



Programowanie deklaratywne

Artur Michalski
Informatyka II rok



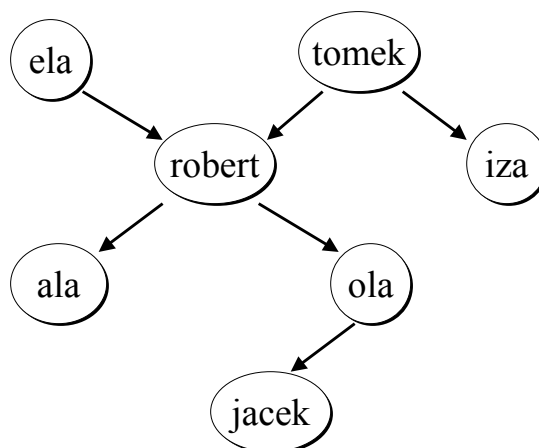
Plan wykładu

- 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

Wprowadzenie do języka Prolog

- Przykład prostego programu w języku Prolog: Relacje pokrewieństwa »
- Rozszerzanie programu przez wprowadzanie reguł prologowych »
- Rekurencyjna definicja reguły prologowej »
- Zasady generowania odpowiedzi na postawione pytania »
- Deklaratywna i proceduralna interpretacja programu prologowego »

Prosty program w języku Prolog: Relacje pokrewieństwa



Prosty program w języku Prolog: Relacje pokrewieństwa

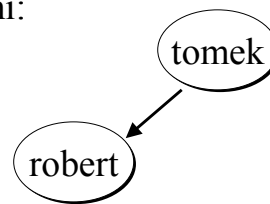
Prolog - język przetwarzania symbolicznego przeznaczony do rozwiązywania problemów dotyczących **obiektów** i **relacji** między nimi.

Przykład relacji między obiektami:

Tomek *jest rodzicem* Roberta.

Zapis w Prologu:

```
rodzic(tomek, robert).
```



Prosty program w języku Prolog: Relacje pokrewieństwa

Wszystkie relacje `rodzic` dla naszego przykładu:

```
rodzic(tomek, robert).
```

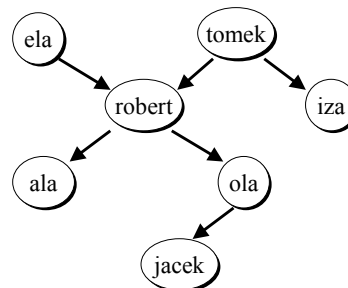
```
rodzic(tomek, iza).
```

```
rodzic(ela, robert).
```

```
rodzic(robert, ala).
```

```
rodzic(robert, ola).
```

```
rodzic(ola, jacek).
```



Prosty program w języku Prolog: Relacje pokrewieństwa

Program prologowy
składa się z *klauzul*.

W naszym
przykładzie klauzule
definiują fakty,
wskazujące, między
którymi obiektami
zachodzi relacja
pokrewieństwa
rodzic.

```
rodzic(tomek, robert).  
rodzic(tomek, iza).  
rodzic(ela, robert).  
rodzic(robert, ala).  
rodzic(robert, ola).  
rodzic(ola, jacek).
```

Prosty program w języku Prolog: Relacje pokrewieństwa

W Prologu możliwe jest formułowanie *pytań*
dotyczących zdefiniowanych relacji.

Przykładowe pytanie:

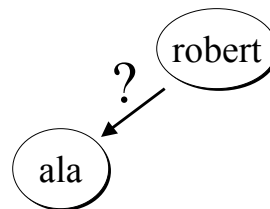
Czy Robert jest rodzicem Ali?

