



# Programowanie deklaratywne

Artur Michalski  
Informatyka II rok



## Plan wykładu

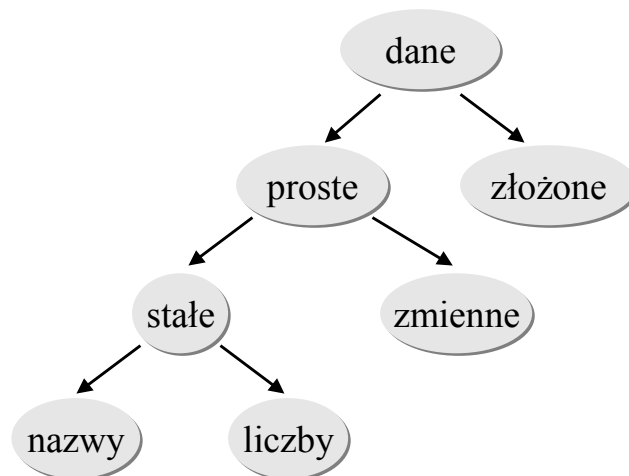
- Wprowadzenie do języka Prolog
- Budowa składniowa i interpretacja programów prologowych
- Listy, operatory i operacje arytmetyczne
- Złożone/abstrakcyjne struktury danych
- Sterowanie mechanizmem nawrotów
- Operacje wejścia/wyjścia w Prologu
- Predefiniowane procedury prologowe
- Styl i technika programowania w Prologu

## Budowa składniowa i znaczenie programów w Prologu

- Reprezentacja danych w Prologu
- Mechanizm dopasowywania (ang. *matching*)
- Deklaratywna interpretacja programu prologowego
- Proceduralna interpretacja programu prologowego
- Przykład interpretacji programu: *Problem małpy i banana*
- Porządek kazuł prologowych i celów
- Związki języka Prolog z logiką formalną
- Podsumowanie

## Reprezentacja danych w Prologu

Klasyfikacja rodzajów danych w Prologu



## Reprezentacja danych w Prologu

- **Nazwy** - składają się z:
  - dużych liter: A, B, ..., Z
  - małych liter: a, b, ..., z
  - cyfr: 0, 1, ..., 9
  - znaków specjalnych: +-\*/<>=:.&\_~
- i są ciągami:
  - liter, cyfr i znaków podkreślenia (zaczynającymi się od małej litery): anna, x12, y\_, \_z, a\_b
  - znaków specjalnych: <--->, ==>, ..
  - lub napisami (ang. string): 'Ania', 'co to', 'kto\_to'

## Reprezentacja danych w Prologu

- **Zmienne** - ciągi liter, cyfr i znaków podkreślenia zaczynające się od dużej litery lub podkreślenia
- Zmienna anonimowa - tylko znak podkreślenia

*Przykład*

```
ma_dziecko(X) :- rodzic(X,Y) .
```

ponieważ konkluzja klauzuli nie zależy od wartości zmiennej **Y**, można ją zastąpić przez zmienną nie posiadającą identyfikatora tzw. *zmienną anonimową*:

```
ma_dziecko(X) :- rodzic(X,_) .
```

## Reprezentacja danych w Prologu

Zmienna anonimowa c.d.

Każde wystąpienie zmiennej anonimowej w danej klauzuli reprezentuje inną zmienną

*Przykład*

**ktos\_ma\_dziecko :- rodzic(,\_).**

jest równoważne:

**ktos\_ma\_dziecko :- rodzic(X,Y).**

a nie:

**ktos\_ma\_dziecko :- rodzic(X,X).**

## Reprezentacja danych w Prologu

Jeśli zmienna anonimowa zostanie użyta w pytaniu, jej wartość *nie zostanie* wyświetlona:

*Przykład*

*Jakie osoby mają dzieci (bez podawania samych dzieci)?*

Zapis w Prologu:

**?- rodzic(X,\_).**

Odpowiedź:

**X = torek;**

...

## Reprezentacja danych w Prologu

Zasięg leksykalny zmiennej (identyfikatora) ograniczony jest tylko do jednej klauzuli:

*Przykład*

```
ma_dziecko(X) :- rodzic(X,_).  
dziecko(X,Y) :- rodzic(Y,X).  
zmienna X oznacza inną zmienną w każdej z  
klauzul, czyli:  
ma_dziecko(X1) :- rodzic(X1,_).  
dziecko(X2,Y) :- rodzic(Y,X2).
```

## Reprezentacja danych w Prologu

- Dane złożone (strukturalne) - dane, które składają się z kilku elementów, tworzących jeden obiekt; w szczególności składowymi *struktury* mogą być również dane złożone
- Dane złożone tworzone są za pomocą *funktora* i *argumentów* traktowanych razem jako jeden obiekt programu

*Przykład*

Zapis daty w Prologu

```
date(1,october,2000)
```

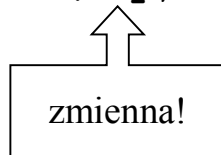
## Reprezentacja danych w Prologu

Argumentami danych złożonych (strukturalnych) mogą być zarówno *stałe*, jak i *zmienne* prologowe

*Przykład*

Dowolony dzień marca 2000 roku:

**date (Day , march , 2000)**

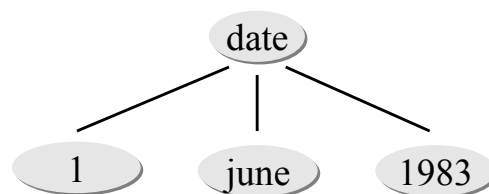


## Reprezentacja danych w Prologu

Wszystkie dane strukturalne w Prologu mogą być reprezentowane za pomocą *drzew*, którego korzeniem jest funktor a węzłami potomnymi - argumenty

