

# PROLOG. Advanced Issues. Meta-Programming in PROLOG

Antoni Ligęza

Katedra Automatyki, AGH w Krakowie

2011

- [1] Ulf Nilsson, Jan Maluszyński: **Logic, Programming and Prolog**, John Wiley & Sons Ltd., pdf, <http://www.ida.liu.se/~ulfni/lpp>
- [2] Dennis Merritt: **Adventure in Prolog**, Amzi, 2004  
<http://www.amzi.com/AdventureInProlog>
- [3] Quick Prolog:  
<http://www.dai.ed.ac.uk/groups/ssp/bookpages/quickprolog/quickprolog.html>
- [4] W. F. Clocksin, C. S. Mellish: **Prolog. Programowanie**. Helion, 2003
- [5] SWI-Prolog's home: <http://www.swi-prolog.org>
- [6] Learn Prolog Now!: <http://www.learnprolognow.org>
- [7] <http://home.agh.edu.pl/~ligeza/wiki/prolog>
- [8] <http://www.im.pwr.wroc.pl/~przemko/prolog>

## Terms

The set of **terms**  $TER$  is one satisfying the following conditions:

- ✘ if  $c$  is a constant,  $c \in C$ , then  $c \in TER$ ;
- ✘ if  $X$  is a variable,  $X \in V$ , then  $X \in TER$ ;
- ✘ if  $f$  is an  $n$ -ary function symbol ( $f/n$ ),  $f \in F$ , and  $t_1, t_2, \dots, t_n$  are terms, then

$$f(t_1, t_2, \dots, t_n) \in TER$$

- ✘ all the elements of  $TER$  are generated only by applying the above rules.

## Two-Argument Terms: Binary Relations/Operators

Consider a term  $f(x, y)$ . It can be represented as:

- ✘  $f(x,y)$  (prefix notation; simplified form  $f x y$  ),
- ✘  $xfy$  (infix notation),
- ✘  $(x,y)f$  (postfix notation; simplified form  $x y f$  ).

## Single-Argument Terms

Consider a term  $f(x)$ . It can be represented as:

- ✖  $fx$  (prefix notation),
- ✖  $xf$  (postfix notation)

## Terms: n-argument relations

Prefix notation:

$$f(t_1, t_2, \dots, t_n) \in TER$$

can be replaced by:

$$t_1 f t_2 f t_3 \dots t_{n-1} f t_n$$

## Nested terms

Prefix notation:

$$f(f(f(y)))$$

can be represented as:

$$f f f y$$

## Operators: basic information

- 1 Operators are defined to improve the **readability of source-code**. For example, without operators, to write  $2*3+4*5$  one would have to write  $+(*(2,3),*(4,5))$ .
- 2 In PROLOG, a number of operators have been predefined. All operators, except for the comma (,) can be redefined by the user.
- 3 Declaration of operators:  
`:- op(+Precedence, +Type, :Name) .`
- 4 Name can be a list of names; all elements are declared to be identical operators.
- 5 Precedence is an integer between 0 and 1200. Precedence 0 removes the declaration.
- 6 Type is one of: `xf`, `yf`, `xfx`, `xfy`, `yfx`, `fy` or `fx`.
  - Infix operators: `xfx`, `xfy`, `yfx`, `yfy`.
  - Prefix operators: `fx`, `fy`.
  - Postfix operators: `xf`, `yf`.
- 7 `xfy` — right-hand commutativity.
- 8 `yfx` — left-hand-commutativity.
- 9 `:- op(450, yfx, '+' ) .`  
means:  $a+b+c = (a+b)+c$  ( but not  $a+(b+c)$  ).

## text

- 1 **f** indicates position of the functor; **x** and **y** — positions of the arguments.
- 2 **y** should be interpreted as “on this position a term with precedence **lower or equal** to the precedence of the functor should occur”.
- 3 For **x** the precedence of the argument must be strictly lower.
- 4 The precedence of a term is 0, unless its principal functor is an operator, in which case the precedence is the precedence of this operator.
- 5 A term enclosed in brackets ( ... ) has precedence 0.
- 6 It is not allowed to redefine the comma (',').
- 7 (!) can only be (re-)defined as infix operator with priority not less than 1001.
- 8 It is not allowed to define the empty list [] or the curly-bracket-pair {} as operators.

