

Programowanie w logice

PROLOG

- PROLOG – język wysokiego poziomu
- Powstał w 1972 na Uniwersytecie w Marsylii (Francja) w zespole A.Colmerauer'a i F.Roussel'a
- **PRO**grammation en **LOG**ique,
- PROgramming in LOGic,
- PROramowanie w LOGice

Zastosowania

- systemy komputerowe wykorzystujące metody sztucznej inteligencji (Artificial Intelligence),
- Systemy ekspertowe (Expert System)
- systemy z bazą wiedzy (Knowledge-based Systems)

AI obejmuje:

- Systemy ekspertowe (doradcze)
- Uczenie się przez komputer
- Automatyczne dowodzenie twierdzeń
- Przetwarzanie języka naturalnego
- Robotyka
- Rozpoznawanie obrazów
- Rozpoznawanie mowy
- Gry komputerowe

- Podstawa utworzenia PROLOGu: **logika predykatów pierwszego rzędu** oraz **zasada rezolucji**.
- Program napisany w PROLOGu zawiera **zbiór asercji** (faktów), które mogą być traktowane jako aksjomaty pewnej teorii oraz **zbiór reguł wnioskowania** (dedukcji) dla tej teorii. Problem do rozwiązania jest twierdzeniem teorii, które należy udowodnić.

Podstawowe elementy języka

- Duże litery A, B, ..., Z;
- Małe litery a, b, ..., z;
- cyfry 0, 1, 2, ..., 9;
- Symbol „_”;
- Znaki specjalne: np. +, -, *, /, <, >, =, :, ., &, ~.
- string – ciąg znaków

Atomy

Stałe (obiekty proste, struktury atomowe, atomy) to symboliczne nazwy obiektów występujących w programie.

- Ciągi znaków zbudowane z dużych i małych liter, cyfr i znaku podkreślenia „_” zaczynający się małą literą.

Przykłady: `wilk`, `lubi`, `słucha_muzyki`, `sto`.

- Ciąg znaków ujętych w apostrofy.

Przykłady: `'Jan'`, `'Prolog'`, `'Dwa_razy_dwa'`, `'&^%&#@$_&*&'`.

- Ciąg znaków specjalnych.

Przykłady: `@=` `====>` `;` `:-`

Liczby

- Liczby całkowite `...`, `-2`, `-1`, `0`, `1`, `2`, `...`

- Liczby rzeczywiste nie mają szczególnego znaczenia w typowych aplikacjach w Prologu.

Zmienne

- Ciąg dużych i małych liter, cyfr i znaku podkreślenia.

Przykłady: `X`, `Y`, `Zmienna`, `_znak`, `X_123`, `Lista`, `ListaAB`, `_head`, `Tail`, `_input`, `Output`

- Zmienna „_” to zmienna anonimowa

Termy

- **Term** to stała, zmienna lub struktura.

- Term złożony (**struktura**) - obiekt składający się z innych obiektów, określony jest przez funktor oraz nazwy obiektów składowych (argumentów funktora). Funktor musi być atomem.

Przykłady:

`ksiazka(adam,mickiewicz,pan_tadeusz,2000)`

`ksiazka(autor(gatnar,stapor),tytul(prolog),wydanie(pwn,warszawa,2017))`

Predykaty

- **Predykaty** opisują związki zachodzące między obiektami

Symbol predykatu: atom.

Przykłady:

`lubi`, `wiekszy_od`, `młodszy_od`, `jest_przodkiem`

Argumenty predykatu: termy oddzielone przecinkami, dowolna liczba termów.

Przykłady:

`lubi(anna,ksiazka(autor(adam,mickiewicz),pan_tadeusz,1960))`

`suma(X,Y,Z)`

`ojciec(maciej,marek)`

Fakty

- Opisują związki między obiektami, opisują obiekty. Przedstawia się je za pomocą **predykatów**.

predykat(object₁, object₂, object₃, ..., object_n).

`lubi(anna, piotr).`

`lubi(piotr, anna).`

`kobieta(anna).`

`ojciec(jan, marek).`

`odleglosc(poznan,berlin,300).`

Fakty

- Nazwy obiektów występujące w nawiasach nazywamy **argumentami**.

