

# Programowanie w Logice

## Działanie Prologu

Przemysław Kobylański  
na podstawie [CM2003]

# Działanie Prologu

## Składnia

- ▶ Programy Prologu składają się z *termów*.
- ▶ Term to *stała*, *zmienna* lub *struktura* (term złożony).
- ▶ Term zapisuje się jako ciąg znaków.
- ▶ Znaki podzielono na cztery kategorie:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
a b c d e f g h i j k l m n o p q r s t u v w x y z  
0 1 2 3 4 5 6 7 8 9  
+ - * / \ ~ ^ < > : . ? @ # $ &
```

# Działanie Prologu

## Stałe

- ▶ Stałe nazywają konkretne obiekty lub konkretne relacje.
- ▶ Istnieją dwa rodzaje stałych: atomy i liczby.
- ▶ Przykłady atomów:  
`lubi maria jan ksiazka wino posiada moze_ukrasc`
- ▶ Specjalne symbole `?` i `:-` to również atomy.
- ▶ Atomy alfanumeryczne zaczynają się małą literą.
- ▶ Atomy zawierające symbole składają się tylko z symboli.
- ▶ Atomem jest również dowolny ciąg znaków ujęty w apostrofy.
- ▶ Kolejne przykłady atomów:  
`a void = 'Jan Kowalski' --> jan_kowalski ieh2304`
- ▶ To nie są poprawne atomy:  
`2304ieh jan-kowalski Void _alfa`
- ▶ Przykłady liczb:  
`-17 -2.67e2 0 1 99.9 512 8192 14765 67344 6.02e-23`

# Działanie Prologu

## Zmienne

- ▶ Zmienne mają postać atomów ale ich nazwy zaczynają się od wielkiej litery lub podkreślenia.
- ▶ Zmienną należy traktować jak zastępstwo obiektu, którego nie możemy w danej chwili nazwać.
- ▶ Gdy nazwa zmiennej nie ma znaczenia bo nie będziemy używać jej wartości, to można użyć *zmiennej anonimowej* zapisanej jako znak podkreślenia.

?- lubi(jan, \_).

# Działanie Prologu

## Struktury

- ▶ Struktury w standardzie Prologu nazywane są *termami złożonymi*.
- ▶ Struktura to pojedynczy obiekt złożony z zestawu innych obiektów, nazywanych *składnikami* struktury.

# Działanie Prologu

## Struktury

```
posiada(jan ,  ksiazka ).  
posiada(maria ,  ksiazka ).
```

Oboje posiadają tę samą książkę!

# Działanie Prologu

## Struktury

```
posiada(jan , wicherowe_wzgorza ).  
posiada(maria , moby_dick ).
```

Czy wicherowe wzgorza to tytuł książki czy np. imię królika?

# Działanie Prologu

## Struktury

Struktury zapisuje się podając *funktor* oraz jego *składniki*.

```
posiada(jan ,  ksiazka(wichrowe_wzgorza ,  bronte)).
```

W powyższym przykładzie funktorem jest *ksiazka* a jego dwoma składnikami *wichrowe\_wzgorza* i *bronte*.

```
posiada(jan ,  ksiazka(wichrowe_wzgorza ,  
                        autor(emily ,  bronte))).
```

```
?- posiada(jan, ksiazka(X, autor(_, bronte))).
```

```
posiada(jan ,  ksiazka(wichrowe_wzgorza ,  
                        autor(emily ,  bronte),  
                        3129)).
```



# Działanie Prologu

## Znaki

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
a b c d e f g h i j k l m n o p q r s t u v w x y z  
0 1 2 3 4 5 6 7 8 9  
! " # \$ % & ' ( ) = - ~ ^ | \ { } [ ] \_ ' @ + ; \* :  
< > , . ? /

# Działanie Prologu

## Operatory

- ▶ Zamiast funktorów przed składnikami wygodniej czasami pisać operatory między argumentami:
- ▶ Operatory nie powodują wykonania jakichkolwiek obliczeń.
- ▶ W Prologu  $3+4$  nie jest równoważne z 7, jest to inny zapis termu  $+(3, 4)$ .
- ▶ Aby poprawnie zinterpretować term zawierający operatory musimy znać ich: położenie, priorytety i łączność.

# Działanie Prologu

## Równość i unifikacja

?-  $X = Y$ .

- ▶ Prolog stara się *dopasować*  $X$  i  $Y$ .
- ▶ O unifikacji można myśleć jako o próbie *uczynienia*  $X$  i  $Y$  *równymi*.
- ▶ Predykat *równości* jest predykatem wbudowanym.
- ▶ Działa tak, jakby był zdefiniowany za pomocą faktu:

$X = X$ .

# Działanie Prologu

## Równość i unifikacja