*Przykład*

**date (1 , june , 1983)**



## Reprezentacja danych w Prologu

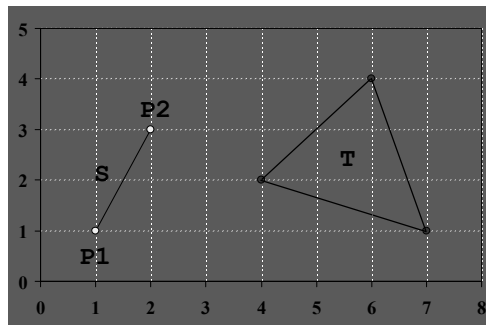
Przykład reprezentacji danych strukturalnych

**P1=point(1,1)**

**P2=point(2,3)**

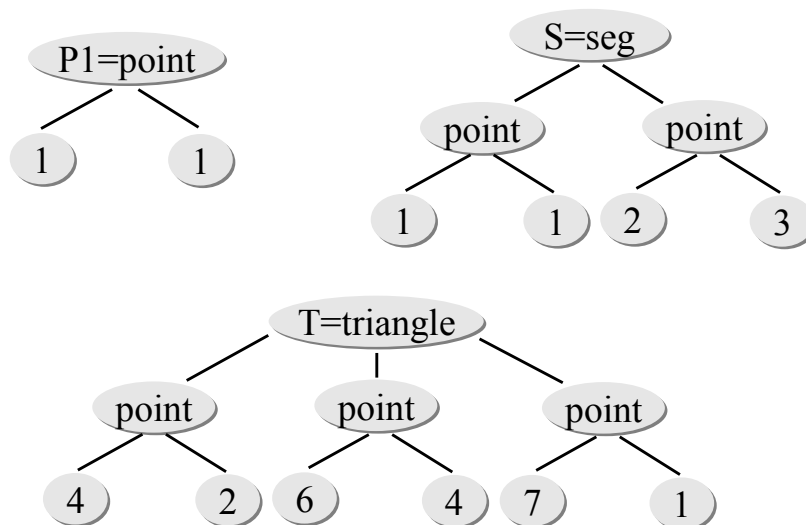
**S=seg(P1,P2)=seg(point(1,1),point(2,3))**

**T=triangle(point(4,2),point(6,4),point(7,1))**



## Reprezentacja danych w Prologu

Przykład danych strukturalnych



## Reprezentacja danych w Prologu

Dane strukturalne określone są zarówno przez nazwę funktora, jak i liczbę argumentów tzw. *arność*.

Oznacza to, iż dwa funktory o tej samej nazwie, lecz innej arności są różnymi obiektami.

### Przykład

Punkt w przestrzeni 2-wymiarowej:

**point(7,3)**

funktor i  
2 argumenty

punkt w przestrzeni 3-wymiarowej:

**point(7,3,1)**

funktor i  
3 argumenty

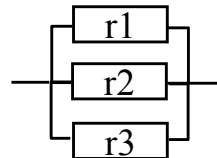
## Reprezentacja danych w Prologu

Inny przykład zastosowania struktur

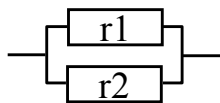
**seq(r1, r2)**



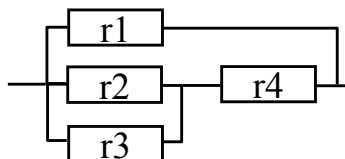
**par(r1, par(r2, r3))**



**par(r1, r2)**



**par(r1, seq(par(r2, r3), r4))**







## Reprezentacja danych w Prologu

Pojęcie formalne termu

Do termów zaliczamy:

- stałe i zmienne prologowe
- struktury reprezentowane za pomocą funktorów o dowolnej arności, których argumentami są inne termy



## Mechanizm dopasowywania (ang. *matching*)

O dopasowaniu dwóch termów mówimy wtedy, gdy:

- termy są *identyczne*,
- zmienne zawarte w obu termach mogą przyjąć wartości, dla których termy te staną się identyczne

*Przykład*

Termy: **date (D ,M ,1993)** i **date (D1 ,may ,Y1)**  
można dopasować, gdy:

**D** przyjmie wartość **D1**

**M** przyjmie wartość **may**

**Y1** przyjmie wartość **1993**

## Mechanizm dopasowywania (ang. *matching*)

Proces dopasowywania może zakończyć się *porażką*, kiedy nie istnieje żadne możliwe podstawienie zmiennych, które spowodowałoby, iż termy będą identyczne.

### *Przykład*

Termów: **date (D,M,1993)** i **date (D1,M1,1644)** nie można dopasować, bo: **1993** i **1644**, to różne stałe i podobnie **date (X,Y,Z)** i **point (X,Y,Z)**, bo: **date** i **point**, to różne funktory.

## Mechanizm dopasowywania (ang. *matching*)

Operator dopasowania w Prologu oznaczany jest znakiem '='.

### *Przykład*

?- **date (D,M,1993)=date (D1,may,Y1)** .

można dopasować, gdy:

**D=1, D1=1, M=may, Y1=1993** albo

**D=third, D1=third, M=may, Y1=1993** albo

...

lecz dopasowania te nie są *najogólniejsze*, w przeciwieństwie do:

**D=D1, M=may, Y1=1993**

## Mechanizm dopasowywania (ang. *matching*)

Mechanizm dopasowania w Prologu zawsze wyznacza *najogólniejsze możliwe dopasowanie*, tzn. takie dopasowanie, które w najmniejszym możliwym stopniu ogranicza zbiór wartości zmiennej.

