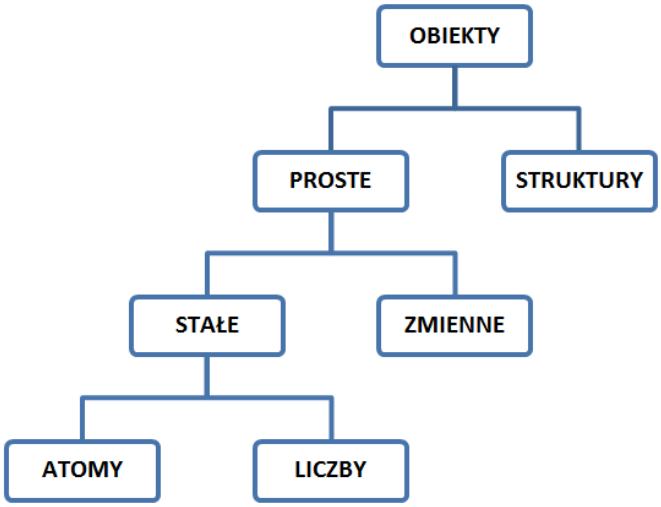
Podstawowe elementy języka Prolog



Termy

Program Prologu składa się z termów. Wyróżniamy cztery rodzaje termów:

- atomy (ang. atoms),
- liczby (ang. numbers),
- zmienne (ang. variables)
- termy złożone (ang. compound terms).

Atomy i liczby wspólnie określane są jako stałe (ang. constants). Zbiór złożony z atomów i termów złożonych nazywany jest tez zbiorem predykatów. Każdy term zapisywany jest jako ciąg znaków pochodzących z następujących czterech kategorii:

- duże litery: A-Z
- małe litery: a-z
- cyfry: 0-9
- znaki specjalne: % + - * / \ ~ ^ < > : . ? @ # $ &

Zbiór ten uzupełnia znak podkreślenia (_), który zwykle traktowany jest jak litera.

Atomy

Atom jest ciągiem znaków utworzonym z:

- małych i dużych liter, cyfr i znaku podkreślenia z zastrzeżeniem, ze pierwszym znakiem musi być mała litera, np. jas, a, aLA, x_y_z, abc
- dowolnego ciągu znaków ujętego w apostrofy, np. 'To tez jest atom'. Wykorzystujemy ten sposób do nazywania stałych zgodnie z zasadami gramatycznymi tzn. zaczynając nazwę z dużej litery.
- symboli, np. ?- lub :- .

Wybrane termy wbudowane

- **read(X)** – odczyt termu z klawiatury,
- **write(X) –** wypisanie termu na ekran,
- **nl** – przejście do nowego wiersza,
- **open('nazwa_pliku',read,X)** – odczyt pliku do X
- **open('nazwa_pliku',write,X)** – zapis X do pliku

- **consult(X)** – jeśli X jest nazwa pliku, wczytuje klauzule i cele
- **['nazwa_pliku']** – działanie identyczne z consult
- **var(X)** – cel nie zawodzi jeśli X jest zmienną nieukonkretnioną,
- **nonvar(X)** - cel nie zawodzi jeśli X jest zmienną ukonkretnioną,
- **atom(X)** – nie zwodzi jeśli X jest atomem,
- **number(X)** – nie zawodzi, jeśli X jest liczbą,
- **atomic(X)** – nie zawodzi jeśli X jest liczbą lub atomem,
- **listing(A)** – wypisuje wszytkie klauzule dla predykatu A,
- **asserta(X**), **assertz(X) –** dodaje klauzule do bazy danych, asserta dodaje na początek, a assertz na koniec bazy,
- **retract(X)** – pozwala na usuwanie klauzul z bazy danych,

Liczby

W SWI-Prologu dostępne są zarówno liczby całkowite jak i rzeczywiste
-17, 23, 99.9, 123e-3

Zmienne

Zmienna jest ciągiem znaków utworzonym z małych i dużych liter, cyfr i znaku podkreślenia z zastrzeżeniem, ze pierwszym znakiem musi być duża litera lub znak podkreślenia, np.

```
X, Kto, _123, X_1_2, _
```

Ostatni z wymienionych przykładów, pojedynczy znak podkreślenia, to tak zwana zmienna anonimowa. Korzystamy z niej zawsze wtedy, gdy interesuje nas tylko czy cos jest prawda, ale zupełnie nie interesuje nas co, np.

Czy ktoś lubi Jasia?

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

Należy pamiętać, ze wielokrotnym wystąpieniom zmiennej anonimowej w jednym wyrażeniu mogą być przypisane różne wartości.

```
?- a(1,2)=a(X,Y).
X = 1,
Y = 2
?- a(1,2)=a(X,X).
No
?- a(1,2)=a(_,_).
Yes
```

Struktury

Term złożony, inaczej struktura, to obiekt złożony z innych obiektów, czyli atomów, liczb, zmiennych a także innych termów złożonych. Termy złożone maja postać:

`funktor(arg_1,...,arg_n)`

gdzie arg_1,. . . ,arg_n są termami, natomiast **funktor** jest atomem (nazwa relacji). Korzystając z możliwości zagnieżdżania innych termów w termach złożonych, możemy lepiej opisać interesujący nas problem.

