Programowanie
w logice

PROLOG

Predykaty obsługi wejścia/wyjścia

Czytanie i pisanie znaków:

get(X) – umożliwiają pobranie pojedynczych znaków z bieżącego urządzenia wejściowego

```
?-get(X).
|: a
X=97.
```

put(X) – powoduje wypisanie do bieżącego urządzenia wyjściowego znaku, którego reprezentację w kodzie ASCII stanowi zmienna X

```
?- put(104),put(101),put(108),put(108),put(111).
hello
```

Predykaty obsługi wejścia/wyjścia

Czytanie i pisanie termów:

write(X) – powoduje wypisanie termu (jeśli X jest ukonkretniona z prologowym termem) do bieżącego urządzenia wyjściowego (domyślnie monitor)

```
?-write('hallo').
  hallo
?-write("hallo").
  [104,97,108,108,111]
```

Predykaty obsługi wejścia/wyjścia

display(X) – równoważny predykatowi write z różnicą dotyczącą traktowania operatorów

```
?-display(a*b+c*d).
 +(*(a,b),*(c,d))

?-write(a*b+c*d).
 a*b+c*d
```

Predykaty obsługi wejścia/wyjścia

read(X)

w przypadku, gdy zmienna X jest nieukonkretniona, spowoduje ukonkretnienie tej zmiennej termem wczytanym z bieżącego urządzenia wejściowego

Predykaty obsługi wejścia/wyjścia

Czytanie i pisanie do plików:

tell(X) – ze zmienną X ukonkretnioną nazwą pliku kojarzy bieżące urządzenie wejściowe z plikiem o podanej nazwie, przygotowując go do operacji pisania (otwarcie pliku)

Jeśli X oznacza nazwę pliku istniejącego, to poprzednia jego zawartość zostanie usunięta.

W przypadki pliku nie istniejącego, zostanie on utworzony.

append(X) – otwarcie pliku do zapisu, bez usunięcia zawartości pliku (dopisanie)

told – zamknięcie pliku

Przykład

Wprowadzenie przez użytkownika elementów listy i zapisanie ich do pliku.

```
pisz_plik :-
    write('Podaj listę:'),
    read(L1),
    tell('plik.txt'), /*lub append*/
    write(L1),
    write(.),
    nl,
    told.
```

Predykaty obsługi wejścia/wyjścia

see(X) - ze zmienną X ukonkretnioną nazwą pliku kojarzy bieżące urządzenie wejściowe z plikiem o podanej nazwie, przygotowując go do operacji czytania (otwarcie pliku)

seen - zamknięcie pliku

Przykład

Odczytanie elementów listy liczbowej z pliku, obliczenie i wyświetlenie ich sumy

```
czytaj_plik :- write('Czytam z pliku...'),
        nl,
        see('przyk.txt'),
        read(L),
        seen,
        write('suma elementow listy z pliku wynosi:'),
       sumlist(L,Suma),
       write(Suma).
```

Predykaty obsługi wejścia/wyjścia

Predykaty dynamicznej zmiany pamięci:

asserta(X) – umożliwia dołączenie do bazy danych – **na początek** – klauzuli, którą jest ukonktretniona zmienna X

assertz(X) – umożliwia dołączenie do bazy danych – **na koniec** – klauzuli, którą jest ukonktretniona zmienna X

Predykaty obsługi wejścia/wyjścia

retract(X) – usunięcie z bazy danych pierwszej klauzuli dającej się uzgodnić z argumentem predykatu

np. asserta(student(adam,kowalski,s12345)).
retract(film(ziemia_obiecana,wajda)).

Predykaty obsługi wejścia/wyjścia

consult(X) - umożliwia rozszerzenie prologowej bazy danych o zbiór klauzul zawartych w określonym pliku lub wprowadzanych bezpośrednio z klawiatury.

Klauzule odczytywane z danego pliku są dołączane na koniec bazy danych.

Np. consult('dane.txt').

Sprawdzanie typu termów

Można stwierdzić, czy term jest niewiadomą (zmienną logiczną) o nieustalonej wartości, czy też ma określoną wartość, czyli jest stałą, lub zmienną o wcześniej ustalonej wartości (po pomyślnej unifikacji).

var(X) – sprawdza, czy zmienna X ma przypisaną wartość

var(X) nie zawodzi, gdy X jest zmienną nieukonkretnioną

nonvar(X) – przeciwieństwo predykatu var(X)

```
1 ?- var(X).
true.

2 ?- var(X),X=5.
X = 5.

3 ?- X=5,var(X).
false.

4 ?- nonvar(X).
false.

5 ?- nonvar(X),X=2.
false.

6 ?- X=2,nonvar(X).
X = 2.
```

