Programowanie deklaratywne

Artur Michalski
Informatyka II rok

Plan wykładu

- Wprowadzenie do języka Prolog
- Budowa składniowa i interpretacja programów prologowych
- Listy, operatory i operacje arytmetyczne
- Sterowanie mechanizmem nawrotów
- *Predefiniowane procedury prologowe*
- Styl i technika programowania w Prologu

Predefiniowane predykaty prologowe

- Sprawdzanie typu termów
- Kompozycja i dekompozycja termów: =.., *functor*, *arg*, *name*
- Różne rodzaje operacji równości w Prologu
- Manipulacja bazą danych w Prologu
- Manipulowanie przepływem sterowania w Prologu
- Predykaty: *bagof*, *setof* i *findall*

Sprawdzanie typu termów

Prolog umożliwia manipulowanie termami (stałymi, zmiennymi, liczbami, atomami) za pomocą specjalnych procedur systemowych:

- predykat `var(X)` jest spełniony, jeżeli `X` zmienną wolną (nie związaną!)
- predykat `nonvar(X)` jest spełniony, jeżeli `X` termem innym niż zmienna lub zmienną związaną
- predykat `atom(X)` jest spełniony, jeżeli `X` jest stałą lub zmienną (związaną!) atomową

Sprawdzanie typu termów

Prolog umożliwia manipulowanie termami (stałymi, zmiennymi, liczbami, atomami) za pomocą specjalnych procedur systemowych:

- predykat `integer(X)` jest prawdziwy, jeżeli `X` jest stałą lub zmienną (związaną!) całkowitoliczbową
- predykat `real(X)` jest prawdziwy, jeżeli `X` jest stałą lub zmienną (związaną!) zmiennoprzecinkową (rzeczywistą)
- predykat `atomic(X)` jest prawdziwy, jeżeli `X` jest stałą lub zmienną (związaną!) liczbową lub atomową

Sprawdzanie typu termów

Przykład zastosowania predykatów typu:

```
?- var(Z),Z=2.
Z=2
?- Z=2,var(Z).
No
?- integer(X),X=2.
No
?- Y=2,integer(Y),nonvar(Y).
Y=2
?- atom(22).
No
?- atomic(22).
Yes
?- atom([]).
Yes
?- atomic(p(1)).
No
```

Sprawdzanie typu termów

Przykład zastosowanie predykatu **atom**:

Treść zadania

Predykat **count(A,L,N)** ma podawać liczbę wystąpień (**N**) atomu (!) **A** na liście obiektów **L**.

Definicja pierwsza

```
count(_,[],0).
count(A,[A|L],N):- !,
                   count(A,L,N1),N is N1+1.
count(A,[_|L],N):- count(A,L,N).
```

Sprawdzanie typu termów

Przykład zastosowanie predykatu **atom** c.d.:

Przykłady użycia

```
?- count(a,[a,b,a,a],N).
N=3
?- count(a,[a,b,X,Y],Na).
Na=3
...
?- count(b,[a,b,X,Y],Nb).
Nb=3
...
?- L=[a,b,X,Y],count(a,L,Na),count(b,L,Nb).
Na=3
Nb=1
X=a
Y=a
```

Zgodnie z definicją **count(A,L,N)** sprawdzane są nie faktyczne wystąpienia szukanego atomu, lecz możliwe dopasowania termu!!!

Sprawdzanie typu termów

Przykład zastosowanie predykatu **atom** c.d.:

Definicja poprawna

```
count(_,[],0).
count(A,[B|L],N):- atom(B), A=B,!,
                   count(A,L,N1),N is N1+1.
count(A,[_|L],N):- count(A,L,N).
```

Przykład użycia

```
?- L=[a,b,X,Y],count(a,L,Na),count(b,L,Nb).
L=[a,b,_G378,_G381]
Na=1
Nb=1
X=_G378
Y=_G381
```

Sprawdzanie typu termów

Przykład zastosowanie predykatu **nonvar**:

Treść zadania

Napisać program rozwiązujący arytmograf sformułowany za pomocą równań postaci:

```
  D O N A L D
+ G E R A L D
-----------------
  R O B E R T
```

przy czym każdej literze musi zostać przypisana inna cyfra.

