PROLOG

Prolog jest językiem deklaratywnym – deklarujemy fakty w oparciu o nasze deklaracje oczekujemy odpowiedzi na zapytania. Podczas wnioskowania luki pomiędzy wprowadzonymi faktami Prolog stara się sam wypełniać.

Prolog jest językiem programowania w logice:
--> Standard ISO
--> Implementacja SWI Prolog
--> Implementacja GNU Prolog

OBSŁUGA

Używając gprolog (GNU Prolog). Interesują nas dwie komendy:

```
gplc plik_programu.pro
```
która kompiluje plik_programu.pro do postaci wykonywalnej. Uruchamiamy wynikowy plik

```
./plik_programu
```
i lądujemy w konsoli.

```
gprolog
```
która uruchamia konsolę interaktywną. Skróty klawiszowe:

- ^c przerywa działanie (h -- help, e -- exit)
- ^d kończy działanie
- <ENTER> zaakceptowanie wyniku
- ; następny wynik
- a wszystkie wyniki

FAKTY

- nazwy relacji (lubi) i obiektów (jan oraz piwo) muszą zaczynać się małymi literami,
- nazwa relacji poprzedza nazwy objętych nią obiektów,
- obiekty oddziela się przecinkami i ujmuje w nawias okrągły,
- fakt musi kończyć się kropką,
- w nazwach faktów i obiektów nie należy stosować polskich liter,
- fakty są uporządkowane. Poniższy fakt oznacza, że jan lubi piwo, ale nie oznacza, że piwo lubi jana.

```
lubi(jan,piwo).
```

```
lubi(jan,PIWO).   →   BŁĘDNE!!!
Lubi(jan,pIWO).   →   BŁĘDNE!!!
lubi(Jan,pIWO).   →   BŁĘDNE!!!
```

```
luBI(jAn,pIWO).
```

PREDYKAT – nazwa relacji przed nawiasem

ARGUMENT – nazwa obiektu w nawiasie

Nazwy relacji i obiektów są dowolne. Zamiast deklarować:

```
lubi(jan,piwo).
```

równoważne, aczkolwiek znacznie mniej samo-komentujące, byłoby:

```
a(b,c).
```

Relacje mogą mieć dowolną liczbę argumentów

```
gra(jan,maria,warcaby).
```

BAZA DANYCH – to zbiór faktów i reguł używanych w Prologu do rozwiązania pewnego problemu

Deklarowane w prologu fakty nie muszą być prawdziwe w świecie zewnętrznym.

```
slodki(cytryna).
```

ZAPYTANIA

Po zbudowaniu bazy danych możemy zadawać zapytania dotyczące zgromadzonych w niej faktów.

Zapytanie wygląda tak samo jak fakt, poza tym, że poprzedzone jest następującymi po sobie symbolami pytajnika i myślnika (?-)

```
?- posiada(maria,gazeta).
```

Powyższe zapytanie można interpretować jako zapytanie, czy konkretny obiekt maria znajduje się relacji posiadania z konkretnym obiektem gazeta.

Prolog w reakcji na tak postawione zapytanie przeszuka stworzoną wcześniej bazę danych w poszukiwaniu faktów pasujących do tego podanego w zapytaniu:

- Jeżeli fakt zostanie odnaleziony w bazie, prolog odpowie **yes**
- Jeżeli fakt nie zostanie odnaleziony w bazie, prolog odpowie **no**

Odpowiedź **no** nie jest więc odpowiedzią zapewniającą, że maria nie posiada gazety, a jedynie stwierdzeniem, że w bazie nie występują fakty pozwalające jednoznacznie stwierdzić, że maria wspomnianą gazetę posiada.

