

**LOGIKA**  
**jako**  
**JEZYK PROGRAMOWANIA.**  
**PROLOG W 33 MINUTY**

© Antoni Ligeza

## Zarys planu wykładu

1. Intuicyjne wprowadzenie do problematyki PL (1).
2. Zarys *filozofii* PROLOGU (1).
3. Wprowadzenie do PROLOGU (2).
4. Przegląd typowych zastosowań (1).
5. Alfabet i język (1).
6. Mechanizm unifikacji (2).
7. Wnioskowanie – reguła rezolucji (1).
8. Mechanizm wnioskowania (SLD) (1).
9. SLD-drzewa – strategia wnioskowania (1).
10. Negacja jako porażka (1).
11. Sterowanie wnioskowaniem: nawroty (*fail*) i obcięcia (*cut*) (2).
12. Modyfikacja bazy wiedzy: *retract* i *assert* (1).

## Zarys planu wykładu – c.d.

13. Elastyczne struktury danych: listy (1).
14. Operacje na listach: *member*, *append* (1).
15. Przykłady operacji na listach (3).
16. Przykład: min i porządkowanie listy (2).
17. Przykład obliczeń – silnia (1).
18. Przykład problemu – "Wieże w Hanoi".
19. Przykład symulacji układu cyfrowego.
20. Przykład problemu – "8 Królowych"
21. Przykład: planowanie trasy przejazdu (1).
22. Przykład: system ekspertowy (4).
23. Niektóre dalsze problemy.
24. Zarys literatury problemu (1).
25. Rys historyczny i perspektywy.

# Problemy Inżynierii Wiedzy

## Problemy:

1. Reprezentacja wiedzy (stany, własności, transformacje, heurystyki). Prolog: fakty + reguły.
2. Mechanizm wnioskujący. Prolog: Rezolucja
3. Sterowanie wnioskowaniem. Strategia SLD
4. Akwizycja wiedzy. Prolog: programowanie deklaratywne
5. Weryfikacja wiedzy. Prolog: specyfikacja wykonywalna.
6. Interfejs użytkownika. Prolog: ?-
7. Uczenie się Prolog: retract + assert.

## Zarys filozofii Prologu

- programowanie deklaratywne (a nie proceduralne),
- indeterminizm,
- brak rozróżnienia we/wy (relacje vs. funkcje),
- rekurencja,
- brak "instrukcji".

ALGORYTM = LOGIKA + STEROWANIE

- unifikacja (dopasowanie wzorca),
- rezolucja (wnioskowanie),
- strategia SLD (sterowanie wnioskowaniem).

## Wprowadzenie do Prologu

Termy – nazwy obiektów:

- stałe, np.:  $a, b, c, \dots$ , ewa, jan, tomek, jacek
- zmienne, np.:  $X, Y, Z, \dots$
- obiekty złożone, np.:  $f(a), f(X), g(a, b), g(f(a), X), \dots$

Formuły atomowe – zapis faktów:  $p(t_1, t_2, \dots, t_n)$ .

kobieta(ewa).

mezczyzna(tomek).

ojciec(jan, jacek).

Klauzule – reguły wnioskowania:  $h : -p_1, p_2, \dots, p_n$ .

(równoważnik logiczny:  $p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow h$ )

```
brat(B,X):-    /* B jest bratem X */
    ojciec(P,X),
    ojciec(P,B),
    mezczyzna(B),
    B<>X.
```

Cel – pytanie użytkownika: g.

brat(tomek, jacek), brat(X, jacek),

brat(jacek, X), brat(X, Y).

## Przykład:

kobieta(ewa).

mezczyzna(tomek).

mezczyzna(jacek).

ojciec(jan,ewa).

ojciec(jan,tomek).

ojciec(jan,jacek).

ojciec(jacek,wojtek).

brat(B,X):- /\* B jest bratem X \*/

ojciec(P,X),

ojciec(P,B),

mezczyzna(B),

B<>X.

wuj(U,X):- /\* U jest wujem X \*/

ojciec(P,X),

brat(U,P).