Sprawdzanie typu termów

Przykład zastosowanie predykatu **`nonvar`**:

Sformułowanie w Prologu

Zdefiniować predykat **`sum(N1,N2,N)`**, gdzie **`N1`**, **`N2`**, **`N`** reprezentują odpowiednio pierwszą, drugą i trzecią liczbę łamigłówki, zaś cel **`sum`** jest spełniony, jeżeli istnieje takie przypisanie cyfr do liczb, że **`N1+N2=:=N`**.

Reprezentacja danych: listy zmiennych, którym zostaną przypisane odpowiednie cyfry.

Cel główny

```
?- sum([D,O,N,A,L,D], [G,E,R,A,L,D],
[R,O,B,E,R,T]).
```

Sprawdzanie typu termów

Przykład zastosowanie predykatu **`nonvar`**:

Rozwiązanie

Rozwiązanie zadania wymaga realizacji algorytmu dodawania dziesiętnego i przechowywania następujących informacji:

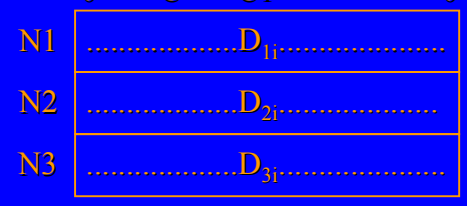
- cyfry przeniesienia dziesiętnego przed sumowaniem
- cyfry przeniesienia dziesiętnego po sumowaniu
- zbioru cyfr dostępnych przed sumowaniem
- zbioru cyfr nie wykorzystanych w sumowaniu

Sprawdzanie typu termów

Mechanizmy dodawania dziesiętnego

$N1 = [D_{11},D_{12},D_{13},\ldots,D_{1i},\ldots]$
$N2 = [D_{21},D_{22},D_{23},\ldots,D_{2i},\ldots]$
$N3 = [D_{31},D_{32},D_{33},\ldots,D_{3i},\ldots]$

Tu przeniesienie musi wynosić 0 — Przeniesienie z prawej: C, C1 — Tu przeniesienie wynosi 0



$D_{3i} = (C1 + D_{1i} + D_{2i})$ **mod** 10 $\qquad C = (C1 + D_{1i} + D_{2i})$ **div** 10

Sprawdzanie typu termów

Rozwiązanie c.d.

Predykat `sum1(N1,N2,N,C1,C,Ds1,Ds)` realizuje dodawanie dziesiętne z cyfrą przeniesienia z prawej (`C1`) i cyfrą przeniesienia na lewo (`C`) oraz zbiorami cyfr dostępnych (`Ds1`) i nie wykorzystanych (`Ds`).

Przykładowe wywołanie predykatu `sum1`:

```
?- sum1([H,E],[6,E],[U,S],1,1,
        [1,3,4,7,8,9],Ds).
H=8
E=3
U=4
S=7
Ds=[1,9]
```

Sprawdzanie typu termów

Rozwiązanie c.d.

Związek predykatu `sum(N1,N2,N)` z predykatem `sum1(N1,N2,N,C1,C,Ds1,Ds)` wynika w warunków początkowych zadania:

```
sum(N1,N2,N):-
sum1(N1,N2,N,0,0,[0,1,2,3,4,5,6,7,8,9],Ds).
```

Dla uproszczenia rozważań założymy, że dodawane liczby są jednakowej długości (nie zmniejsza, to ogólności rozważań, bo „brakujące" cyfry zawsze możemy zastąpić z lewej strony zerami).

Sprawdzanie typu termów

Rozwiązanie c.d.

Definicja predykatu `sum1(N1,N2,N,C1,C,Ds1,Ds)` obejmuje następujące możliwe przypadki:

- wszystkie trzy liczby są reprezentowane przez puste listy:
  `sum1([],[],[],0,0,Ds,Ds)`
- wszystkie trzy liczby składają się z najbardziej znaczącej cyfry i pozostałych cyfr, czyli mają postać:
  `[D1|N1] [D2|N2] [D|N]`, wtedy:
  - pozostałe cyfry również muszą spełniać relację `sum1`, dając pewne przeniesienie na lewo `C2` oraz pozostawiając pewien podzbiór nie wykorzystanych cyfr `Ds2`,
  - najbardziej znaczące cyfry `D1`, `D2` muszą razem z cyfrą przeniesienia `C2` sumować się w `D` oraz w dalsze przeniesienie na lewo (zależność ta zostanie wyrażona za pomocą predykatu `digitsum`)

Sprawdzanie typu termów

Rozwiązanie c.d.

