# Programowanie logiczne w Prologu

matma6 (tech. Michał Gabor)

21 stycznia 2013 r.

# Co to jest Prolog

# Fakty

- Francja, rok 1972
- Alain Colmerauer i Philippe Roussel
- Programowanie w logice (PROgrammation en LOGique)

# Implementacje

Istnieje wiele implementacji Prologu.

- SWI-Prolog
- YAP
- GNU Prolog
- SICStus Prolog
- Visual Prolog
- ...

# SWI-Prolog

Używam SWI-Prolog.
Cechy:

- LGPL/GPL
- CLP
- XPCE
- Serwer WWW
- Doskonała dokumentacja
- ...

# Instalacja

**Arch Linux** posiada SWI-Prolog w AURze (np. `yaourt -S swi-prolog`)

**Debian** ma SWI-Prolog w repozytorium (np. `sudo apt-get install swi-prolog`)

**MacOS X** paczka jest do pobrania ze strony `http://swi-prolog.org/`

**Windows** paczka jest do pobrania ze strony `http://swi-prolog.org/`

W GNU/Linuksie pojawia się wtedy polecenie `swipl`.

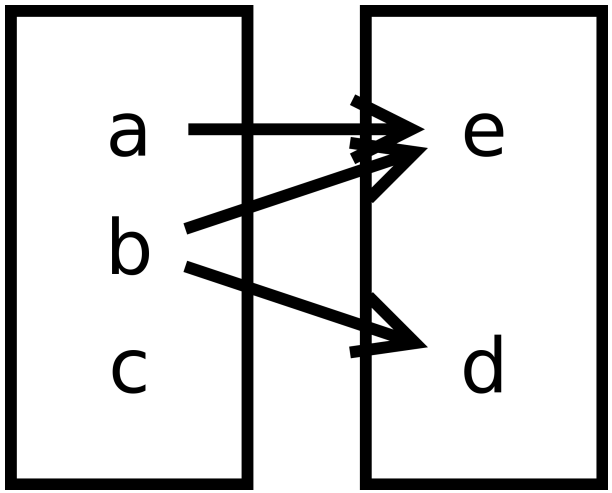
# Paradygmat logiczny

**Imperatywnie** wykonujemy instrukcje

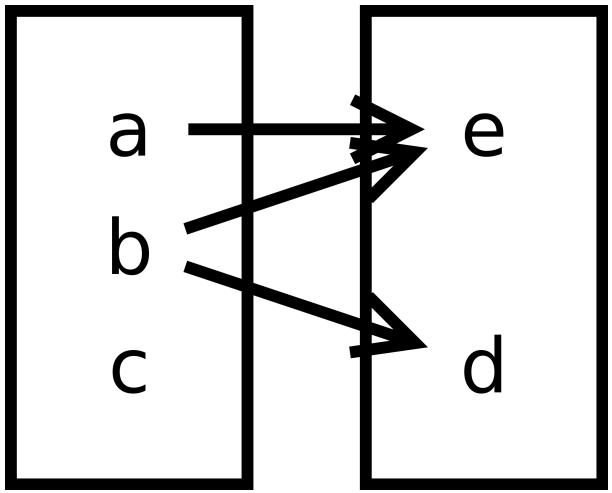
**Funkcyjnie** obliczamy wartość funkcji

**Logicznie** pytamy o relacje

# Funkcja a relacja

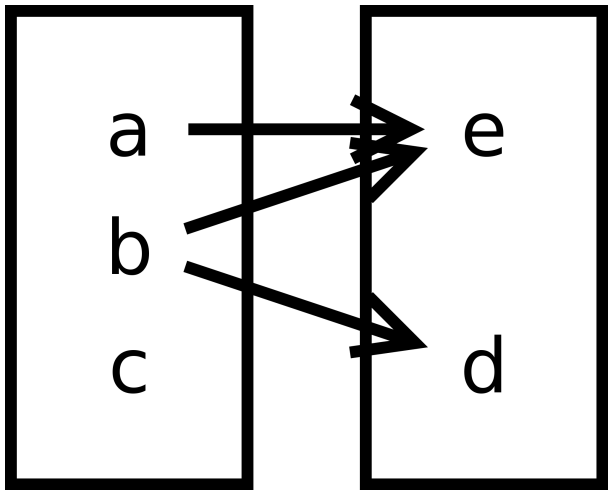


# Funkcja a relacja




Relacja $\rho$ to nie funkcja:

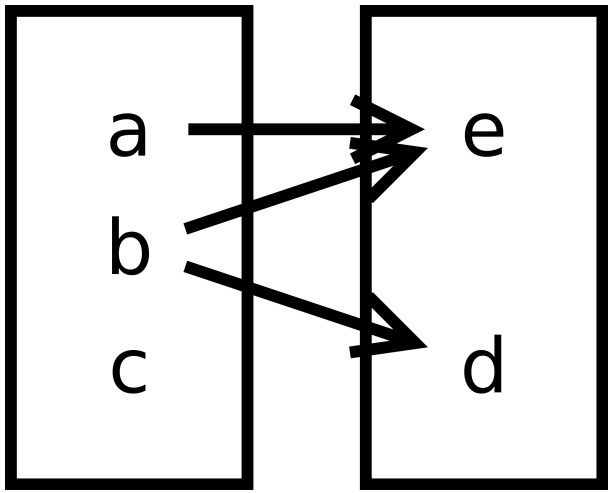
# Funkcja a relacja




Relacja $\rho$ to nie funkcja:

$\rho(a) = e$ ok...

# Funkcja a relacja

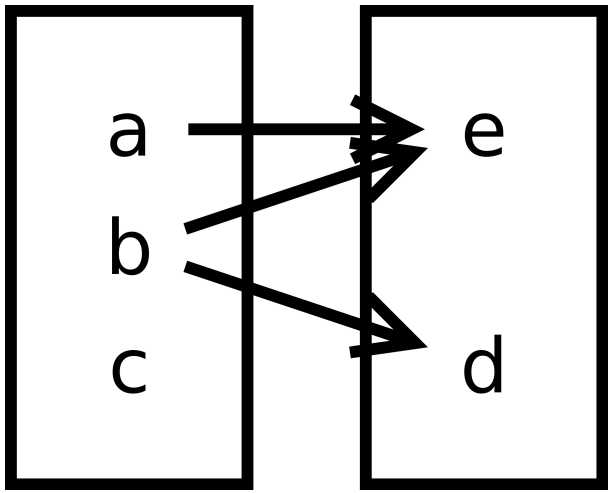



Relacja $\rho$ to nie funkcja:

$\rho(a) = e$ ok...

$\rho(b) = ?$ dwie wartości

# Funkcja a relacja




Relacja $\rho$ to nie funkcja:

$\rho(a) = e$ ok...

$\rho(b) = ?$ dwie wartości

$\rho(c) = ?$ brak wartości

# W Prologu

```
ro(a, e).
ro(b, e).
ro(b, d).
```

# Czas na przykład z życia - sinus :)

```
lubi(kasia, koty).
lubi(kasia, psy).
lubi(jan, slodycze).
lubi(jan, koty).
```

# Podstawy

# Komentarze, atomy i zmienne

**Komentarz** zaczyna się od %.
**Atom** zaczyna się z małej litery.
**Zmienna** zaczyna się z dużej litery.

```
%atomy
jan
kot
pies

%Zmienne
Lista
X
N1
ODP
```

# Źródło to nie REPL

Źródło to baza faktów!

```
lubi(kasia, koty).
lubi(kasia, psy).
lubi(jan, slodycze).
lubi(jan, koty).
```

REPL służy do zadawania pytań!

```
lubi(jan, koty).
lubi(jan, Co).
lubi(X, Y).
```

Zarówno fakty, jak i pytania kończymy kropką.