## Przegląd Typowych Zastosowań

1. Dowodzenie twierdzeń (odpowiedź typu: tak/nie):
  - weryfikacja własności opisywalnych aksjomatycznie
2. Wyszukiwanie wartości parametrów (odpowiedź typu:  $X=...$ ):
  - bazy danych, systemy dialogowe
  - wspomaganie decyzji, diagnozowanie
  - systemy ekspertowe, systemy doradcze
  - sterowanie inteligentne, systemy regułowe
3. Synteza planów (odpowiedź typu: wykonaj a, b, c, ...):
  - synteza algorytmów sterowania (planów)
  - poszukiwanie drogi
  - rozwiązywanie problemów
  - szukanie heurystyczne
4. Problemy złożone (odpowiedź typu: jeżeli ... to ...):
  - gry, wybór strategii
  - symulacja
  - synteza programów



## Alfabet i Język

Stałe: a,b,c,...

Zmienne: X,Y,Z,... (-)

Symbole funkcyjne: f,g,h,...

Symbole relacyjne: p,q,r,...

Symbole specjalne: :- (if), przecinek (and), kropka (koniec klauzuli), nawiasy.

Termy:

- każda stała i zmienna jest termem,
- jeżeli  $t_1, t_2, \dots, t_n$  są termami, to  $f(t_1, t_2, \dots, t_n)$  jest termem.

Formuły atomowe: jeżeli  $t_1, t_2, \dots, t_n$  są termami, to  $p(t_1, t_2, \dots, t_n)$  jest formułą atomową (literał = f. atomowa lub jej negacja).

Klauzule (Horna):

$$h : - p_1, p_2, \dots, p_n.$$

Zapis logiczny:

$$h \vee \neg p_1 \vee \neg p_2 \vee \dots \neg p_n$$

Warianty klauzul:  $h$ . (fakt),  $: - p_1, p_2, \dots, p_n$ . (wywołanie, cel).

## Mechanizm unifikacji

Podstawienie: Odwzorowanie zbioru zmiennych w zbiór termów (jedynie skończonej liczbie zmiennych przypisuje termy różne od nich samych).

Reprezentacja podstawienia:  $\sigma = \{X_1/t_1, X_2/t_2, \dots, X_n/t_n\}$ .  
( $t_i \neq X_i, X_i \neq X_j$ )

Unifikacja wyrażeń: Podstawienie  $\theta$  jest unifikatorem wyrażeń  $E_1$  oraz  $E_2$  wtw gdy  $E_1\theta = E_2\theta$ .

Najbardziej ogólne podstawienie unifikujące (mgu): Podstawienie  $\theta$  jest mgu pewnych wyrażeń wtw gdy dla każdego podstawienia  $\sigma$  będącego unifikatorem istnieje podstawienie  $\lambda$ , takie że  $\sigma = \theta\lambda$ .

## Mechanizm Unifikacji – Algorytm

Dane:  $W$  – zb. wyrażeń do unifikacji, Szukane:  $\theta = \text{mgu}(W)$

1. Podstawić:  $k = 0$ ,  $W_k = W$ ,  $\theta_k = \epsilon$ .
2. Jeżeli  $W_k$  ma jeden element to stop;  $\theta_k = \text{mgu}(W)$ . W przeciwnym przypadku znaleźć  $D_k$  – zbiór niezgodności (podwyrażeń) dla  $W_k$ .
3. Jeżeli w  $D_k$  mamy  $X_k$  i  $t_k$  ( $X_k \notin t_k$ ) to przejść do 4; w przeciwnym przypadku stop – brak unifikacji.
4.  $\theta_{k+1} = \theta_k\{X_k/t_k\}$ ,  $W_{k+1} = W_k\{X_k/t_k\}$ .
5.  $k = k + 1$ , przejść do 2.