Zapis predykatu **sum1** w Prologu:

```
sum1([],[],[],0,0,Ds,Ds).
sum1([D1|N1],[D2|N2],[D|N],C1,C,Ds1,Ds):-
 sum1(N1,N2,N,C1,C2,Ds1,Ds2),
 digitsum(D1,D2,C2,D,C,Ds2,Ds).
```

Sprawdzanie typu termów

Rozwiązanie c.d.

Predykat **del** usuwa wykorzystane cyfry ze zbioru dostępnych cyfr:

- jeżeli zmienna reprezentująca literę nie jest związana, to może zostać zainicjowana dowolną wartością ze zbioru dostępnych cyfr
- jeżeli zmienna jest już związana, to wszystkie wartości w zbiorze dostępnych cyfr zostaną zachowane

```
del(Var,L,L):- nonvar(Var),!.
del(Var,[Var|L],L).
del(Var,[X|L1],[X|L2]):- del(Var,L1,L2).
```

Predykat **del** ma charakter niedeterministyczny - usunięty może być każdy element, o ile zmienna nie jest związana.

Sprawdzanie typu termów

Rozwiązanie c.d.

Predykat **digitsum** weryfikuje zależności arytmetyczne wynikające z dodawania dwóch cyfr i przeniesienia dziesiętnego oraz generuje wynik w postaci najmniej znaczącej cyfry sumy i dalszego przeniesienia:

```
digitsum(D1,D2,C1,D,C,Ds1,Ds):-
 del(D1,Ds1,Ds2),
 del(D2,Ds2,Ds3),
 del(D,Ds3,Ds),
 S is D1+D2+C1,
 D is S mod 10,
 C is S // 10.
```

Sprawdzanie typu termów

Rozwiązanie - kompletny program:

```
sum(N1,N2,N):-
  sum1(N1,N2,N,0,0,[0,1,2,3,4,5,6,7,8,9],Ds).
sum1([],[],[],0,0,Ds,Ds).
sum1([D1|N1],[D2|N2],[D|N],C1,C,Ds1,Ds):-
 sum1(N1,N2,N,C1,C2,Ds1,Ds2),
 digitsum(D1,D2,C2,D,C,Ds2,Ds).
digitsum(D1,D2,C1,D,C,Ds1,Ds):-
 del(D1,Ds1,Ds2), del(D2,Ds2,Ds3),
 del(D,Ds3,Ds), S is D1+D2+C1,
 D is S mod 10, C is S // 10.
del(Var,L,L):- nonvar(Var),!.
del(Var,[Var|L],L).
del(Var,[X|L1],[X|L2]):- del(Var,L1,L2).
```

Sprawdzanie typu termów

Rozwiązanie - przykładowe zadania:

```
?- sum( [D,O,N,A,L,D],
        [G,E,R,A,L,D],
        [R,O,B,E,R,T]).
D=5
O=2
N=6
A=4
L=8
G=1
E=9
R=7
B=3
T=0
```

```
?- sum( [O,S,E,N,D],
        [O,M,O,R,E],
        [M,O,N,E,Y]).
S=7
E=5
N=3
D=1
M=0
O=8
R=2
Y=6
```

=.., *functor,arg,name*

W języku Prolog mamy do dyspozycji predykaty systemowe przeznaczone do konstruowania i dekomponowania termów.

Predykat `=..` (*ang.* univ) służy do konstruowania termu z listy atomów. Cel `Term =.. L` jest spełniony, jeżeli lista `L` zawiera nazwę funktora termu `Term` i wszystkie jego kolejne argumenty.

Przykład

```
?- f(a,b)=..L.
L=[f,a,b]
?- T=..[rectangle,3,5].
T=rectangle(3,5).
?- Z =..[p,X,f(X,Y)].
Z=p(X,f(X,Y)).
```

=.., *functor,arg,name*

Zastosowanie predykatu `=..`:

Przykład 1

Manipulowanie termami opisującymi różne klasy obiektów w celu wykonania na nich jednej (tej samej) operacji, ale mającej inny przebieg dla każdej klasy. Przykładowo, operacja powiększania termów, reprezentujących różne figury geometryczne.

Klasy obiektów: `square(A)`, `rectangle(A,B)`, `circle(R)`, `...`

Predykat `enlarge(Fig,Factor,Fig1)` powiększa figurę `Fig` w `Fig1` o `Factor` razy.