*Przykład*

?- **date (D,M,1993)=date (D1,may,Y1) ,  
date (D,M,1993)=date (15,M,Y) .**

najpierw:	<b>D=D1</b>	ostateczny	<b>D=15</b>
	<b>M=may</b>	rezultat:	<b>D1=15</b>
	<b>Y1=1983</b>		<b>M=may</b>
			<b>Y1=1993</b>
			<b>Y=1993</b>

## Mechanizm dopasowywania (ang. *matching*)

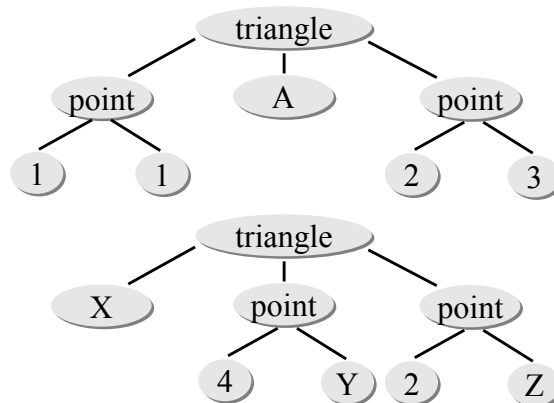
Przebieg procesu dopasowania w Prologu:

- Jeżeli dwa termy S i T, są stałymi, to dopasowanie zachodzi, gdy są identyczne,
- Jeżeli S jest zmienną a T dowolnym termem, to dopasowanie zachodzi, gdy S przypiszemy wartość T; podobnie w sytuacji odwrotnej, kiedy T jest zmienną, a S dowolnym termem, to dopasowanie zajdzie, gdy T przypiszemy wartość S
- Jeżeli S i T są obiektami złożonymi, to dopasowanie zachodzi, gdy:
  - S i T mają ten sam funktor, i
  - zachodzi dopasowanie pomiędzy wszystkimi ich składowymi

Ostateczny rezultat dopasowania zależy wtedy od dopasowań poszczególnych składowych.

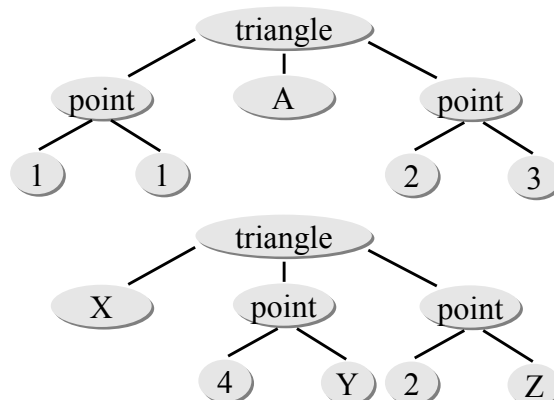
## Mechanizm dopasowywania (ang. *matching*)

?- triangle(point(1,1),A,point(2,3))=  
triangle(X,point(4,Y),point(2,Z)).



## Mechanizm dopasowywania (ang. *matching*)

Proces dopasowania przebiega od korzenia do liści struktur drzewiastych, reprezentujących dopasowywane terminy.



## Mechanizm dopasowywania (ang. *matching*)

Proces dopasowania przebiega od korzenia do liści struktur drzewiastych, reprezentujących dopasowywane termy.

Poszczególne etapy procesu dopasowania:

```
triangle=triangle,  
point(1,1)=X,  
A=point(4,Y),  
point(2,3)=point(2,Z).
```

Sukces dopasowania końcowego wynika z udanych dopasowań poszczególnych składowych termów:

```
X=point(1,1)  
A=point(4,Y)  
Z=3
```

## Mechanizm dopasowywania (ang. *matching*)

Proces dopasowania może być wykorzystany sam w sobie do przeprowadzenia pewnych „obliczeń”.

*Przykład*

*Sprawdzenie czy odcinek jest pionowy, czy też poziomy?*

Zapis własności w Prologu:

```
vertical(seg(point(X,Y1),point(X,Y2))).  
horizontal(seg(point(X1,Y),point(X2,Y))).
```

Zapytanie w Prologu:

```
?- vertical(seg(point(1,1),point(1,2))).
```

**Yes**

```
?- horizontal(seg(point(1,1),point(2,Y))).
```

**Y=1**

## Mechanizm dopasowywania (ang. *matching*)

ciąg dalszy zapytań:

*Czy istnieje odcinek pionowy zaczynający się w punkcie (2,3)?*

?- **vertical**(**seg**(**point**(2,3),**P**)).

**P=point**(2,**Y**)

inne zapytanie:

*Czy istnieje odcinek zarówno pionowy, jak i poziomy?*

?- **vertical**(**S**),**horizontal**(**S**).

**S=seg**(**point**(**X**,**Y**),**point**(**X**,**Y**))

odpowiedź jest twierdząca, gdyż każdy odcinek „zdegenerowany” do punktu jest zarówno pionowy, jak i poziomy.

## Deklaratywna i proceduralne interpretacja programu prologowego

Rozważmy klauzulę **P** postaci: **P**:-**Q**,**R**.

- Interpretacja deklaratywna klauzuli **P**:

***P** jest prawdziwe wtedy i tylko wtedy, gdy **Q** i **R** są prawdziwe.*

albo:

*Z **Q** i **R** wynika **P**.*

- Interpretacja proceduralna klauzuli **P**:

*Żeby osiągnąć cel **P**, najpierw musisz osiągnąć podcel **Q**, a potem podcel **R**.*

albo:

*Żeby spełnić **P** najpierw spełnij **Q** a potem **R**.*

Różnica: interpretacja proceduralna określa nie tylko logiczny związek między nagłówkiem reguły i jej ciałem, ale również porządek w jakim mają być osiągnane podcele.

## Deklaratywna interpretacja programu prologowego

Deklaratywna interpretacja programu prologowego określa czy dany cel jest spełniony, jeśli tak, to dla jakich wartości zmiennych.