## Wnioskowanie – Reguła Rezolucji

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

$$\frac{: -r_1, r_2, \dots, r_k, \quad h : -p_1, p_2, \dots, p_n}{: -p_1\theta, p_2\theta, \dots, p_n\theta, r_2\theta, \dots, r_k\theta}$$

gdzie  $\theta = mgu(r_1, h)$ .

## Mechanizm wnioskowania (SLD)

Cel  $C_1$

$C'_1$   $C_2$

$C'_2$   $C_3$

$C'_{n-1}$   $C_n$

$C'_n$

**SLD** – Linear resolution for **D**efinite clauses with **S**election function.

## SLD-Drzewa – Strategia wnioskowania

Niech  $P$  będzie programem logicznym (zbiorem klauzul),  $G_0$  – celem, a  $R$  – regułą wnioskowania (SLD-rezolucji).

SLD-Drzewo jest zdefiniowane jak następuje:

- $G_0$  jest korzeniem,
- $G_{i+1}$  jest węzłem bezpośrednio pochodnym od  $G_i$  wtw gdy istnieje rezolwenta  $G_i$  oraz pewnej klauzuli z  $P$ .

$G_0$

$G_i$

$G_{i+1}$

## Negacja jako porażka

$$\frac{P \not\models q}{\neg q}$$

W PROLOGU negacją atomu  $q$  realizowana jest jako porażka dowodzenia celu  $q$ ; służy do tego standardowy predykat  $not(.)$ . Dowodzenie celu  $not(q)$  kończy się sukcesem, o ile dowodzenie celu  $q$  (wywołane jako podprocedura) zakończy się porażką.

```
telefon(adam,111111).  
telefon(bogdan,222222).  
telefon(czeslaw,333333).  
nie_ma_telefonu(X):-not(telefon(X,_)).
```

## Sterowanie wnioskowaniem: nawroty (*fail*) i obciecia (*cut*)

Standardowy predykat *fail* wymusza nawrót – próba jego wywołania zawsze kończy się porażką.

Standardowy predykat ! powoduje:

- podział klauzuli na dwie części, lewą i prawą, w stosunku do miejsca wystąpienia !,
- uniemożliwienie nawracania (ponownego uzgadniania) literałów z lewej części klauzuli.

```
not(X):- X,!,fail.
```

```
not(X).
```



## Modyfikacje bazy wiedzy: *retract* i *assert*

W trakcie wykonania programu możliwa jest modyfikacja bazy wiedzy (programu).

Predykat *retract(.)* usuwa zadany atom z bazy wiedzy (programu).

Predykat *assert(.)* dopisuje zadany atom do bazy wiedzy (programu).

```
retract_all(X):- retract(X), fail.  
retract_all(X).
```

## Elastyczne struktury danych – listy

Lista: ciąg elementów (długość listy nie jest predefiniowana).

Reprezentacja listy:

$$[e_1, e_2, \dots, e_n]$$

Bezpośredni dostęp jest tylko do pierwszego elementu listy (głowy);  
operator | oddziela głowę listy od reszty (ogona).

$$[H|T]$$

```
list([]).
```

```
list([H|T]):-list(T).
```

## Operacje na listach: *member* i *append*

```
member(Name, [Name|_]) :- !.  
member(Name, [_|Tail]) :- member(Name, Tail).
```

```
select(Name, [Name|_]).  
select(Name, [_|Tail]) :- select(Name, Tail).
```

```
append([], L, L).  
append([X|L1], L2, [X|L3]) :-  
    append(L1, L2, L3).
```

## Przykłady operacji na listach

```
add(X,L,[X|L]).
del(X,[X|L],L).
del(X,[Y|L],[Y|L1]):- del(X,L,L1).
insert(X,L,LX):- del(X,LX,L).
sublist(S,L):- append(_,L2,L), append(S,_,L2).
```

```
reverse([],[]).
reverse([X|L],R):-
    reverse(L,RL),
    append(RL,[X],R).
```

```
inverse(L,R):- do([],L,R).
do(L,[],L).
do(L,[X|T],S):-
    do([X|L],T,S).
```

