Лабораторна робота №6
Керування процесом виконання програми в мові Пролог

Мета роботи: вивчити методи організації циклічних процесів в мові Пролог; познайомитись з предикатом відсікання (рус. – отсечение, англ. – cut, cutting); набути навичок застосування прийомів керування ходом виконання Prolog-програми при розв'язанні практичних задач.

Завдання роботи полягає у практичному вивченні наступних тем:

Частина 1: Методи організації циклічних процесів в мові Пролог.

Частина 2: Предикат відсікання.

Частина 3: Використання прийомів керування ходом виконання програми при розв'язанні практичних задач.

Ця лабораторна лякає тим, що *много буков*. Насправді це якраз добре: завдання займає всього дві сторінки (с. 3–4), а решта документу – це тексти програм для виконання завдань, які вже написані за Вас ☺

Короткі теоретичні відомості

Оператори циклу в тих формах, до яких ми звикли в процедурних мовах програмування – не можуть бути використані в чистому Пролозі з двох причин:

1) Оскільки змінні в Пролозі зберігають свої значення лише в межах одного речення, то до поточного значення лічильника циклу неможливо повернутись після виконання тіла циклу. З цієї причини оператор циклу *for* у Пролозі відсутній.

(*Правда, зберігання значення лічильника можна організувати вручну за допомогою спеціального, досить громіздкого прийому, але про це буде трохи пізніше.*)

2) І саме поняття «виконання тіла циклу» неможливо механічно перенести на Пролог-програми, оскільки в

декларативній програмі послідовність виконання «операторів» не співпадає з тим порядком, в якому вони записані.

Тому циклічні процеси в програмах мовою Prolog доводиться організовувати за допомогою різних спеціальних прийомів, – як правило, основаних на рекурсивних викликах.

Для ознайомлення з цими прийомами в завданні лабораторної роботи наведено ряд програм-прикладів. Студенту необхідно розглянути кожну програму, запустити її, провести трасування і простежити, як вона працює. Після цього зробити висновок.

Мета цих дій – навчитися в своїх програмах використовувати ті прийоми, які використали автори прикладів. Для цього слід по кожній програмі розібратися, як вона працює, і чому працює саме так.

Для зручності наведені приклади на організацію циклів «занумеровані» латинськими буквами: **a**Перша_програма.pro, **b**Друга_програма.pro і т.д.

Підказка: В деяких випадках замість трасування зручно вставляти в програму оператори write, які будуть виводити повідомлення про проходження контрольних точок та значення змінних в цих точках.

Предикат відсікання (*рус.* – отсечение, *англ.* – cut, cutting).

В програмі на Пролозі предикат відсікання має форму знака оклику (!). Призначення предиката відсікання – управління порядком перебору варіантів розв'язків в програмі. Сутність відсікання в тому, що воно відкидає («відсікає») деякі гілки з дерева пошуку. Детальні правила виконання предиката відсікання розглянуто у лекціях.

Відсікання найчастіше застосовують при програмуванні взаємовиключних тверджень, тобто у випадках, коли в програмі розглядаються несумісні альтернативи. Наприклад, натуральне число може бути або парним, або непарним, але не може бути парним і непарним одночасно. Отже, якщо виявлено, що число, з яким ми маємо справу – парне, то виконувати операції для іншої

альтернативи не має сенсу. Оператор відсікання якраз і забезпечує відмову від такого виконання.

Відсікання також вживають для усунення нескінченних циклів (при цьому сценарії його вживання знову-таки базуються на розгляді несумісних альтернатив), при необхідності завершення доказу мети в певних випадках, тощо.

Порядок виконання роботи

Підготовча частина:

1. Ознайомтеся з наведеними теоретичними відомостями.
2. За конспектом лекцій розгляньте приклади організації циклів при підрахунку факторіала, та повторіть тему «Предикати керування ходом виконання програми (fail, відсікання)»

Далі йдуть практичні завдання:

Частина 1: Методи організації циклічних процесів в мові Пролог.

3. Розберіться в роботі програм, що підраховують факторіал: aFACT1.pro, bFACT2.pro, cFACT3.pro. (*Програми наведені в Додатку 1.*)
4. Розберіться в роботі програм dProbcik.pro, eZacik.pro, fForever.pro. (*Програми наведені в Додатку 2.*)

Вказівка: Якщо ваша програма зациклилась і середовище програмування не відповідає, нажміть Ctrl-Break.

5. Розберіться в роботі програм gCycle0.pro, hCycle1.pro, iCycle2.pro. (*Програми наведені в Додатку 3.*)
6. Після ознайомлення з роботою цих програм доробіть одну з них (на Ваш вибір) так, щоб можна було виконувати не лише цикл for(1,5,1), а і цикл з від'ємним кроком (наприклад: for (5,1,-1), for (-1,-5,-1), тощо).