=.., *functor,arg,name*

Zastosowanie predykatu `=..`:

Definicja pierwsza predykatu `enlarge`:

```
enlarge(square(A),F,square(A1)):- A1 is F*A.
enlarge(rectangle(A,B),F,rectangle(A1,B1)):-
  A1 is F*A, B1 is F*B.
enlarge(circle(R),F,circle(R1)):- R1 is F*R.
...
```

Zaproponowane rozwiązanie choć poprawne, ma jedną poważną wadę - musimy z góry uwzględnić wszystkie możliwe figury geometryczne. Jeśli jakaś figura nie zostanie wzięta pod uwagę, konieczne będzie rozszerzenie definicji predykatu `enlarge` o nowy przypadek.

=.., *functor,arg,name*

Zastosowanie predykatu `=..`:

Rozwiązanie poprawne - uogólniona definicja predykatu `enlarge`:

```
enlarge(Fig,F,Fig1):-
 Fig=..[Type|Params],      % dekompozycja
 multiplylist(Params,F,Params1),
 Fig1=..[Type|Params1].    % konstrukcja

multiplylist([],_,[]).
multiplylist([X|L],F,[X1|L1]):-
 X1 is X*F, multiplylist(L,F,L1).
```

=.., *functor,arg,name*

Zastosowanie predykatu `=..`:

Przykład 2

Podstawianie za term występujący w wyrażeniu innego termu: predykat `subst(SubT,T,SubT1,T1)` jest spełniony, jeżeli wszystkie wystąpienia termu `SubT` w termie `T` zostaną zastąpione przez term `SubT1` i otrzymamy w efekcie term `T1`.

Przykład użycia

```
?- subst(sin(x),2*sin(x)*f(sin(x)),t,F).
F=2*t*f(t)
?- subst(a+b,f(a,A+B),v,T).
T=f(a,v)
A=a
B=b
```

Sprawdzanie wystąpienia termu odbywa się przez jego unifikację !!!

=.., *functor,arg,name*

Zastosowanie predykatu `=..`:

Definicja predykatu `subst(SubT,T,SubT1,T1)`:

Jeżeli `SubT=T` *to wtedy* `SubT1=T1`
w przeciwnym przypadku jeżeli `T` *jest termem prostym, to*
`T1=T` *(podstawienia jest bowiem niemożliwe)*
w przeciwnym przypadku podstawienia są prowadzone na argumentach `T`.

=.., *functor,arg,name*

Zastosowanie predykatu `=..`:

Prologowa definicja predykatu `subst(SubT,T,SubT1,T1)`:

```
subst(Term,Term,Term1,Term1):- !.
subst(_,Term,_,Term):- atomic(Term),!.
subst(SubT,T,SubT1,T1):-
 T=..[F|Args],          % dekompozycja
 substlist(SubT,Args,SubT1,Args1),
 T1=..[F|Args1].        % konstrukcja

substlist(_,[],_,[]).
substlist(Sub,[T|Ts],Sub1,[T1|Ts1]):-
 subst(Sub,T,Sub1,T1), % pośrednia rekursja
 substlist(Sub,Ts,Sub1,Ts1).
```

=.., functor,arg,name

Pozostałe predykaty systemowe przeznaczone do konstruowania i dekomponowania termów.

Predykat **`functor(Term,F,N)`** jest spełniony jeżeli **`F`** jest głównym funktorem termu **`Term`**, którego arność wynosi **`N`**.

Predykat **`arg(N,Term,A)`** jest spełniony, jeżeli **`A`** jest **`N`** -tym argumentem termu **`Term`**, przy założeniu, że numerowanie zaczyna się od 1.

Przykład zastosowania

```
?- functor(t(f(X),X,t),Func,Arity).
Func=t
Arity=3
?- arg(2,f(X,t(a),t(b)),Y).
Y=t(a)
```

=.., functor,arg,name

Przykład zastosowania

```
?- functor(D,date,3),
   arg(1,D,29),
   arg(2,D,june),
   arg(3,D,1982).
D=date(29,june,1982)
```

Predykat **`functor`** zastosowany powyżej generuje „uogólniony term", rozpoczynający się funktorem **`date`** o arności 3, którego składowe początkowo są nieokreślone (niezwiązane). Zostają one ustalone dopiero na drodze spełniania następnych celów wskutek wywołania predykatu **`arg`**.

=.., *functor,arg,name*

Predykat systemowy `name(A,S)` służy do przekształcania wczytanej informacji reprezentowanej w postaci listy kodów znaków (`S`) w stałą symboliczną lub liczbę (`A`). Konwersji można dokonywać w jedną albo drugą stronę.

