших елементів списку, перелічених через кому), а T – список, що складається з елементів, що залишилися від вихідного списку. Прийнято називати H головою списку, а T – хвостом списку.

У розділі опису доменів списки описуються так:

```
DOMAINS
```
<ім'я спискового домену> = <ім'я домену елементів списку> *.

Зірочка після імені домену вказує на те, що описується список, який складається з об'єктів відповідного типу. Наприклад:

*mylist = elementDom**.

У цьому прикладі *mylist* – домен, що складається зі списку елементів із домену *elementDom*. Домен *elementDom* може бути або визначеним користувачем доменом, або одним із стандартних доменів. Читається символ зірочка * як «список». Наприклад,

*numberlist = integer **

це оголошення домену для списку цілих чисел, наприклад [1, –5, 2, –6].

Вбудований предикат `findall` використовує цільові твердження як один зі своїх аргументів і збирає всі розв'язки для цього цільового твердження до єдиного списку. Предикат `findall` має три аргументи:

– `VarNam` (ім'я змінної) – визначає параметр, який необхідно зібрати в список;

– `MyPredicate` (мій предикат) – визначає предикат, з якого потрібно зібрати значення;

– `ListParam` (список параметрів) – містить список значень, зібраних методом пошуку з поверненням.

Порядок виконання практичного заняття

1. Ознайомитися із запропонованим прикладом 1 написання програми на Prolog, який дозволяє визначити середній вік групи людей з використанням предиката `findall`.

Приклад 1.

```
DOMAINS
    name, address = symbol
    age = integer
```

```
    list  = age*

PREDICATES
    nondeterm  person (name, address, age)
    sumlist (list, age, integer)

CLAUSES
    sumlist ([], 0, 0).
    sumlist ([H | T], Sum, N) :-  sumlist (T, S1, N1), Sum = H + S1,
    N = 1 + N1.
    person ("Sherloch Holmes", "22B Baker Street", 42).
    person ("Pete Spiers", "21st Street", 36).
    person ("Mary Darrow", "suite 2, Omega Home", 61).

GOAL
    findall  (Age, person (_ , _,  Age), L),
    sumlist (L, Sum, N),
    Ave = Sum div N,
    write  ("Average=", Ave), nl.
```

2. Запустити програму на виконання та проаналізувати отримані результати.

3. Перефразувати одне з підцільових тверджень так, щоб скласти список людей, вік яких складає 42 роки:

findall (Who, person (Who, _, 42), List).

Ця ціль потребує від програми, щоб та вміщувала оголошення домену для результуючого списку:

slist = symbol.*

Відредагована програма має наступний вигляд:

```
DOMAINS
    name, address  =  symbol
    age  =  integer
    list = age*
    slist =  symbol*

PREDICATES
    nondeterm  person (name, address, age)
    sumlist (list, age, integer)
```

```
CLAUSES
    sumlist ([], 0, 0).
    sumlist ([H | T], Sum, N):- sumlist (T, S1, N1), Sum = H + S1,
    N = 1 + N1.
    person ("Sherloch Holmes", "22B Baker Street", 42).
    person ("Pete Spiers", "21st Street", 36).
    person ("Mary Darrow", "suite 2, Omega Home", 61).
    person ("Watson", "22B Baker Street", 42).


GOAL
    findall (Who, person (Who, _, 42), List).
```

4. Доповнити базу даних інформацією про студента, обчислити середній вік запропонованої групи осіб з урахуванням даних про студента та вивести на екран список 20-річних студентів.

5. Ознайомитися із запропонованим прикладом 2 написання програми на Prolog: завдання розстановки *N* ферзів. Сенс проблеми полягає в розміщенні *N* ферзів на шаховій дошці так, щоб жодних два ферзі не били один одного і не були розміщені в одному рядку, стовпчику або на одній діагоналі. Для вирішення проблеми необхідно пронумерувати рядки і стовпчики шахової дошки від *1* до *N*. Нумерація діагоналей визначається за типом та номером діагоналі:

$$Diagonal = N + Column - Row \ (Type\ 1)$$
$$Diagonal = Row + Column - 1 \ (Type\ 2).$$

При погляді на шахову дошку з лівого верхнього кута діагональ першого типу схожа на риску з нахилом вліво (\), а другого типу – на риску з нахилом вправо (/). Рисунок 2.3 показує нумерацію діагоналей другого типу на дошці 4 x 4.

	1	2	3	4
1	1	2	3	4
2	2	3	4	5
3	3	4	5	6
4	4	5	6	7

Рисунок 2.3 – Шахова дошка для задачі *N* ферзів

Для вирішення завдання розстановки *N* ферзів за допомогою програми на Prolog необхідно записувати рядки, стовпчики і діагоналі, які є незайнятими, і занотовувати позиції розміщення ферзів, які описуються номером рядка та номером стовпчика, як у цьому оголошенні домену:

queen = q (integer, integer).

Це оголошення становить позицію одного ферзя. Для описання більшої кількості позицій необхідно використовувати список:

*queens = queen *.*

Аналогічно необхідно ввести декілька списків для зазначення вертикалей, горизонталей і діагоналей, які незайняті ферзями:

*freelist = integer *.*

Шахова дошка обробляється як одиничний об'єкт із наступним оголошенням домену:

board = board (queens, freelist, freelist, freelist, freelist).

Списки *freelists* є вільні рядки, стовпчики, діагоналі першого і другого типу відповідно. Зробимо припущення, що *board* є шаховою дошкою 4 х 4 у двох ситуаціях:

1) дошка без ферзів:

board ([], [1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4, 5, 6, 7], [1, 2, 3, 4, 5, 6, 7]);

2) дошка з одним ферзем у верхньому лівому куті:

board ([q (l, l)], [2, 3, 4], [2, 3, 4], [1, 2 ,3, 5, 6 ,7], [2, 3, 4, 5, 6, 7]).

Визначимо предикат:

placeN (integer, board, board)

з двома наступними реченнями:

а) ферзі розміщуються по одному, поки всі рядки і стовпчики не стають зайнятими. Це знайшло відображення в першому реченні, де два списки *freerows* (горизонталей) і *freecols* (вертикалей) порожні:

```
placeN (_, board (D, [ ], [ ], X, Y), board (D, [ ], [ ], X, Y)) :- !.
placeN (N, Boardl, Result) :- place_a_queen (N, Boardl, Board2),
placeN (N, Board2, Result);
```

б) у другому реченні предикат *place a queen* визначає зв'язок між *Board1* і *Board2*. Позиція *Board2* має на один ферзь більше, ніж позиція *Board1*. Використовуємо таке оголошення предиката:

place_a_queen (integer, board, board).

Ключове питання під час вирішення розстановки *N* ферзів полягає в описі додавання ферзів, поки вони всі не будуть успішно розміщені на порожній дошці. Для вирішення цієї проблеми додається новий ферзь до списку вже розміщених на дошці:

[Q (R, C) | Queens].

Серед вільних рядків *Rows*, що залишаються, необхідно знайти рядок *R*, на який можна розмістити наступний ферзь. У той самий час горизонталь *R* має бути вилучена зі списку вільних рядків, що веде до нового списку вільних рядків *NewR*:

findandremove (R, Rows, NewR).