- Zbiór faktów nazywamy **bazą danych**.

Cel

- Bazę danych można wykorzystać poprzez **zadawanie pytań** (lub inaczej **celów do realizacji**).
- Celem**, w zależności od formy, jest:
 - pytanie o prawdziwość faktów,
 - połączenie znalezienia obiektów będących w relacji z innymi obiektami

Zapytania (cele)

```

1 ?- studiuje(anna,X).
   X = informatyka ;
   X = matematyka.

2 ?- studiuje(hanna,X).
   X = informatyka.

3 ?- studiuje(Y,informatyka).
   Y = anna ;
   Y = hanna.

4 ?- studiuje(Y,matematyka).
   Y = anna ;
   Y = jan.

```

studiuje(anna, informatyka).
 studiuje(hanna, informatyka).
 studiuje(anna, matematyka).
 studiuje(jan, matematyka).

Zapytania (cele)

```

lubi(jan, wycieczki).
lubi(jan, książki).
lubi(jan, kino).
lubi(ewa, książki).
lubi(ewa, teatr).

1 ?- lubi(jan,A).
   A = wycieczki ;
   A = książki ;
   A = kino.

2 ?- lubi(ewa,A).
   A = książki ;
   A = teatr.

3 ?- lubi(B,książki).
   B = jan ;
   B = ewa.

4 ?- lubi(B,teatr).
   B = ewa.

```

Reguły

- stwierdzenia dotyczące obiektów i ich powiązań, opisują zależności między obiektami

```

predykat(objekt1, obiekt2, obiekt3,... obiektn) if
predykat1(objekt11, obiekt12, obiekt13,...,obiekt1m1) and
predykat2(objekt21, obiekt22, obiekt23,...,obiekt2m2) and
.....
predykatk(objektk1, obiektk2,obiektk3,... obiektkmk).

```

Reguły

- Przykład:


```
siostra(X,Y) :- kobieta(X),rodzice(M,O,X),
               rodzice(M,O,Y), X\=Y.
```

X jest siostrą Y, **jeśli** X jest kobietą **oraz** X i Y mają takich samych rodziców
- Predykat **siostra** jest tutaj **nagłówkiem reguły** (nagłówek składa się tylko z jednego predykatu), zaś warunki: **kobieta(X)**, **rodzice(M,O,X)**, **rodzice(M,O,Y)**, **X\=Y** tworzą **treść reguły**.

Zapytania (cele)

```

1 ?- siostra(ewa,S).
   S = marek.

2 ?- siostra(R,W).
   R = ewa,
   W = marek ;
   false.

kobieta(ewa).
kobieta(marta).
mezczyzna(jan).
mezczyzna(marek).
rodzice(marta,jan,ewa).
rodzice(marta,jan,marek).
siostra(X,Y) :- kobieta(X),rodzice(M,O,X),
               rodzice(M,O,Y), X\=Y.

```

Poszukiwanie odpowiedzi

- Zapisane w bazie danych fakty i reguły analizowane są **od góry do dołu w kolejności wprowadzenia**. Szukany jest fakt potwierdzający zapytanie.
- Jeżeli w pytaniu jest zmienna, to w trakcie wyszukiwania odpowiedzi jest **ukonkretniana** (podstawiane są pod nią stałe wartości).
- Jeżeli zapytanie jest złożone, to zawsze poszukuje się potwierdzenia predykatów od lewego do skrajnie prawego. Powrót do wcześniejszych predykatów celem sprawdzenia wszystkich kombinacji nazywa się **nawracaniem** (backtracking).

Fakty i reguły

- Fakty i reguły stanowią tzw. **klauzule**.
- Fakt to klauzula składająca się tylko z nagłówka (nie posiada treści).
- Zbiór klauzul, w których predykaty tworzące nagłówki mają tę samą nazwę i liczbę argumentów tworzą **procedurę**.

Deklaratywna interpretacja klauzuli

Każdą klauzulę o ogólnej postaci:

A :- B₁, B₂, ..., B_n.

można interpretować w następujący sposób:

A zachodzi, jeśli zachodzą (są prawdziwe) **B₁** i **B₂** i ... i **B_n**.