PRZYKŁAD

```
lubi(jan,ryby).
lubi(jan,piwo).
lubi(maria,czekolada).
lubi(maria,ksiazka).
```

```
?- lubi(jan,pieniadze).
no
?- lubi(jan,ryby).
yes
```

Oczywiście nie możemy kierować zapytań o niezdefiniowane uprzednio relacje, np.

```
?- pije(jan,piwo).          →          BŁĘDNE!!!
```

ZMIENNE

Prolog odróżnia zmienne od nazw obiektów na podstawie nazwy, która dla zmiennych zawsze zaczyna się od wielkiej litery. Zmienne traktujemy jako zastępstwo dla obiektu, którego w danej chwili nie jesteśmy w stanie nazwać.

Zmienna w Prologu może być:

- ukonkretniona – w sytuacji gdy odpowiada jakiemuś konkretnemu obiektowi
- nieukonkretniona – w sytuacji gdy nie wiemy, jakiemu obiektowi może odpowiadać

```
lubi(jan,ryby).
lubi(jan,piwo).
lubi(jan,maria).
```

```
?- lubi(jan,X).
```

Po zadaniu zapytania zmienna X nie jest początkowo ukonkretniona i może być zastępowana dowolnym innym argumentem występującym w faktach (na tej samej pozycji). Prolog przegląda bazę danych (w takiej kolejności, w jakiej ją wpisano – od góry do dołu) w poszukiwania pasujących faktów. W sytuacji, kiedy pasujący fakt zostanie odnaleziony, zmienna X staje się ukonkretniona oraz zostaje oznaczone miejsce w którym znaleziono fakt.

Dla naszego przykładu pierwszym pasującym faktem jest fakt lubi(jan,ryby) i powoduje, że zmienna X od tej pory oznacza obiekt ryby (została ukonkretniona jako ryby).

Prolog w takiej sytuacji pokaże obiekt podstawiony pod zmienną:

```
X = ryby
```

a następnie będzie oczekiwał na naszą decyzję:

- ENTER – odpowiedź nas satysfakcjonuje
- ; - szukaj dalej

Żądanie dalszego wyszukiwania powoduje, że zmienna X ponownie staje się nieukonkretniona. Następują dalsze próby podstawienia, począwszy od miejsca wstawienia poprzedniego znacznika.

ZMIENNA ANONIMOWA

Jest specjalną zmienną oznaczaną przez podkreślenie (_) i oznacza dowolną, nieistotną wartość. Zapytanie czy jan cokolwiek lubi wyglądałoby jak niżej:

```
?- lubi(jan,_).
```

Funktory

, i
; lub
:- jeśli

KONIUNKCJE ,

```
lubi(maria,czekolada).
lubi(maria,wino).
lubi(jan,wino).
lubi(jan,maria).
lubi(jan,spacery).
lubi(maria,spacery).
```

Spójnik i oznaczany przez przecinek (,) oznacza, że interesuje nas koniunkcja celów (chcemy spełnić je wszystkie).

```
?- lubi(jan,maria), lubi(maria,jan).
no

?- lubi(maria,X), lubi(jan,X).
X = wino;
X = spacery
```

W przypadku koniunkcji celów Prolog stara się spełnić je kolejno, od lewej do prawej. W naszym przypadku zachodzi kolejno:
- zmienna X zostaje ukonkretniona jako czekolada
- cel zawodzi (brakuje w bazie lubi(jan,czekolada)), zachodzi **nawracanie** i zmienna X ponownie jest nieukonkretniona
- zmienna X zostaje ukonkretniona jako wino
- odnaleziony zostaje fakt lubi(jan,wino), po czym Prolog oczekuje na naszą decyzję
- ; powoduje ponowne wyszukiwanie

REGUŁY :-

Mając daną bazę:

```
alkohol(piwo).
alkohol(wino).
alkohol(rum).
```

I chcąc wprowadzić fakt, że Jan lubi wszystko, co jest alkoholem musielibyśmy wprowadzić odrębne fakty dla każdego z alkoholi:

```
lubi(jan,piwo).
lubi(jan,wino).
lubi(jan,rum).
tani(piwo).
```