## Example definitions of operators

```
1 :- op(800, fx, if) .
2 :- op(700, xfx, then) .
3 :- op(250, xfy, and) .
```

## Set of predefined operators of SQR-Prolog

```
1 1200 xfx -->, :-
2 1200 fx :-, ?-
3 1150 fx dynamic, discontiguous, initialization,
4         meta_predicate, module_transparent, multifile,
5         thread_local, volatile
6 1100 xfy ;, |
7 1050 xfy ->, op*->
8 1000 xfy ,
9 900 fy \+
10 900 fx ~
11 700 xfx <, =, =.., =@=, =:=, =<, ==, =\=, >, >=,
12         @<, @=<, @>, @>=, \=, \==, is
13 600 xfy :
14 500 yfx +, -, /\, \/ , xor
15 500 fx ?
16 400 yfx *, /, //, rdiv, <<, >>, mod, rem
17 200 xfx **
18 200 xfy ^
19 200 fy +, -, \
```

A motivational example: two equivalent terms.

```
1 p(X,Z) :-q(X,Y),r(Y,Z),s(Z).
2 ':-' (p(X,Z), ('', ' (q(X,Y), ', ' (r(Y,Z), s(Z))))).
```

Meaning of operators

```
1 op(Priority,Type,Name)
2     |           |
3     |           -- xfy, yfx, xfx, fx, fy, xf, yf
4     -- the higher number the priority has, the lower priority
```

A motivational example: two equivalent terms.

```
1 op(400,yfx,'*'). % a*b*c means ((a*b)*c)
2 op(500,yfx,'+').
3 op(500,yfx,'-'). % be careful a-b-c means ((a-b)-c)
4 op(700,xfx,'=') % it is not possible to write a=b=c
5 op(900,fy,not). % not not a means not(not(a))
6 not 1=2+3+4*5 is equivalent to: not(1=((2+3)+(4*5)))
7     not ('=' (1, '+' ('+' (2,3), '* (4,5)))).
```



## Tic-tac-toe: spatial notation of terms

```
1 :- op(500,xfx,===>).
2 :- op(400,xfy,=====).
3 :- op(300,xfy,|).
4 x|o|x
5 =====
6 x|o|x
7 =====
8 o|x|o.
9
10 x|o|x
11 =====
12 x|o|x
13 =====
14 o|_|o
15
16 ===>
17
18 x|o|x
19 =====
20 x|o|x
21 =====
22 o|x|o.
```

## Animal Expert System: Direct use of Prolog

```

1  :- dynamic xpositive/2.
2  :- dynamic xnegative/2.
3  run:-
4      write('An Expert Systems for Animals'), nl, nl,
5      write('Answer the following questions with yes/no'), nl,nl,
6      animal_is(X),
7      write('The animal may be: '), write(X), nl, nl.
8  % clear_facts.
9
10     positive(X,Y):- xpositive(X,Y),!.
11     positive(X,Y):- not(negative(X,Y)), !, ask(X,Y).
12     negative(X,Y):- xnegative(X,Y), !.
13 ask(X,Y):-
14     rite(X), write(' it '), write(Y), nl, read(Reply),
15     remember(X,Y,Reply).
16 remember(X,Y,yes):-
17     asserta(xpositive(X,Y)).
18 remember(X,Y,no):-
19     asserta(xnegative(X,Y)),fail.
20 clear_facts:- retract(xpositive(_, _)),fail.
21 clear_facts:- retract(xnegative(_, _)),fail.
22 % clear_facts:- nl,write('Press the dot to Exit'),read(_).