Unifikacja

Unifikacja termów T1 i T2 polega na szukaniu wyrażień, jakie trzeba podstawić pod zmienne występujące w T1 i T2, by po ich podstawieniu termy stały się identyczne. Jeśli takiego postawienia nie ma, to unifikacja zawodzi.

T1=T2

Unifikacja

- Jeśli T1 i T2 są **stałymi** (atomami lub liczbami) to równość zachodzi, gdy ta sama stała występuje po obu stronach predykatu =.

np.

praga=praga.

2017=2017.

'Kowalski'='Kowalski'.

1 ?- praga=praga.

true.

2 ?- praga='praga'.

true.

3 ?- praga='pragA'.

false.

4 ?- 2017=2017.

true.

5 ?- 'Kowalski'='Kowalski'.

true.

6 ?- 'Kowalski'='kowalski'.

false.

Unifikacja

Jeżeli termy T1 i T2 są zmiennymi, np. X i Y to przy próbie uzgodnienia tych zmiennych możliwe są przypadki:

- Zmienna X jest ukonkretniona, czyli związana z pewną stałą (strukturą), a Y jest wolna – wtedy Y zostanie ukonkretniona przez wartość zmiennej X.

np.

stolica(warszawa,polska)=Y.

1=Y.

1+2=Y.

10 ?- 1=Y.

Y = 1.

11 ?- 1+2=Y.

Y = 1+2.

12 ?- stolica(warszawa,polska)=Y.
Y = stolica(warszawa, polska).

Unifikacja

- Zmienna X jest wolna, a Y ukonkretniona, wtedy X zostanie ukonkretniona przez wartość zmiennej Y.

np.

X=madryt.	14 ?- X=jan. X = jan.
X=77.	15 ?- X='Jan'. X = 'Jan'.
X=adres(poznan, 60-661, mieszkaj).	16 ?- X=uam(wmi,morasko). X = uam(wmi, morasko).

Unifikacja

- Jeśli obie zmienne są wolne, to wtedy następuje ich **powiązanie**, czyli jeśli w pewnym momencie działania programu jedna z nich zostanie ukonkretniona, to druga automatycznie przyjmie tę samą wartość.

np.

X=Y.	17 ?- X=Y. X = Y.
	18 ?- A=Z. A = Z.
	19 ?- A=Z, Z=5. A = Z, Z = 5.
	20 ?- A=Z, Z=5, X=A+Z. A = Z, Z = 5, X = 5+5.

Unifikacja

Dwie struktury są sobie równe, jeśli

- są opisane przez ten sam funktor,
- funktory mają tę samą liczbę argumentów,
- odpowiednie argumenty są sobie równe.

np.

kolor(niebieski,auto)=kolor(niebieski,auto).

Uzgadnianie:

staw(morskie_oko,tatry)=staw(X,tatry).

Unifikacja

1 ?- a(X,Y,Z)=a(s,t,v).

X = s,

Y = t,

Z = v.

2 ?- X=uczelnia(uam).

X = uczelnia(uam).

3 ?- stolica(X,polska)=stolica(warszawa,P).

X = warszawa,

P = polska.

4 ?- a(1,2)=b(1,2).

false.

Porównywanie wartości

- X=Y**

Porównanie kończy się sukcesem, gdy oba wyrażenia są identyczne lub da się je uzgodnić

- X\=Y**

Porównanie kończy się sukcesem, gdy wyrażen nie daje się uzgodnić

Porównywanie wartości

- X==Y**

Predykat X==Y również oznacza równość, ale w węższym znaczeniu niż X=Y

X=Y

Jeśli X lub Y w wyrażeniu X=Y jest zmienną, to następuje uzgodnienie.

W przypadku X==Y uzgodnienie nie nastąpi, jeśli jedna ze zmiennych ma przypisaną wartość, a druga nie.