Lub wprowadzić odpowiednią regułę

```
lubi(jan,X) :- alkohol(X).
```

- Reguła to ogólne stwierdzenie dot. obiektów i ich powiązań.
- Reguła składa się z głowy i treści połączonych następującymi po sobie symbolami dwukropka i myślnika (:-)
- Reguły także kończą się kropką

Dla przytoczonego przykładu:
- głową reguły jest *lubi(jan,X)*
- treścią reguły jest *alkohol(X)*

treść reguły może zawierać koniunkcję celów
lubi(jan,X) :- alkohol(X), tani(X)

TERM – stała, zmienna lub struktura. Programy w Prologu składają się z termów.

Termy to wyrażenia

--> proste (stałe i zmienne), np.
jan, maria, ryby, X, Y

--> złożone (termy proste połączone funktorami) – inaczej **STRUKTURY** – pojedynczy obiekt, na który składa się zestaw innych obiektów, nazywanych składnikami struktury.
posiada(jan,ksiazka(harrypotter,rowling)).
W przykładzie powyżej na książkę składają się tytuł i autor.

STAŁE – ich nazwy zaczynają się od małej litery. Nazywają konkretne obiekty lub konkretne relacje.

Istnieją dwa rodzaje stałych:
--> atomy, np.

Rozróżniamy przy tym dwa typy atomów:
- składające się z liter i cyfr
- składające się wyłącznie z symboli

--> liczby, np.
-17, -2.67e2, 0, 1, 99.9 512, 8192, 6.02e-23

UNIFIKACJA

T=S. unifikacja (czyli przyrównanie logiczne)

A=b. Unifikacja pod A zostaje podstawione b
b=A.

jan=maria. Unifikacja unifikacja niemożliwa, dwa różne obiekty nie mogą być równoważne

lubi(X,Y) = lubi (X,b). Y zostanie ukonkretnione jako b
lubi(X,Y) = lubi (Z,V).

ARYTMETYKA is

prawy element is traktowany jest jak wyrażenie,którego wartość należy wyznaczyć

suma(X,Y,Z) :- X is Y+Z.

argumenty(X,Y) = argumenty(5,5), Z is X+Y.

SKŁADNIA

Zad. Sprawdź następujące predykaty (warunki) w odniesieniu do zmiennych i stałych:

var(T). czy T jest zmienną?
nonvar(T). czy T nie jest zmienną?
atom(T). czy T jest stałą, ale nie liczbą?
number(T). czy T jest liczbą?
compound(T). czy T jest termem złożonym?

Zad. Sprawdź odpowiedzi systemu na poniższe zapytania

para(a,b)=para(x,y).
para(a,b)=para(X,Y).
para(a,b)=para(X,Y),Z=para(Y,X).

FUNKCJE vs. RELACJE

Relacja jest podzbiorem iloczynu kartezjańskiego XxY, przy czym jeden element z X może być w relacji z więcej niż jednym elementem z Y.

Funkcja jest relacją, która jest jednoznacznym odwzorowaniem X-->Y.

Prolog operuje nie tylko na funkcjach, ale również na relacjach. Pozwala to na znajdowanie więcej niż jednego wyniku spe³niaj±cego wszystkie zadane warunki.

LISTY

Funktor . (kropka) oznacza dodanie elementu (głowy) do reszty listy (ogona)

SWI Prolog w celu dodania elementu do głowy stosuje zapis [a|[]].

Zapis z kropką	Zapis równoważny
.(a,[])	[a]
.(a,.(b,[]))	[a,b]
.(.(a,[]),.(.(a,.(b,[])),[]))	[[a],[a,b]]

Przykład: równoważność zapisów

```
| ?- [X,Z] = .(.(a,[]),.(.(a,.(b,[])),[])).
```

Przykład: odwrócenie listy

```
L1=[1,2,3,4],L1=[A,B,C,D],L2=[D,C,B,A].
```

Inne operacje na listach

member(X,[1,2,3]).	<==>	L=[1,2,3],member(X,L).
append([1,2,3],[2,3,4],N).	<==>	L=[1,2,3],M=[2,3,4],append(L,M,N)
select(1,[1,2,3],L).	...	

