Podstawy Prolog-a cz. 2

Maciej Hojda

1 Stałe, a predykaty

Próby bezpośredniej deklaracji stałej:

```
?- x = 5.  % fałsz, bo nie ma faktu "x = 5"
?: x is 5. % fałsz, bo x <> 5
?: X := 5. % fałsz, bo nie ma odpowiedniego faktu w bazie
|: x is 5. % błąd - tutaj faktycznie chcemy redefiniować "is"
|: x := 5. % błąd - tutaj faktycznie chcemy redefiniować ":="
|: x = 5.  % "działa" ale czy faktycznie

% jedyne co zdefiniowaliśmy, to prawdziwość wyrażenia "x := 5".
?- x is 5. % prawda
?- x is 4. % fałsz
?- X := 5. % prawda dla X = x.
?- X := 4. % fałsz
```

Jak utworzyć coś w rodzaju zmiennej/stałej? Trzeba to zrobić w predykacie.

```
|: stala(X) :- X is 5.
?= stala(X).
```

2 Listy

W Prologu listy tworzymy składnią `[e1, e2, e3, ...]`. Np.

```
|: aaa(X) :- X = [1, 2, 3, 8, 9, 0].
|: bbb(X) :- X = [x, y, z, 4, 5, 6].

?- aaa(X).
?- bbb(X).
```

Specjalna składnia pozwala wyróżnić pierwszy element listy (głowę): `[G|O]` gdzie `G` to głowa, a `O` to pozostałe elementy listy.

Przykłady reguł i faktów operujących na listach.

```
|: polacz([], X, X).
|: polacz([X|Y], V, [X|W]) :- polacz(Y, V, W).
|: odetnij([X|Y], X, Y).

?- polacz([a, b, c], [4, 5, 6], X).
?- odetnij([1, 2, 3, 4, 5], X, Y).
```

(dokładnie przeanalizuj działanie tych procedur w trybie *trace*)

3 Śledzenie

Śledzenie pozwala na ustawienie breakpointów (miejsc zatrzymania programu).

```
|: dodaj2(0, X) :- X is 2.
|: dodaj2(1, X) :- X is 3.
|: dodaj2(2, X) :- X is 4.
|: dodaj2(3, X) :- X is 5.
|: dodaj3(0, X) :- X is 3.
|: dodaj3(1, X) :- X is 4.
|: dodaj3(2, X) :- X is 5.
|: dodaj3(3, X) :- X is 6.
|: suma_i_dodaj5(X, Y, Z) :- dodaj2(X, A), dodaj3(Y, B), Z is A + B.

?- spy(dodaj3).
?- suma_i_dodaj5(1, 2).

?- spy(dodaj2).
?- nospy(dodaj3).
?- notrace.
?- suma_i_dodaj5(1, 2).
?- nodebug.
```

4 Formatowanie tekstu

Funkcja `format` organizuje tekst (i liczby) w jedną całośc.

```
?- format('~w ~w ~D abdc ~w', [veni, vidi, 1000, vici]).
?- format(atom(S), '~w ~w ~D abdc ~w', [veni, vidi, 1000, vici]).
```

(pierwsza linijka tyko wyświetla, druga robi również przypisanie do identyfikatora/atomu S)

```
|: info([X|Y], S) :- length(Y, 0), format(atom(S), '~w.', [X]).
|: info([X|Y], S) :- not(length(Y, 0)), info(Y, V),
                     format(atom(S), '~w and ~w', [X, V]).
|: info(X) :- info(X, S), format('~w', S).

?- info([one, two, three]).
```

(**uwaga:** jeśli jest wystąpi „operand expected", to zmień apostrofy na napisane ręcznie.)