Zapis w Prologu:

```
?- rodzic(robert, ala).
```

Odpowiedź:

Yes



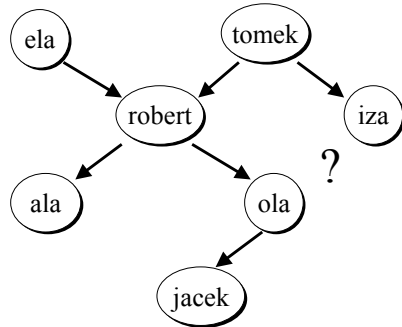
Prosty program w języku Prolog: Relacje pokrewieństwa

Inne pytanie:

`?- rodzic(iza, ola).`

Odpowiedź:

No



Kolejne pytanie:

`?- rodzic(iza, piotr).`

Odpowiedź:

No



Prosty program w języku Prolog: Relacje pokrewieństwa

Inne pytanie:

`?- rodzic(iza, ola).`

Odpowiedź:

No

Kolejne pytanie:

`?- rodzic(iza, piotr).`

Odpowiedź:

No

W Prologu, jeśli nie jest możliwa weryfikacja odpowiedzi w oparciu o zdefiniowane fakty, to odpowiedź brzmi **No**.

Prosty program w języku Prolog: Relacje pokrewieństwa

W Prologu możliwe jest formułowanie *pytań szczegółowych*.

Przykładowe pytanie:

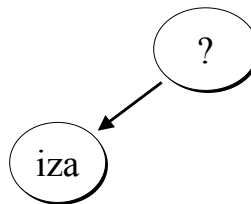
Kto jest rodzicem Izy?

Zapis w Prologu:

```
?- rodzic(X, iza).
```

Odpowiedź:

```
X = tomek
```



Prosty program w języku Prolog: Relacje pokrewieństwa

Odpowiedzi na pytanie szczegółowe może być *kilka*.

Przykładowe pytanie:

Kto jest dzieckiem Roberta?

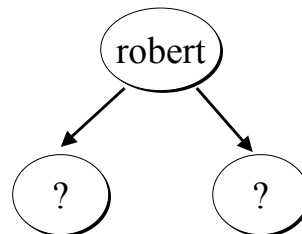
Zapis w Prologu:

```
?- rodzic(robert, X).
```

Odpowiedź:

```
X = ala;
```

```
X = ola
```





Prosty program w języku Prolog: Relacje pokrewieństwa

Pytanie szczegółowe może mieć węższy lub szerszy zakres.

Przykładowe pytanie:

Dla jakich X i Y, X jest rodzicem Y?

Zapis w Prologu:

?- `rodzic(X, Y)`.

Odpowiedź:

`X = ela`

`Y = robert;`

`X = tomek`

`Y = robert;`

`X = tomek`

`Y = iza;`

...



Prosty program w języku Prolog: Relacje pokrewieństwa

W Prologu można również formułować pytania złożone.

Przykładowe pytanie:

Kto jest babcią lub dziadkiem Jacka?

Zapis w Prologu:

?- `rodzic(Y,jacek), rodzic(X,Y)`.

Odpowiedź:

`X = robert`

`Y = ola`



Prosty program w języku Prolog: Relacje pokrewieństwa

Jeżeli zmieniamy porządek w pytaniu złożonym logiczny sens pytania pozostaje taki sam.

Wariant 1:

```
?- rodzic(Y,jacek) , rodzic(X,Y) .
```

Wariant 2:

```
?- rodzic(X,Y) , rodzic(Y,jacek) .
```

Odpowiedź (taka sama):

```
x = robert
```

```
y = ola
```



Prosty program w języku Prolog: Relacje pokrewieństwa

Inne pytanie:

Czy Ala i Ola mają wspólnego rodzica?

Zapis w Prologu:

```
?- rodzic(X,ala) , rodzic(X,ola) .
```

Odpowiedź:

```
x = robert
```




Prosty program w języku Prolog: Relacje pokrewieństwa

Podsumowanie przykładu:

- Prolog umożliwia prosty zapis relacji, takich jak **rodzic**, poprzez definiowanie n-tek, składających się z obiektów, które spełniają daną relację
- Użytkownik może w łatwy sposób formułować pytania dotyczące tych relacji
- Program prologowy składa się z *klauzul*
- Argumentami relacji mogą być: konkretne obiekty, stałe (takie jak **tomek** czy **ala**) lub ogólne obiekty, czyli zmienne (takie jak **X** czy **Y**)