Відповідно необхідно знайти і видалити незайнятий стовпчик *C*. Зі значень *R* і *C* може бути обчислено кількість зайнятих діагоналей, що дозволить визначити, чи знаходяться *D1* і *D2* в списку незайнятих діагоналей. Предикат *place_a_queen* реалізує дані функції:

```
place_a_queen (N, board (Queens, Rows, Columns, Diag1, Diag2),
board ([q(R, C) | Queens], NewR, NewS, NewDl, NewD2))  :-
findandremove (R, Rows, NewR),
findandremove (C, Columns, NewC),
Dl = N + S – R, findandremove (Dl, Diag1, NewD1),
D2 = R + S – l, findandremove (D2, Diag2, NewD2).
```

Повний лістинг програми приклада 2 наведено нижче.

```
DOMAINS
    queen = q(integer, integer)
    queens = queen*
    freelist = integer*
    board = board (queens, freelist, freelist, freelist, freelist)

PREDICATES
    nondeterm placeN(integer, board, board)
    nondeterm place_a_queen(integer, board, board)
```

```
    nondeterm nqueens (integer)
    nondeterm makelist(integer, freelist)
    nondeterm findandremove (integer, freelist, freelist)
    nextrow(integer, freelist, freelist)

CLAUSES
    nqueens (N) :-
    makelist (N, L), Diagonal = N * 2 – 1, makelist (Diagonal, LL),
    placeN (N, board ([], L, L, LL, LL), Final), write(Final).
    placeN ( _, board (D, [], [], Dl, D2), board(D, [] , [ ], Dl, D2)) :- !.
    placeN (N, Board1, Result) :-
    place_a_queen (N, Board1, Board2),
    placeN (N, Board2 ,Result).
    place_a_queen (N, board (Queens, Rows, Columns, Diag1, Diag2),
    board ([q(R,C) | Queens], NewR, NewC, NewDl, NewD2)) :-
    nextrow (R, Rows, NewR),
    findandremove (C, Columns, NewC),
    Dl = N + C – R, findandremove (Dl, Diag1,NewDl),
    D2 = R + C – 1, findandremove (D2 , Diag2, NewD2).
    findandremove (X, [X|Rest], Rest).
    findandremove (X, [Y|Rest], [Y | Tail]) :-
    findandremove (X, Rest, Tail).
    makelist (1, [1]).
    makelist (N, [N | Rest]) :-
    N1 = N – 1, makelist (N1, Rest).
    nextrow (Row, [Row | Rest], Rest).

GOAL
    nqueens (5),
    nl, readchar(_).
```

6. Ознайомитися із запропонованим прикладом 3 написання програми на Prolog: пригоди в небезпечній печері, яка складається з лабіринту переходів, що поєднують різні кімнати, у яких знаходяться небезпечні істоти (монстри, розбійники тощо), та кімнату, де знаходяться всі скарби. Необхідно визначити маршрут руху, щоб знайти скарби й уникнути зіткнення з мешканцями печери. Карту печери наведено на рисунку 2.4.

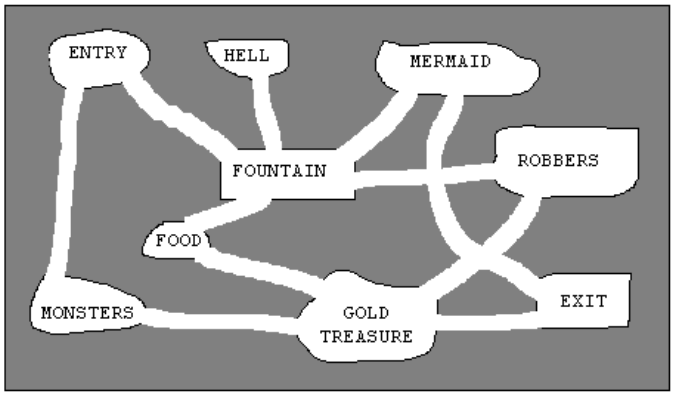

Рисунок 2.4 – Лабіринт переходів

Для вирішення цього завдання необхідно побудувати в Prolog уявну карту, яка допоможе знайти безпечний маршрут. Кожний прохід описується як факт. Предикати *go* і *route* становлять правила. Запит до програми у вигляді:

go (entry, exit)

дозволить отримати список кімнат, які потрібно відвідати для отримання скарбів і благополучного повернення з ними.

Лістинг програми прикладу 3 наведено нижче.

```
DOMAINS
   room = symbol
   roomlist = room*

PREDICATES
   nondeterm gallery (room, room)
 % There is a gallery between two rooms
 % Necessary because it does not matter
 % which direction you go along a gallery
   nondeterm neighborroom (room, room)
   avoid (roomlist)
   nondeterm go (room, room)
   nondeterm route (room, room, roomlist)
 % This is the route to be followed
 % roomlist consists of a list of rooms already visited.
   nondeterm member (room, roomlist)

CLAUSES
   gallery (entry, monsters).
   gallery (entry, fountain).
   gallery (fountain,hell).
```

```
gallery (fountain, food).
gallery (exit, gold_treasure).
gallery (fountain, mermaid).
gallery (robbers, gold_treasure).
gallery (fountain, robbers).
gallery (food, gold_treasure).
gallery (mermaid, exit).
gallery (monsters, gold_treasure).
gallery (gold_treasure,exit).

neighborroom (X, Y) :- gallery (X, Y).
neighborroom (X, Y) :- gallery (Y, X).
avoid ([monsters, robbers]).
go (Here, There) :- route (Here, There, [Here]).
go ( _, _).

route (Room, Room, VisitedRooms) :-
member (gold_treasure, VisitedRooms),
write (VisitedRooms), nl.
route (Room, Way_out, VisitedRooms) :-
neighborroom (Room, Nextroom),
avoid (DangerousRooms),
not (member (NextRoom, DangerousRooms)),
not (member (NextRoom, VisitedRooms)),
route (NextRoom,Way_out, [NextRoom | VisitedRooms]).
member (X, [X | _]).
member (X, [ _ | H]) :- member (X, H).

GOAL
go (entry, exit).
```

7. Запустити програму на виконання та проаналізувати отримані результати.

8. Реверсувати список або записати предикат виведення списку так, щоб він виводив список у зворотному порядку: (*go (exit, entry)).*

9. Скласти свій маршрут пересування в лабіринті.

Контрольні питання

1. Дати рекурсивне визначення списку.
2. Як у розділі опису доменів описуються списки?
3. Навести структуру списку.
4. Описати призначення вбудованого предиката `findall`.
5. Описати аргументи вбудованого предиката `findall`.

ПРАКТИЧНЕ ЗАНЯТТЯ 10
РОБОТА З БАЗАМИ ДАНИХ

Мета заняття – використання баз даних в Prolog–програмах.

Завдання до виконання практичного заняття

Виконати наведену програму, доповнити базу даних програми своїми фактами і правилами, перефразувати цілі до програми, отримані результати проаналізувати, відповісти на контрольні питання.

Загальні положення

Мовою Prolog легко реалізуються реляційні бази даних, що зараз найбільш розповсюджені [13–14]. Будь-яка таблиця реляційної бази даних може бути описана відповідним набором фактів, де кожному запису вихідної таблиці буде відповідати один факт, а кожному полю – аргумент предиката, що реалізує таблицю. Структура реляційних баз даних включається в структуру Prolog–програм. З іншого боку, Prolog має вбудовані засоби для роботи з двома типами баз даних: внутрішніми та зовнішніми. Внутрішні бази даних мають таку назву тому, що вони обробляються винятково в оперативній пам'яті комп'ютера, на відміну від зовнішніх баз даних, які можуть оброблятися на диску або в пам'яті. Внутрішня база даних, яка називається ще динамічною, складається з фактів, які можна динамічно, у процесі виконання програми, додавати до бази даних і видаляти з неї, зберігати у файлі, завантажувати факти з файлу в базу даних. Ці факти можуть використовувати тільки предикати, описані в розділі опису предикатів бази даних:

```
DATABASE [ – < ім'я бази даних >]
<ім'я предиката>(< ім'я домену першого аргументу >,…,
< ім'я домену n-го аргументу>)
…
```

Один розділ опису предикатів бази даних у програмі автоматично отримає стандартне ім'я `dbasedom`. У разі наявності в програмі декількох розділів опису предикатів бази даних тільки один з них може бути неназваним, усі інші повинні мати унікальне ім'я, яке вказується після назви розділу `DATABASE`, і символу «–». Після оголошення розділу опису предикатів бази даних, компілятор внутрішньо оголошує відповідний домен із таким самим ім'ям. Це дозволяє спеціальним предикатам обробляти факти як терми. Опис предикатів бази даних збігається з їхнім описом у розділі опису предикатів `PREDICATES`. Однак ці предикати можна задіяти як параметри вбудованих предикатів. Крім того, факти, які використовують ці предикати, можуть додаватися і видалятися під час виконання програми. У базі даних можуть міститися тільки факти, а не

правила виведення, до того ж факти бази даних не можуть містити вільних змінних. У назві розділу опису предикатів внутрішньої бази даних Prolog–програми слово DATABASE замінено на синонім FACTS, що ще більше підкреслює, що у внутрішній базі даних можуть зберігатися тільки факти, а не правила.