Переконайтеся, що дороблена Вами програма працює коректно – наприклад, запит for (-1,-5, 1) не має виконуватися взагалі (в противному випадку його виконання привело б до зациклювання), і т.д.

Коротше кажучи, доведіть програму до розуму, до якого її не довели автори. І тоді Вам буде чим пишатися, бо автори цих програм – розробники системи Visual Prolog з компанії PDC, з хелпів яких узятий цей матеріал ☺.

7. Розробіть програму «хрестики-нулики» (*див. Додаток 4*).

Частина 2: Предикат відсікання.

8. Розгляньте та дослідіть приклади програм на застосування відсікання, наведені у *Додатку 5*. Намалюйте відповідні дерева пошуку.

9. Реалізуйте приклад програми підрахунку факторіалу з відсіканням (див. лекції). Дослідіть роботу цієї програми.

Частина 3: Використання прийомів керування ходом виконання програми при розв'язанні практичних задач.

10. Розгляньте приклад предметної області (*Додаток 6*). Використовуючи набуті уміння працювати з циклічними процесами та відсіканням, реалізуйте предикати, які виконують вказані запити.

11. Реалізуйте аналогічні запити для предметної області за своїм індивідуальним варіантом.

Заключна частина:

12. Складіть звіт з виконаної роботи.

13. Захистіть звіт. Покажіть викладачу роботу розроблених програм (програму, перероблену в завданні 6, програму «хрестики-нулики», та інші).

14. Дайте відповіді на контрольні запитання.

15*. Запропонуйте приклади, якими можна б було доповнити додатки.

Приклади запитань для самоперевірки:

1. Поясніть, чому в Пролозі неможливе використання оператора циклу for?
2. Як змоделювати в Пролозі оператори циклу do-while, repeat-until?
3. *** У Вас є при собі конспект з записаними правилами застосування відсікання?
4. Що таке «відсікання»?
5. Як відсікання записується в Prolog-програмі?
6. Поясніть правила виконання предиката відсікання (точніше, правила його розгляду інтерпретатором Прологу).
7. Для чого може бути використано відсікання?
8. Як ви вважаєте, чи є зв’язок між відсіканням і відкатом?

ДОДАТОК 1

Програми до завдання 3.

(Розберіться в роботі програм aFACT1.pro, bFACT2.pro, cFACT3.pro.)

Програма FACT1.pro – підраховує факторіал числа.

Програма FACT2.pro перероблена з FACT1.pro з дослідницькою метою. Мені хотілося дізнатись, якою є глибина стеку виклику в середовищі Турбо-Прологу. Програма написана так, що вона переповнює стек і по цьому зупиняється з помилкою Stack overflow (переповнення стека). Розміри стека можна настроїти в опціях компілятора середовища TP, але мені незрозумілий зв'язок між числом, яке там стоїть, і числом циклів, яке виконує програма до зупинки. Спробуйте його якось встановити, змінюючи це число.

Програма FACT3.pro – це варіація на тему FACT1.pro , лише з тією відмінністю, що Турбо-Пролог не розуміє виразу "is (N1, -(N,1))" (ця програма зроблена під інший Пролог). У нас те ж саме записується як N1=(N-1) .

```
aFACT1.pro
/*FACTORIAL*/

predicates
f(integer,real)
clauses
f(1,1).
f(N,FN):-N>1,N1=N-1,f(N1,FN1),FN=FN1*N.

bFACT2.pro
/* Eta programma peredelana
iz programmy factoriala dla togo, chtoby uznat' glubinu steka.
Ona rabotaet pri celi f(382,X),
```

no pishet [Stack overflow] pri f(383,X).
V opcijah kompilatora stoit glubina steka = 600...
*/

```
predicates
f(integer,real)
clauses
f(1,1).
f(N,FN) :- N>1, N1=N-1, f(N1,FN1), FN=FN1+N.
```

cFACT 3.pro

/*Takoe tozhe byvaet, no ne s nami.
Potomu chto nash Prolog etogo ne ponimaet. */

```
domains
    n, f = integer
 predicates
    factorial(n,f)
clauses
   factorial(1,1).
   factorial(N,Res) if
         N > 0 and
         is (N1, -(N,1)) and
         factorial(N1,FacN1) and
         Res = N*FacN1.
```

ДОДАТОК 2.

Тексти програм до завдання 4.

Розберіться в роботі програм dProbcik.pro, eZacik.pro, fForever.pro.

