Programowanie w Prolog-u cz. 1

Maciej Hojda

1 Podstawy

Prolog (ang. programming in logic) to *deklaratywny* język programowania (w odróżnieniu od *imperatywnych* jak C czy Python).

Prolog jest rozbudowanym językiem na którym można zrealizować maszynę Turinga (pomijając ograniczoną pamięć), toteż wszystkie klasyczne algorytmy można zrealizować także w Prologu.

Prolog służy do wnioskowania na podstawie wprowadzonych faktów i reguł. Sposób wykorzystania tychże zależy od samego Prologa. Mechanizm wnioskowania w Prologu składa się z dwóch elementów (realizowanych w tle, bez bezpośredniego udziału programisty):

- poszukiwanie – przeszukiwanie bazy wiedzy (faktów i reguł),
- unifikacja – łączenie faktów z wykorzystaniem reguł w celu stwierdzenia czy zapytanie ma prawdziwe wartościowanie zmiennych (i zwrócenie przynajmniej jednego takiego wartościowania).

Np. w bazie wiedzy mamy fakty reprezentujące pogodę w miastach

```
miasto_pogoda(wroclaw, slonecznie).
miasto_pogoda(krakow, deszczowo).
miasto_pogoda(warszawa, slonecznie).
```

Po zadaniu pytania

```
miasto_pogoda(X, slonecznie).
```

Prolog odszuka takie wartości `X`, dla których powyższy fakt jest prawdziwy, czyli zwróci

```
X = wroclaw ;
X = warszawa.
```

Uwaga: faktycznie zwróci tylko pierwszą znalezioną wartość `X`, chyba, że nakażemy mu zrobić inaczej.
Uwaga: średnik robi tutaj za alternatywę, kropka to koniec formuły (przecinek to koniunkcja).

W bazie wiedzy możemy mieć stałe, np.

```
wroclaw_slonecznie.
krakow_deszczowo.
warszawa_slonecznie.
```

a pytanie `wroclaw_slonecznie.` zwróci po prostu prawdę `true.`

Dalej korzystamy z SWI Prolog `www.swi-prolog.org`

2 Polecenia i przykłady

2.1 Tryby

Obsługa prologa z wiersza poleceń odbywa się w dwóch trybach: *definiowania* i *odpytywania*.

W trybie definiowania użytkownik dodaje fakty i reguły do bazy wiedzy.

W trybie odpytywania Prolog stwierdza prawdziwość (lub brak możliwości jej wykazania) formuł podanych przez użytkownika (zwraca też zbiór wartości zmiennych, dla których formuła jest prawdziwa). Wnioskowanie wykonywane przez Prologa odbywa się na podstawie faktów i reguł wprowadzonych w trybie definiowania.

`consult(user).` lub `[user].`	przejście do trybu definiowania (`|:`)
`CTRL+d`	powrót do trybu odpytywania (`?-`)

> polecenia kończymy kropką

2.2 Pomoc

`help.`	wyświetlenie wbudowanej pomocy
`help([polecenie]).`	wyświetlenie pomocy na temat `[polecenie]`

2.3 Fakty

Przykłady faktów.
Uwaga: `|:` oznacza wpisywanie w trybie definiowania, `?-` oznacza wpisywanie w trybie odpytywania. Nie wpisuj `|:` ani `?-` do konsoli.

Dodaj do bazy wiedzy następujące fakty.

```
|: slonecznie.
|: pogoda(slonecznie).
|: miasto_pogoda(wroclaw, slonecznie).
|: miasto_pogoda(krakow, pochmurno).
|: miasto_pogoda(warszawa, deszczowo).
|: miasto_pogoda(lodz, deszczowo).
|: miasto_pogoda(szczecin, slonecznie).
|: miasto_czas_pogoda(wroclaw, poranek, slonecznie).
```

Sprawdź, czy podane formuły są prawdziwe (i dla jakich argumentów).

```
?- slonecznie.
?- pogoda(X).
?- miasto_pogoda(wroclaw, X).
?- miasto_pogoda(_, X).   % wyjaśnij co robi podkreślnik
?- pogoda(wroclaw, _, _). % wyjaśnij na czym polega błąd
```

> do wyświetlania kolejnej wartości służy średnik `;`
> wpisanie kropki `.` kończy wyświetlanie

2.4 Pliki i katalogi

`?- [nazwapliku].`	umieszcza plik `nazwapliku.pl` w bazie wiedzy
`?- working_directory(CWD, b).`	przechodzi do katalogu `b`
`?- working_directory(CWD, CWD).`	zwraca aktualny katalog
`?- edit(file(nazwapliku)).`	otwiera edytor pliku o zadanej nazwie (domyślnie nie zapisuje pliku na dysku)
`?- edit(nazwapliku).`	otwiera do edycji wcześniej utworzony plik

2.4.1 Przyklad

Utwórz i załaduj plik `rodzina.pl` z następującymi faktami.

```
rodzic(jan, tomek).
rodzic(kasia, tomek).
rodzic(jan, magda).
rodzic(kasia, malgorzata).

potomek(jan).
potomek(kasia).

przodek(tomek).
przodek(magda).
przodek(malgorzata).
```

Zweryfikuj utworzoną bazę reguł.

```
?- rodzic(X, Y).
?- rodzic(X, _).
?- potomek(Zmienna).
?- przodek(tomek).
?- syn(jan).
?- dziecko(_).
```

2.5 Analiza kodu

Do analizy kodu w trakcie wykonywania służą polecenia:

```
|: trace.
|: debug.
```

Do wychodzenia z trybów analizy służą polecenia:

```
|: notrace.
|: nodebug.
```

Jeśli chcemy analizować tylko wybrany fakt, to korzystamy ze składni

```
|: trace(pogoda).
|: pogoda(slonecznie).
```

Przez `trace` przechodzimy wpisując `c` lub wciskając `[enter]`. Lista poleceń: `?`.