## Przykłady operacji na listach

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

## Przykłady operacji na listach

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

```
inclusion([],_).
inclusion([X|L],Set):-
    member(X,Set),
    inclusion(L,Set).
```

## Przykłady: – min i porządkowanie

```
min([X],X):-!.  
min([P|R],P):- min(R,X),X>P,!.  
min([P|R],X):- min(R,X),X<=P.
```

```
order([],[]).  
order([H|T],R):-  
    order(T,TR),  
    put(H,TR,R).
```

```
put(H,[],[H]):-!.  
put(H,[X|Y],[H,X|Y]):-  
    H<X,!.  
put(H,[X|Y],[X|Z]):-  
    put(H,Y,Z),!.
```

## Przykłady operacji na listach – *quicksort*

```
append([],X,X).  
append([H|L],L1,[H|L2]) :-  
    append(L,L1,L2).
```

```
split(_,[],[],[]).  
split(H,[A|X],[A|Y],Z) :-  
    A <= H,!, split(H,X,Y,Z).  
split(H,[A|X],Y,[A|Z]) :-  
    A > H,!, split(H,X,Y,Z).
```

```
qsort([],[]).  
qsort([H|T],S) :-  
    split(H,T,A,B),  
    qsort(A,A1),  
    qsort(B,B1),  
    append(A1,[H|B1],S).
```



## Przykład obliczeń – silnia

factorial(1,1).

factorial(N,Res) if

N > 0 and

N1 = N-1 and

factorial(N1,FacN1) and

Res = N\*FacN1.

## Przykład: planowanie trasy przejazdu

droga(krakow,katowice).

droga(katowice,opole).

droga(wroclaw,opole).

przejazd(X,Y) if droga(X,Y).

przejazd(X,Y) if droga(Y,X).

member(X,[X|\_]) if !.

member(X,[\_|T]) if member(X,T).

eq(X,X).

szukaj\_trasy(X,X,S,W) if eq(S,W) and !.

szukaj\_trasy(X,Y,S,W) if

przejazd(X,Z) and not(member(Z,W)) and

eq(W1,[Z|W]) and szukaj\_trasy(Z,Y,S,W1).

plan(Y,X,S) if eq(W,[X]) and

szukaj\_trasy(X,Y,S,W).

## Przykład problemu – "Wieże w Hanoi"

```
hanoi(N) :- move(N,left,middle,right).
```

```
move(1,A,_,C) :- inform(A,C),!.
```

```
move(N,A,B,C) :-
```

```
    N1=N-1,move(N1,A,C,B),inform(A,C),move(N1,B,A,C).
```

```
inform(Loc1,Loc2):-
```

```
    write("\nMove a disk from ",Loc1," to ",Loc2).
```

## Przykład symulacji układu cyfrowego

```
not_(1,0).
not_(0,1).
and_(0,0,0).
and_(0,1,0).
and_(1,0,0).
and_(1,1,1).
or_(0,0,0).
or_(0,1,1).
or_(1,0,1).
or_(1,1,1).
xor(Input1,Input2,Output) if
    not_(Input1,N1) and not_(Input2,N2) and
    and_(Input1,N2,N3) and and_(Input2,N1,N4) and
    or_(N3,N4,Output).
```

## Przykład problemu – "8 Królowych"

```
nqueens(N) :-
    makelist(N,L),Diagonal=N*2-1,makelist(Diagonal,LL),
    placeN(N,board([],L,L,LL,LL),board(Final,_,_,_,_)),
    write(Final).
placeN(_,board(D,[],[],D1,D2),board(D,[],[],D1,D2)):-!.
placeN(N,Board1,Result):-
    place_a_queen(N,Board1,Board2),
    placeN(N,Board2,Result).
place_a_queen(N,board(Queens,Rows,Columns,Diag1,Diag2),
    board([q(R,C)|Queens],NewR,NewC,NewD1,NewD2)):-
    findandremove(R,Rows,NewR),
    findandremove(C,Columns,NewC),
    D1=N+C-R,findandremove(D1,Diag1,NewD1),
    D2=R+C-1,findandremove(D2,Diag2,NewD2).
findandremove(X,[X|Rest],Rest).
findandremove(X,[Y|Rest],[Y|Tail]):-
    findandremove(X,Rest,Tail).
makelist(1,[1]).
makelist(N,[N|Rest]):-
    N1=N-1,makelist(N1,Rest).
```