!!! Якщо ваша програма зациклилась, нажміть Ctrl-Break !!!

В тексті цих програм зверніть увагу на таку конструкцію (або похідні від неї):

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

Це – стандартна конструкція, яка часто використовується для зациклювання будь-чого. В деяких Прологах предикат **repeat** навіть визначено як вбудований предикат мови. (І визначено саме так, як тут). В Турбо-Пролозі предикат **repeat** не визначено, тому програмісту доводиться визначати його самостійно.

Перевірте, чи заповнюється стек викликів при використанні **repeat** ?

dProbcik.pro

```
/*proba cycla*/
/*Chto delaet eta programma ?????*/

predicates
  repeat
  www
clauses
  repeat:-write('c'), fail.
  repeat:-repeat.
  www:-write('k'), nl, repeat.
```

eZacik.pro

```
trace
predicates
  repeat
  main

clauses

  repeat.
  repeat:-repeat.
  main :- repeat, readln(X), write(X), X="STOP".
```

fForever.pro

```
/* ORGANIZACIJA BESKONECHNOGO CIKLA.

  VNIMANIE !!!
ESLI trace UBRAT' ,
TO PROGRAMMA ZACIKLITSA...

      Ctrl-Break POMOGAET :)))   */

trace
predicates
  repeat
  main

clauses
  repeat.
  repeat:-repeat.
  main :- repeat, write(1), fail.
```

ДОДАТОК 3

Приклади програм до Завдань 5 та 6.
(gCycle0.pro, hCycle1.pro, iCycle2.pro.)

5. Розберіться в роботі програм gCycle0.pro, hCycle1.pro, iCycle2.pro. (Програми наведені в Додатку 3.)

6. Після ознайомлення з роботою цих програм доробіть одну з них (на Ваш вибір) так, щоб виконувався не лише цикл for(1,5,1), а і цикл з від'ємним кроком (наприклад: for (5,1,-1), for (-1,-5,-1), тощо).

Переконайтеся, що дороблена Вами програма працює коректно – наприклад, запит for (-1,-5, 1) не має виконуватися взагалі (в противному випадку його виконання привело б до зациклювання), і т.д.

Коротше кажучи, доведіть програму до розуму, до якого її не довели автори (автори – розробники системи Visual Prolog з компанії PDC).

```
/*gCycle0.pro - An example from VIP5 folder*/

predicates
for (integer, integer, integer)
cyclebody

clauses
  for(Cur,Max,_):-Cur>Max.
  for(Cur,Max,Step):-
                Cur<=Max,
                cyclebody(Cur),
        Cur2=Cur+Step,
        Cur2<=Max,
        for(Cur2,Max,Step).
  cyclebody.   /*<--- Insert your cycle body here */
```

hCycle1.pro

```
/*
THIS PROGRAM FRAGMENT,
STARTED WITH THE GOAL  cycle(3),
EXECUTES THE CYCLE BODY 3 TIMES WITH X=3,2,1
*/

  predicates

cycle(integer)
repeat(integer)
body(integer)
next(integer)

  clauses

cycle(X):-repeat(X).

repeat(X):- X=0.
repeat(X):- body(X), next(X).

body(X):-write(X,' '). % PUT YOUR CYCLE BODY HERE...

next(X):-Y=X-1,repeat(Y).
```

iCycle2.pro

```
/*
ONE MORE PROGRAM FRAGMENT TO RUN A CYCLE
```

```
    Specify the goal: cycle(1,5)
    (works like    for (i=1;i<=5;i++)  (C)
    or             for i:=1 to 5 do    (Pas))

    */

    predicates
      cycle(integer,integer)
      repeat_up (integer,integer)
      body(integer)

    clauses
      cycle(Start,Finish):-repeat_up(Start,Finish).

      repeat_up(LastTry,Finish):- LastTry>Finish.
                    %( OR: LastTry=Finish+1 )

    repeat_up(I,Finish):- body(I),
                    Iplus1=I+1,
                    repeat_up (Iplus1,Finish).

    body(I):-write(I,' '). %  <----INSERT YOUR CYCLE BODY
HERE...
```

ДОДАТОК 4

Задача до Завдання 7. («Хрестики-нулики»)

Штірліц і Вінні-Пух сіли грати в хрестики-нулики. Наприкінці гри на полі склалася наступна ситуація:

```
krestik(1,1).
krestik(1,2).
krestik(2,2).
krestik(3,1).
krestik(3,3).
nolik(X,Y) if not ( krestik(X,Y) ).
```

Необхідно скласти програму, яка за за двома циклами – по X і по Y – обійде всі поля і, *використовуючи наведені факти про* krestik, роздрукує (за допомогою літер X і O) зображення ігрового поля у такому вигляді:

XXO
OXO
XOX

Якщо правило з *not* буде погано працювати в такій формі, придумайте інший спосіб його записати (наприклад, допомагає відсікання). Однак, досвід ваших попередників показує, що застосовувати відсікання не доводиться.