Prosty program w języku Prolog: Relacje pokrewieństwa

Podsumowanie przykładu (ciąg dalszy):

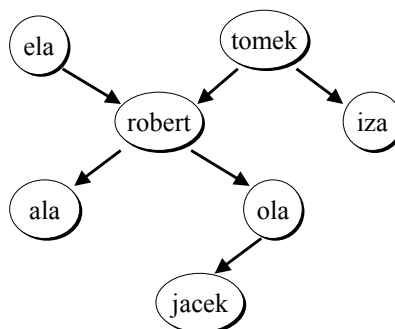
- Pytania składają się z jednego lub więcej *celów*. Ciąg celów, taki jak: **rodzic(X, ala)**, **rodzic(X, ola)** oznacza koniunkcję logiczną celów: *X jest rodzicem Ali i X jest rodzicem Oli.*
- Odpowiedź na pytanie może być pozytywna lub negatywna, w zależności od tego czy odpowiedni cel udało się osiągnąć, czy też nie. Pozytywna odpowiedź oznacza, że cel został *spełniony*. W przypadku przeciwnym cel jest *niespełniony*.
- Jeżeli jest kilka możliwych odpowiedzi na zadane pytanie, Prolog może wygenerować ich tyle ile potrzebuje użytkownik.

Rozszerzanie programu o reguły prologowe

Program prologowy może zawierać również fakty, opisujące wybrane cechy obiektów. Wykorzystujemy do tego celu tzw. *relacje unarne*.

Przykładowo może to być płeć osoby:

kobieta (ela) .
 mezczyzna (tomek) .
 mezczyzna (robert) .
 kobieta (iza) .
 kobieta (ala) .
 kobieta (ola) .
 mezczyzna (jacek) .



Rozszerzanie programu o reguły prologowe

Definicja relacji **dziecko**.

Fakty prologowe:

~~dziecko (robert, ela) .~~
~~dziecko (robert, tomek) .~~
~~dziecko (iza, tomek) .~~
 ...

Wykorzystanie ogólnej zasady:

*Dla wszystkich X i Y,
 Y jest dzieckiem X, o ile
 X jest rodzicem Y.*

Zapis w Prologu:

**dziecko (Y, X) :-
 rodzic (X, Y) .**

Rozszerzanie programu o reguły prologowe

Fakty prologowe

Zawsze
(bezwarunkowo)
prawdziwe.

Reguły prologowe

Prawdziwe, jeżeli
spełniony jest pewien
warunek.

Budowa reguły prologowej:

$\underbrace{\text{dziecko}(Y,X)}_{\text{nagłówek}} \text{ :- } \underbrace{\text{rodzic}(X,Y)}_{\text{ciało}} .$

Rozszerzanie programu o reguły prologowe

Budowa reguły prologowej (ciąg dalszy):

$\underbrace{\text{dziecko}(Y,X)}_{\text{nagłówek}} \text{ :- } \underbrace{\text{rodzic}(X,Y)}_{\text{ciało}} .$

↑
Część konkluzyjna
albo
Lewa strona reguły

↑
Część warunkowa
albo
Prawa strona reguły



Rozszerzanie programu o reguły prologowe

Reguły prologowe „w działaniu”

Przykładowe pytanie:

?- **dziecko(iza,tomek)** .

Brak bezpośrednich faktów powoduje odwołanie do reguł.

Reguła jako rodzaj uogólnienia może być zastosowana do konkretnych obiektów takich jak **iza** i **tomek**.

Użycie reguły wymaga podstawień:

X = tomek Y = iza



Rozszerzanie programu o reguły prologowe

Reguły prologowe „w działaniu” (ciąg dalszy):

Wiązanie zmiennych **X** i **Y** daje przypadek szczególny reguły:

dziecko(iza,tomek) :- rodzic(tomek,iza) .

Sprawdzenie części warunkowej - zastąpienie celu *podcelem*:

rodzic(tomek,iza) .

Weryfikacja podcelu: prawdziwy w oparciu o fakty.