Zad. Wiedząc, że Stefan jest ojcem Romana i Henryka, Roman ma synów: Mariana i Marka, a Henryk synów: Adama i Jana, napisz fakty ojciec(a,b) oraz reguły dziadek (X,Y), wój(X,Y), kuzyn(X,Y), syn(X,Y) bazujące na faktach ojciec(X,Y)

Zad. Stwórz plik tekstowy genealogia.pro o treści:

```
rodzice(uranus, gaia, rhea).
rodzice(uranus, gaia, cronus).
rodzice(cronus, rhea, zeus).
rodzice(cronus, rhea, hera).
rodzice(cronus, rhea, demeter).
rodzice(zeus, leto, artemis).
rodzice(zeus, leto, apollo).
rodzice(zeus, demeter, persephone).

ojciec(X, Y) :- rodzice(X, _, Y).
matka(X, Y) :- rodzice(_, X, Y).

rodzic(X, Y) :- ojciec(X, Y); matka(X, Y).

dziadek(X, Z) :- ojciec(X, Y), rodzic(Y, Z).
babcia(X, Z) :- matka(X, Y), rodzic(Y, Z).
```

ojciec(X, Y) :- rodzice(X, _, Y).	X to ojciec Y gdy X oraz dowolna matka to rodzice Y
matka(X, Y) :- rodzice(_, X, Y).	X to matka Y gdy X oraz dowolny ojciec to rodzice Y
rodzic(X, Y) :- ojciec(X, Y); matka(X, Y).	X to rodzic Y gdy X to ojciec lub matka Y
dziadek(X, Z) :- ojciec(X, Y), rodzic(Y, Z).	X to dziadek Z gdy Z ma rodzica Y i X jest ojcem Y
babcia(X, Z) :- matka(X, Y), rodzic(Y, Z).	X to babcia Z gdy Z ma rodzica Y i X jest matką Y

Skompiluj go i uruchom komendami:

```
gplc genealogia.pro
./genealogia
```

Znajdź rodziców Zeusa poleceniami:

```
| ?- rodzic(Rodzic,zeus).
| ?- matka(Rodzic,zeus).
```

itd.

ZADANIA (LISTY)

Zad. Arytmetyka list.
Poniższe reguły definiują liczbę elementów listy, ich sumę i średnią arytmetyczną.

```
dlugosc(0,[]).
dlugosc(DlListy,[Glowa|Ogon]) :- dlugosc(DlOgona,Ogon), DlListy is DlOgona + 1.
```

- dlugosc listy pustej jest zerowa
- DlListy = DlOgona + 1
- DlOgona to 0 gdy ogon pusty dlugosc(DlOgona,Ogon) = dlugosc(DlOgona,[])
lub rekurencyjnie dlugosc(DlOgona,[glowa_ogonowa|ogon_ogonowy]) gdy niezerowy

mamy warianty:
1. Lista pusta brak głowy i ogona
dlugosc(DlListy, lista) => dlugosc(DlListy,[]) => **DlListy = 0**

2. Lista 1 elem. glowa, brak ogona
dlugosc(DlListy, lista) => *DlListy = DlOgona + 1 (glowa)*
=> dlugosc(DlOgona,[]) => **DlListy = 0 + 1**

3. Lista 2 elem. glowa, ogon
dlugosc(DlListy, lista) => *DlListy = DlOgona + 1 (glowa)*
=> dlugosc(DlOgona,[elem_ogonowy])
=> dlugosc(DlOgona,[]) + 1 (glowa ogonowa)
=> **DlListy = 1 + 1**