```
?- jedzie(student, rower) = X.
```

```
X = jedzie(student, rower).
```

```
?- jedzie(student, rower) = jedzie(student, X).
```

```
X = rower.
```

```
?- a(b, C, d(e, F, g(h, i, J))) =
```

```
    a(B, c, d(E, f, g(H, i, j))).
```

```
C = c,
```

```
F = f,
```

```
J = j,
```

```
B = b,
```

```
E = e,
```

```
H = h.
```

# Działanie Prologu

## Arytmetyka

Poniższe operatory można zapisywać między dwoma wyrażeniami arytmetycznymi i porównywane są ich wartości liczbowe:

$X ::= Y$	$X$ i $Y$ są tą samą liczbą
$X \neq Y$	$X$ i $Y$ są różnymi liczbami
$X < Y$	$X$ jest mniejsze od $Y$
$X > Y$	$X$ jest większe od $Y$
$X \leq Y$	$X$ jest mniejsze lub równe $Y$
$X \geq Y$	$X$ jest większe lub równe $Y$

W porównywanych wyrażeniach wszystkie zmienne muszą mieć już wcześniej nadane wartości liczbowe.

# Działanie Prologu

## Arytmetyka

*% lata panowania walijskich władców*

wlada(rhodri, 844, 878).

wlada(anarawd, 878, 916).

wlada(hywel\_dda, 916, 950).

wlada(lago\_ap\_idwal, 950, 979).

wlada(hywel\_ap\_ieuaf, 979, 985).

wlada(cadwallon, 985, 986).

wlada(maredudd, 986, 999).

Kto był księciem w danym roku?

X był księciem w roku Y, jeśli:

X panował między rokiem A i B oraz

Y mieści się między A a B z tymi latami włącznie.

# Działanie Prologu

## Arytmetyka

```
ksiaze(X, Y) :-  
    wlada(X, A, B),  
    Y >= A,  
    Y <= B.
```

```
?- ksiaze(cadwallon, 986).
```

```
true.
```

```
?- ksiaze(rhodri, 1979).
```

```
false.
```

```
?- ksiaze(X, 900).
```

```
X = anarawd ;
```

```
false.
```

```
?- ksiaze(X, 979).
```

```
X = lago_ap_idwal ;
```

```
X = hywel_ao_ieuaf ;
```

```
false.
```

# Działanie Prologu

## Arytmetyka

*% populacja w milionach w roku 1976*

`lud(usa, 203).`

`lud(indie, 548).`

`lud(chiny, 800).`

`lud(brazylia, 108).`

*% obszar kraju w milionach mil kwadratowych*

`obszar(usa, 3).`

`obszar(indie, 1).`

`obszar(chiny, 4).`

`obszar(brazylia, 3).`

*% gestosc zaludnienia*

`gestosc(X, Y) :-`

`lud(X, P),`

`obszar(X, A),`

`Y is P/A.`



# Działanie Prologu

## Arytmetyka

- ▶ Prawym argumentem predykatu `is` jest term interpretowany jako wyrażenie arytmetyczne.
- ▶ Obliczona wartość wyrażenia arytmetycznego jest dopasowana do lewego argumentu.

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

```
X = 4.
```

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

```
true.
```

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

```
false.
```

# Działanie Prologu

## Arytmetyka

```
?- gestosc(chiny, X).
```

```
X = 200
```

```
?- gestosc(turcja, X).
```

```
false.
```

Wszystkie implementacje powinny obsługiwać następujące operatory arytmetyczne:

$X + Y$	suma $X$ i $Y$
$X - Y$	różnica $X$ i $Y$
$X * Y$	iloczyn $X$ i $Y$
$X / Y$	iloraz $X$ i $Y$
$X // Y$	całkowity iloraz $X$ przez $Y$
$X \bmod Y$	reszta z dzielenia $X$ przez $Y$

# Działanie Prologu

## Udane spełnienie koniunkcji celów