**Instancja klauzuli** to taka klauzula, w której za każdą zmienną podstawiono jakiś term.

*Przykład*

Klauzula:

```
ma_dziecko(X) :- rodzic(X,Y) .
```

i jej przykładowe instancje:

```
ma_dziecko/piotr) :- rodzic/piotr,Y) .
```

```
ma_dziecko/jan) :- rodzic/jan,mala(ewa)) .
```

## Deklaratywna interpretacja programu prologowego

Formalna definicja interpretacji deklaratywnej

Dla danego programu prologowego i celu  $G$ :

Cel  $G$  jest prawdziwy (spełniony albo wynika logicznie z programu) wtedy i tylko wtedy, gdy:

- istnieje w programie klauzula  $C$  taka, że:
- istnieje instancja  $J$  klauzuli  $C$  taka, że:
  - nagłówek instancji  $J$  jest identyczny z  $G$ , i
  - wszystkie podcele w ciele instancji  $J$  są prawdziwe (spełnione)

## Deklaratywna interpretacja programu prologowego

W przypadku celów złożonych, lista celów jest spełniona, gdy wszystkie jej cele są spełnione dla tych samych podstawień zmiennych (koniunkcja celów).

*Koniunkcja celów* - lista celów oddzielonych przecinkiem - wszystkie cele muszą być spełnione.

*Dysjunkcja celów* - lista celów oddzielonych średnikiem - wystarczy, że jeden z celów zostanie spełniony.

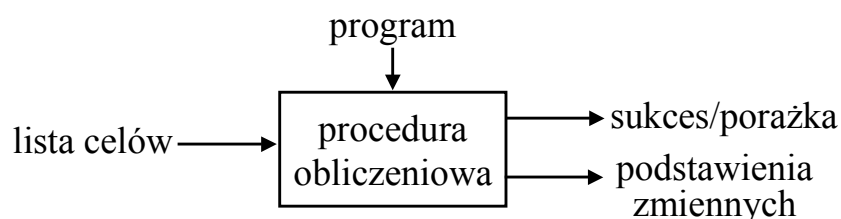
Klauzula postaci:  $P : -Q ; R.$   
jest równoważna dwóm klauzulom:  $P : -Q.$  i  $P : -R.$

Koniunkcja klauzul ma wyższy priorytet niż dysjunkcja klauzul.

## Proceduralna interpretacja programu prologowego

Interpretacja proceduralna programu prologowego określa *jak* Prolog odpowiada na pytania (w *jaki sposób* spełnia cele).

Interpretacja proceduralna oznacza wykonanie procedury obliczeniowej, która doprowadzi do spełnienia listy celów z uwzględnieniem danego programu prologowego.





## Proceduralna interpretacja programu prologowego

Przykład interpretacji proceduralnej

Zbiór klauzul:

<b>big(bear) .</b>	% klauzula 1
<b>big(elephant) .</b>	% klauzula 2
<b>small(cat) .</b>	% klauzula 3
<b>brown(bear) .</b>	% klauzula 4
<b>black(cat) .</b>	% klauzula 5
<b>gray(elephant) .</b>	% klauzula 6
<b>dark(X) :- black(X) .</b>	% klauzula 7
<b>dark(X) :- brown(X) .</b>	% klauzula 8

Pytanie (cel): **dark(X) , big(X) .**

## Proceduralna interpretacja programu prologowego

Przebieg procedury spełniania celu:

- 1) Początkowa lista celów: **dark(X) , big(X) .**
- 2) Analiza całego programu od początku do końca i poszukiwanie klauzuli, której nagłówek pasuje do pierwszego celu **dark(X)** . Klauzula 7:  
**dark(X) :- black(X) .**  
Zamiana pierwszego celu na zainicjowaną podstawieniami zmiennych treść klauzuli 7:  
**black(X) , big(X) .**
- 3) Analiza programu w poszukiwaniu dopasowania dla celu **black(X)** . Dopasowanie do klauzuli 5:  
**black(cat)** . Klauzula nie ma treści, więc lista celów po zastosowaniu podstawień zmiennych ma postać:  
**big(cat) .**

## Proceduralna interpretacja programu prologowego

Przebieg procedury spełniania celu (ciąg dalszy):

- 4) Analiza całego programu w poszukiwaniu klauzuli, której nagłówek pasuje do: **big(cat)**. Brak takiej klauzuli. Nawrót do kroku 3) i anulowanie podstawienia: **X=cat**. Lista celów ma postać:  
**black(X),big(X)**.  
Kontynuacja przeszukiwania programu poniżej klauzuli 5 zakończona porażką. Nawrót do kroku 2) i przeszukiwanie poniżej klauzuli 7. Dopasowanie dla klauzuli 8:  
**dark(X) :- brown(X)**.  
Nowa lista celów: **brown(X),big(X)**.

## Proceduralna interpretacja programu prologowego

Przebieg procedury spełniania celu (ciąg dalszy):

- 5) Analiza całego programu w poszukiwaniu klauzuli, której nagłówek pasuje do **brown(X)**.  
Dopasowanie do **brown(bear)**. Klauzula pozbawiona treści, więc lista celów zmniejsza się do: **big(bear)**.
- 6) Poszukiwanie dopasowania do klauzuli **big(bear)** zakończone sukcesem. Klauzula nie ma treści, więc lista celów zmniejsza się do listy pustej. Cel główny spełniony dla podstawienia zmiennych:  
**X=bear**.

## Proceduralna interpretacja programu prologowego

Formalna definicja interpretacji proceduralnej

Dla danego programu prologowego i celu  $G1, G2, \dots, Gm$ :

- 1) jeśli lista celów jest pusta, to koniec z *sukcesem*
- 2) jeśli lista celów nie jest pusta, to kontynuuj operację ANALIZA.
- 3) ANALIZA: Przeszukiwanie całego programu (od początku do końca) do pierwszej klauzuli  $C$ , której nagłówek pasuje do pierwszego celu  $G1$ . Jeśli nie ma takiej klauzuli, to koniec z *porażką*. Jeśli taka klauzula jest i ma postać:  
 $H :- B1, B2, \dots, Bn$ .  
to zmiana nazw zmiennych w  $C$  na unikalne zmienne (nowy wariant tej klauzuli:  $C'$ ), różne od zmiennych zawartych w liście  $G1, G2, \dots, Gm$ :  
 $H' :- B1', B2', \dots, Bn'$ . wariant  $C$  oznaczony jako  $C'$

## Proceduralna interpretacja programu prologowego

Formalna definicja interpretacji proceduralnej

ciąg dalszy ANALIZY:

Dopasowanie  $G1$  do  $H'$  z rezultatem w postaci zbioru podstawień  $S$ .