До вбудованих предикатів Prolog, які призначені для роботи з внутрішньою базою даних, належать:

1) предикати assert, asserta, assertz. Використання цих предикатів спричиняє додавання фактів до внутрішньої бази даних. Різниця між цими предикатами полягає в тому, що предикат asserta додає факт перед іншими фактами (на початок внутрішньої бази даних), а предикат assertz додає факт після інших фактів (у кінець бази даних). Предикат assert доданий для сумісності з іншими версіями Prolog і працює так само, як і предикат assertz. Як перший параметр у цих предикатах вказується факт, що додається, як другий (необов'язковий) – ім'я внутрішньої бази даних, до якої додається факт;

2) предикати retract і retractall. Використання цих предикатів спричинить видалення фактів із бази даних. Предикат retract видаляє з внутрішньої бази даних перший факт. Для видалення всіх предикатів використовується предикат retractall із застосуванням анонімної змінної як його першого параметра. Предикат retractall може бути замінений комбінацією предикатів retract і fail у такий спосіб:

```
retractall(Fact):- retract(Fact), fail.
retractall(_).
```

Для збереження динамічної бази на диску використовується предикат save, який зберігає її в текстовому файлі з ім'ям, яке було зазначено як перший параметр предиката [2–3].

Порядок виконання практичного заняття

1. Ознайомитися з запропонованим прикладом написання програми на Prolog, яка показує, як видалити з бази даних інформацію про людину, чиє ім'я *Fred* та інформацію про тих, хто любить *broccoli* та *money.*

```
DATABASE
   person(symbol, symbol, integer)

DATABASE - likesDatabase
   likes (symbol, symbol)
   dislikes (symbol, symbol)
```

```
CLAUSES
    person ("Fred", "Capitola", 35).
    person ("Fred", "Omaha", 37).
    person ("Michael", "Brooklyn", 26).

    likes ("John", "money").
    likes ("Jane", "money").
    likes ("Chris", "chocolate").
    likes ("John", "broccoli").
    dislikes ("Fred", "broccoli").
    dislikes ("Michael", "beer").

GOAL
    retract (person("Fred", _, _)),
    retract (likes(_, "broccoli")),
    retract (likes(_, "money"), likesDatabase).
```

2. Запустити програму на виконання та проаналізувати отримані результати.

3. Перефразувати ціль, яка ілюструє, як можна отримати значення з предиката `retract`:

```
GOAL
     retract (person (Name, Address, Age)),
     write (Name, " , ", Age), nl.
```

4. Запустити програму на виконання та проаналізувати отримані результати.

Контрольні питання

1. Дати визначення бази даних.
2. Які типи баз даних Вам відомі?
3. У якому розділі Prolog–програми описуються факти внутрішньої бази даних?
4. Яке стандартне ім’я має розділ опису предикатів бази даних?
5. Які диз’юнкти зберігаються у внутрішній базі даних?
6. За допомогою яких предикатів можна додавати факти у внутрішню базу даних? Яка різниця у використанні цих предикатів?
7. Які предикати використовуються для знищення фактів із бази даних?
8. Описати роботу з динамічною базою даних.
9. Описати призначення вбудованого предиката `save`.

ПРАКТИЧНЕ ЗАНЯТТЯ 11
ЕКСПЕРТНІ СИСТЕМИ

Мета заняття – проектування простих експертних систем.

Завдання до виконання практичного заняття

Виконати наведені програми, доповнити базу даних програм своїми фактами і правилами, перефразувати цілі до програм, отримані результати проаналізувати, відповісти на контрольні питання.

Загальні положення

Експертними системами називають комп'ютерні програми, які здатні накопичувати знання, що містяться в різних джерелах, і моделювати процес експертизи. Основним призначенням експертної системи є розроблення програмних засобів, які під час вирішення завдань, важких для людини, одержують результати, що не поступаються за якістю й ефективністю рішень, одержаним експертом. Використання ЕС виглядає так: користувач уводить інформацію на деякій формалізованій мові й отримує відповідь системи, яка може бути: або рішенням проблемної ситуації, яка описана користувачем, або вказівкою на те, що необхідні додаткові дані, або знання, або судження про те, що рішення не існує. База знань ЕС вміщує факти (дані) та правила, які використовують факти для прийняття рішень. Механізм виводу вміщує інтерпретатор, який визначає як використовувати правила для виведення нових знань, та диспетчер, який установлює порядок використання цих правил.

У програмуванні, орієнтованому на правила, поводження визначається множиною правил вигляду умова – дія. Умова задає образ даних, у разі виникнення якого дія правила може бути виконана. У парадигмі, що орієнтована на правила, поводження (послідовність дій) задається не заздалегідь визначеною послідовністю правил, а формується на основі тих даних і їхніх значень, які в поточний момент обробляються програмою (системою). Формування поводження здійснюється за такою схемою: умови правил зіставляються з поточними даними, і ті правила, умови яких задовольняються значеннями поточних даних, стають претендентами на виконання. На наступному кроці за певним критерієм виконується вибір одного правила серед претендентів і його виконання, тобто виконання дії, зазначеної в правій частині правила [15].

Порядок виконання практичного заняття

1. Ознайомитися із запропонованим прикладом 1 класифікаційної експертної системи [2–3].

Приклад 1.

```
DOMAINS
    thing = string
    conds = cond*
    cond = string

DATABASE
    is_a (thing, thing, conds)
    type_of (thing, thing, conds)
    false (cond)

PREDICATES
    nondeterm run (thing)
    nondeterm ask (conds) Update

CLAUSES
    run (Item) :- is_a (X, Item, List), ask(List),
    type_of (ANS, X, List2), ask (List2),
    write(Item," you need is a/an ", Ans), nl.
    run(_) :-
    write ("This program does not have enough "),
    write ("data to draw any conclusions.").
    ask([]).
    ask([H|T]) :- not(false (H)),
    write("Does this thing help you to"),
    write (H, "(enter y/n)"),
    readchar(Ans), nl, Ans = 'y',
    ask (T).
    ask([H|_]) :-
    assertz (false (H)), fail.
    is_a (language, tool, ["communicate"]).
    is_a (hammer, tool, ["build a house", "fix a fender", "crack a nut"]).
    is_a (sewing_machine, tool, ["make clothing", "repair sails"]).
    is_a (plow, tool, ["prepare fields", "farm"]).
    type_of (english, language, ["communicate with people"]).
    type_of (prolog, language, ["communicate with a computer"]).
    update :-
    retractall (type_of (prolog, language, ["communicate with a
    computer"])),
    asserta (type_of ("Visual Prolog", language, ["communicate with a
    personal computer"])),
```

```
asserta (type_of (prolog, language, ["communicate with a mainframe computer"])).

GOAL
    run (tool).
```

2. Запустити програму на виконання, результат виконання якої наведено на рисунку 2.5.