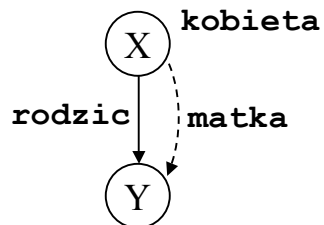
Wniosek: konkluzja reguły prawdziwa.

Odpowiedź: **Yes**

Rozszerzanie programu o reguły prologowe

Przykład innej reguły - *matka*:

*Dla wszystkich X i Y,
X jest matką Y, o ile
X jest rodzicem Y i
X jest kobietą.*



Zapis w Prologu:

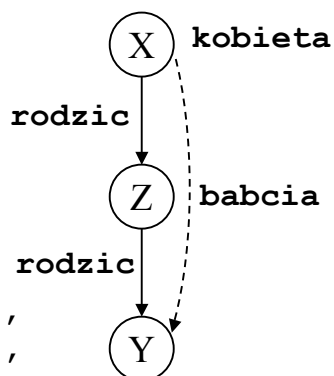
```
matka(X,Y) :- rodzic(X,Y), kobieta(X).
```

↑
koniunkcja
warunków

Rozszerzanie programu o reguły prologowe

Inne reguły pokrewieństwa - *babcia*:

*Dla wszystkich X i Y,
X jest babcią Y, o ile
X jest rodzicem Z,
Z jest rodzicem Y i
X jest kobietą.*



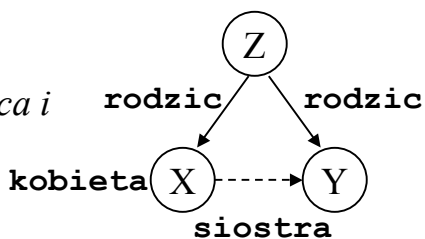
Zapis w Prologu:

```
babcia(X,Y) :- rodzic(X,Z),  
rodzic(Z,Y),  
kobieta(X).
```

Rozszerzanie programu o reguły prologowe

Inne reguły pokrewieństwa - *siostra*:

Dla wszystkich X i Y ,
 X jest siostrą Y , o ile
 X i Y mają wspólnego rodzica i
 X jest kobietą.



Zapis w Prologu:

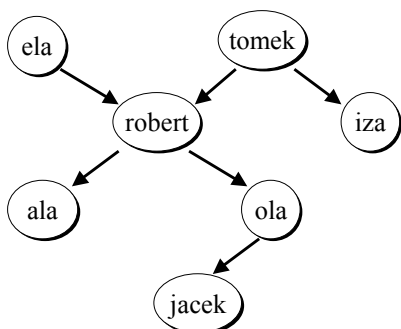
```

siostra(X,Y) :- rodzic(Z,X),
                rodzic(Z,Y),
                kobieta(X).
  
```

Rozszerzanie programu o reguły prologowe

Przykładowe pytanie:
 ?- `siostra(ala,ola)`.

Odpowiedź: **Yes**



Przykładowe pytanie:
 ?- `siostra(X,ola)`.

Odpowiedź:

X = ala;

X = ola



Rozszerzanie programu o reguły prologowe

Przykładowe pytanie:
?- `siostra(ala,ola)`.

Odpowiedź: **Yes**

Poprawna reguła:

```
siostra(X,Y) :- rodzic(Z,X),  
               rodzic(Z,Y),  
               kobieta(X),  
               X\=Y.
```

Przykładowe pytanie:
?- `siostra(X,ola)`.

Odpowiedź:

X = ala;

X = ola



Rozszerzanie programu o reguły prologowe

Podsumowanie:

- Programy prologowe można rozszerzać w prosty sposób poprzez definiowanie nowych klauzul
- Wyróżniamy trzy rodzaje klauzul: *fakty*, *reguły* i *pytania*
- *Fakty* opisują to, co jest zawsze, bezwarunkowo prawdziwe
- *Reguły* opisują to, czego prawdziwość zależy od pewnego warunku
- Za pomocą *pytań* można dowiedzieć się co jest prawdą lub fałszem