Sprawdzanie typu wartości argumentów

Predykaty sprawdzające, czy term jest:

atom
atomem logicznym (stałą, napisem)

atomic
liczbą lub atomem

number
liczbą

compound
złożoną strukturą

integer
liczbą całkowitą

float
liczbą zmiennoprzecinkową

```
7 ?- atom(X).
false.

8 ?- atom(8).
false.

9 ?- atom(a).
true.

10 ?- atom(+).
true.

11 ?- atom(:-).
true.

12 ?- atom(:=).
true.

13 ?- atom('prolog').
true.

14 ?- atomic(X).
false.

15 ?- atomic(5).
true.

16 ?- atomic(+).
true.
```

```
18 ?- number(prolog).
false.

19 ?- number(2016).
true.

20 ?- compound(prolog).
false.

21 ?- compound(5).
false.

22 ?- compound(programowanie(prolog, haskell)).
true.

24 ?- integer(3).
true.

25 ?- integer(3.14).
false.

26 ?- float(3.14).
true.

27 ?- float(3).
false.

28 ?- float(3.0).
true.
```

Konstruowanie i dekompozycja termów

=..

Operator pozwala na dynamiczną zamianę termu na listę i odwrotnie

X=..L

L jest listą składającą się z funktora struktury reprezentowanej przez X oraz następującego po nim zbioru argumentów

```
?-X=..[a,b,c].      ?-X=..[suma,2,5,4,3].
X=a(b,c)            X=suma(2,5,4,3)

?- student(jan,kowalski,wmii,poznan)=..L.
L = [student, jan, kowalski, wmii, poznan]
```

Konstruowanie i dekompozycja termów

name(A,L) – rozkłada wyrażenie atomowe na zbiór znaków ujętych w postaci listy

```
?-name(abcd,L).
  L=[97,98,99,100]

?-name(A, [97,98,99,100]).
  A=abcd
```

Konstruowanie i dekompozycja termów

functor(S,F,N) – umożliwia dostęp do struktury, ustala liczbę argumentów struktury

(predykat jest prawdziwy, jeżeli F pokrywa się z nazwą struktury S o arności N)

S – struktura

F – funktor

N – liczba argumentów

```
?- functor(a(5,2,8),a,3).
   true
?- functor(plus(1,5),F,N).
   F=plus
   N=2
```

Konstruowanie i dekompozycja termów

arg(N,S,A) - umożliwia dostęp do wybranych argumentów struktury

(predykat jest prawdziwy, jeżeli A jest N-tym argumentem struktury)

Dwa pierwsze argumenty arg muszą być ukonkretnione

N – numer argumentu struktury

S – struktura

```
?-arg(4,litery(a,d,r,y,w,q,z,i),A).
  A=y
?-arg(2,[a,b,c,d,e,f],A).
  A=[b,c,d,e,f]
```

Definiowanie operatorów

:- op(P, T, N)

definiuje N, jako operator typu T, o priorytecie P

Każdy operator może występować w jednej lub kilku wersjach, które nazywać będziemy -fixowością

Wyróżniamy operatory:

- **Infixowe** - takie, które występują pomiędzy operandami, np. operator + traktowany jako operator dodawania dwóch liczb;
- **Prefixowe** - takie, które występują przed operandem, np. operator + traktowany jako operator określający znak liczby;
- **Postfixowe** - takie, które występują za operandem, np. operator ! oznaczający silnie

Definiowanie operatorów

Wzorzec	Łączność		Przykłady
fx	prefix	non-associative	-
fy	prefix	łączny (prawostronny)	
xf	postfix	non-associative	
yf	postfix	łączny (lewostronny)	
xfx	infix	non-associative	=, is
xfy	infix	prawostronnie łączny	,
yfy	infix	**nie ma sensu**	
yfx	infix	lewostronnie łączny	+, *

Wzorce określające łączność operatorów w Prologu.

Definiowanie operatorów

Zdefiniowane w standardzie ISO operatory (przykład):

```
700   xfx   <, =, ==.., =:=, ==, =<, <=
500   yfx   +, -
400   yfx   *, /, mod
200   xfx   ^
```

Przykład definicji spójników logicznych:

```
:-op(140, fy, neg).
:-op(160, xfy, [and, or, imp, uparrow,downarrow]).
```

Definiowanie operatorów

Zawartość pliku .pl

```
:- op(100,xfy,matka).
:-op(300, xfx, ma).
:-op(200, xfy, i).
ewa matka jan.
jan ma kota i psa.
ewa ma jana i kota i dosc_prologu.
```

```
39 ?- X matka Y.
X = ewa,
Y = jan.

40 ?- ma(X,Y).
X = jan,
Y = kota i psa ;
X = ewa,
Y = jana i kota i dosc_prologu.

41 ?- display(jan ma kota i psa).
ma(jan,i(kota,psa))
true.

42 ?- display(ewa ma jana i kota i dosc_prologu).
ma(ewa,i(jana,i(kota,dosc_prologu)))
true.

43 ?- display(ewa ma jana i kota i psa).
ma(ewa,i(jana,i(kota,psa)))
true.
```

Literatura

- W. Clocksin, C. Mellish, „Prolog. Programowanie”
- E.Gatnar, K.Stąpor, „Prolog”
- G.Brzykcy, A.Meissner, „Programowanie w Prologu i programowanie funkcyjne”
- M. Ben-Ari, “Logika matematyczna w informatyce”