4. Dla list o większej liczbie elem. rekurencyjnie

```
suma(0,[]).
suma(SListy,[Glowa|Ogon]) :- suma(SBezGlowy,Ogon), SListy is SBezGlowy + Glowa.
```

Suma SBezGlowy rekurencyjnie wyznaczana jest przez sumę elementów dla listy pozbawionej głowy
SListy to suma elementów wchodzących w skład ogona oraz elementu głowy

```
srednia(Sr,Lista) :- suma(S,Lista), dlugosc(D,Lista), Sr is S/D.
```

Sr to S będące sumą elementów listy podzielone przez D będące ilością elementów listy

Umieść je w pliku arytm_list.pro
Uruchom konsolę gprolog
Wczytaj reguły poleceniem
```
['arytm_list'].
```
Wywołaj je wpisując np.

```
| ?- dlugosc(X,[1,2,3,4]).
| ?- suma(X,[1,2,3,4]).
| ?- srednia(X,[1,2,3,4]).
```

```
+-----------------------------------------------------------+
| LISTA WBUDOWANYCH PREDYKATÓW W GPROLOG                    |
| http://www.gprolog.org/manual/html_node/gprolog024.html   |
+-----------------------------------------------------------+
```

Zad. Stwórz relację kolor dla warzyw wg. schematu:

natka zielona
sałata zielona
papryka zielona
pomidor czerwony
papryka czerwona
cytryna żółta
papryka żółta

Relacja smaczne definiuje zielone warzywa jako smaczne. Sprawdź czy
papryka i cytryna są smaczne?

UWAGA: - kod umieść w pliku warzywa.pro
- otwórz konsolê poleceniem gprolog
- za³aduj program poleceniem ['warzywa.pro']

Zad. Rozwiąż równanie 2+x=4.

UWAGA:
Wartości zmiennych można zawęzić do określonej dziedziny. Polecenia fd_xxxx odnoszą się do Finite Domain Solver. Przykładowo:

fd_domain(X,0,10) zawęzi wartości X do 0,1,2,...,10.

Warunki dla wyrażeń FD:
FD1 #= FD2 ogranicza FD1 do równości z FD2
FD1 #\= FD2 ogranicza FD1 do nierówności z FD2
FD1 #< FD2 ogranicza FD1 do bycia mniejszym niż FD2
FD1 #=< FD2 ogranicza FD1 do bycia mniejszym równym FD2
FD1 #> FD2 ogranicza FD1 do bycia większym niż FD2
FD1 #>= FD2 ogranicza FD1 do bycia większym równym niż FD2

Więcej --> par. 9.6 manuala

Odp.:
| ?- 2+X #= 4.

Zad. Rozwiąż zadanie: cena za 72 pary spodni wyniosła -67,9- zł, gdzie pierwsza i ostatnia cyfra jest nieznana. Ile kosztowała jedna para spodni?

Rozwiązanie: 72 x = k*100 + 67,9 + g
Widać, że g jest parzyste 0,2,4,6,8
Wiemy, że k=1,2,...9
Rozpisujemy wielokrotności 72, dopasowujemy możliwe ceny...

Odp. w GNU prologu:
| ?- fd_domain([K,G],0,9),72*X #= K*10000 + 6790 + G.

G = 2
K = 3
X = 511

yes

Zad. Skonstruuj kwadraty magiczne 3x3.

UWAGA: Można wykorzystać ograniczenia symboliczne, np.:
fd_all_different(L) ogranicza elementy listy L do różnych

Zad. Napisz program obliczający sumę elementów o parzystych indeksach dla podanej listy np. dla zapytania s(X, [1, 2, 3, 7]), program odpowiada X = 9

Zad. Napisz program usuwający z podanej listy co 3 element, np. na zapytanie u(L, [1, 2, 3, 4, 5, 6, 7]) program odpowiada L = [1, 2, 4, 5, 7]