ДОДАТОК 5

Приклади програм для вивчення відсікання (до завдання 8)

Пояснення: секція **clauses** наводиться в двох варіантах, які розрізняються тим, що друкується на екрані. В одному випадку це цифри (1 2 3 4 5 6 7 8), в іншому – вказівки на конкретний предикат: a1, a2 і т.д. Оберіть самі те, що вам більше до вподоби.

Перша програма OTSECH1.PRO – "еталонна" : для початку дивимось, як відбувається перебір, якщо відсікання в програмі немає. В наступних програмах вже додамо відсікання і будемо порівнювати результати.

OTSECH1.PRO

```
predicates
a
b
c

m /*main predicate. Run the program with the goal: m*/

clauses
m:-a,b,c.
a:-write(1).
a:-write(2).
b:-write(3).
b:-write(4).
c:-write(5).
c:-write(6).
```

clauses (другий варіант)

```
m:-a,b,c.
a:-write("a1").
a:-write("a2").
b:-write("b3").
b:-write("b4").
c:-write("c5").
c:-write("c6").
```

OTSECH2.PRO

```
predicates
a
b
c
m /*main predicate. Run the program with the goal: m */

clauses
m:-a,b,!,c.
a:-write(1).
a:-write(2).
b:-write(3).
b:-write(4).
c:-write(5).
c:-write(6).
```

clauses (другий варіант)

```
m:-a,b,!,c.
a:-write("a1").
a:-write("a2").
b:-write("b3").
b:-write("b4").
c:-write("c5").
c:-write("c6").
```

OTSECH3.PRO

```
predicates
a
b
c

m /*main predicate*/


clauses
m:-a,b,c.
a:-write(1).
a:-write(2).
b:-write(3).
b:-write(4).
c:-write(5),fail.
c:-write(6),fail.
```

clauses (другий варіант)

```
m:-a,b,c.
a:-write("a1").
a:-write("a2").
b:-write("b3").
b:-write("b4").
c:-write("c5"),fail.
c:-write("c6"),fail.
```

OTSECH4.PRO

```
predicates
a
b
c

m /*main predicate*/


clauses
m:-a,b,!,c.
a:-write(1).
a:-write(2).
b:-write(3).
b:-write(4).
c:-write(5),fail.
c:-write(6),fail.
```

clauses (другий варіант)

```
m:-a,b,!,c.
a:-write("a1").
a:-write("a2").
b:-write("b3").
b:-write("b4").
c:-write("c5"),fail.
c:-write("c6"),fail.
```

OTSECH5.PRO

```
predicates
a
b
c
d
m /*main predicate*/


clauses
m:-a,b,!,c,d.
a:-write(1).
a:-write(2).
b:-write(3).
b:-write(4).
c:-write(5).
c:-write(6).
d:-write(7),fail.
d:-write(8),fail.
```

clauses (другий варіант)

```
m:-a,b,!,c,d.
a:-write("a1").
a:-write("a2").
b:-write("b3").
b:-write("b4").
c:-write("c5").
c:-write("c6").
d:-write("d7"),fail.
d:-write("d8"),fail.
```

OTSECH6.PRO

```
predicates
a
b
c
d
m /*main predicate*/


clauses
m:-a,b,!,c,d.
a:-write(1).
a:-write(2).
b:-write(3).
b:-write(4).
c:-write(5).
c:-write(6).
d:-write(7),fail.
d:-write(8),fail.
```

clauses (другий варіант)

```
m:-a,b,!,c,d.
a:-write("a1").
a:-write("a2").
b:-write("b3").
b:-write("b4").
c:-write("c5").
c:-write("c6").
d:-write("d7"),fail.
d:-write("d8"),fail.
```

OTSECH7.PRO

```
predicates
a
b
c
d
m /*main predicate*/

clauses
m:-a,b,!,c.
m:-d.
a:-write(1).
a:-write(2).
b:-write(3).
b:-write(4).
c:-write(5),fail.
c:-write(6),fail.
d:-write(7),fail.
d:-write(8),fail.
```

clauses (другий варіант)

```
m:-a,b,!,c.
m:-d.
a:-write("a1").
a:-write("a2").
b:-write("b3").
b:-write("b4").
c:-write("c5"),fail.
c:-write("c6"),fail.
d:-write("d7"),fail.
d:-write("d8"),fail.
```