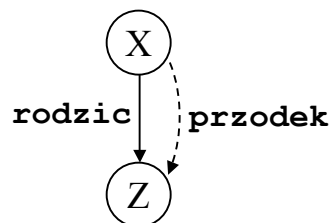
Rozszerzanie programu o reguły prologowe

Podsumowanie (ciąg dalszy):

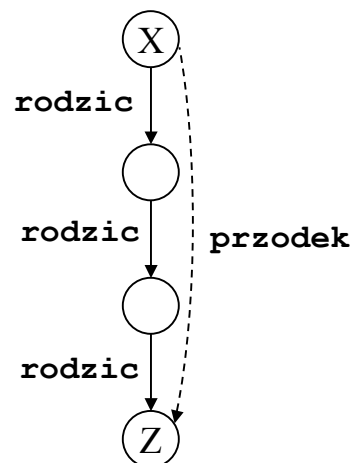
- Klauzula prologowa składa się z: *nagłówka* i *ciała*; ciało to lista *celów* oddzielonych przecinkami, które taktujemy jako *koniunkcję*
- Fakty to klauzule, które mają nagłówek i puste ciało; pytania posiadają tylko ciało; reguły mają zarówno nagłówek, jak i niepuste ciało
- W trakcie działania zmienne mogą być zamieniane przez inne obiekty; proces ten określamy mianem *wiązania (unifikacji)*
- Zakładamy, że wszystkie zmienne klauzuli są poprzedzone kwantyfikatorem uogólnionym

Rekurencyjna definicja reguły

Definicja relacji *przodek*



Przodek *bezpośredni*

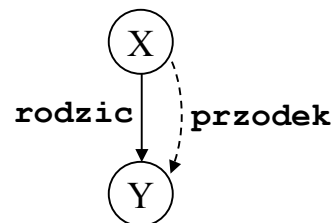


Przykład przodka *pośredniego*

Rekurencyjna definicja reguły

Pierwsza definicja relacji *przodek*:

Dla wszystkich X i Y ,
 X jest przodkiem Y , o ile
 X jest rodzicem Y .



Przodek *bezpośredni*

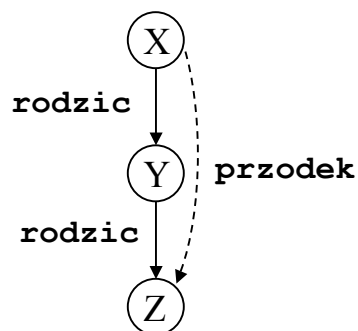
Zapis w Prologu:

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

Rekurencyjna definicja reguły

Druga definicja relacji *przodek*:

```
przodek(X,Z) :-  
    rodzic(X,Y) ,  
    rodzic(Y,Z) .
```

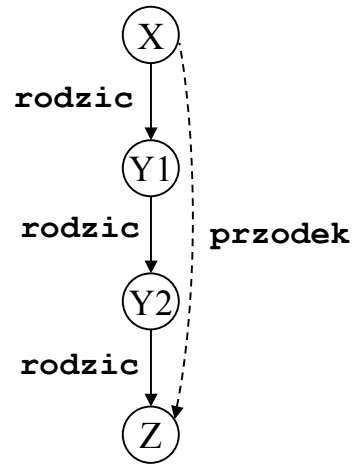


Przodek *pośredni*

Rekurencyjna definicja reguły

Trzecia definicja relacji *przodek*:

```
przodek (X, Z) :-  
    rodzic (X, Y1) ,  
    rodzic (Y1, Y2) ,  
    rodzic (Y2, Z) .
```

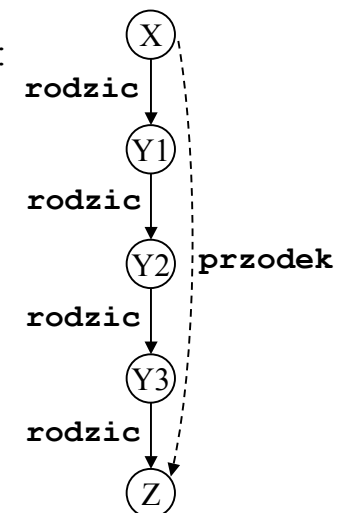


Przodek *pośredni*

Rekurencyjna definicja reguły

Czwarta definicja relacji *przodek*:

```
przodek (X, Z) :-  
    rodzic (X, Y1) ,  
    rodzic (Y1, Y2) ,  
    rodzic (Y2, Y3) ,  
    rodzic (Y3, Z) .
```