```
kobieta(maria).
```

```
rodzic(C, M, F) :- matka(C, M), ojciec(C, F).
```

```
matka(jan, anna).
```

```
matka(maria, anna).
```


```
ojciec(maria, ferdynand).
```

```
ojciec(jan, ferdynand).
```

```
?- kobieta(maria), rodzic(maria, M, F), ojciec(jan, M, F).
```

# Działanie Prologu

Udane spełnienie koniunkcji celów



kobieta(maria)

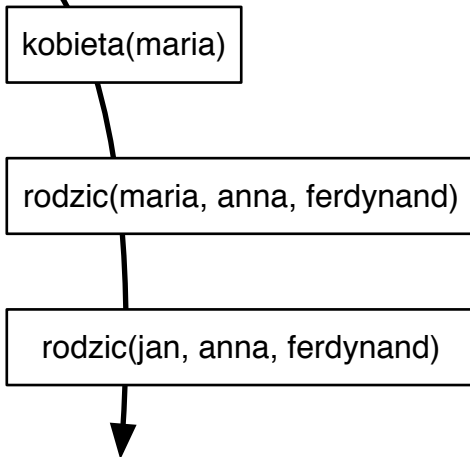
rodzic(maria, M, F)

rodzic(jan, M, F)

Sekwencja jeszcze nie spełnionych celów.

# Działanie Prologu

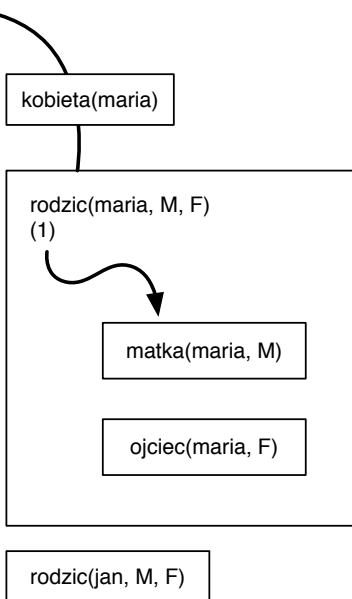
Udane spełnienie koniunkcji celów



Sekwencja spełnionych celów, zmienne zostały już ukonkretnione.

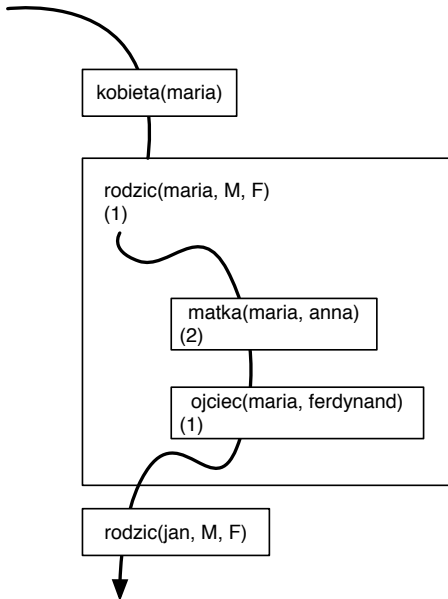
# Działanie Prologu

Udane spełnienie koniunkcji celów



# Działanie Prologu

Udane spełnienie koniunkcji celów



# Działanie Prologu

## Udane spełnienie koniunkcji celów