Рисунок 2.5 – Рекомендації ЕС з вибору предмета вивчення для користі спілкування

3. Ввести таку ціль:

```
GOAL
   update, run(tool).
```

Необхідно звернути увагу на те, що предикат *update*, який включено до початкового коду програми, знищує факт

(type_of (prolog, language, ["communicate with a computer"]))

з внутрішньої бази фактів та додає два нових факти в програму:

(type_of ("Visual Prolog", language, ["communicate with a personal computer"])),
(type_of (prolog, language, ["communicate with a mainframe computer"])).

4. Доопрацювати програму так, щоб видавалися реальні висновки про вподобання студента.

5. Ознайомитися з запропонованим прикладом 2 написання програми на Prolog, у якому показано, як конструювати невелику ЕС для визначення однієї з семи тварин. ЕС буде обирати тварину, поставивши питання та використовуючи метод дедукції залежно від отриманої відповіді. Цей приклад демонструє механізм повернення з використанням фактів і способи ефективного використання предиката `not`.

Приклад 2.

```
DATABASE
   xpositive (symbol, symbol)
```

```
    xnegative (symbol, symbol)

PREDICATES
    nondeterm animal_is (symbol)
    nondeterm it_is (symbol)
    ask (symbol, symbol, symbol)
    remember (symbol, symbol, symbol)
    positive (symbol, symbol)
    negative (symbol, symbol)
    clear_facts
    run

CLAUSES
    animal_is (cheetah) :-
    it_is (mammal),
    it_is (carnivore),
    positive (has, tawny_color),
    positive (has, dark_spots).

    animal_is (tiger) :-
    it_is (mammal),
    it_is (carnivore),
    positive (has, tawny_color),
    positive (has, black_stripes).

    animal_is (giraffe) :-
    it_is (ungulate),
    positive (has, long_neck),
    positive (has, long_legs),
    positive (has, dark_spots).

    animal_is (zebra) :-
    it_is (ungulate),
    positive (has,black_stripes).

    animal_is (ostrich) :-
    it_is (bird),
    negative (does, fly),
    positive (has, long_neck),
    positive (has, long_legs),
    positive(has, black_and_white_color).

    animal_is (penguin) :-
```

```
it_is (bird),
negative (does, fly),
positive (does, swim),
positive (has, black_and_white_color).

animal_is (albatross) :-
it_is (bird),
positive (does, fly_well).

it_is (mammal) :- positive(has, hair).
it_is (mammal) :- positive (does, give_milk).
it_is (bird) :- positive (has, feathers).
it_is (bird) :- positive (does, fly), positive (does, lay_eggs).
it_is (carnivore) :- positive (does, eat_meat).
it_is (carnivore) :- positive (has, pointed_teeth),
positive (has, claws),
positive (has, forward_eyes).
it_is (ungulate) :- it_is (mammal), positive (has ,hooves).
it_is (ungulate) :- it_is (mammal), positive (does, chew_cud).

positive (X, Y) :- xpositive (X, Y), !.
positive (X,Y) :- not (xnegative (X, Y)),
ask (X, Y, yes).

negative (X, Y) :- xnegative (X, Y), !.
negative (X, Y) :- not (xpositive (X, Y)),
ask (X, Y, no).

ask (X, Y, yes) :- !,
write (X, " it ", Y, '\n'),
readln (Reply), nl,
frontchar (Reply, 'y' , _),
remember(X, Y, yes).

ask (X, Y, no) :- !,
write (X, " it ",Y, '\n'),
readln (Reply), nl,
frontchar(Reply, 'n' , _),
remember(X, Y, no).
remember (X, Y, yes):- assertz (xpositive (X, Y)).
remember (X, Y, no) :- assertz (xnegative (X, Y)).
```

```
clear_facts :-
write("\n\n Please press the space bar to exit\n"),
retractall (_, dbasedom) , readchar (_).

run :- animal_is (X), !,
write ("\n Your animal may be a ", X),
nl, nl, clear_facts.
run :- write ("\n Unable to determine what"),
write ("your animal is. \n\n"),
clear_facts.

GOAL
run.
```

6. Запустити програму на виконання. Результат виконання програми наведено на рисунку 2.6.

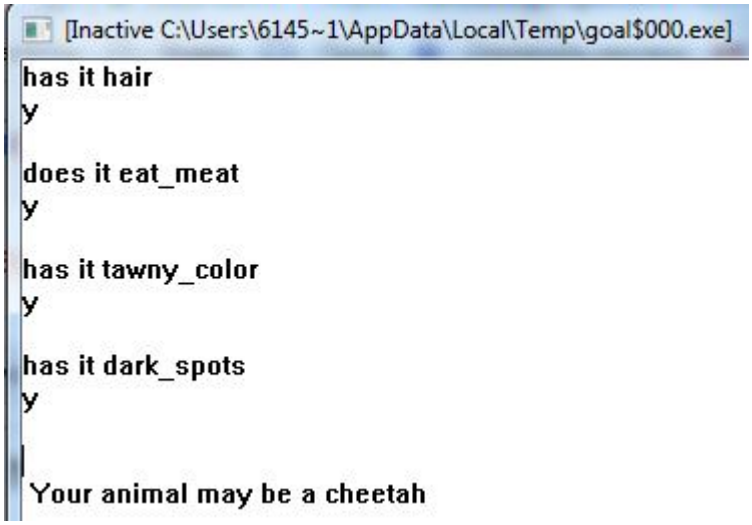

Рисунок 2.6 – Рекомендації ЕС на підставі вибору характерних ознак гепарда

Пояснення щодо роботи експертної системи. Кожна тварина описана рядом ознак, які вона має (або не має). Для простоти цей приклад програми розглядає тільки позитивні та негативні відповіді, для яких використовуються два предикати:

```
DATABASE
    xpositive (symbol, symbol)
    xnegative (symbol, symbol).
```

Наприклад, факт, що тварина не має шерсті подається так:

xnegative (has, hair).

Питання ЕС до користувача знаходяться в правилах *positive (X, Y)* і *negative (X, Y),* які перевіряють, чи відома відповідь, перед тим як з'ясувати це у користувача:

```
positive(X,Y)  :-  xpositive (X, Y).
positive(X,Y)  :-  not  (xnegative (X, Y)), ask (X, Y, yes).

negative(X,Y)  :-  xnegative (X, Y).
negative(X,Y)  :-  not  (xpositive (X, Y)), ask (X, Y, no).
```

Друге правило для обох предикатів виконується обов'язково для того, щоб перед питанням не виникло протиріччя. Предикат *ask* задає питання та організовує запам'ятовування відповіді. Якщо відповідь починається з літери *Y*, то система визначає відповідь *Так,* а, якщо – з літери *N*, то відповідь – *Ні*.

```
/*  Asking Questions and Remembering-Answers  */
    ask(X, Y, yes)  :-  !,
    write(X, " it ", Y,  '\n'),
    readln(Reply),
    frontchar  (Reply,  'y' ,  _  ),
    remember (X, Y, yes).

    ask (X, Y, no)  :-  !,
    write  (X, " it ", Y,  '\n'),
    readln  (Reply),
    frontchar  (Reply,  'n',  _),
    remember(X, Y, no).

    remember (X, Y, yes)  :-
    assertz  (xpositive (X, Y)).

    remember (X, Y, no)  :-
    assertz  (xnegative (X, Y)).

/*  Clearing Out Old Facts  */
    clear_facts  :-
    write  ("\n\n  Please press the space bar to  exit\n"),
    retractall  (_,dbasedorn),  readchar  (_).
```

7. Доповнити програму інформацією про обрану студентом тварину та під час виконання програми показати викладачеві результати доопрацювання програми.