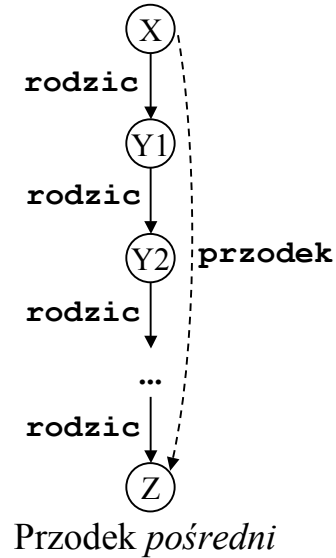


Przodek *pośredni*

Rekurencyjna definicja reguły

Kolejna definicja relacji *przodek*:

```
przodek(X, Z) :-
    rodzic(X, Y1),
    rodzic(Y1, Y2),
    ...
    rodzic(Yn, Z).
```



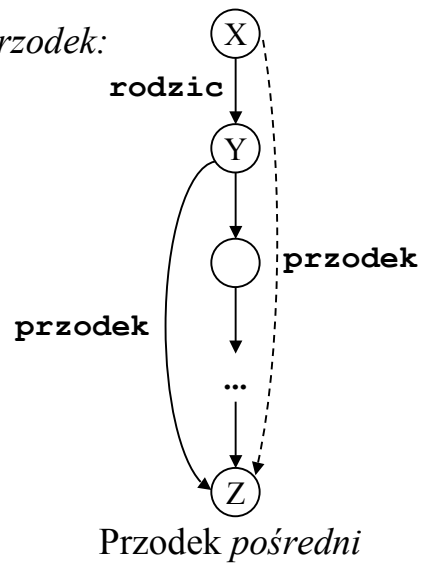
Rekurencyjna definicja reguły

Ostateczna definicja relacji *przodek*:

*Dla wszystkich X i Z,
X jest przodkiem Z, o ile
X jest rodzicem Y i
Y jest przodkiem Z.*

Zapis w Prologu:

```
przodek(X, Z) :-
    rodzic(X, Y),
    przodek(Y, Z).
```



Rekurencyjna definicja reguły

Pełna definicja relacji *przodek* w Prologu:

```
przodek(X,Z) :- rodzic(X,Z). %bezpośredni  
przodek(X,Z) :- rodzic(X,Y), %pośredni  
                przodek(Y,Z).
```

Programowanie z wykorzystaniem rekurencji jest podstawową metodą programowania w Prologu.

Generowanie odpowiedzi na postawione pytania

Jak w Prologu generowane są odpowiedzi na pytania?

Pytanie (cel lub ciąg celów)



Sprawdzić, czy cel jest *spełniony*



Czy cel jest *prawdziwy* przy założeniu, że zdefiniowane relacje są prawdziwe?



Dowieść, że cel jest *logiczną konsekwencją* zdefiniowanych faktów i reguł



Jeśli cel zawiera *zmienne*, podać dla jakich *wartości* jest spełniony

Generowanie odpowiedzi na postawione pytania

Przykład generowania odpowiedzi

Relacje (fakty):

Wszyscy ludzie są omylni.

Sokrates jest człowiekiem.

Pytanie:

Czy Sokrates jest omylny?