# Zmienne

Prolog ma zmienne jednokrotnego przypisania:

```
?- X = a.
X = a.

?- X = a, X = b.
false.

?- X = Y, X = 2.
X = Y, Y = 2.
```

# Zmienne związane i wolne

Jeśli zmienna ma wartość to jest związana (ang. grounded).
Zmienne mogące przyjąć różne wartości to zmienne wolne.

```
?- X = 2, ground(X).
X = 2.

?- X = 2, ground(Y).
false.
```

# Dopasowanie do wzorca

Prolog wszędzie automatycznie dopasowuje się do wzorca.

```
?- X = (a, b).
X = (a, b).

?- (X, Y) = (a, b).
X = a,
Y = b.

?- [_|T] = [a,b,c].
T = [b, c].
```

_ to wieloznacznik
[X|Y] to podział listy na głowę i ogon (OCamlowe H::T)

# Zmienne a dane

W Prologu zmienna może być częścią danej.

```
?- (a, X) = (Y, b).
X = b,
Y = a.

?- length(X, 3).
X = [_G1106, _G1109, _G1112].

?- append([a, b, c], X, Y).
Y = [a, b, c | X].
```

# 2+2 to nie 4

Prolog nie dokonuje ewaluacji automatycznie.
2+2 to wyrażenie
4 to liczba
Wyrażenie nie jest liczbą.
Żeby coś policzyć, używamy is.

```
?- X = 2+2.
X = 2+2.

?- X is 2+2.
X = 4.
```

# Długość listy

## Myśl logicznie!

Długość listy pustej to 0.
Długość listy niepustej to długość ogona + 1.

inaczej

Długość niepustej listy to N + 1 wtedy, gdy długość ogona to N.

```
dl([], 0).

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

# Długość listy

## Składnia reguły

wniosek :- warunki.
Warunki oddzielamy przecinkiem (oraz)

```
dl([], 0).

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

# Długość listy

**Zagadka**

Co jest nie tak z tym kodem?

```
dl([], 0).

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

# Długość listy

**Zagadka**

Co jest nie tak z tym kodem?

Nie ma rekursji ogonowej.

```
dl([], 0).

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

# Wersja optymalna z rekursją ogonową

```
%dlugosc listy z rekursja ogonowa

dl(L, W) :-
        dl(L, 0, W).

%       V-- tu jest akumulator
dl([], X, X).
dl([_|T], X, W) :-
        X1 is X + 1,
        dl(T, X1, W).
```

# Czas na coś ciekawego

```
polacz([], X, X).
polacz([H|T], X, [H|W]) :-
        polacz(T, X, W).
```

Pytamy o relacje
A co Prolog na to?

```
?- polacz(X, Y, [a,b,c]).
```

# Rozwidlanie

W programach imperatywnych i funkcyjnych program „idzie” jedną ścieżką.
W Prologu tworzone są równoległe ścieżki.

# Ciekawe przykłady

# Listy różnicowe

# Definicja

Lista różnicowa to lista z odjętym ogonem.

```
[a, b, c, d] - [d]
[a, b, c | X] - X
```

1. Prolog nie ewaluuje, więc nie muszę definiować tego odejmowania
2. Mam daną listę [a, b, c], ale nie wiem, jaki ogon odejmę - zmienna częścią danej(!)
3. Te listy daje się (zwykle) łączyć w czasie stałym(!)

# Łączenie w czasie stałym

Lista różnicowa to lista z odjętym ogonem.

```
polacz(X–Y, Y–Z, X–Z).
```

Oto CAŁA implementacja.
Prolog cechuje bardzo zwięzły zapis.

# Programy „samouczące się”

# Ciąg Fibonacciego - z definicji

```
fib1(0, 0).
fib1(1, 1).
fib1(N, X) :-
        N > 1,
        N1 is N-1,
        N2 is N-2,
        fib1(N1, X1),
        fib1(N2, X2),
        X is X1+X2.
```

# Ciąg Fibonacciego - z definicji

```
fib1(0, 0).
fib1(1, 1).
fib1(N, X) :-
        N > 1,
        N1 is N-1,
        N2 is N-2,
        fib1(N1, X1),
        fib1(N2, X2),
        X is X1+X2.
```