OTSECH8.PRO

```
predicates
a
b
c
d
m /*main predicate*/


clauses
m:-a,b,c.
m:-d.
a:-write(1).
a:-write(2).
b:-write(3).
b:-write(4).
c:-write(5),fail.
c:-write(6),fail.
d:-write(7),fail.
d:-write(8),fail.
```

clauses (другий варіант)

```
m:-a,b,c.
m:-d.
a:-write("a1").
a:-write("a2").
b:-write("b3").
b:-write("b4").
c:-write("c5"),fail.
c:-write("c6"),fail.
d:-write("d7"),fail.
d:-write("d8"),fail.
```

OTSECH9.PRO

```
predicates
a
b
c
d
m /*main predicate*/

clauses
m:-a,b,!,c.
m:-d.
a:-write(1).
a:-write(2).
b:-write(3).
b:-write(4).
c:-write(5),fail.
c:-write(6),fail.
d:-write(7),fail.
d:-write(8),fail.
```

clauses (другий варіант)

```
m:-a,b,!,c.
m:-d.
a:-write("a1").
a:-write("a2").
b:-write("b3").
b:-write("b4").
c:-write("c5"),fail.
c:-write("c6"),fail.
d:-write("d7"),fail.
d:-write("d8"),fail.
```

OTSECH10.pro

```
predicates
a
b
c(symbol)
d(symbol)
m1(symbol)
m2(symbol)

/* Run this program with the goal: m1(X)
   and then         : m2(X)  */

clauses
m1(X):-a,b, c(X).
m1(X):-d(X).

m2(X):-a,b,!,c(X).
m2(X):-d(X).

a:-write(a1).
a:-write(a2).
b:-write(b1).
b:-write(b2).
c(X):-X="c".
c(X):-X="C".
d(X):-X="d".
d(X):-X="D".
```

clauses (другий варіант)

```
m1(X):-a, b, c(X).
m1(X):-d(X).

m2(X):-a,b,!,c(X).
m2(X):-d(X).
```

```
a:-write("a1").
a:-write("a2").
b:-write("b1").
b:-write("b2").
c(X):-X="c".
c(X):-X="C".
d(X):-X="d".
d(X):-X="D".
```

ДОДАТОК 6

Модельна предметна область для відпрацювання практичних прийомів обробки інформації

Дано перелік товарів, що продаються в магазині. Про кожен товар відома його назва і ціна.

```
domains
    nomer = integer
    nazva = symbol
    cina = integer

predicates
    tovar (nomer, nazva, cina)

clauses

   tovar (1, lampochka, 8).
   tovar (2, velosiped, 4000).
   tovar (3, tapochki, 80).
   tovar (4, kastrulya, 200).
   tovar (5, kedy, 300).
```

Напишіть предикати, що реалізують наступні запити:

1) Надрукуйте перелік товарів в порядку номерів у списку.

2) Надрукуйте перелік товарів у зворотньому порядку номерів у списку.

3) Визначте сумарну ціну всх товарів у списку.

4, а) Визначте найдорожчий товар.
4, б) Визначте найдешевший товар.

5, а) Опишіть предикат, який визначить кількість товарів, що задовольняють певній умові. Наприклад, мають ціну від 100 до 400.

5, б) Опишіть предикат, який визначить сумарну ціну товарів, що задовольняють тій же умові, що і у пункті завдання 5а.

6, а) Надрукуйте список товарів в порядку збільшення ціни.

6, б) Надрукуйте список товарів в порядку зменшення ціни.

7) Опишіть предикат, який отримає на вхід ціну C і визначить, скільки є в списку товарів, дешевших за введену ціну. (Відповіддю має бути число – кількість товарів. Наприклад, якщо ввести C=400, то відповідь має бути 4, бо в списку є чотири товари, дешевших за 400 грн.)

8) Опишіть предикат, який отримає на вхід число N і визначить N-й за ціною товар (в порядку збільшення ціни). Наприклад, якщо введено N=3, то має бути знайдена каструля, оскільки вона третя за ціною (200), після велосипеду (4000) і кедів (300).

Підказка: цю задачу досить просто вирішити, якщо скористатися результатами виконання попереднього пункту. Але тепер в якості C підставляйте не довільне число «зі стелі», а послідовно ціну кожного конкретного товару із списку. Наприклад, якщо Ви дійшли до розгляду каструлі і виявилося, що у списку знайшлося 2 товари, дешевших за неї, то каструля третя за ціною. Якщо було введено запит з N=3, то відповідь знайдена. Якщо ж ні, то пробуйте усі інші товари, поки не знайдете потрібний.