Zapis w Prologu:

```
omylny(X) :-
```

```
    czlowiek(X) .
```

```
czlowiek(sokrates) .
```

```
?- omylny(sokrates) .
```

Odpowiedź:

Yes

Generowanie odpowiedzi na postawione pytania

Inny przykład:

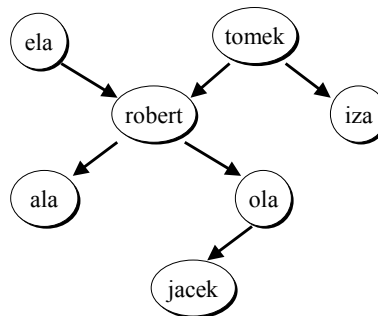
```
przodek(X,Z) :- rodzic(X,Z) . %prb
```

```
przodek(X,Z) :- rodzic(X,Y) , %prp
```

```
    przodek(Y,Z) .
```

Pytanie:

```
?- przodek(tomek,ola) .
```






Generowanie odpowiedzi ...

Nieformalna interpretacja deklaratywna:

1. Znany fakt: `rodzic(robert,ola)`
2. Na podstawie reguły *prb* wnioskujemy, że `przodek(robert,ola)`, bo:
$$\text{rodzic(robert,ola)} \Rightarrow \text{przodek(robert,ola)}$$
3. Znany fakt: `rodzic(tomek,robert)`
4. Korzystając z faktów: `przodek(robert,ola)` i `rodzic(tomek,robert)` wnioskujemy na podstawie reguły *prp*, że `przodek(tomek,ola)` :
$$\text{rodzic(tomek,robert)} \wedge \text{przodek(robert,ola)} \Rightarrow \text{przodek(tomek,ola)}$$



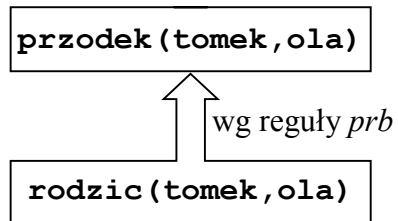
Generowanie odpowiedzi ...

Nieformalna interpretacja proceduralna:

1. Cel: `przodek(tomek,ola)`
2. *Poszukiwanie* odpowiednich klauzul
3. *Dopasowanie* do nagłówków klauzul *prb* i *prp*
4. Klauzula *prb*: `przodek(X,Z) :- rodzic(X,Z)` .
Wiązanie zmiennych: `X = tomek, Z = ola`
5. Nowy cel: `rodzic(tomek,ola)`

Generowanie odpowiedzi ...

Nieformalna interpretacja proceduralna:



6. Brak klauzuli pasującej do celu `rodzic(tomek, ola)`

7. *Nawrót* do pierwotnego celu (`przodek(tomek, ola)`) i analiza drugiej pasującej klauzuli, *prp*:

`przodek(X, Z) :- rodzic(X, Y), przodek(Y, Z) .`

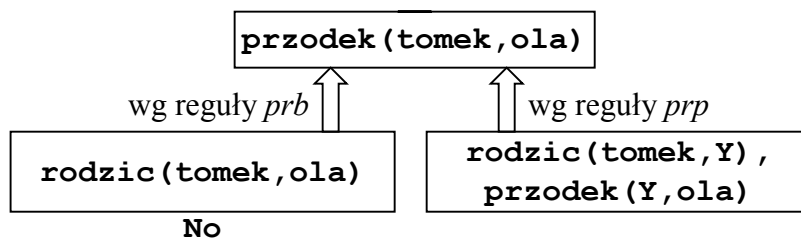
Generowanie odpowiedzi ...

Nieformalna interpretacja proceduralna:

8. *Wiązanie* zmiennych: `x = tomek, z = ola`

9. Nowy cel (dwa *podcele*):

`rodzic(tomek, Y), przodek(Y, ola)`





Generowanie odpowiedzi ...