8. Ознайомитися з прикладом 4 простої ЕС, яка розбиває введене слово на склади.

Приклад 4.

```
DOMAINS
    letter = char
    word_ = letter*

PREDICATES
    nondeterm divide (word_, word_, word_, word_)
    vocal (letter)
    consonant (letter)
    nondeterm string_word (string, word_)
    append (word_, word_, word_)
    nondeterm repeat

CLAUSES
    divide (Start, [T1, T2, T3 | Rest], D1, [T2, T3 | Rest]) :-
    vocal (T1), consonant (T2), vocal (T3),
    append (Start, [T1], D1).
    divide (Start, [T1, T2, T3, T4 | Rest], D1, [T3, T4 | Rest]) :-
    vocal (T1), consonant (T2), consonant (T3), vocal (T4),
    append (Start, [T1, T2], D1).
    divide (Start, [T1 | Rest], D1 ,D2) :-
    append (Start, [T1], S),
    divide (S, Rest, D1, D2).
    vocal ('a'). vocal ('e'). vocal ('i').
    vocal ('o'). vocal ('u'). vocal ('y').
    consonant (B) :-
    not (vocal (B)), B <= 'z' , 'a' <= B.
    string_word ("", [] ) :- !.
    string_word (Str, [H | T] ) :-
    bound (Str), frontchar (Str, H, S), string_word (S, T).
    string_word (Str, [H | T] ) :-
    free (Str), bound (H), string_word (S, T), frontchar (Str, H, S).
    append ([], L, L) :- !.
    append ([X | L1], L2, [X | L3]) :-
    append (L1, L2, L3).
    repeat.
```

```
repeat :- repeat.

GOAL
    repeat,
    write ("Write a multi-syllable word: "),
    readln (S), nl,
    string_word (S, Word),
    divide ( [], Word, Part1, Part2),
    string_word (Syllable1, Part1),
    string_word (Syllable2, Part2),
    write ("Division: ", Syllable1, "-", Syllable2), nl,
    fail.
```

Пояснення до алгоритму роботи програми. Програма пропонує ввести слово, а потім робить спробу розбити його на склади, наприклад, ruler > ru-ler, prolog > pro-log. У подальшому програма створює список із літер:

```
letter = char
word_ = letter*.
```

У програмі задекларовані предикати, які розрізняють голосні (*vocal*) та приголосні літери (*consonant*). Предикат *vocal* об'єднує такі голосні літери: *a, e, i, o;* і виокремлює літеру *y*. У програмі використані вбудовані предикати: `append`, `frontchar`, `free`, `bound`, а створений предикат *divide* розбиває слова на склади.

9. Запустити програму на виконання. Результат виконання програми для слова «визначити» наведено на рисунку 2.7.

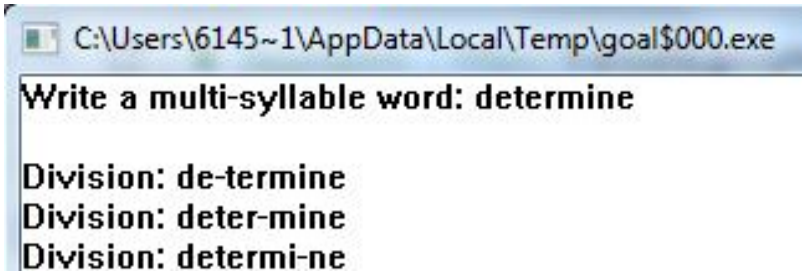

Рисунок 2.7 – Результат розбиття слова «determine » на склади

Контрольні питання

1. Дати визначення експертної системи.
2. Описати структуру правила в продукційних системах.
3. Які переваги та недоліки має логічна модель подання знань?
4. Описати призначення вбудованого предиката `frontchar`.
5. Описати призначення вбудованого предиката `free`.

6. Описати призначення вбудованого предиката `bound`.

7. Описати призначення предиката *ask* (`symbol, symbol, symbol`).

8. Описати призначення предиката *remember* (`symbol, symbol, symbol`).

9. Описати синтаксис та призначення вбудованих предикатів `frontchar, free, bound`.

10. Описати призначення предиката *run*.

ПРАКТИЧНЕ ЗАНЯТТЯ 12
ЕКСПЕРТНА СИСТЕМА «ВИЗНАЧЕННЯ МОВИ ПРОГРАМУВАННЯ ДЛЯ НАВЧАННЯ»

Мета заняття – розроблення та використання ЕС для вирішення завдання комп'ютерного спрямування.

Завдання до виконання практичного заняття

Виконати наведені програми, доповнити базу даних програм своїми фактами і правилами, перефразувати цілі до програми, проаналізувати отримані результати, відповісти на контрольні питання.

Загальні положення

ЕС – це інтелектуальна прикладна програмна система, яка здатна накопичувати знання, що містяться в різних джерелах, і моделювати процес експертизи. Основною ознакою класичної ЕС є наявність бази знань (БЗ), написаної на спеціальних мовах, близьких до природньої мови. БЗ відображає модель та спосіб прийняття нею рішень, тому містить декларативну і процедурну частини. Загалом, така система займається доказом теореми, що дозволяє встановити істинність виводу за певної інтерпретації змінних. Засобом реалізації подібних моделей є мови та середовища логічного програмування [1–3, 4–5, 9–12].

Логічне програмування гарно підходить для вирішення проблем, для роботи з формальними й природними мовами, для баз даних, запитальних й експертних систем і для інших дискретних необчислювальних завдань. Користувача приваблює ясність, змістовність програм та їхній нетехнічний характер. У програмі не потрібно описувати, яким чином вирішується завдання. Досить опису самого завдання й того, що бажано довідатися.

Prolog і логічне програмування безупинно розширюються, охоплюючи все нові методи програмування й форми зображення саме в напрямку процедурного й об'єктно-орієнтованого програмування, а також у напрямку паралельних обчислень [1–3, 16].

Порядок виконання практичного заняття

1. Ознайомитися з запропонованим прикладом роботи експертної системи для визначення мови програмування для навчання. Наразі існує велика кількість мов програмування, кожна з яких має свої недоліки і переваги через те, що вони орієнтовані на різні сфери програмної діяльності.

Під час розроблення експертної системи були розглянуті лише найпопулярніші мови програмування: одна відмінно працює з браузерами – JavaScript, але абсолютно не придатна для написання Flash, інша для роботи з базою даних – PHP. Кожен програміст розпочинає з найпростіших із них і наприкінці навчання вибирає або ту, яка найбільше підходить йому через напрямки його діяльності, і стає фахівцем у цій галузі, або ж продовжує потроху використовувати кожну з цих мов програмування.