Nieefektywne :(

# Ciąg Fibonacciego - zapamiętajmy wynik

```
:- dynamic fib/2. %bo chce dodawac nowe fakty

fib(0, 0).
fib(1, 1).
fib(N, X) :-
        N > 1,
        N1 is N-1,
        N2 is N-2,
        fib(N1, X1),
        fib(N2, X2),
        X is X1+X2,
        !,                          %jedyny sluszny wynik
        asserta(fib(N, X)). %zapamietaj go
```

# Zagadki

# Opis

*Zagadka pochodzi z „Jaki jest tytuł tej książki?” autorstwa Raymonda Smullyana*

Wszyscy mieszkańcy pewnej wyspy są albo rycerzami (ludźmi, którzy nigdy nie kłamią) albo łotrami (ludźmi, którzy kłamią zawsze).

Wędrując po tej wyspie spotykamy trzech tubylców:

Zadaliśmy osobie A pytanie czy jest łotrem czy rycerzem ale ten odpowiedział niewyraźnie i nie zrozumieliśmy jego odpowiedzi.

Pytamy się osobę B co odpowiedział A. B odpowiada nam, że A powiedział o sobie, że jest łotrem.

Słysząc to C mówi: ”Nie wierz B! To B jest łotrem!”.

Kim są B i C?

# Kod

*Kod autorstwa dra Przemysława Kobylańskiego*

```
rycerz(rycerz).
lotr(lotr).

powiedzial(rycerz, Zdanie) :-
        call(Zdanie).
powiedzial(lotr, Zdanie) :-
        \+ Zdanie. %\+ to negacja
```

# Pytanie

Pytamy się osobę B co odpowiedział A. B odpowiada nam, że A powiedział o sobie, że jest łotrem.
Słysząc to C mówi: "Nie wierz B! To B jest łotrem!".

```
?- powiedzial(B, powiedzial(A, lotr(A))),
powiedzial(C, lotr(B)).
```

```
B = lotr ,
C = rycerz .
```

# Więzy

# Czym są więzy

Za pomocą więzów można zapisywać ograniczenia, np.

- A jest pomiędzy 0 i 9
- A i B są różne

Przykłady podane za dokumentacją SWI-Prologu

# SEND + MORE = MONEY

$$\begin{array}{r} \mathrm{SEND} \\ +\ \mathrm{MORE} \\ \hline \mathrm{MONEY} \end{array}$$

# SEND + MORE = MONEY

Pytanie:

```
?- puzzle(X + Y = Z), label(X).

:- use_module(library(clpfd)).

puzzle([S,E,N,D] + [M,O,R,E] = [M,O,N,E,Y]) :-
        Vars = [S,E,N,D,M,O,R,Y],
        Vars ins 0..9,
        all_different(Vars),
                  S*1000 + E*100 + N*10 + D +
                  M*1000 + O*100 + R*10 + E #=
       M*10000 + O*1000 + N*100 + E*10 + Y,
       M #\= 0, S #\= 0.
```

# Sudoku

```
:- use_module(library(clpfd)).
sudoku(Rows) :-
        length(Rows, 9), maplist(length_(9), Rows),
        append(Rows, Vs), Vs ins 1..9,
        maplist(all_distinct, Rows),
        transpose(Rows, Columns),
        maplist(all_distinct, Columns),
        Rows = [A,B,C,D,E,F,G,H,I],
        blocks(A,B,C),blocks(D,E,F),blocks(G,H,I).
length_(L, Ls) :- length(Ls, L).
blocks([], [], []).
blocks([A,B,C|Bs1], [D,E,F|Bs2], [G,H,I|Bs3]) :-
        all_distinct([A,B,C,D,E,F,G,H,I]),
        blocks(Bs1, Bs2, Bs3).
```

# Sudoku - konkretna zagadka

```
:- [sudoku].

problem(1, [[_,_,_,_,_,_,_,_,_],
            [_,_,_,_,_,_,3,_,8,5],
            [_,_,1,_,2,_,_,_,_],
            [_,_,_,_,5,_,7,_,_,_],
            [_,_,4,_,_,_,1,_,_],
            [_,9,_,_,_,_,_,_,_],
            [5,_,_,_,_,_,_,_,7,3],
            [_,_,2,_,1,_,_,_,_],
            [_,_,_,_,4,_,_,_,9]]).
```