Nieformalna interpretacja proceduralna:

10. Pierwszy podcel: `rodzic(tomek, Y)`
11. Dopasowanie do klauzuli `rodzic(tomek, robert)`
Wiązanie zmiennych: `Y = robert`
12. Drugi podcel : `przodek(robert, ola)`
13. Dopasowanie do klauzul `prb` i `prp`
14. Klauzula `prb`:
`przodek(X', Z') :- rodzic(X', Z') .`
Wiązanie zmiennych: `X' = robert, Z' = ola`



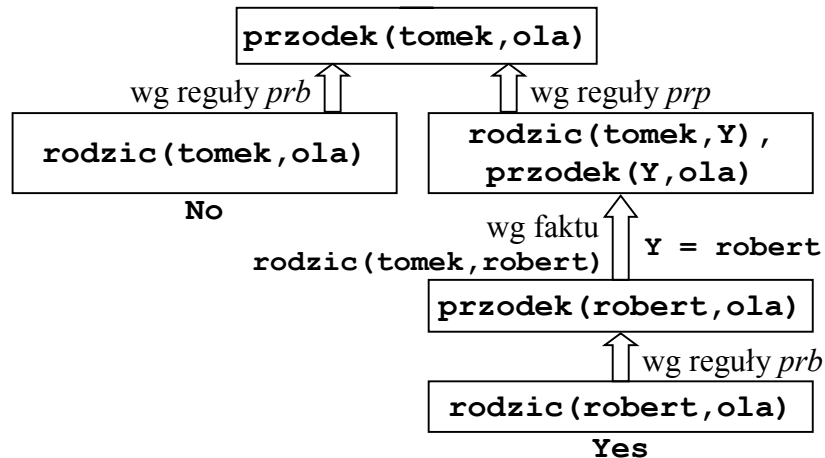
Generowanie odpowiedzi ...

Nieformalna interpretacja proceduralna:

15. Nowy cel: `rodzic(robert, ola)`
16. Dopasowanie do klauzuli (faktu)
`rodzic(robert, ola)`
17. Koniec wnioskowania - cel główny spełniony

Generowanie odpowiedzi ...

Nieformalna interpretacja proceduralna:



Deklaratywna i proceduralna interpretacja programu prologowego

- *Interpretacja deklaratywna* pozwala określić *jaki* będzie rezultat programu prologowego
- *Interpretacja proceduralna* pozwala określić *w jaki sposób* ten rezultat zostanie otrzymany, czyli *w jaki sposób* zostaną użyte relacje zdefiniowane w programie

Programowanie w języku Prolog powinno opierać się tylko - o ile to jest możliwe - na interpretacji deklaratywnej programu prologowego.



Wprowadzenie do języka Prolog

Podsumowanie:

- Programowanie w Prologu opiera się na definiowaniu relacji i zadawaniu pytań dotyczących relacji
- Program prologowy składa się z *klauzul*. Mamy trzy rodzaje klauzul: *fakty*, *reguły* i *pytania*
- Relacje definiowane są za pomocą faktów, które mają postać n-tek zawierających obiekty, które relacje spełniają lub reguł, które je opisują
- *Pytania* dotyczące relacji przypominają zadawanie pytań w systemach baz danych; odpowiedzi składają się ze zbioru obiektów, spełniających cel zawarty w pytaniu
- Sprawdzenie czy dany obiekt spełnia cel opiera się na złożonym procesie obliczeniowym, wykorzystującym mechanizmy logicznego wnioskowania i nawroty; wszystkie szczegóły tego procesu pozostają ukryte przed programistą



Wprowadzenie do języka Prolog

Podsumowanie c.d.:

- Wyróżniamy dwie interpretacje programu prologowego: deklaratywną i proceduralną; interpretacja proceduralna powinna być jedną wykorzystywaną przez programistę; są jednak sytuacje, kiedy nie da się uniknąć interpretacji proceduralnej programu
- podstawowe pojęcia:
klauzula, fakt, reguła, pytanie,
nagłówek i ciało klauzuli,
reguła rekurencyjna, definicja rekurencyjna,
zmienna, wiązanie zmiennej,
cel, cel spełniony, cel niespełniony,
nawroty,
interpretacja deklaratywna i proceduralna



Literatura

- R.A. Kowalski, *Logika w rozwiązywaniu zadań*, WNT, Warszawa 1989.
- E. Gatnar, K. Stapor, *Prolog*, Wydawnictwo PLJ, Warszawa 1991.
- F. Kluźniak, S. Szpakowicz, *Prolog*, WNT, Warszawa, 1983.
- I. Bratko, *Prolog – programming for AI*, Addison-Wesley 1990.