2.6 Reguły złożone

Konsekwencja jest oznacznana przez `a:-b`, co oznacza: gdy `b` jest prawdą, to `a` również jest prawdą. Wyrażenie `b` to złożona funkcja zdaniowa złożona z funkcji połączonych funktorami *lub* (`;`) oraz *i* (`,`). Dopuszczalne jest też wykorzystanie nawiasów.

Do pliku `rodzina.pl` dodaj następujące formuły.

```
|: dziecko(X):- ojciec(_, X); matka(_, X).
|: syn(X):- dziecko(X), mezczyzna(X).
|: corka(X):- dziecko(X), kobieta(X).

|: ojciec(jan, janek).
|: ojciec(jan, gosia).
|: matka(malgorzata, janek).
|: kobieta(gosia).
|: kobieta(malgorzata).
|: mezczyzna(jan).
|: mezczyzna(janek).
```

Załaduj plik i sprawdź, jakie są odpowiedzi do następujących zapytań.

```
?- corka(X).
?- dziecko(X).
?- dziecko(janek).
?- syn(jan).
?- dziecko(_).
```

2.7 Baza wiedzy i moduły

Do wyświetlania wszystkich faktów i reguł w bazie wiedzy służy polecenie `listing/0`
(gdzie `/0` to liczba argumentów – zero):

```
?- listing.
```

Problem: wyświetla to też niemało predefiniowanych predykatów.

Rozwiązanie: przydziel predykaty do modułu.

Utwórz plik `data.pl`

```
:- module(fun, [fun/1, fun/2]).

fun(cake).
fun(rock, roll).
```

Predykat `module` deklaruje, że plik to moduł o nazwie `fun` z widocznymi predykatami `fun/1` i `fun/2`, gdzie `/1` `/2` to liczba argumentów.

Plik załaduj i przetestuj:

```
?- fun(X).
?- fun(X, Y).

?- listing(fun:_).
?- fun:listing.    % inny zapis powyższego
```

2.8 Operacje matematyczne

Typowe operacje: `+, -, *, /, ^, sin, cos`. Przypisywanie wartości do zmiennych.

```
|: testa(X) :- X is 1 + 2   przypisuje wartość wyrażenia
|: testb(X) :- X = 1 + 2    przypisuje wyrażenie

?- testa(3).
?- testa(1+2).
?- testb(3).
?- testb(1+2).
```

2.9 Rekurencja

```
|: potega(_,0,1).

|: potega(A,B,C) :- B > 0, D is B - 1,
             potega(A,D,E), C is E*A.

?- potega(5, 3, X).
```

3 Zadania

Uwaga: do realizacji poniższych zadań wykorzystaj tylko te elementy języka Prolog, które zostały zawarte na niniejszej liście. W szczególności, nie korzystaj z polecenia `aggregate` i podobnych
(one istnieją i pomagają, ale najpierw zobaczmy jak działa podstawowa składnia).

3.1 Zadanie nr 1

a) Utwórz bazę reguł opisującą rodzinę, a zawierającą relacje (fakty) takie jak:

```
corka(X, Y) % prawda gdy X jest córką Y
syn(X, Y)   % itd.
matka(X, Y)
ojciec(X, Y)
kobieta(X)
mezczyzna(X)
```

Przykład (•):

```
kobieta(Anna).
kobieta(Monika).
matka(Anna, Monika).
% itd.
```

b) Zdefiniuj relacje (predykaty)

```
dziadek(X, Y)
babcia(X, Y)
wnuk(X, Y)
wnuczka(X, Y)
```

w oparciu o relacje zdefiniowane w a).
Np. dziadek może być zdefiniowany przez relacje `ojciec/2` i `matka/2`.

Wypełnij bazę wiedzy tak aby dla każdej relacji istniał przynajmniej jeden zestaw argumentów, dla których ta relacja jest prawdziwa (czyli musi być przynajmniej jeden syn, dziadek itd.).

3.2 Zadanie nr 2

Utwórz (i zrozum) reguły liczące sumy n pierwszych elementów ciągów $n \in \{0, 1, 2, \ldots\}$. Ciągi zdefiniowano rekurencyjnie:

- $f(n+1) = f(n) + f(n-1), f(0) = f(1) = 1$,
- $f(n) = f(n+1) + f(n-2), f(0) = f(1) = f(2) = 2$,
- $f(n+2) = f(n) * n, f(0) = 2, f(1) = 3$,
- indywidualnie podany przez prowadzącego.

3.3 Zadanie nr 3

Na podstawie dokumentacji, zapoznaj się z instrukcjami
`not` (negacja)
`\=` (nierówność)
`!` (cięcie).

Napisz przykładowe programy (predykaty + zapytania), które zobrazują działanie tych instrukcji.

3.4 Zadanie nr 4*

Niech dana będzie relacja utworzona z wykorzystaniem predykatów jak w zadaniu nr 1. Dany jest graf, którego wierzchołki to osoby a krawędzie to połączenia relacjami `corka, syn, matka, ojciec`. (czyli dla przykładu • wierzchołki Anna i Monika są połączone krawędzią).

Utwórz regułę sprawdzającą, czy między **każdą** parą wierzchołków w tak utworzonym grafie istnieje ścieżka, tzn. czy mamy jedną rodzinę czy jest ich więcej. Reguła ma działać poprawnie dla dowolnej rodziny, której graf jest lasem.