## Przykład: – system ekspertowy

```
positive(X,Y) if xpositive(X,Y),!.
positive(X,Y) if not(negative(X,Y)),! and ask(X,Y).
negative(X,Y) if xnegative(X,Y),!.
ask(X,Y):-
    write(X," it ",Y,"\n"),
    readln(Reply),
    remember(X,Y,Reply).
remember(X,Y,yes):-
    asserta(xpositive(X,Y)).
remember(X,Y,no):-
    asserta(xnegative(X,Y)),
    fail.
clear_facts:-
    retract(xpositive(_,_)),fail.
clear_facts:-
    retract(xnegative(_,_)),fail.
clear_facts:-
    write("\n\nPlease press the space bar to Exit"),
    readchar(_).
animal_is(cheetah) if
    it_is(mammal),
    it_is(carnivore),
```

```
    positive(has,tawny_color),
    positive(has,black_spots),!.
animal_is(tiger) if
    it_is(mammal) and
    it_is(carnivore) and
    positive(has,tawny_color) and
    positive(has,black_stripes),!.
animal_is(giraffe) if
    it_is(ungulate) and
    positive(has,long_neck) and
    positive(has,long_legs) and
    positive(has,dark_spots),!.
animal_is(zebra) if
    it_is(ungulate) and
    positive(has,black_stripes),!.
animal_is(ostrich) if
    it_is(bird) and
    not(positive(does,fly)) and
    positive(has,long_neck) and
    positive(has,long_legs),!.
animal_is(penguin) if
    it_is(bird) and
    not(positive(does,fly)) and
```

```
        positive(does,swim) and
        positive(has,black_and_white_color),!.
animal_is(albatross) if
    it_is(bird) and
    positive(does,fly),
    positive(does,fly_well),!.
it_is(mammal) if
    positive(has,hair),
    positive(does,give_milk),!.
it_is(carnivore) if
    it_is(mammal),
    positive(does,eat_meat),
    positive(has,pointed_teeth),
    positive(has,claws),!.
it_is(ungulate) if
    it_is(mammal),
    positive(has,hooves),
    positive(does,chew_cud),!.
it_is(bird) if
    not(positive(has,hair)),
    not(positive(does,give_milk)),
    positive(has,feathers),
    positive(does,lay_eggs),!.
```



## Zarys literatury problemu

1. Bratko, I: *Prolog Programming for Artificial Intelligence*. Addison-Wesley Publ. Co., Wokingham, England; Readings, Massachusetts; Menlo Park, California, 1986, 1987.
2. Chang, C.L. and Lee, R.C.T.: *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York and London, 1973.
3. Clocksin, W.F. and C. Mellish: *Programming in Prolog*. Springer-Verlag, Berlin, 1984.
4. Ligeza, A., Czarkowski, D., Marczyński, P. i Włodarczyk, M.: *Programowanie w języku Turbo-Prolog*. Skrypt AGH (złożono do druku).
5. Nilsson, U. and J. Małuszyński: *Logic, Programming and Prolog*. John Wiley and Sons, Chichester, 1990.
6. Sterling, L. and Shapiro, E.: *The Art of Prolog. Advanced Programming Techniques*. The MIT Press, Cambridge, Massachusetts; London, England, 1986.
7. Szajna, J., M. Adamski i T. Kozłowski: *Turbo Prolog. Programowanie w języku logiki*. WN-T, W-wa, 1991.