Przykłady

```
?- name(kot,X).
X=[107,111,116]
?- name(Y,[112,105,101,115])
Y=pies
```

Alternatywna reprezentacja list kodów znaków
|w Prologu - napis ograniczony znakami cudzysłowu

```
?- [112,105,101,115]="pies".
Yes
?- name(Y,"pies").
Y=pies
```

=.., *functor,arg,name*

Przykład zastosowania

Przetwarzanie identyfikatorów numerowanych np. postaci:

`item1,item2,...,itemk,...`

Predykat `isitem(X)` ma służyć do sprawdzania, czy mamy do czynienia z właściwym identyfikatorem:

```
isitem(X):- name(X,Xlist),
            name(item,T),
            conc(T,_,Xlist).

conc([],L,L). % przypomnienie definicji
conc([H|T],L,[H|T2]):- conc(T,L,T2).
```

Manipulacja bazą danych w Prologu

Program prologowy traktowany jako baza danych, to:
- klauzule bezwarunkowe - fakty reprezentujące jawne relacje
- klauzule warunkowe - fakty reprezentujące niejawne relacje

Predykaty systemowe umożliwiające manipulowanie bazą klauzul:
- `assert(C)` - zawsze spełniony cel, dodający klauzulę `C`
- `asserta(C)` - -//-, dodający klauzulę `C` na początku bazy
- `assertz(C)` - -//-, dodający klauzulę `C` na końcu bazy
- `retract(C)` - zawsze spełniony cel, usuwający klauzulę `C`

Dodawane klauzule funkcjonują dokładnie tak samo jak klauzule zawarte w pierwotnym programie. Zastosowanie powyższych predykatów umożliwia adaptowanie programu do zmieniających się warunków działania.

Manipulacja bazą danych w Prologu

Przykłady manipulacji bazą klauzul

Dany jest zbiór klauzul:

```
nice:- sunshine,
       not(raining).
funny:- sunshine,
        raining.
distgusting:- raining,
              fog.

raining.
fog.
```

Dialog:

```
?- nice.
No
?- disgusting.
Yes
?- retract(fog).
Yes
?- disgusting.
No
?- assert(sunshine).
Yes
?- funny.
Yes
?- retract(raining).
Yes
?- nice.
Yes
```

Manipulacja bazą danych w Prologu

Przykłady manipulacji bazą klauzul

Operacja **retract** jest realizowana w sposób niedeterministyczny - można usunąć cały zbiór klauzul dzięki mechanizmowi nawrotów.

Początkowy zbiór klauzul

```
fast(ann).
slow(tom).
slow(pat).
?- assert((faster(X,Y):-
     fast(X), slow(Y))).
Yes
```

Dodawane klauzule warunkowe umieszczamy w nawiasach!

Dialog

```
?- faster(A,B).
A=ann
B=tom
?- retract(slow(X)).
X=tom;
X=pat;
No
?- faster(ann,_).
No
```

niedeterminizm!

Manipulacja bazą danych w Prologu

Przykłady manipulacji bazą klauzul

Operacje **asserta** i **assertz** pozwalają wskazywać miejsce, w którym zostanie dodana nowa klauzula, co ma znaczenie w przypadku nawrotów dokonywanych dla nowej klauzuli.

Przykład

```
?- assert(p(a)),assertz(p(b)),asserta(p(c)).
Yes
?- p(X).
X=c;
X=a;
X=b
```

Manipulacja bazą danych w Prologu

Przykłady manipulacji bazą klauzul

Operacje **asserta**, **asserta** lub **assertz** mogą zostać wykorzystane do przechowywania wyników wcześniejszych obliczeń lub generowania faktów na potrzeby przyszłych zadań.

Przykład

Generowanie tabliczki mnożenia w postaci faktów

```
maketable:-
 L=[0,1,2,3,4,5,6,7,8,9],
 member(X,L),member(Y,L),
 Z is X*Y,
 assert(product(X,Y,Z)),
 fail;true.
```

Manipulacja bazą danych w Prologu

Uwaga

Predykaty dodawania i usuwania klauzul są bardzo użytecznym narzędziem programistycznym, lecz ich zastosowania wymaga dużej ostrożności. Są to bowiem mechanizmy, które dokonują modyfikacji programu w trakcie jego działania, czyli samomodyfikacji, i jako takie mogą zmieniać jego funkcjonowanie z upływem czasu. Utrudnia to zarówno zrozumienie programu, jego ewentualne poprawki, jak i ogranicza nasze przekonanie co do jego prawidłowego działania.