W liście celu głównego  $G1, G2, \dots, Gm$  zamieniamy  $G1$  na listę  $B1', B2', \dots, Bn'$  i otrzymujemy listę celów:

$B1', B2', \dots, Bn', G2, \dots, Gm$

(jeśli  $C$  jest faktem wtedy  $n=0$  i nowa lista celów jest krótsza i niż lista pierwotna; zmniejszanie się tej listy doprowadzi w końcu do listy pustej i tym samym osiągnięcia celu głównego). Zamieniamy zmienne na nowej liście celów zgodnie z podstawieniami ze zbioru  $S$  i otrzymujemy nową listę:

$B1'', B2'', \dots, Bn'', G2', \dots, Gm'$

## Proceduralna interpretacja programu prologowego

Formalna definicja interpretacji proceduralnej

- 4) Wykonujemy rekurencyjnie całą powyższą procedurę dla nowej listy celów  $B1'', B2'', \dots, Bn'', G2', \dots, Gm'$ . Jeśli realizacja tego celu zakończy się sukcesem, to realizacja celu pierwotnego  $G1, G2, \dots, Gm$  również kończy się sukcesem. Jeśli lista celów  $B1'', B2'', \dots, Bn'', G2', \dots, Gm'$  zakończy się porażką, to porzucamy dalsze przetwarzanie tej listy i powracamy do kontynuowania operacji ANALIZA dla pozostałej części programu prologowego. Przeglądamy program, poczynając od klauzuli następnej po klauzuli  $C$  ( $C$  jest klauzulą, której użyliśmy jako ostatniej) i próbujemy wykorzystać inną pasującą klauzulę do spełnienia celu  $G1$ .

## Proceduralna interpretacja programu prologowego

Formalna definicja interpretacji proceduralnej

Uwagi do definicji:

1. Procedura nie określa w jaki sposób otrzymywany jest ostateczny zbiór podstawień zmiennych  $S$ . Zbiór podstawień, który prowadzi do spełnienia pierwszego celu z listy podlega najczęściej dalszym uściśleniom w wyniku realizacji kolejnych celów.
2. Kiedy rekurencyjne wywołanie procedury prowadzi do porażki dokonujemy *nawrotu* do tego miejsca w programie, w którym wybrano złą klauzulę (klauzulę  $C$ ), porzucając wszystkie rezultaty jej przetwarzania, włącznie z dokonanymi podstawieniami zmiennych. Taka realizacja celów gwarantuje systematyczną analizę wszystkich alternatywnych ścieżek przetwarzania prowadzących do spełnienia celu lub kończy się stwierdzeniem (po sprawdzeniu ich wszystkich), że cel nie jest spełniony.

## Przykład interpretacji programu: *Problem małpy i banana*

Sformułowanie problemu:

Małpa przy drzwiach wejściowych do pomieszczenia.

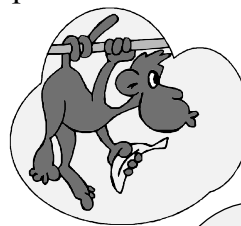
Małpa stoi na podłodze.

Pudło stoi pod oknem.

Małpa nie ma banana.

Banan wisi na środku sufitu.

Małpa nie może dosięgnąć banana.



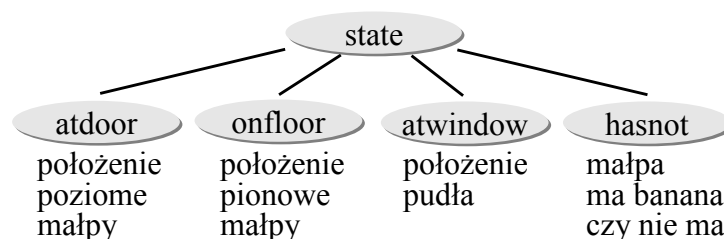
Pytanie: *Czy małpa może zjeść banana?*



## Przykład interpretacji programu: *Problem małpy i banana*

Reprezentacja stanu problemu:

Aktualna sytuacja w naszym „świecie małpy” reprezentowana jest jako jeden z możliwych stanów w całej przestrzeni stanów, opisujących wszystkie możliwe położenia poszczególnych obiektów.



## Przykład interpretacji programu: *Problem małpy i banana*

Reprezentacja stanu problemu w Prologu:

Term postaci: **state(<phm>, <pwm>, <pp>, <mnm>)**

gdzie: <phm> - położenie horyzontalne małpy

<pwm> - położenie wertykalne małpy

<pp> - położenie pudła

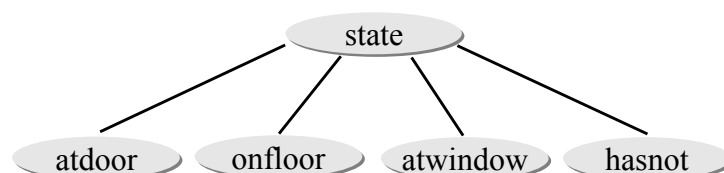
<mnm> - małpa *ma* albo *nie ma* banana

Stan docelowy:

**state(\_, \_, \_, has)**

Stan początkowy:

**state(atdoor, onfloor, atwindow, hasnot)**



## Przykład interpretacji programu: *Problem małpy i banana*

Możliwe ruchy małpy:

- 1) chwytanie banana
- 2) wspinanie się na pudło
- 3) przesuwanie pudła
- 4) poruszanie po pomieszczeniu

Nie wszystkie ruchy są dopuszczalne w każdym stanie.

