Programowanie w logice

PROLOG cz.3

Operacje na listach

Predykat sprawdzający, czy podana lista stanowi początek innej listy:

```
poczatek([],X).
poczatek([H1|T1], [H2|T2]) :- H1 = H2, poczatek(T1,T2).
-----------------------------------------------------------------
poczatek([],X).
poczatek([H1|T1],[H1|T2]):-poczatek(T1,T2).
```

Operacje na listach

Procedura znajdująca ostatni element listy:

```
ostatni([H], H).
ostatni([_|T], O) :- ostatni(T, O).
```

Predykat wbudowany: **last**

Operacje na listach

Procedura znajdująca największy element w liście:

```
max([X], X).
max([H|T], X) :- max(T, X1), X1 > H, X is X1.
max([H|T], X) :- max(T, X1), X1 < H, X is H.
-----------------------------------------------------------------
max([], 'Brak elementów w liście').
max(L, W) :- sort(L, L1), last(L1, W).
```

Operacje na listach

Procedura sprawdzająca, czy kolejne elementy listy tworzą ciąg ściśle rosnący:

```
rosnacy([_]).
rosnacy([G|[H|T]]) :- G < H, rosnacy([H|T]).
```

Operacje na listach

Procedura znajdująca n-ty element w liście:

```
nty(X,[X|_],1).
nty(X,[_|T],N) :- nty(X,T,N1), N is N1+1.
```

Predykat wbudowany: **nth1**

Predykat odcięcia „cut” („!”)

- „Cut” – bezargumentowy predykat jest interpretowany logicznie jako zawsze prawdziwy i służy do ograniczania nawrotów.
- Realizacja tego predykatu, występującego jako jeden z podcelów w ciele klauzuli, uniemożliwia nawrót do któregokolwiek z poprzedzających go podcelów przy próbie znajdowania rozwiązań alternatywnych.

Cut

- Wszystkie zmienne, które zostały ukonkretnione podczas realizacji poprzedzających odcięcie podcelów w ciele klauzuli, zachowują nadane im wartości w trakcie realizacji występujących po predykacie odcięcia warunków.
- Odcięcie nie ma wpływu na nieukonkretnione zmienne występujące w następujących po nim podcelach.

Wpływ na nawracanie

repeat – generowanie wielu rozwiązań danego problemu poprzez „wymuszanie” nawrotów

```
a(1).
a(2).
a(3).
a(4).
?-repeat, a(X),write(X), X==3,!.
123
X=3
```

Predykat „fail”

- „fail” powoduje niepowodzenie wykonywania klauzuli. Wykonanie tego predykatu zawsze zawodzi. Najczęściej używany w celu wymuszenia nawrotów.
- Użyty w kombinacji z „cut” (!,fail) zapobiega użyciu innej klauzuli przy próbie znalezienia rozwiązań alternatywnych, co oznacza niepowodzenie wykonywania całej procedury.

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

read, write

```
 17 ?- read(X), read(Y),write('wczytane liczby to '),write(X),write(' i '),write(Y).
|: 345.
|: 456.
wczytane liczby to 345 i 456
X = 345,
Y = 456.

 18 ?- X=2,Y=3,Z is X+Y,write(X),write('+'),write(Y),write('='),write(Z).
2+3=5
X = 2,
Y = 3,
Z = 5.
```

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 ozn. nazwę pliku istniejącego, to poprzednia jego zawartość zostanie usunięta.

W przypadku pliku nie istniejącego, zostanie on utworzony.

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

told – zamknięcie pliku

Wprowadzenie przez użytkownika elementów listy i zapisanie ich do pliku.

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

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)).

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').

Zastosowania matematyczne Prologu

Przynależność do zbioru (member)

```
nalezy_do_listy(X,[X|_]).
nalezy_do_listy(X,[_|Y]):-
                nalezy_do_listy(X,Y).
```

Zawieranie się zbiorów (subset)

```
podzbior([],Y).
podzbior([A|X],Y):-nalezy_do_listy(A,Y),
                podzbior(X,Y).
```

Zastosowania matematyczne Prologu

Część wspólna zbiorów (intersect)

```
czesc_wspolna([],X,[]).
czesc_wspolna([X|R],Y,[X|Z]):-
        nalezy_do_listy(X,Y),!,
        czesc_wspolna(R,Y,Z).
czesc_wspolna([X|R],Y,Z):-
        czesc_wspolna(R,Y,Z).
```

Zastosowania matematyczne Prologu

Suma zbiorów (union)

```
suma_zb([],X,X).
suma_zb([X|R],Y,Z):-nalezy_do_listy(X,Y),!,
                suma_zb(R,Y,Z).
suma_zb([X|R],Y,[X|Z]):-suma_zb(R,Y,Z).
```

Zastosowania matematyczne Prologu

Różnica zbiorów (difference)

```
roznica([],_,[]).
roznica([X|L],Set,[X|Z]):-
                not(member(X,Set)),!,
                roznica(L,Set,Z).
roznica([_|L],Set,Z):-roznica(L,Set,Z).
```

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”