Programowanie w prologu
(programming in logic)

- język deklaratywny – określa cel/ wzorzec do osiągnięcia, bez określania „planu dojścia”
- język programowania używany do rozwiązywania problemów dotyczących *obiektów* i *relacji* między nimi

  np. zdanie *jan posiada książkę* opisuje relację *własności* między janem a książką
  posiada(jan, ksiazka)
  relacja argumenty relacji

- Programowanie polega na opisaniu problemu za pomocą **faktów** i **reguł** i osiągnięciu celu poprzez (automatyczne) **wnioskowanie**

2

Programowanie nieimperatywne vs sztuczna inteligencja

- Programowanie funkcyjne, programowanie oparte na logice ignorują architekturę komputera, na którym będą realizowane
- Koncepcja prologu – lata 70 XX wieku
- Wielkie nadzieje na rozwój tzw sztucznej inteligencji
  - komputerowe dowodzenie twierdzeń matematycznych
  - systemy eksperckie (80/90 XX wieku)
- Dziś metody sztucznej inteligencji to raczej
  - algorytmy oparte na obserwacji Natury: ewolucyjne, mrówkowe,
  - obserwacji działania ludzkiego mózgu - sieci neuronowe
  - operowanie na wielkich zbiorach danych (big data) znajdowanie grup o wspólnych cechach, znajdowanie związków, badanie sieci (społecznościowych) , rozszerzone badania statystyczne, eksploracja danych

3

Prolog – obszary zastosowań

- Logika matematyczna
- Przetwarzanie języka naturalnego
- bazy danych
- Automatyzacja projektowania
- Symboliczne rozwiązywanie równań
- Analiza struktur biochemicznych
- Zagadnienia z obszaru sztucznej inteligencji
- Rozwiązywanie zagadek logicznych ☺

4

Program w prologu

- Fakty
- Reguły
- Zapytania o obiekty i związki między nimi

```
/* fakty   */
rodzic(elzbieta, karol).  /* Elżbieta jest rodzicem Karola */
rodzic(filip, karol).
rodzic(elzbieta, anna).
rodzic(filip, anna).
/* reguły */
rodzeństwo(X, Y) :- rodzic(Z, X), rodzic(Z, Y) /* X jest bratem/siostrą Y jeśli mają
     wspólnego rodzica*/
/* zapytania*/
?- rodzic(elzbieta, karol).  odp: true
?- rodzic(X, karol).          odp: elzbieta  filip
?- rodzenstwo(karol, Rodz)  odp: karol  anna
```

Poszukiwanie odpowiedzi na pytanie: przeszukiwanie zbioru faktów i dopasowywanie do wzorca podanego w pytaniu

5

Podstawowe pojęcia

- **term**
  - stała np. elzbieta, posiada, - stałe to teksty pisane małymi literami
  - zmienna np. X Q Glowa Koniec – zmienne zaczynają się od Wielkiej litery
  - struktura np. posiada(jan, ksiazka(Ulisses, Joyce))
- **predykat** – nazwa relacji, predykaty są jedno- lub wieloargumentowe) np. kobieta(elzbieta) , rodzic(elzbieta, karol) , rodzice(elzbieta, filip, karol), prezydent(macron, francja, 2013, 2020)
- **zapytanie**
  - o fakt: rodzice(elzbieta, filip, karol). odp. yes/no
  - Ze zmienną: rodzice(elzbieta, filip, Dzieci). Odp dzieci elzbiety i filipa zapisane w bazie faktów
  - Ze zmienną anonimową rodzice(elzbieta, filip, _). odp yes/no
  - O koniunkcje faktów: rodzice(elzbieta, filip, karol), rodzice(elzbieta, filip, anna) odp yes/no

6

Podstawowe pojęcia cd.

- reguła
  X jest ptakiem jeśli
  X jest zwierzęciem
  X ma pióra

  S jest siostrą Y jeśli
  S jest kobietą
  S i Y mają tych samych rodziców
- reguła zapisana w prologu

```
ptak(X) :- zwierze(X), ma(X, piora).

siostra(S, Y) :- kobieta(S), rodzice(Rm, Ro, S), rodzice(Rm, Ro, Y).
```

- W terminologii zdań logicznych stosowanych w matematyce mamy

$$\forall S, Y, Ro, Rm \ \{kobieta(S) \wedge rodzice(Rm, Ro, S) \wedge rodzice(Rm, Ro, Y)\} \Longrightarrow siostra(S, Y)\}$$