Każdy ruch wymaga spełnienia określonych warunków początkowych, niezbędnych do jego realizacji.

Reprezentacja ruchów w Prologu:

Term postaci: **move(<stan1>, <ruch>, <stan2>)**

gdzie: <stan1> - sytuacja przed ruchem małpy

<ruch> - ruch małpy

<stan2> - sytuacja po ruchu małpy

## Przykład interpretacji programu: *Problem małpy i banana*

Przykłady ruchów w Prologu:

Ruch „chwytanie banana”:

```
move ( state (middle, onbox, middle, hasnot) ,  
        grasp,  
        state (middle, onbox, middle, has) ) .
```

*Małpa może złapać banana tylko wtedy, gdy stoi na pudle, znajdującym się na środku pokoju i nie ma jeszcze banana.*

Klauzuli definiującej ruch nie musi odpowiadać tylko jeden z ruchów możliwych do wykonania w modelowanym świecie, może to być cały zbiór ruchów.

## Przykład interpretacji programu: *Problem małpy i banana*

Przykłady ruchów w Prologu c.d.:

Ruch „poruszanie po pomieszczeniu” :

```
move ( state (Pos1, onfloor, Box, Has) ,  
        walk (Pos1, Pos2) ,  
        state (Pos2, onfloor, Box, Has) ) .
```

*Małpa może zmienić swoje położenie (w poziomie) niezależnie od położenia pudła i faktu schwytania banana:*

- *Zmiana położenia z miejsca **Pos1** na miejsce **Pos2**.*
- *Małpa znajduje się na podłodze zarówno przed, jak i po wykonaniu ruchu.*
- *Pudło pozostaje w tym samym położeniu **Box**.*
- *Stan posiadania małpy **Has** również nie ulega zmianie.*

Wykorzystanie zmiennych uogólniło definicję na wszystkie przypadki, w których możliwe jest poruszanie się po pomieszczeniu.

## Przykład interpretacji programu: *Problem małpy i banana*

Sformułowanie celu w Prologu:

Klauzula **canget** (<stan>)

gdzie: <stan> - stan docelowy

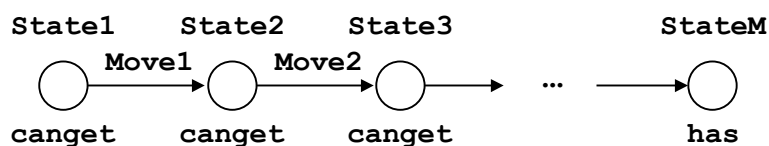
Możliwe scenariusze:

- W dowolnym stanie, w którym małpa ma już banana klauzula **canget** powinna być zawsze prawdziwa - nie ma potrzeby wykonywania wtedy żadnych ruchów:  
**canget** (state (\_,\_,\_,has)) .
- W każdym innym przypadku należy wykonać co najmniej jeden ruch - małpa może schwycić banana w stanie **State1**, o ile istnieje ruch ze stanu **State1** do takiego stanu **State2**, w którym małpa jest w stanie osiągnąć banana:  
**canget** (State1) :- move (State1,Move,State2) ,  
canget (State2) .

## Przykład interpretacji programu: *Problem małpy i banana*

Sformułowanie celu w Prologu c.d.:

Klauzula **canget** ma charakter rekurencyjny.



Postać zapytania dla naszego problemu:

?- **canget** (state (atdoor, onfloor, atwindow, hasnot)) .



## Przykład interpretacji programu: *Problem małpy i banana*

Kompletny program:

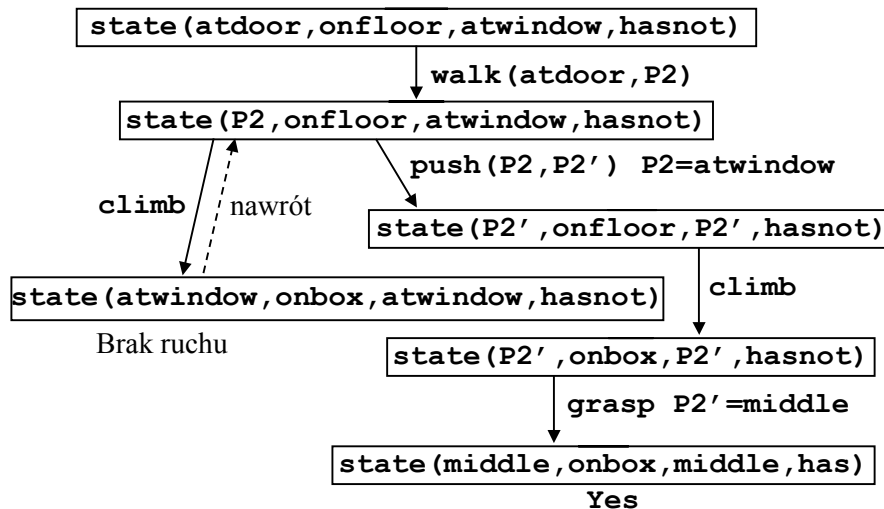
```

move ( state (middle,onbox,middle,hasnot) ,
      grasp,                                     % chwytanie
      state (middle,onbox,middle,has) ) .
move ( state (P,onfloor,P,H) ,
      climb,                                     % wspinaczka
      state (P,onbox,P,H) ) .
move ( state (P1,onfloor,P1,H) ,
      push (P1,P2) ,                             % przesuwanie
      state (P2,onfloor,P2,H) ) .
move ( state (P1,onfloor,B,H) ,
      walk (P1,P2) ,                             % chodzenie
      state (P2,onfloor,B,H) ) .

canget (state (_,_,_,has)) .                    % już ma banana!
canget (State1) :-                             % ruch, żeby mieć
    move (State1,Move,State2) ,
    canget (State2) .
  