```

## Animal Expert System: Direct use of Prolog

```
1   animal_is(cheetah):-
2       it_is(mammal),
3       it_is(carnivore),
4       positive(has,tawny_color),
5       positive(has,black_spots),!.
6   animal_is(tiger):-
7       it_is(mammal),
8       it_is(carnivore),
9       positive(has,tawny_color),
10      positive(has,black_stripes),!.
11  animal_is(giraffe):-
12      it_is(ungulate),
13      positive(has,long_neck),
14      positive(has,long_legs),
15      positive(has,dark_spots),!.
16  animal_is(zebra):-
17      it_is(ungulate),
18      positive(has,black_stripes),!.
19  animal_is(ostrich):-
20      it_is(bird),
21      not(positive(does,fly)),
22      positive(has,long_neck),
23      positive(has,long_legs),!.
```

## Animal Expert System: Direct use of Prolog

```
1  animal_is(penguin):-
2      it_is(bird),
3      not(positive(does,fly)),
4      positive(does,swim),
5      positive(has,black_and_white_color),!.
6  animal_is(albatross):-
7      it_is(bird),
8      positive(does,fly),
9      positive(does,fly_well),!.
10 it_is(mammal):-
11     positive(has,hair),positive(does,give_milk),!.
12 it_is(carnivore):-
13     it_is(mammal),
14     positive(does,eat_meat),positive(has,pointed_teeth),
15     positive(has,claws),!.
16 it_is(ungulate):-
17     it_is(mammal),
18     positive(has,hooves),positive(does,chew_cud),!.
19 it_is(bird):-
20     not(positive(has,hair)),
21     not(positive(does,give_milk)),
22     positive(has,feathers), positive(does,lay_eggs),!.
```

## Simple Backward-Chaining Expert System

```
1 :- dynamic pos/1.
2 :- dynamic neg/1.
3 rule(1,informatyka,[lubi_komputery,zna_angielski,
4                     mysli_analitycznie,mysli_po_inzyniersku]).
5 rule(2,gornictwo,[lubi_wegiel,kocha_slask,mysli_po_inzyniersku]).
6 rule(3,mysli_analitycznie,[rozwiazuje_sudoku,planuje_kalendarz]).
7 rule(4,mysli_po_inzyniersku,[dokreca_srubki,rysuje_techicznie]).
8 rule(5,kocha_slask,[mieszka_familiok,godo_gwaro]).
9
10 decision(D):- rule(_,D,List), prove(List).
11 prove([]):-!.
12 prove([D|R]):- fact(D),prove(R).
13
14 fact(D):- pos(D),!,write('Found in fact base: '),write(D),nl.
15 fact(D):- decision(D),assert(pos(D)),!,write(D),nl.
16 fact(D):- not(neg(D)), write('Is that true: '),write(D),write('?'),
17             read(X),X='yes',assert(pos(D)),!,write(D),nl.
18 fact(D):- not(neg(D)),write('Then probably no? '),
19             read(X),X='no',assert(neg(D)),!,write(D),fail.
20 fact(D):- not(pos(D)),not(neg(D)),
21             write('Once more please: yes/no'),fact(D),!.
22 clear:- retract(pos(_)),fail. clear:- retract(neg(_)),fail. clear.
```

## Backward-Chaining Expert Systems. Operators in use

```
1 :- dynamic pos/1.
2 :- dynamic neg/1.
3 :- op(830,fx,rule) .
4 :- op(780,xfx,::) .
5 :- op(750,xfx,==>) .
6
7 rule 1 :: [lubi_komputery, zna_angielski, mysli_analitycznie,
8           mysli_po_inzyniersku] ==> informatyka.
9 rule 2 :: [lubi_komputery, zna_angielski, mysli_analitycznie,
10          mysli_po_inzyniersku] ==> gornictwo.
11 rule 3 :: [rozwiazuje_sudoku, planuje_w_kalendarzu]
12          ==> mysli_analitycznie.
13 rule 4 :: [dokreca_srubki, rysuje_techniczne]
14          ==> mysli_po_inzyniersku.
15 rule 5 :: [mieszka_familiok, godo_gwaro] ==> kocha_slask.
16
17 decision(D):-
18 % rule(_,D,List),
19   rule _ :: List ==> D,
20   prove(List) .
21
22 prove([]):-!.
23 prove([D|R]):- fact(D),prove(R) .
```