7

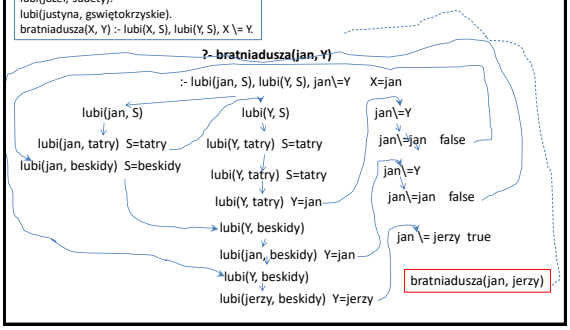
Poszukiwanie rozwiązania w prologu

- Zaczynamy od celu i próbujemy znaleźć ciąg pasujących stwierdzeń, które prowadzą do pewnego zbioru faktów w programie – to jest **dowód** celu
- Stosuje się metodę zstępującą
- Dla stwierdzeń (zapytań) złożonych stosuje się **przeszukiwanie w głąb** – dowodzi się pierwsze od lewej stwierdzenie, następnie przetwarza kolejne podcele.
- Jeśli nie uda się udowodnić jednego z podcelów, porzuca się go i **nawraca** do poprzednich podcelów próbując znaleźć rozwiązania alternatywne
- Przeszukiwanie faktów i reguł odbywa się od początku bazy danych, jednak dla sensownej realizacji nawracania, zaczyna się od faktu następnego po oznaczonym

8

```
lubi(jan, tatry).
lubi(jan, beskidy).
lubi(jerzy, beskidy).
lubi(jerzy, bieszczady).
lubi(józef, sudety).
lubi(justyna, gświętokrzyskie).
bratniadusza(X, Y) :- lubi(X, S), lubi(Y, S), X \= Y.
```

Jak to działa?



9

Jak to działa

Rozważmy program:

```
> lubi(jan, tatry).
  lubi(jan, beskidy).
  lubi(jerzy, beskidy).
  lubi(jerzy, bieszczady).
  lubi(józef, sudety).
  lubi(justyna, gświętokrzyskie).
  bratniadusza(X, Y) :- lubi(X, S), lubi(Y, S), X \= Y.
```

Oraz pytanie: ?- bratniadusza(jan, Y)

ukonkretniamy zmienną X wartością jan otrzymując regułę:
bratniadusza(jan, Y) :- lubi(jan, S), lubi(Y, S), jan\=Y

Szukamy dowodu dla pierwszego celu: lubi(jan, S). odp lubi(jan, tatry)

Zmienna S ukonkretnia się do S = tatry

Szukamy dowodu drugiego podcelu: lubi(Y, tatry) odp lubi(jan, tatry)
czyli Y=jan, co nie spełnia warunku X\=Y

Nawracamy do podcelu: lubi(Y, tatry) startując od faktu lubi(jan, beskidy) – cel nie da się udowodnić

10

Jak to działa cd

```
  lubi(jan, tatry).
  >lubi(jan, beskidy).
  >>lubi(jerzy, beskidy).
  lubi(jerzy, bieszczady).
  lubi(józef, sudety).
  lubi(justyna, gświętokrzyskie).
  bratniadusza(X, Y) :- lubi(X, S), lubi(Y, S), X \= Y.
```

Oraz pytanie: ?- bratniadusza(jan, Y)

Nawracamy do pierwszego podcelu: lubi(jan, S). startując od lubi(jan beskidy) - S może być ukonkretnione S = beskidy

Szukamy dowodu drugiego podcelu: lubi(Y, beskidy) – przeglądamy fakty od początku i trafiamy na lubi(jan, beskidy), ale nie jest spełniony cel jan \= jan

Nawracamy dla drugiego podcelu: lubi(Y, beskidy) startując od lubi(jerzy, beskidy) cel spełniony Y= jerzy oraz jan \=jerzy

Wyprowadzamy Y=jerzy

Prolog w tym momencie „uważa" swoje zadanie za spełnione, chyba, że nakażemy kontynuację, wówczas nawracamy do pierwszego podcelu, ale startując od faktu lubi(jerzy, beskidy) - i tu już nie mamy szans na spełnienie podcelu

11

Inny przykład

Osoba może coś ukraść, jeśli jest złodziejem i lubi coś, co jest wartościowe.

```
/* fakty */
>>/* 1 */ zlodziej(jan).
/* 2 */ lubi(maria, czekolada).   /* predykat lubi zdefiniowany przez fakt */
/* 3 */ lubi(maria, wino).
/* reguły */
/* 4 */ lubi(jan, X) :- lubi(X, wino). /* predykat lubi zdefiniowany przez regułę*/
> /*5 */ moze_ukrasc(X,Y)  :-  zlodziej(X),  lubi(X, Y).
```

Co może ukraść jan?