# Sudoku - rozwiązanie

```
?- problem(1, Rows), sudoku(Rows),
   maplist(writeln, Rows).
[9, 8, 7, 6, 5, 4, 3, 2, 1]
[2, 4, 6, 1, 7, 3, 9, 8, 5]
[3, 5, 1, 9, 2, 8, 7, 4, 6]
[1, 2, 8, 5, 3, 7, 6, 9, 4]
[6, 3, 4, 8, 9, 2, 1, 5, 7]
[7, 9, 5, 4, 6, 1, 8, 3, 2]
[5, 1, 9, 2, 8, 6, 4, 7, 3]
[4, 7, 2, 3, 1, 9, 5, 6, 8]
[8, 6, 3, 7, 4, 5, 2, 1, 9]
Rows = [[9, 8, 7, 6, 5, 4, 3, 2|...], ... , [...|..
```

# Funkcje anonimowe

# Funkcje anonimowe

```
substAB(\(X -> Y, P), X -> Y) :-
        call(P).
subst(F, D) :-
        copy_term(F, Klon),
        substAB(Klon, D).
```

# Surgunt

# Funkcje anonimowe

```
F = (X -> Y -> do(Z is X+Y) -> Z)

subst([2,3], F, R)

subst([1], F, Incr)
%Incr = (Y -> do(Z is 1+Y) -> Z)
```

# Funkcje anonimowe - kod 1/2

```
substAB(L, do(P) -> Cont, R) :-
        nonvar(P),
        call(P),
        substAB(L, Cont, R).
substAB([], X -> Y, X -> Y) :-
        var(X).
substAB([], X, X) :-
        X \= do(_),
        X \= (do(_) -> _).
substAB([X|Xs], X -> Cont, R) :-
        substAB(Xs, Cont, R).
```

# Funkcje anonimowe - kod 2/2

```
subst(V, E, R) :-
        copy_term(E, Ce),
        substAB(V, Ce, R).
```

# Predykaty anonimowe

```
%zamiast pisac
length_(X, L) :- length(L, X).
?- length(Rows, 3), maplist(length_(3), Rows).

%mozemy napisac:
%anonPred(kod, interfejs, argument)
anonPred(length(L, 3), [L], List)

maplist(anonPred(length(L, 3), [L]), Rows)

%zatem dla kazdego wiersza:
anonPred(length(L, 3), [L], Row)
```

# Predykaty anonimowe - kod

```
anonPred(Gp, Gd, Cx) :-
        copy_term([Gp|Gd], [Cp, Cx]),
        call(Cp).
anonPred(Gp, Gd, Cx, Cy) :-
        copy_term([Gp|Gd], [Cp, Cx, Cy]),
        call(Cp).
anonPred(Gp, Gd, Cx, Cy, Cz) :-
        copy_term([Gp|Gd], [Cp, Cx, Cy, Cz]),
        call(Cp).
```

# Składanie

```
%zamiast pisac
p1(Input, X1),
p2(Arg, X1, X2),
p3(X2, Output)

%piszemy
compose(Input, [p1, p2(Arg), p3], Output)
```

# Składanie - kod

```
compose(X, [], X).
compose(X, [P|Ps], Z) :-
        call(P, X, Y),
        compose(Y, Ps, Z).
```

# Więcej informacji


- E. Gatnar, K. Stąpor, *Prolog, język sztucznej inteligencji*, PLJ, Warszawa 1991, ISBN: 83-85190-63-5
- W. F. Clocksin, C. S. Mellish, *Prolog, programowanie*, Helion, Gliwice 2003, ISBN: 83-7197-918-5
- Jan Wielemaker, *SWI-Prolog Reference Manual 6.2.6*, `http://www.swi-prolog.org/download/stable/doc/SWI-Prolog-6.2.6.pdf`

Dziękuję za uwagę.
Czas na pytania.

matma6 (tech. Michał Gabor)
matma6.net
matma6@matma6.net