Manipulowanie przepływem sterowania w Prologu

W języku Prolog mamy do dyspozycji następujące mechanizmy systemowe przeznaczone do modyfikacji sterowania:

- odcięcie (`!`) – cel, eliminujący nawroty
- `fail` – cel, który zawsze jest niespełniony
- `true` – cel, który zawsze jest spełniony
- `not(P)` – negacja (przez niepowodzenie!) celu `P`
- `call(P)` – cel spełniony, gdy wywołany cel `P` jest spełniony
- `repeat` – cel zawsze spełniony; niedeterministyczny - prowadzi poprzez nawroty do poszukiwania alternatywnych rozwiązań ze względu na definicję:

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

Manipulowanie przepływem sterowania w Prologu

Zastosowanie predykatu `repeat`

Przykład

Generowanie kwadratów liczb wczytywanych z terminala, aż do zakończenia sygnalizowanego atomem `stop`.

```
makesqr:-  repeat,read(X),proc(X).
proc(stop):- !.
proc(X):-  Y is X*X,
           write(Y), nl,
           fail.
```

odczyt z klawiatury

zapis na ekran

Predykaty: *bagof*, *setof* i *findall*

Mechanizm nawrotów stosowany w Prologu umożliwia sprawdzenie wszystkich obiektów lub relacji, które spełniają zadany cel . Po dokonaniu nawrotu nie jest jednak możliwe odwołanie się do wcześniej wygenerowanych rozwiązań (tych przed nawrotem). Efekt taki zapewnia użycie predykatów:

- `bagof(X,P,L)` - generuje listę `L` wszystkich obiektów `X` takich, że cel `P` jest spełniony; ma sens kiedy `P` i `X` mają wspólne zmienne
- `setof(X,P,L)` - podobnie jak `bagof` tyle, że lista `L` zostanie uporządkowana i pozbawiona powtórzeń elementów;
- `findall(X,P,L)` - podobnie jak `bagof` tyle, że generowane są wszystkie obiekty niezależnie od wartości tych zmiennych w `P`, które nie występują w `X`

Predykaty: *bagof*, *setof* i *findall*

Zastosowanie predykatu `bagof`

Przykład

Dane są fakty:

```
age(peter,7).
age(ann,5).
age(pat,8).
age(tom,5).
```

Wszystkie dzieci w wieku 5 lat:

```
?- bagof(Ch,age(Ch,5),L).
L=[ann,tom]
```

Wszystkie dzieci w dowolnym wieku:

```
?- bagof(Ch,age(Ch,Age),L).
Age=7
L=[peter];
Age=5
L=[ann,tom];
Age=8
L=[pat]
```

Agregacja wyników dotyczy tylko imion dzieci a nie ich wieku

Predykaty: *bagof*, *setof* i *findall*

Wynik predykat **bagof** może zawierać powtórzenia, jeżeli znaleziony obiekt wielokrotnie spełniał podany cel.
Predykat **setof** eliminuje powtórzenia i porządkuje obiekty alfabetycznie kiedy są atomami, a rosnąco gdy są liczbami.
Jeśli obiekty są termami, to porządkowanie alfabetyczne odnosi się do funktorów tych termów, a gdy funktory są takie same i termy złożone, to dotyczy skrajnie lewych, najmniej zagnieżdżonych w nich, rożnych od siebie funktorów.

Struktura generowanych obiektów jest dowolna!!!

Przykład

Wszystkie dzieci posortowane według wieku a potem wg imion:

```
?- setof(Age/Child,age(Child,Age),L).
L=[5/ann,5/tom,7/peter,8/pat]
```

Predykaty: *bagof*, *setof* i *findall*

Predykat **findall(X,P,L)** umożliwia uzyskanie na liście **L** obiektów **X** z wszystkich rozwiązań celu **P** niezależnie od wartości pozostałych zmiennych w **P**, które nie należą do **X**. Jeżeli nie istnieje żaden taki obiekt **X**, który spełniałby **P**, to w wyniku zwracana jest lista pusta!

Przykład

```
?- bagof(Ch,age(Ch,Age),L).
Age=7
L=[peter];
Age=5
L=[ann,tom];
Age=8
L=[pat]
?- findall(Ch,age(Ch,Age),L).
L=[peter,ann,pat,tom]
```