Приклад.

```
DATABASE
    xpositive (symbol, symbol)
    xnegative (symbol, symbol)

PREDICATES
    nondeterm language_is (symbol)
    ask (symbol, symbol, symbol)
    remember (symbol, symbol, symbol)
    positive (symbol, symbol)
    negative (symbol, symbol)
    clear_facts
    run

CLAUSES
    language_is ("PHP") :-
    positive (" язык", "объектно-ориентированного программирования"),
    positive (" язык,", "который имеет объекты в качестве основных
    логических конструктивных элементов"),
    positive (" язык,", "который поддерживает наследования"),
    positive (" язык,", "который доступный в освоении"),
    positive (" язык", "скриптовый язык"),
    positive (" язык", "кросплатформенный"),
    negative (" язык", "с написанием оконного приложения").

    language_is ("Visual Prolog"):-
    positive (" язык","логический"),
    positive (" язык,", "который доступный в освоении"),
    positive (" язык,", "который использует факты"),
```

```
positive (" язык,", "который имеет конкретные запросы"),
positive (" язык,", "который использует правила").

language_is ("Visual Basic") :-
positive (" язык","объектно-ориентированного программирования"),
positive ("язык", "который имеет объекты в качестве основных
логических конструктивных элементов"),
positive (" язык", "который поддерживает наследования"),
positive (" язык", "который доступный в освоении"),
positive (" язык", "который имеет графический интерфейс"),
positive (" язык", "который поддерживает инкапсуляцию"),
positive (" язык", "который включает в себя процедуры").

language_is ("C++") :-
positive (" язык","объектно-ориентированного программирования"),
positive (" язык,", "который имеет объекты в качестве основных
логических конструктивных элементов"),
positive (" язык,", "который поддерживает наследования"),
positive (" язык,", "который поддерживает инкапсуляцию"),
positive (" язык,", "который поддерживает полиморфизм"),
positive (" язык,", "который тяжелый в освоении").

language_is("Delphi") :-
positive (" язык","объектно-ориентированного программирования"),
positive (" язык,", "который имеет объекты в качестве основных
логических конструктивных элементов"),
positive (" язык,", "который поддерживает наследования"),
positive (" язык,", "который поддерживает инкапсуляцию"),
positive (" язык,", "который поддерживает полиморфизм"),
positive (" язык,", "который доступный в освоении").

language_is ("Lisp") :-
positive (" язык","функциональный"),
positive (" язык,", "который доступный в освоении"),
positive (" язык,", "который поддерживет динамическую смену типов"),
positive (" язык,", "который поддерживает полноценные средства
символьной обработки").

language_is ("Miranda") :-
positive (" язык", "функциональный"),
positive (" язык,", "который тяжелый в освоении"),
positive (" язык,", "который поддерживет динамическую смену типов"),
positive (" язык,", "который имеет строгую полиморфную систему
типов"),
```

positive (" язык,", "который поддерживает типы данных пользователя").

language_is ("Perl") :-
positive (" язык", "динамический"),
positive (" язык,", "который имеет богатые возможности для работы с текстом"),
positive (" язык,", "который поддерживает переменные"),
positive (" язык,", "который тяжелый в освоении"),
positive (" язык,","который поддерживает выражение присваивания").

language_is ("JavaScript") :-
positive (" язык", "динамический"),
positive (" язык,", "который доступный в освоении"),
positive (" язык,", "который применение находит в браузерах"),
positive (" язык,", "который имеет динамическую типизацию"),
positive (" язык,", "который поддерживает стандартные интерфейсы к веб-серверам и браузерам").

positive (X,Y) :- xpositive (X, Y), !.
positive (X,Y) :- not (xnegative (X, Y)), ask (X, Y, yes).
negative (X,Y) :- xnegative (X, Y), !.
negative (X,Y) :- not (xpositive (X,Y)), ask (X, Y, no).
ask (X, Y, yes) :- !, write ("Вам нужен", X, " ", Y, '\n'),
readln (Reply), nl, frontchar (Reply, 'y', _), remember (X, Y, yes).
ask (X, Y, no) :- !, write ("Вам нужен", X, " ", Y, '\n'),
readln (Reply), nl, frontchar (Reply ,'n', _), remember (X, Y, no).
remember (X, Y, yes) :- assertz (xpositive (X, Y)).
remember (X, Y, no) :- assertz (xnegative (X, Y)).

run :- write ("Добро пожаловать!\n"),
write ("Вас приветствует программа, которая позволит Вам отпределить, какой именно язык программирования\n Вам больше подойдет для достижения ваших целей.\n Давайте ответы y – Да, n – Нет, если Вам это подходит.\n"), language_is (X), !,
write ("\Вам подойдет язык программирования ", X),
nl, nl, clear_facts.
run :-
write ("\n Невозможно определить"),
write ("Вам подойдет язык программирования .\n\n"),
clear_facts.
clear_facts :-
write ("\n\n Пожалуйста, нажмите пробел, чтобы выйти\n"),

```
    retractall (_, dbasedom), readchar (_).

GOAL
    run.
```

Результати роботи ЕС наведено на рисунку 2.8.

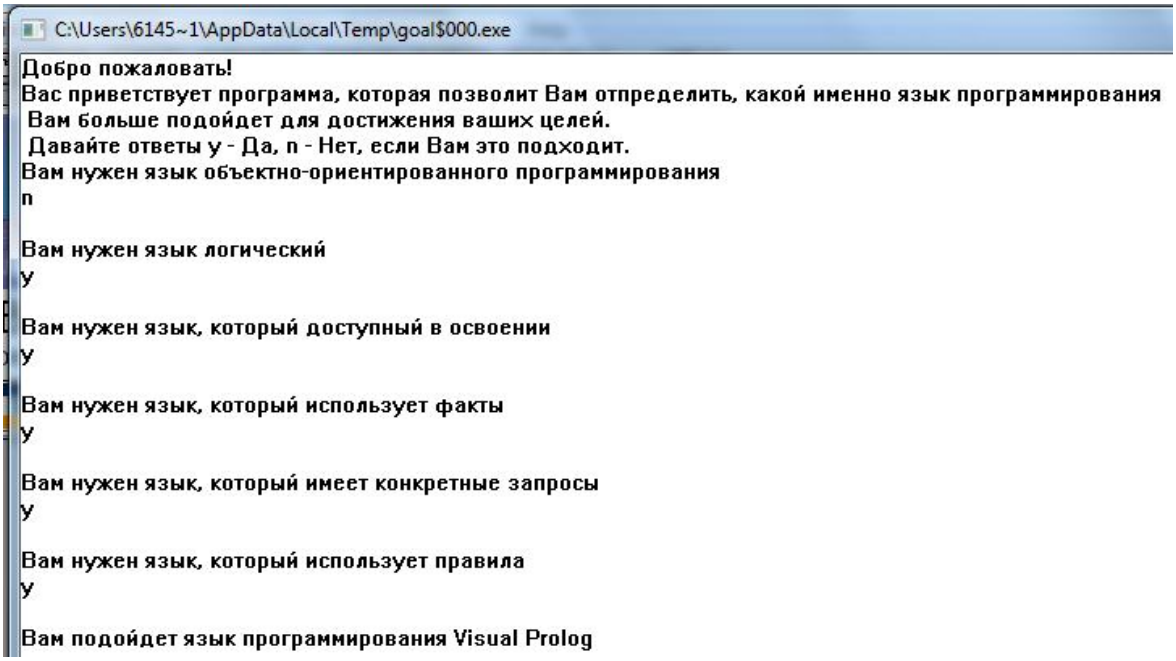

Рисунок 2.8 – Рекомендації ЕС із вибору мови програмування на підставі аналізу вимог для поставленого завдання

2. Додати до бази даних власні факти, правила для визначення мови програмування для навчання. Виконати програму, проаналізувати отримані результати.

Контрольні питання

1. Як організовано інтерфейс користувача в цих ЕС?
2. Пояснити призначення предикатів у розділі `PREDICATES`.
3. Пояснити призначення фактів та предикатів, які використані в розділі `CLAUSES`.
4. Пояснити призначення предиката `fail`.
5. Пояснити призначення предиката `retractall`.
6. Пояснити призначення використання предиката `assertz` у цій програмі.
7. Які вбудовані предикати використані в цих ЕС?

ПРАКТИЧНЕ ЗАНЯТТЯ 13
ЕКСПЕРТНА СИСТЕМА «ВИБІР ТРИГЕРА»

Мета заняття – розроблення та використання ЕС для вирішення завдань, пов'язаних з електронними приладами та пристроями.

Завдання до виконання практичного заняття

Виконати наведені програми, доповнити базу даних програм своїми фактами і правилами, перефразувати цілі до програми, проаналізувати отримані результати, відповісти на контрольні питання.