?- może_ukrasc(jan, X)

12

Poszukiwanie odpowiedzi

- Poszukiwanie klauzuli pasującej do może_ukrasc odp 5 . Prolog oznacza tę klauzulę. Następnie próbuje spełnić jej cele: zlodziej(X) jest ukonkretniany do zlodziej(jan) – X = jan oraz lubi(jan, Y)
- Szukamy wzorca dla celu zlodziej(jan) - odp 1. Następnie szukamy szukamy drugiego podcelu w postaci: lubi(jan, Y)
- Cel lubi(jan, Y) pasuje do głowy klauzuli 4. Aby spełnić regułę, szukamy celu: lubi(X, wino)
- Wzorzec pasujący to 3 – lubi(maria, wino), X ukonkretnia się do maria. Cel klauzuli 4 jest osiągnięty, więc lubi(jan, maria) .
- Mamy więc spełniony drugi podcel z klauzuli 5, a zatem może_ukrasc(jan, maria)

Wnioski z rozumowań mogą być czasem dość nieoczekiwane…

13

Arytmetyka

- Używa się standardowych operatorów arytmetycznych (+, *, /) i porównań (=, >=,..), spójników logicznych (or, and) , które w istocie są predykatami, ale używa się notacji infiksowej

  formalnie: +(1, 2) zapisuje się jednak 1 + 2,
  =(X, Y) zapisuje się X=Y (specyficzne X\=Y nierówne)
- Przykład użycia arytmetyki

```
ludnosc(usa, 203).   /*  203 miliony */
ludnosc(indie, 548).
ludnosc(chiny, 1000).

powierzchnia(usa, 3) . /* miliony kilometrów kwadratowych  */
powierzchnia(indie, 1) .
powierzchnia(chiny, 4).
```

14

Używanie arytmetyki

```
ludnosc(usa, 203).   /*  203 miliony */
ludnosc(indie, 548).
ludnosc(chiny, 1000).

powierzchnia(usa, 3) . /* miliony kilometrów kwadratowych  */
powierzchnia(indie, 1) .
powierzchnia(chiny, 4).
/* gestosc zaludnienia   */
gestosc(X, G) :- ludnosc(X, L), powierzchnia(X, P),  G  is  L/P.
```

Operatory arytmetyczne: +, -, *, /, // dzielenie całkowitoliczbowe, mod reszta z dzielenia

15

Listy

- Oznaczenie listy: [1, 2, 3 ,4], [karol, anna, andrzej, edward]
- W operacjach na listach używa się oznaczeń [Glowa| Ogon] (jak ML) przykłady:

lista	głowa	Ogon
[a, b, c]	a	[b,c]
[]		
[X+Y, x+y]	X+Y	x+y

Listy, struktury i pytania:

```
pojazd( rower, [kolo, kolo, rama, bagaznik]).
pojazd(samochod, [kolo, kolo, kolo, kolo, maska, silnik, bagaznik]).
pojazd(sanki, [ploza, ploza, siedzisko]).
?- pojazd(X, [kolo| Y]).  /* pojazd kołowy */
?- pojazd silnikowy?????
```

16

Dzialania na listach

- Przynależność do listy

```
member(X, [X, _]).   (albo: member(X, [Y, _]) :- X = Y).
member(X, [_ | Lista]) :- member(X, Lista).
```

Użycie: ?- member(lolek, [bolek, lolek, olek, kaska]) odp true

```
?- pojazd(P, L), member(silnik, L).  /* odpowiedź na pytanie z poprzedniej strony */
```

- Łączenie list;

```
append([], Lista,  Lista).
append([G | L1], L2, [G | L3]) :- append(L1, L2, L3).
```

Użycie: ?- append([1, 2, 3], [4, 5, 6], L). Odp L=[1, 2, 3, 4, 5, 6]
?- append([1, 2], [3, 4], [1,2,3,4]). Odp true
Pokaż w debuggerze

17

Działania na listach

- Odwracanie listy.

```
odwroc([], []).
odwroc([G | Og], Lista) :- odwroc(Og, Wyn), append(Wyn, [G], Lista).
```

Użycie: ?- odwroc([1, 2, 3, 4], [4, 3, 2, 1]) odp true
odwroc([1, 2, 3, 4], L) odp L=[4, 3, 2, 1])

- Dlugosc listy

```
dl_listy([], 0).
dl_listy([G|O], N) :- dl_listy(O, N1) , N is N1+1.
```

Użycie: ?- dl_listy([a, b, c, d, e], N). Odp N = 5
dl_listy([1, 2, 3], 5). Odp: false

18