Wybrane parametry `format`

∼d oraz ∼D	liczba	∼e oraz ∼E	liczba w notacji wykładniczej
∼i	pomiń następny	∼n	nowy wiersz

5 Przekazywanie predykatów przez argument

```
|: dodaj(X, Y, Z) :- Z is X + Y.
|: odejmij(X, Y, Z) :- Z is X - Y.
|: suma_lub_roznica(A, X, Y, Z) :- call(A, X, Y, Z).

?- suma_lub_roznica(dodaj, 10, 5, X).
```

6 Zadania

Uwaga: do realizacji zadań wykorzystaj tylko te elementy języka Prolog, które zostały zawarte na niniejszej i poprzedniej liście. W szczególności, nie korzystaj z polecenia `aggregate` i podobnych.

6.1 Zadanie nr 1

Utwórz bazę reguł opisującą rodzinę zawierającą przynajmniej 10 członków. Baza reguł powinna zostać utworzona z wykorzystaniem tylko i wyłącznie relacji:

```
malzenstwo(X, Y), dziecko(X, Y), kobieta(X).
```
(*)

Upewnij się, że w bazie są takie osoby jak pradziadek, prababcia, babcia, dziadek, wuj, ciotka, kuzyn, kuzynka, ojciec, matka, brat, siostra, bratowa, szwagier, syn, synowa, córka, zięć (ale jeszcze nie dodawaj reguł do ich wyłuskiwania).

Diagram poprzyj rysunkiem drzewa genealogicznego rodziny.

6.2 Zadanie nr 2

Korzystając z faktów (*), utwórz reguły definiujące wszystkie relacje wymienione w poprzednim zadaniu (pradziadek, prababcia, babcia, dziadek, itd.), np.

- `dziadek(X, Y)` - jest prawdziwa, gdy `X` jest w relacji (tu: dziadek) z `Y`,

Dodatkowo, utwórz reguły zwracające listę osób w relacji j.w., np.

- `dziadek(X)` - jest prawdziwa, gdy `X` to lista wszystkich osób w danej relacji (tu: dziadek) z kimkolwiek.

Czyli `dziadek(X, Y)` jest prawdziwy, gdy `X` jest dziadkiem `Y`, a `dziadek(X)` jest prawdziwy, gdy `X` to **lista** (lista w rozumieniu jak wcześniej) wszystkich dziadków w rodzinie.

6.3 Zadanie nr 3

Utwórz reguły zwracające:

- listę wszystkich przodków,
- listę wszystkich potomków,
- liczbę wszystkich przodków,
- liczbę wszystkich potomków.

dla każdego wskazanego członka rodziny (konkretnego, nie dla wszystkich członków jednocześnie).

6.4 Zadanie nr 4

Utwórz regułę wyświetlającą listę wszystkich potomków zadanego członek rodziny, wraz z opisem relacji.

Np. „ adam (syn), agnieszka (córka), julia (synowa) ”

6.5 Zadanie nr 5

Rozwiąż przydzielone zadanie indywidualne, np. utwórz predykat tworzący listę i zliczający osoby zadane przez prowadzącego (np. braci, sióstr, osób bez wstępnych, osób bez zstępnych, bezpośrednich krewnych osoby itd.).

Zadanie wykonaj z wykorzystaniem list.

6.6 Zadanie nr 6*

Utwórz regułę podającą odległość między dwiema osobami, rozumianą jako (minimalną) długość ciągu relacji „malzenstwo" i „dziecko", które prowadzą od pierwszej osoby, do drugiej.

Utwórz regułę zwracającą wspomniany ciąg relacji.

Wskazówka (do wszystkich zadań)

Jeśli `niejestw(X, L)` zwraca prawdę, gdy lista `L` nie zawiera elementu `X`, a `osoba(X)` zwraca prawdę, gdy `X` to dowolna osoba w rodzinie (oba predykaty łatwo zdefiniować), to predykaty

```
dodaj(L, M) :- osoba(X), niejestw(X, L), dodaj([X|L], M), !.
dodaj(M, M).
```

zwrócą listę wszystkich osób, przy wywołaniu

```
dodaj([], M).
```