Загальні положення

Тригер – це електронний пристрій, який призначається для запису і зберігання інформації і який має два виходи: прямий і інверсний, і деяку кількість входів, залежно від виконуваного завдання. Під дією вхідних сигналів змінюється стан виходів. Напруга на виходах змінюється різко – стрибкоподібно. Для виготовлення тригерів зазвичай використовуються напівпровідникові прилади [17].

Інформація може записуватися в тригери вільно (безперервно), тобто при подачі сигналів на вхід, стан виходу змінюється в реальному часі. Такі тригери називаються асинхронними.

Синхронні тригери характеризуються тим, що інформація може записуватися тільки коли активний синхронизуючий сигнал, і у разі відсутності позитивного рівня напруги на ньому інформація на виходах змінитися не може.

Найрозповсюдженішими є тригери *RS*, *JK*, *D* і *T*-типів.

RS-тригер (від англ. `Set/Reset` – встановити/скинути) – асинхронний тригер, який зберігає свій попередній стан при неактивному стані обох входів і змінює свій стан при подачі на один з його входів активного рівня. При подачі на обидва входи активного рівня стан тригера, взагалі кажучи, невизначено, але в конкретних реалізаціях на логічних елементах обидва виходи приймають стан або логічного нуля, або логічної одиниці.

Схема *RS*-тригера дозволяє запам'ятовувати стан логічної схеми, але так як в початковий момент часу може виникати перехідний процес (в цифрових схемах цей процес називається «небезпечні гонки»), то запам'ятовувати стан логічної схеми в тригерах потрібно тільки в певні моменти часу, коли всі перехідні процеси закінчені. Це означає, що більшість цифрових схем вимагають сигналу синхронізації (тактового сигналу). Всі перехідні процеси в комбінаційній логічній схемі повинні закінчитися за час періоду синхросигналу, що подається на входи тригерів. Тригери, що запам'ятовують вхідні сигнали тільки в момент часу, який визначається сигналом синхронізації, називаються синхронними тригерами.

В *RS*-тригерах для запису логічного нуля і логічної одиниці потрібні різні входи, що не завжди зручно. При запису і зберіганні даних один біт може приймати значення, як нуля, так і одиниці, і для його передачі досить одного

дроту. Сигнали установки і скидання тригера не можуть з'являтися одночасно, тому можна об'єднати ці входи за допомогою інвертора. Такий тригер отримав назву *D*-тригер, і назва його походить від англійського слова Delay – затримка. Конкретне значення затримки визначається частотою проходження імпульсів синхронізації. *D*-тригер запам'ятовує стан входу і видає його на вихід. *D*-тригери мають, як мінімум, два входи: інформаційний D і синхронізації C. Вхід синхронізації C може бути статичним (потенційним) і динамічним.

Таблиця істинності *JK*-тригера практично збігається з таблицею істинності *RS*-тригера. Для того щоб виключити заборонений стан, його схема змінена таким чином, що при подачі двох одиниць *JK*-тригер перетворюється в рахунковий тригер. Це означає, що при подачі на тактовий вхід C імпульсів цей тригер змінює свій стан на протилежний.

T-тригер (від англ. Toggle – перемикач) – це рахунковий тригер. У даного тригера є тільки один вхід. Принцип роботи *T*-тригера полягає в наступному. Після надходження на вхід *T* імпульсу стан тригера змінюється на прямо протилежний. Рахунковим він називається тому, що *T*-тригер як би підраховує кількість імпульсів, що надійшли на його вхід. При отриманні другого імпульсу *T*-тригер знову скидається в початковий стан [18].

Порядок виконання практичного заняття

1. Ознайомитися з ЕС, що призначена для знаходження необхідного тригера з видачею рекомендації про режими його роботи.

Приклад.

```
DATABASE
    yes (string)
    no (string)

PREDICATES
    nondeterm repeat
    nondeterm run
    nondeterm trigger (string, string)
    positive (string)
    negative (string)
    nondeterm xpositive (string)
    nondeterm xnegative (string)
    ask (string, char)
    remember (string, char)
    nondeterm clear

    CLAUSES
    repeat.
```

```
repeat :- repeat.
trigger ("RS-тригер асинхронний.\n При подачі одиниці на вхід S вихідний стан стає рівним логічній одиниці. При подачі одиниці на вхід R вихідний стан стає рівним логічному нулю.\n Подача одиниці одночасно на обидва входи R і S є забороненою,", "оскільки вводить схему в режим генерації. В складніших реалізаціях RS-тригер \n переходить в третій стан QQ=00.") :-
positive ("Тригер зберігає свій попередній стан при нульових входах та змінює свій вихідний стан при подачі на один з його входів одиниці? \n"),
positive ("Тригер має 3 стани, а саме 2 стійких та 1 нестійкий? \n"),
positive ("Використовується для створення сигналу з позитивним та негативним фронтами? \n"), ! .

trigger ("RS-тригер синхронний.\n При подачі одиниці на вхід S вихідний стан стає рівним логічній одиниці. При подачі одиниці на вхід R вихідний стан стає рівним логічному нулю.\n Подача одночасно на обидва входи R і S одиниці є забороненою,", "оскільки вводить схему в режим генерації. В складніших реалізаціях RS-тригер \n переходить в третій стан QQ=00.") :-
positive ("Схема збігається зі схемою одноступеневого парафазного (двофазного) D-тригера ? \n"),
positive ("У тригера алгоритм функціонування Q(t+1)=R*(Q(t)+S)+x*S*R ? \n"), ! .

trigger ("D - тригер синхронний.\n Після приходу активного фронту імпульсу синхронізації на вхід C D-тригер відкривається. Збереження інформації в D-тригерах \n відбувається після спаду імпульсу синхронізації C.", " Оскільки інформація на виході залишається незмінною до приходу чергового імпульсу \n синхронізації, D - тригер називають також тригером із запам’ятовуванням інформації або тригером-засувкою") :-
positive ("Тригер запам’ятовує стан входу та видає його на вихід? \n"),
positive ("Має два входи: інформаційний і синхронізації? \n"),
positive ("Використовується для реалізації засувки? \n"), ! .

trigger ("D - тригер двоступеневий.\n Спочатку інформація записується в першу сходинку, а вже потім переписується у другу та з’являється на виході.\n Двоступеневий D - тригер називають тригером з динамічним керуванням.", "") :-
positive ("Має дві рівня запам’ятовування інформації? \n"),
positive ("Спочатку інформація записується в першу сходинку, а вже потім переписується у другу та з’являється на виході? \n"), ! .
```

```
trigger ("T - тригер асинхронний.\n Асинхронний Т-тригер не має входу дозволу рахунку - T і переключається за кожним тактовим імпульсом на вході C.", "") :-
positive ("Не має входу дозволу рахунку - T? \n"),
positive ("Переключається за кожним тактовим імпульсом на вході C? \n"), !.

trigger ("T - тригер синхронний.\n При одиниці на вході T, за кожним тактом на вході C змінює свій логічний стан на протилежний,\n і не змінює вихідний стан при нулі на вході T.", "T - тригер можна побудувати на JK - тригері, \n на двоступеневому D - тригері і на двох одноступеневих D - тригерах та інверторі.") :-
positive ("Застосовують для пониження частоти в 2 рази? \n"), !.

trigger ("JK - тригер.\n Працює так само як RS - тригер, з одним лише винятком: при подачі логічної одиниці на обидва входи \n J і K стан виходу тригера змінюється на протилежний.", "При подачі одиниці на вхід J і нуля на вхід K \n вихідний стан тригера стає рівним логічній одиниці. А при подачі одиниці на вхід K і нуля на вхід J \n вихідний стан тригера стає рівним логічному нулю. ") :-
positive ("При подачі логічної одиниці на обидва входи J і K стан виходу тригера змінюється на протилежний? \n" ),
positive ("На базі цього тригера можливо побудувати D-тригер або T-тригер? \n"), !.

positive(X) :- xpositive (X), !; xnegative (X), !, fail; ask (X, 'y').
negative (X) :- xnegative (X), !; xpositive (X), !, fail; ask (X, 'n').
ask (X, R) :- write (X), readchar (Reply), write (Reply, "\n"),
remember (X, Reply), R = Reply.

xpositive (X) :- yes (X).
xnegative (X) :- no (X).

remember (X, 'y') :- asserta (yes (X)).
remember (X, 'n') :- assertz (no (X)).
clear :- retract (yes (_)), clear.
clear :- retract (no (_)), clear.
clear.
run :- clear, repeat,
write ("Ласкаво просимо в експертну систему, в якій Ви можете підібрати тригер і дізнатися як він працює. \n"),
write ("Для отримання результату прийміть рішення: y-yes або n-no
```

```
\n"), write ("\n"),
trigger (X, Y),
write ("Ми підібрали для Вас: ", X, Y, "\n"),
write ("\n\n Щоб вийти з експертної системи натисніть spacebar\n"),
retractall (_, dbasedom), readchar (_).

GOAL
run.
```

Результат роботи ЕС наведено на рисунку 2.9.