## Rules for 2 blocks

```
1 :- dynamic fact/1 .
2 fact (on(b,a)) . fact (onfloor(a)) .
3     run:- repeat,
4         findrule (Number),executerule (Number),
5         test,!.
6 % rule(<id>,<preconditions>,<action>,<retract>,<assert>)
7 rule(1,stop,[on(a,b),onfloor(b)],stop,[],[]).
8 rule(2,put,[onfloor(a),onfloor(b)],put,[onfloor(a)],[on(a,b)]).
9 rule(3,dec,[on(b,a),onfloor(a)],dec,[on(b,a)],[onfloor(b)]).
10 findrule (Number):-
11     rule (Number,_,Preconditions,_,_,_),
12     consistent (Preconditions),!.
13 executerule (Number):-
14     rule (Number,Name,Preconditions,Action,Dels,Adds),
15     consistent (Preconditions),
16     remove (Dels), add (Adds).
17 test:- rule(1,_,Preconditions,_,_,_),
18     consistent (Preconditions).
19 clearstate:- retract (fact (_)),fail. clearstate.
20 remove ([]):- !. remove ([F|T]):- retract (fact (F)),remove (T).
21 add ([]):- !. add ([F|T]):- assert (fact (F)),add (T).
22 consistent ([]):- !. consistent ([F|T]):- fact (F),consistent (T).
```

## Three Blocks World Example: Rule-Based Approach

```
1 :- dynamic fact/1 .
2
3 fact (on(b,a)) .
4 fact (on(c,b)) .
5 fact (onfloor(a)) .
6 fact (cleartop(c)) .
7
8     run:-
9         liststate,
10        repeat,
11        findrule(Number),
12        executerule(Number),
13        liststate,
14        test,!.
15
16 % rule(<id>,<preconditions>,<action>,<retract>,<assert>)
17 rule(1,stop,[cleartop(a),on(a,b),on(b,c),onfloor(c)],stop,[],[]).
18 rule(2,putbc,[cleartop(c),cleartop(b),onfloor(b),onfloor(c)],
19        putbc,[onfloor(b),cleartop(c)],[on(b,c)]).
20 rule(3,putab,[cleartop(a),onfloor(a),on(b,c),onfloor(c)],putab,
21        [onfloor(a),cleartop(b)],[on(a,b)]).
22 rule(4,dec,[cleartop(_X),on(_X,_Y)],dec,[on(_X,_Y)],
23        [cleartop(_Y),onfloor(_X)]).
```



## Path planning

```
1   droga(krakow, katowice) .
2   droga(katowice, opole) .
3   droga(wroclaw, opole) .
4   droga(krakow, zakopane) .
5   droga(krakow, kielce) .
6   droga(krakow, tarnow) .
7   droga(kielce, radom) .
8   droga(radom, warszawa) .
9
10  %%% Symmetric closure
11  przejazd(X, Y) :- droga(X, Y) .
12  przejazd(X, Y) :- droga(Y, X) .
13
14  %%% Transitive closure
15  szukaj_trasy(Cel, Cel, Trasa, Trasa) :- !.
16  szukaj_trasy(Miasto, Cel, Robocza, Trasa) :-
17      przejazd(Miasto, NoweMiasto),
18      not(member(NoweMiasto, Robocza)),
19      szukaj_trasy(NoweMiasto, Cel, [NoweMiasto|Robocza], Trasa) .
20  plan(Start, Cel, Trasa) :-
21      szukaj_trasy(Cel, Start, [Cel], Trasa) .
```