Predykat = traktuje zmienną nieukonkretnioną jako równą dowolnej wartości,

dla predykatu == zmienna nieukonkretniona jest równa jedynie zmiennej z nią związanej

- X\==Y**

```

11 ?- X=1.
X = 1.

12 ?- X==1.
false.

13 ?- X=Y.
X = Y.

14 ?- X==Y.
false.

15 ?- X=1,Y==X.
false.

16 ?- X=1,Y=1,X==Y.
X = Y, Y = 1.

17 ?- X=1,Y=1,X=Y.
X = Y, Y = 1.

```

```

vertical(line(point(X,Y),point(X,Z))).
horizontal(line(point(X,Y),point(Z,Y))).

```

```

1 ?- vertical(line(point(1,1),point(1,3))).
true.

2 ?- vertical(line(point(1,1),point(3,2))).
false.

3 ?- horizontal(line(point(1,1),point(2,Y))).
Y = 1.

4 ?- horizontal(line(point(2,3),P)).
P = point(_G2437, 3).

```

```

f(a).
f(b).

g(a).
g(b).

h(b).

k(X) :- f(X), g(X), h(X).

1 ?- k(a).
false.

2 ?- k(b).
true.

3 ?- k(X).
X = b.

```

```

[trace] 5 ?- k(a).
Call: (6) k(a) ? creep
Call: (7) f(a) ? creep
Exit: (7) f(a) ? creep
Call: (7) g(a) ? creep
Exit: (7) g(a) ? creep
Call: (7) h(a) ? creep
Fail: (7) h(a) ? creep
Fail: (6) k(a) ? creep
false.

```

```

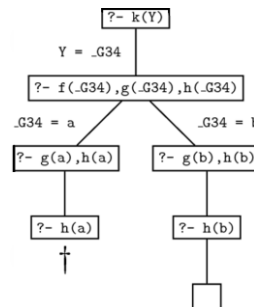
[trace] 6 ?- k(b).
Call: (6) k(b) ? creep
Call: (7) f(b) ? creep
Exit: (7) f(b) ? creep
Call: (7) g(b) ? creep
Exit: (7) g(b) ? creep
Call: (7) h(b) ? creep
Exit: (7) h(b) ? creep
Exit: (6) k(b) ? creep
true.

```

```

[trace] 4 ?- k(X).
Call: (6) k(_G2389) ? creep
Call: (7) f(_G2389) ? creep
Exit: (7) f(a) ? creep
Call: (7) g(a) ? creep
Exit: (7) g(a) ? creep
Call: (7) h(a) ? creep
Fail: (7) h(a) ? creep
Redo: (7) f(_G2389) ? creep
Exit: (7) f(b) ? creep
Call: (7) g(b) ? creep
Exit: (7) g(b) ? creep
Call: (7) h(b) ? creep
Exit: (7) h(b) ? creep
Exit: (6) k(b) ? creep
X = b.

```



Lista operatorów arytmetycznych i porównania

+ dodawanie
 - odejmowanie
 / dzielenie
 // dzielenie całkowite
 * mnożenie
 ** potęga
 mod reszta z dzielenia
 is znak równości (wynik obliczeń arytmetycznych) np. (X is 1 mod 3)
 == czy wartości równe
 != czy wartości różne
 > większe
 < mniejsze
 >= większe lub równe
 <= mniejsze lub równe

Operacja równości (:=) a unifikacja (=)

1 ?- 1+2:=2+1.
true.

2 ?- 1+2=2+1.
false.

3 ?- 1+A=B+2.
 A = 2,
 B = 1.

4 ?- 1+A:=B+2.

ERROR: :=/2: Arguments are not sufficiently instantiated

Operator "is"

- „is” służy do uokretniania występującej po lewej stronie zmiennej przez wyrażenie arytmetyczne znajdującą się po prawej stronie.

5 ?- X is 2*5.
 X = 10.

6 ?- Y is (2+19)-11.
 Y = 10.

7 ?- X=45,Y=2*X.
 X = 45,
 Y = 2*45.

Literatura

- W. Clocksin, C. Mellish, „Prolog. Programowanie”
- E. Gatnar, K. Stąpor, „Prolog”
- G. Brzykcy, A. Meissner, „Programowanie w Prologu i programowanie funkcyjne”
- M. Ben-Ari, „Logika matematyczna w informatyce”
- <http://lpn.swi-prolog.org/lpnpag.php?pageid=online>