```

## Przykład interpretacji programu: *Problem małpy i banana*

Interpretacja proceduralna programu dla problemu *Małpy i banana*:



Przykład interpretacji programu:  
*Problem małpy i banana*



Porządek kauzyl prologowych i  
celów

Porządek klauzul w programie prologowym ma kluczowe znaczenie dla efektywności i skuteczności programu.

*Przykład*

Klauzula poprawna deklaratywnie

**p :- p.**

Pytanie:

**?- p.**

Efekt: nieskończona pętla!!!

Program poprawny w sensie interpretacji deklaratywnej może być bezużyteczny ze względu na interpretację proceduralną.

## Porządek klauzul prologowych i celów

Pętle mogą być efektem niewłaściwej kolejności reguł lub celów. Proceduralna interpretacja programu prologowego powoduje, że preferowane są reguły i/lub cel zdefiniowane wcześniej w programie.

### *Przykład*

W programie dla problemu małpy i banana kolejność klauzul jest następująca:

```
move(...,grasp,...) .  
move(...,climb,...) .  
move(...,push(P1,P2),... ) .  
move(..., walk(P1,P2),... ) .
```

Oznacza to, iż preferowane jest „chwytanie banana”, potem „wspinanie na pudło”, następnie „pchanie pudła” i na końcu „poruszanie po pomieszczeniu”.

## Porządek klauzul prologowych i celów

Wpływ porządku klauzul na realizację celu

### *Przykład c.d.*

Cel główny:

```
?- canget(state(atdoor,onfloor,atwindow,hasnot)) .
```

Przyjmijmy następującą kolejność klauzul:

```
move(..., walk(P1,P2),... ) .          <=== zmiana!!!  
move(...,grasp,...) .  
move(...,climb,...) .  
move(...,push(P1,P2),... ) .
```

Preferowane jest „poruszanie po pomieszczeniu” nad inne ruchy.

## Porządek klauzul prologowych i celów

*ciąg dalszy przykładu*

Kolejność realizowanych celów:

1) **canget**(state(atdoor,onfloor,atwindow,hasnot))

dopasowanie do klauzuli **canget** (wersji drugiej) i nowy cel:

2) **move**(state(atdoor,onfloor,atwindow,hasnot),M',S2'),  
**canget**(S2')

pierwszy możliwy do zastosowania ruch **walk**(atdoor,P2') daje nam cel:

3) **canget**(state(P2',onfloor,atwindow,hasnot))

dopasowanie do klauzuli **canget** i kolejny podcel:

4) **move**(state(P2',onfloor,atwindow,hasnot),M'',S2''),  
**canget**(S2'')

pierwszy możliwy do zastosowania ruch **walk**(P2',P2'') daje nam cel:

5) **canget**(state(P2'',onfloor,atwindow,hasnot)),

z dopasowaniem S2''=state(P2'',onfloor,atwindow,hasnot)

kolejne zastosowanie klauzuli **canget** prowadzi do celu:

6) **move**(state(P2'',onfloor,atwindow,hasnot),M''',S2'''),  
**canget**(S2''')

w którym pierwszym zostanie zastosowany ruch **walk**(P2'',P2''')

...

## Porządek klauzul prologowych i celów

Wpływ porządku klauzul na realizację celu programu.

Wnioski:

Program prologowy może nie znaleźć rozwiązania, nawet jeżeli rozwiązanie to istnieje (pętle nieskończone!).

Program prologowy może być poprawny w sensie deklaratywnym, ale błędny w sensie proceduralnym.

Istnieją ogólne metody eliminacji nieskończonych pętli (bezproduktywnych dróg poszukiwania rozwiązania), które można zastosować w programie prologowym.

## Porządek klauzul prologowych i celów

Wpływ porządku klauzul na realizację celu

*Przykład*

```
przodek (P,D) :- rodzic (P,D) .                %prb
przodek (P,D) :- rodzic (P,R) ,przodek (R,D) .    %prp
```

Możliwe modyfikacje porządku:

- zmiana kolejności klauzul *prb* i *prp* w programie
- zmiana kolejności podcelów w ciele klauzuli *prp*

## Porządek klauzul prologowych i celów

*Przykład c.d.*

Efekt: cztery warianty tego samego programu - identyczne w sensie deklaratywnym, lecz różne w sensie proceduralnym.

```
przodek1 (P,D) :- rodzic (P,D) .
przodek1 (P,D) :- rodzic (P,R) ,przodek1 (R,D) .

przodek2 (P,D) :- rodzic (P,R) ,przodek2 (R,D) .
przodek2 (P,D) :- rodzic (P,D) .

przodek3 (P,D) :- rodzic (P,D) .
przodek3 (P,D) :- przodek3 (P,R) ,rodzic (R,D) .

przodek4 (P,D) :- przodek4 (P,R) ,rodzic (R,D) .
przodek4 (P,D) :- rodzic (P,D) .
```

## Porządek kauzul prologowych i celów

*Przykład c.d.*

?-przodek1(tomek,ola) .

Yes

?-przodek2(tomek,ola) .

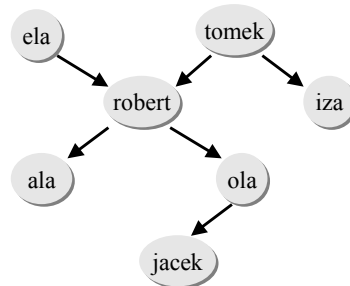
Yes

?-przodek3(tomek,ola) .

Yes

?-przodek4(tomek,ola) .

ERROR: out of local stack



## Porządek kauzul prologowych i celów

*Przykład c.d.*

przodek1(tomek,ola)

rodzic(tomek,ola)

No

rodzic(tomek,Y) ,  
przodek1(Y,ola)

wg faktu

rodzic(tomek,robert)

Y = robert

przodek1(robert,ola)

rodzic(robert,ola)

Yes

## Porządek kauzul prologowych i celów

*Przykład c.d.*

przodek2 (tomek , ola)



???