## Path planning with length calculation

```
1      droga(krakow, rzeszow, 162) .
2      droga(krakow, lublin, 273) .
3      droga(krakow, kielce, 117) .
4      droga(krakow, lodz, 241) .
5      droga(krakow, katowice, 81) .
6      droga(rzeszow, lublin, 168) .
7      droga(rzeszow, kielce, 168) .
8      droga(lublin, kielce, 179) .
9      przejazd(X, Y, D) :- droga(X, Y, D) .
10     przejazd(X, Y, D) :- droga(Y, X, D) .
11     %%% szukaj_trasy(<start>, <goal>, <temp_route>, final_route>,
12     %%%                <temp_length>, <final_length>)
13     szukaj_trasy(Cel, Cel, Trasa, Trasa, Dystans, Dystans, _) .    %%% !.
14     szukaj_trasy(Miasto, Cel, Robocza, Trasa, Dlugosc,
15                 Dystans, Ograniczenie) :-
16         przejazd(Miasto, Kandydat, D) ,
17         not(member(Kandydat, Robocza)) ,
18         NowaDlugosc is Dlugosc + D,
19         NowaDlugosc < Ograniczenie,
20         szukaj_trasy(Kandydat, Cel, [Kandydat | Robocza], Trasa,
21                     NowaDlugosc, Dystans, Ograniczenie) .
22     plan(Miasto, Cel, Trasa, Dystans, Ograniczenie) :-
23         szukaj_trasy(Cel, Miasto, [Cel], Trasa, 0, Dystans, Ograniczenie) .
```

## Missionaries and Cannibals

```
1 %% mision.pl
2 droga(s(X,Y,l),s(X1,Y1,p)):- X >= 1, X1 is X-1, Y1 is Y.
3 droga(s(X,Y,l),s(X1,Y1,p)):- Y >= 1, X1 is X, Y1 is Y-1.
4 droga(s(X,Y,l),s(X1,Y1,p)):- X > 1, X1 is X-2, Y1 is Y.
5 droga(s(X,Y,l),s(X1,Y1,p)):- Y > 1, X1 is X, Y1 is Y-2.
6 droga(s(X,Y,l),s(X1,Y1,p)):- X >= 1,Y >= 1,X1 is X-1,Y1 is Y-1.
7 droga(s(X,Y,p),s(X1,Y1,l)):- X < 3, X1 is X+1, Y1 is Y.
8 droga(s(X,Y,p),s(X1,Y1,l)):- Y < 3, X1 is X, Y1 is Y+1.
9 droga(s(X,Y,p),s(X1,Y1,l)):- X < 2, X1 is X+2, Y1 is Y.
10 droga(s(X,Y,p),s(X1,Y1,l)):- Y < 1, X1 is X, Y1 is Y+2.
11 droga(s(X,Y,p),s(X1,Y1,l)):- X < 3,Y < 3,X1 is X+1,Y1 is Y+1.
12
13 zabronione(s(X,Y,_)):-X >= 1,X < Y.
14 zabronione(s(X,Y,_)):-X1 is 3-X,Y1 is 3-Y,X1 >= 1,X1 < Y1.
15
16 szukaj(X,X,S,S):- !.
17 szukaj(X,Y,W,S):-
18     droga(X,Z),
19     not(zabronione(Z)),
20     not(member(Z,W)),
21     szukaj(Z,Y,[Z|W],S).
22 plan(X,Y,S):-
23     szukaj(X,Y,[X],S), writelist(S).
```

## Modeling a simple Robot

```
1 fact (n(a,b)).
2 fact (n(a,c)).
3 fact (n(b,d)).
4 fact (n(b,a)).
5 fact (n(c,a)).
6 fact (n(d,c)).
7 fact (n(d,b)).
8 fact (n(c,d)).
9
10 %%% Initial State
11 state ([at(robot,d), at(box,a)]).
12 %%% Goal State
13 end ([at(robot,a), at(box,d)]).
14 %%% Actions
15 action (go(X,Y),
16         [at(robot,X), n(X,Y)],
17         [at(robot,X)],
18         [at(robot,Y)]).
19 action (move(X,Y),
20         [at(box,X), n(X,Y), at(robot,X)],
21         [at(robot,X), at(box,X)],
22         [at(robot,Y), at(box,Y)]).
```