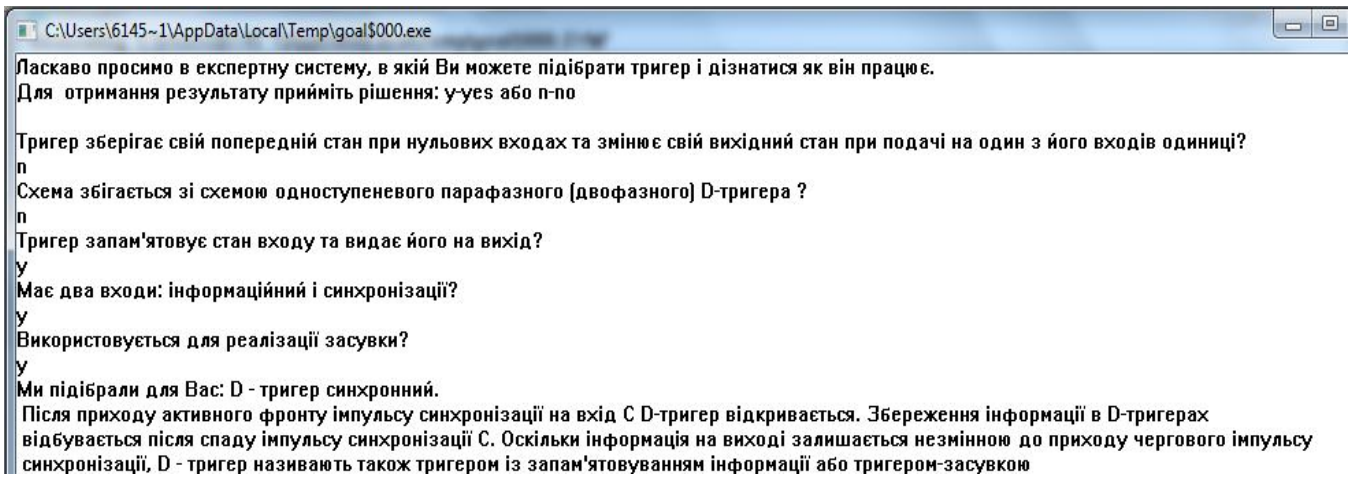

Рисунок 2.9 – Результат роботи ЕС, яка допоможе обрати необхідний тригер

2. Додати до бази даних власні факти, правила для вибору тригера. Виконати програму, проаналізувати отримані результати.

Контрольні питання

1. Як організовано інтерфейс користувача в ЕС?
2. Пояснити призначення предикатів у розділі `PREDICATES`.
3. Пояснити призначення фактів та предикатів, які використані в розділі `CLAUSES`.
4. Пояснити призначення предиката `fail`.
5. Пояснити призначення предиката `retractall`.
6. Пояснити призначення використання предиката `assertz` у цій програмі.
7. Які вбудовані предикати використані в цих ЕС?

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Джарратано Дж. Экспертные системы : принципы разработки и программирование / Дж. Джарратано, Г. Райли : пер. с англ. – 4-е изд. – М. : Вильмс, 2007. – 1152 с.
2. Prolog Development Center A/S (PDC) (Дания) [Електронний ресурс]. – Режим доступу : http://tecon.com.ua/prolog-development-center-denmark.html.
3. Visual Prolog [Електронний ресурс]. – Режим доступу : http://www.visual-prolog.com/.
4. Хоггер К. Введение в логическое программирование / К. Хоггер : пер. с англ. – М. : Мир, 1988. – 348 с.
5. Доорс Дж. Пролог язык программирования будущего / Дж. Доорс, А. Р. Рейблейн, С. Вадера. – М. : ФиС, 1990. – 144 с.
6. Люгер Дж. Ф. Искусственный интеллект. Стратегии и методы решения сложных проблем / Дж. Ф. Люгер. – М. : Вильямс, 2003. – 539 с.
7. John Alan Robinson: A Machine-Oriented Logic Based on the Resolution Principle. J. ACM, 1965. – 12(1). – P. 23–41.
8. Чень Ч. Математическая логика и автоматическое доказательство теорем / Ч. Чень, Р. Ли : пер. с англ. ; под ред. С. Ю. Маслова. – М. : Наука, 1983 . – 287 с.
9. Адаменко А. Логическое программирование и Visual Prolog / А. Адаменко, А. Кучуков. – СПб : БХВ – Петербург, 2003. – 990 с.
10. Братко И. Программирование на языке «Prolog» для искусственного интеллекта / И. Братко. – М. : Мир, 1990. –315с.
11. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG / И. Братко. – М. : Вильямс, 2004. – 635 с.
12. Малпас Дж. Реляционный язык Prolog и его применение / Дж. Малпас. – М. : Наука, 1990. – 304 с.
13. Логический подход к искусственному интеллекту от классической логики к логике баз данных : пер с франц. ; под ред. П. Грибомон и др. – М. : Мир, 1998. – 356 с.
14. Гаврилова Т. А. Базы знаний интеллектуальных систем / Т. А. Гаврилова, В. Ф. Хорошевский. – СПб. : Питер, 2000. – 384 с.
15. Построение экпертных систем / под ред. Ф. Хейес-Рота и др. – М. : Мир, 1987. – 286 с.
16. Джексон П. Введение в экспертные системы / П. Джексон – М. : Вильямс, 2001. – 624 с.
17. Токхейм Р. Основы цифровой электроники / Р. Токхейм. – М. : Мир, 2009. – 289 с.
18. Димитрова М . 33 схемы на триггерах / М. Димитрова, В. Пунджев. – М. : Мир книг ws, 2008. – 128 с.

Навчальне видання

НОВОЖИЛОВА Марина Володимирівна
ПЕТРОВА Олена Олександрівна

ВИКОРИСТАННЯ МОВИ ЛОГІЧНОГО ПРОГРАМУВАННЯ VISUAL PROLOG ДЛЯ РОЗРОБКИ ЕКСПЕРТНИХ СИСТЕМ

НАВЧАЛЬНИЙ ПОСІБНИК

Відповідальний за випуск *М. В. Новожилова*

Редактор *О. В. Михаленко*

Комп’ютерне верстання *О. О. Петрова*

Дизайн обкладинки *Т. А. Лазуренко*

Підп. до друку 23.01.2019. Формат 60 х 84/16.
Друк на ризографі. Ум. друк. арк. 3,6.
Тираж 50 пр. Зам. № .

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Маршала Бажанова, 17, Харків, 61002.
Електронна адреса: rectorat@kname.edu.ua
Свідоцтво суб’єкта видавничої справи:
ДК № 5328 від 11.04.2017.