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
- PROgrammation en LOGique,
- PROgramming in LOGic,
- PROramowanie w LOGice

Zastosowania

- systemy komputerowe wykorzystujące metody sztucznej inteligencji (Artifical Inteligence),
- 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.

Predykaty

- **Predykaty** opisują związki zachodzące między obiektami

Symbol predykatu: dowolna długość, dowolne litery cyfry i „_”, zaczyna się od małej litery.
Przykłady: lubi, wiekszy_od, mlodszy_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(obiekt$_1$, obiekt$_2$,obiekt$_3$,...,obiekt$_n$).

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,
  - polecenie znalezienia obiektów będących w relacji z innymi obiektami

Zapytania (cele)

lubi(jan, wycieczki).
lubi(jan, ksiazki).
lubi(jan, kino).
lubi(ewa, ksiazki).
lubi(ewa, kwiaty).

?- lubi(jan, kino).
yes
?- lubi(jan, konie).
no

Zapytania

?- lubi(jan,X).
X=wycieczki
X=ksiazki;
X=kino.

?- lubi(X,ksiazki).
X=jan;
X=ewa.

Reguły

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

predykat(obiekt$_1$, obiekt$_2$, obiekt$_3$,... obiekt$_n$) **if**
predykat1(obiekt1$_1$, obiekt1$_2$, obiekt1$_3$,...,obiekt1$_{m1}$) **and**
predykat2(obiekt2$_1$, obiekt2$_2$, obiekt2$_3$,...,obiekt2$_{m2}$) **and**
.....
predykatk(obiektk$_1$, obiektk$_2$,obiektk$_3$,... obiektk$_{mk}$).

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.**

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:- B_1, B_2,..., B_n.

można interpretować w następujący sposób:

A zachodzi, jeśli zachodzą (są prawdziwe) **B_1** i **B_2** i **...** i **B_n.**

Podstawowe elementy języka

Term to stała, zmienna lub struktura.

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

Przykłady:

j23, 1, 2, 007, jan, aula_A,
jan_kowalski, 'Jan Kowalski', '125,00'

Podstawowe elementy języka

- **Zmienne** symbolicznie przedstawiają nazwy obiektów, które nie są w danej chwili znane.

Przykłady: X, A, Ogon_listy, _miasto

Zmienna anonimowa „_"

(nie ma znaczenia dla programisty)

Struktury

- **Struktura** – obiekt składający się z innych obiektów, określona jest przez funktor oraz nazwy obiektów składowych (argumentów funktora)

Przykłady:

ksiazka(adam,mickiewicz,pan_tadeusz,2012)

ksiazka(autor(adam,mickiewicz), tytul(pan_tadeusz),wydanie(100,2012))

Unifikacja

Unifikacja termów T1 i T2 polega na szukaniu wyrażeń, 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.

2010=2010.

'Kowalski'='Kowalski'.

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(berlin,niemcy)=Y.

1=Y.

1+2=Y.

Unifikacja

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

np.

X=madryt.

X=77.

X=adres(poznan, 60-661, mieszka_l).

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.

Unifikacja

Dwie struktury są sobie równe, jeśli

a) są opisane przez ten sam funktor,

b) funktory mają tę samą liczbę argumentów,

c) 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 wyrażeń nie daje się uzgodnić

Porównywanie wartości

- **X==Y**

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

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
>= iększe równe
=< mniejsze równe

Operacja równości (=:=) a unifikacja (=)

Operator = sprawdza dopasowanie dwóch obiektów i jeśli jest ono możliwe, prowadzi do wiązania zmiennych w tych obiektach (bez obliczania wartości wyrażeń!).
Operator =:= powoduje obliczenie wartości argumentów bez wiązania zmiennych (muszą być one już związane).

Przykład

```
?- 1+2=:=2+1.        ?- 1+A=B+2.
Yes                  A=2
                     B=1

?- 1+2=2+1.          ?- 1+A=:=B+2.
No                   ERROR:Arguments are not
                     sufficiently instantiated
```

Operator "is"

- „is" służy do ukonkretniania występującej po lewej stronie zmiennej przez *wyrażenie arytmetyczne* znajdującą się po prawej stronie.

```
?- X is 2+3.
   X = 5
   Yes
```

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"