## Path planning

```
1  run :-
2      initialize(Statelist,Actionlist),
3      solve(Statelist,Actionlist,Result),
4      stop(Result).
5  initialize(Statelist,Actionlist) :-
6      state(State),
7      Statelist = [State], Actionlist = [],
8      asserta(node(1)).
9  %% Man Loop - Search for Plan
10 solve(Statelist,Actionlist,Result) :-
11     end(Endlist),
12     Statelist=[State|_],
13     inlist(State,Endlist),!,
14     Actionlist=Result.
15 solve(Statelist,Actionlist,Result) :-
16     action(Action,Prec,Del,Add),
17     Statelist=[State|_],
18     check(Prec,State),
19     delete(State,Del,DelState),
20     add(DelState,Add,NewState),
21     not(instatelist(NewState,Statelist)), modify,
22     solve([NewState|Statelist],[Action|Actionlist],Result).
23 modify :- retract(node(N)),!,N1 is N+1,asserta(node(N1)).
```

## Three-Blocks Worlds

```
1  %%% Knowledge Base for Three-Blocks Worlds
2  %%% Initial state
3  state([onfloor(a), onfloor(b), on(c, a), cl(c), cl(b)]).
4  %%% Goal state
5  end([on(a, b), on(b, c)]).
6
7  %%% Actions
8  %%% put X on Y; X initially on the floor
9  action(put(X, Y),
10         [onfloor(X), cl(X), cl(Y)],
11         [onfloor(X), cl(Y)],
12         [on(X, Y)]).
13 %%% move X from Y to Z; X initially on Y
14 action(move(X, Y, Z),
15         [on(X, Y), cl(X), cl(Z)],
16         [on(X, Y), cl(Z)],
17         [on(X, Z), cl(Y)]).
18 %%% free Y by removing X; X is taken from Y and put on the floor
19 action(free(X, Y),
20         [on(X, Y), cl(X)],
21         [on(X, Y)],
22         [onfloor(X), cl(Y)]).
```

## A Cryptarithmic Problem Solution

```

1  Program do rozwiązywania zagadek kryptoarytmetycznych
2  %%% From: I.Bratko: Prolog Programming for AI
3  % Example calls:
4  % sum([D,O,N,A,L,D],[G,E,R,A,L,D],[R,O,B,E,R,T]).
5  % sum([0,S,E,N,D],[0,M,O,R,E],[M,O,N,E,Y]).
6  % sum([0,C,R,O,S,S],[0,R,O,A,D,S],[D,A,N,G,E,R]).
7  solve(S,E,N,D,M,O,R,Y):-sum([0,S,E,N,D],[0,M,O,R,E],[M,O,N,E,Y]),M\=0.
8  del(A,L,L):-nonvar(A),!.
9  del(A,[A|L],L).
10 del(A,[B|L],[B|L1]):-del(A,L,L1).
11 digitsum(D1,D2,C1,D,C,Digs1,Digs):-
12 del(D1,Digs1,Digs2),
13 del(D2,Digs2,Digs3),
14 del(D,Digs3,Digs),
15 S is D1+D2+C1, D is S mod 10, C is S // 10.
16 sum1([],[],[],0,0,Digits,Digits).
17 sum1([D1|N1],[D2|N2],[D|N],C1,C,Digs1,Digs):-
18 sum1(N1,N2,N,C1,C2,Digs1,Digs2),
19 digitsum(D1,D2,C2,D,C,Digs2,Digs).
20 sum(N1,N2,N):-
21 sum1(N1,N2,N,0,0,[0,1,2,3,4,5,6,7,8,9],_).

```

## Prolog in Prolog

```
1  %%% Interpreter of Prolog %%%
2
3  runprolog(true):- !.
4  runprolog((G1,G2)):- !, runprolog(G1), runprolog(G2).
5  runprolog(G):- clause(G,Body), runprolog(Body).
6
7  %%% Example program %%%
8
9  c(a,b).
10 c(b,c).
11 c(c,d).
12
13 sc(X,Y):- c(X,Y).
14 sc(X,Y):- c(Y,X).
15
16 tc(X,Z):- c(X,Z).
17 tc(X,Z):- c(X,Y),tc(Y,Z).
```



## Path planning

text

text

example

example

1 `pred(arg1, arg2, ... argN).`

alert

alert

text

text

example

example

1 `pred(arg1, arg2, ... argN).`

alert

alert

text

text

example

example

1 `pred(arg1, arg2, ... argN).`

alert

alert