## Porządek kauzul prologowych i celów

*Przykład c.d.*

przodek3 (tomek , ola)



rodzic (tomek , ola)

No



przodek3 (tomek , Y) ,  
rodzic (Y , ola)



rodzic (tomek , Y) ,  
rodzic (Y , ola)



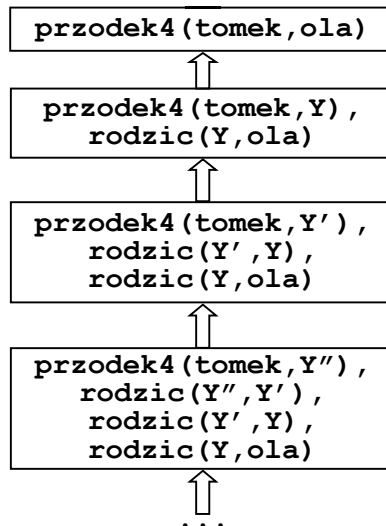
Y = robert

rodzic (robert , ola)

Yes

## Porządek kazułów prologowych i celów

*Przykład c.d.*



## Porządek kazułów prologowych i celów

*Przykład c.d.*

Podsumowanie

**przodek1** - najlepsza wersja

**przodek2** - wersja o najgorszej efektywności, ale skuteczna

**przodek3** - nie zawsze skuteczna, np. **?-przodek3(iza, jacek)** .

**przodek4** - wersja nieskuteczna (nieskończona pętla!)

W definicji klauzuli prologowej należy najpierw korzystać z klauzul, opisujących proste relacje między obiektami, zanim użyjemy relacji bardziej złożonych (rekurencyjnych).



## Porządek kauzul prologowych i celów

*Przykład c.d.*

```
przodek1(P,D) :- rodzic(P,D) .  
przodek1(P,D) :- rodzic(P,R),przodek1(R,D) .  
rodzic → rodzic → przodek1
```

```
przodek2(P,D) :- rodzic(P,R),przodek2(R,D) .  
przodek2(P,D) :- rodzic(P,D) .  
rodzic → przodek2 → rodzic
```

```
przodek3(P,D) :- rodzic(P,D) .  
przodek3(P,D) :- przodek3(P,R),rodzic(R,D) .  
rodzic → przodek3 → rodzic
```

```
przodek4(P,D) :- przodek4(P,R),rodzic(R,D) .  
przodek4(P,D) :- rodzic(P,D) .  
przodek4 → rodzic → rodzic
```

## Porządek kauzul prologowych i celów

*Czy interpretacja deklaratywna programu jest potrzebna? Czy nie byłoby lepiej ograniczyć się tylko do interpretacji proceduralnej skoro poprawność programu w pierwszej nie jest jednoznaczna z poprawnością w drugiej?*

Postęp w językach programowania wskazuje na konieczność porzucania języków proceduralnych na rzecz języków deklaracyjnych, w których łatwiej i jaśniej można sformułować wiele zadań i w których ciężar przetwarzania w większym stopniu spoczywa na systemie programowania niż programiście.

Język programowania Prolog jest tylko pewnym krokiem uczynionym w kierunku „czystego” programowania deklaratywnego.



## Związki języka Prolog z logiką formalną

### Podstawowe pojęcia

- logika predykatów I rzędu
- klauzule Horn'a
- zasada rezolucji
- pojęcie najogólniejszego unifikatora (MGU)
- mechanizm unifikacji



## Podsumowanie

- Proste obiekty w Prologu to *atomy*, *stale*, *zmiennie*. Obiekty złożone (strukturalne) to *termy*.
- Termy składają się z *funktora* (*operatora*, *symbolu funkcyjnego*), określonego przez nazwę i arność
- Typ obiektu jest rozpoznawany jedynie w oparciu o jego budowę składniową
- Zakresem leksykalnym zmiennej jest jedna klauzula, zatem zmienne o tej samej nazwie w różnych klauzulach są różne
- Termy można reprezentować w postaci drzew, zaś ich przetwarzanie traktować jako przetwarzanie struktur drzewiastych
- Proces dopasowania polega na sprawdzeniu czy dwa termy są identyczne i dla jakich podstawień zmiennych są takie same

## Podsumowanie

- Jeżeli proces dopasowania zmiennych zakończy się sukcesem, to w wyniku otrzymujemy *najogólniejsze* możliwe podstawienia zmiennych
- Semantyka deklaratywna w Prologu określa czy dla danego programu cel jest spełniony, i jeśli tak, to dla jakich podstawień zmiennych
- Przecinek między klauzulami oznacza koniunkcję celów, a średnik - dysjunkcję celów
- Semantyka proceduralna w Prologu to procedura spełniania listy celów w kontekście danego programu. Procedura ta stwierdza fałszywość lub prawdziwość listy celów i podaje ewentualne podstawienia zmiennych. Procedura korzysta z mechanizmu nawrotów i analizuje alternatywne rozwiązania
- „Czysta” semantyka deklaratywna nie zależy od kolejności klauzul i kolejności celów w klauzulach

## Podsumowanie

- Semantyka deklaratywna jest ściśle określona przez kolejność klauzul i celów, która może mieć decydujący wpływ na efektywność i skuteczność programu
- Mając dany poprawnie w sensie deklaratywnym program możemy zwiększyć jego efektywność poprzez zmianę kolejności klauzul i celów, nie naruszając jego poprawności deklaratywnej. Jest to dobra metoda wykrywania i eliminacji nieskończonych pętli w programie
- Istnieją inne ogólne techniki (np. z dziedziny Szt. Int.) eliminacji nieskończonych pętli
- podstawowe pojęcia: atomy, liczby, zmienne, term prosty, term złożony, funktor, arność, dopasowanie termów, najogólniejsze dopasowanie, semantyka deklaratywna, semantyka proceduralna, instancja klauzuli