```
?- trace.
```

```
true.
```

```
[trace] ?- kobieta(maria), rodzic(maria, M, F), rodzic(jan, M, F).
```

```
Call: (8) kobieta(maria) ? creep
```

```
Exit: (8) kobieta(maria) ? creep
```

```
Call: (8) rodzic(maria, _G16787, _G16788) ? creep
```

```
Call: (9) matka(maria, _G16787) ? creep
```

```
Exit: (9) matka(maria, anna) ? creep
```

```
Call: (9) ojciec(maria, _G16788) ? creep
```

```
Exit: (9) ojciec(maria, ferdynand) ? creep
```

```
Exit: (8) rodzic(maria, anna, ferdynand) ? creep
```

```
Call: (8) rodzic(jan, anna, ferdynand) ? creep
```

```
Call: (9) matka(jan, anna) ? creep
```

```
Exit: (9) matka(jan, anna) ? creep
```

```
Call: (9) ojciec(jan, ferdynand) ? creep
```

```
Exit: (9) ojciec(jan, ferdynand) ? creep
```

```
Exit: (8) rodzic(jan, anna, ferdynand) ? creep
```

```
M = anna,
```

```
F = ferdynand.
```

```
[trace] ?- notrace.
```

```
true.
```



# Działanie Prologu

## Cele i nawracanie

Rozpatrzmy następujący prosty program:

$p(a).$

$p(b).$

$q(b).$

$q(c).$

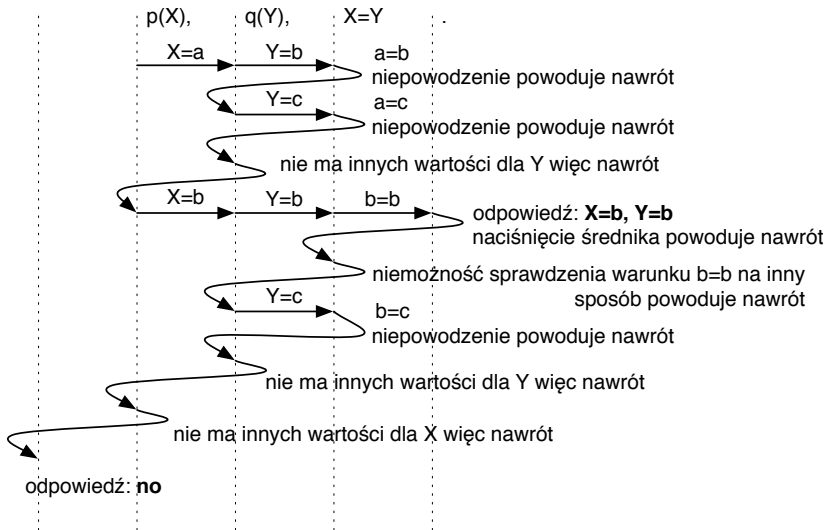
Zadajmy cel:

$?- p(X), q(Y), X = Y.$

Odpowiada on poszukiwaniu części wspólnej zbiorów  $\{a, b\}$  i  $\{b, c\}$ .

# Działanie Prologu

## Cele i nawracanie



(rys. własny)

# Działanie Prologu

## Cele i nawracanie

```
[trace] ?- p(X), q(Y), X = Y.  
  Call: (8) p(_G2260) ? creep  
  Exit: (8) p(a) ? creep  
  Call: (8) q(_G2262) ? creep  
  Exit: (8) q(b) ? creep  
  Call: (8) a=b ? creep  
  Fail: (8) a=b ? creep  
  Redo: (8) q(_G2262) ? creep  
  Exit: (8) q(c) ? creep  
  Call: (8) a=c ? creep  
  Fail: (8) a=c ? creep  
  Redo: (8) p(_G2260) ? creep  
  Exit: (8) p(b) ? creep  
  Call: (8) q(_G2262) ? creep  
  Exit: (8) q(b) ? creep  
  Call: (8) b=b ? creep  
  Exit: (8) b=b ? creep  
X = Y, Y = b .
```

# Działanie Prologu

## Cele i nawracanie

- ▶ Zwróć uwagę, że SWI-Prolog po znalezieniu pierwszej odpowiedzi nie kontynuuje poszukiwania następnej, bo wie, że jej nie ma.
- ▶ Ten sam cel można zapisać prościej i uzyskać odpowiedź jeszcze mniejszą liczbą nawrotów:

```
[trace] ?- p(X), q(X).  
  Call: (8) p(_G2260) ? creep  
  Exit: (8) p(a) ? creep  
  Call: (8) q(a) ? creep  
  Fail: (8) q(a) ? creep  
  Redo: (8) p(_G2260) ? creep  
  Exit: (8) p(b) ? creep  
  Call: (8) q(b) ? creep  
  Exit: (8) q(b) ? creep  
X = b.
```