Fakty:

```
posiada(piotr,auto).
posiada(marcin,auto).
```
pozwalają jedynie stwierdzić, ze obiekty piotr i marcin związane są relacją posiada z obiektem auto, czyli mówiąc „normalnie”, Piotr i Marcin maja auto. Trudno powiedzieć, jakie to jest auto i czy przypadkiem to nie jest to samo auto. Zapisując te fakty inaczej
```
posiada(piotr,auto(nissan,almera)).
posiada(marcin,auto(fiat,punto)).
maAuto(X) :- posiada(X,auto(_,_)).
```
możemy zapytać o cos bardziej szczegółowego, np. marka i model
```
?- maAuto(piotr).
Yes
?- posiada(piotr,auto(X,Y)).
X = nissan,
Y = almera
```

Operatory

Każde wyrażenie arytmetyczne to też pewna struktura. Funktorami w tym przypadku są działania +, *, itp. Przy omówieniu struktur zostało wspomniane, że w Prologu obowiązuje notacja prefiksowa. Jednak tworząc wyrażenia arytmetyczne możemy skorzystać ze zwykłej notacji infiksowej (operator pomiędzy argumentami).

Używając operatorów arytmetycznych należy pamiętać o pozycji, na której występują, priorytecie i łączności. W prologu mamy następujące priorytety wykonania:

OPERATOR	PIORYTET
+ -	1
* /	2
mod	3
- unarne	4

Operator „- unarne” oznacza liczbę ujemną np. -5. Oczywiście programista może zdefiniować swoje własne operatory, podając priorytet ich wykonania.
```
op(500, yfx, $).
```
Powyższy zapis oznacza definicję operatora o nazwie $, z priorytetem wykonania 500. Zapis `yfx` mówi, że zdefiniowany został operator infiksowy.

Łączność operatorów określa kolejność wykonania działań, jeśli pojawiają się wątpliwości w tym względzie wystarczy posłużyć się większą ilością nawiasów.

Arytmetyka

Operacje arytmetyczne przydają się do porównywania liczb i obliczania wyników wyrażeń. Prolog zawiera predykaty wbudowane pozwalające porównywać liczby. Predykaty te wyliczają wartości termów, traktując je jako wyrażenia algebraiczne. Argumentami tych predykatów mogą być zmienne ukonkretnione liczbami całkowitymi lub liczby zapisane jako stałe, mogą być też ogólniejszymi wyrażeniami algebraicznymi.

Predykaty porównujące liczby mogą być zapisywane infiksowo:

`X =:= Y`	X i Y są tą samą liczbą
`X =\= Y`	X i Y są różnymi liczbami
`X < Y`	X jest mniejsze od Y
`X>Y`	X jest większe od Y
`X =< Y`	X jest mniejsze lub równe Y % zapis nie jest błędny !!! %
`X >= Y`	X jest większe lub równe Y

Przykład sesja prologowa:

```
?-2=:=2.
true.
?-2=:=3.
false.
?-2=\=3.
true.
?-2=<3.
true.
?-2>3.
false.
?-2<X.
ERROR: </2: Arguments are not sufficiently instantiated
```

Do wyliczenia wartości wyrażenia arytmetycznego używa się predykatu ***is***.

Do jakich operatorów arytmetycznych można użyć po prawej stronie operatora is, zależy od używanego systemu. Wszystkie implementacje Prologu obsługują:

- `X + Y` suma X + Y
- `X - Y` różnica X – Y
- `X * Y` iloczny X i Y
- `X / Y` iloraz X i Y
- `X // Y` całkowity iloraz X przez Y
- `X mod Y` reszta z dzielenia X przez Y

Przykład predykatu dodaj:

```
dodaj(X,Y,Z) :-
     Z is X + Y.
```

Z samej definicji X i Y muszą być ukonkretnione

Przykład. Niech baza wiedzy zawiera następujące fakty:

```
samochód(fiat_uno, 30, 5.5, lpg).
samochód(ford_escort, 70, 6, benzyna).
samochód(vw_golf, 65, 7, diesel).
```

W powyższych faktach pierwszy argumenty to w kolejności: nazwa samochodu, pojemność baku, zużycie paliwa na 100 km, typ paliwa zasilającego. Chcąc obliczyć możliwy zasięg do przejechania na pełnym baku musimy wykorzystać operator „is”. Przykładowa reguła obliczająca maksymalny zasięg ma postać:

```
ilekm(X, Km) :-
     samochód(X, Y, Z, _),
     Km is Y/Z*100.
```

Wpisując zapytanie:
```
? - ilekm(fiat_uno, Km).
```
Otrzymujemy odpowiedź:
```
Km = 545.455
```

Równość i uzgadnianie

Dosyć ważnym predykatem, jest infiksowy operator „=”.
Kiedy staramy się spełnić cel:
```
?- X = Y.
```
Prolog stara się dopasować X i Y, a jeśli się to uda, cel jest uzgodniony. O unifikacji można myśleć jako o próbie uczynienia X i Y równymi. Predykat równości jest predykatem wbudowanym, co oznacza, że jest z góry zdefiniowany w Prologu. Predykat ten działa tak, jakby był zdefiniowany za pomocą faktu
```
X = X.
```
Jeśli mamy cel w postaci X=Y, gdzie X i Y są dowolnymi termami mogącymi zawierać zmienne nieukonkretnione, czy X i Y są równe sprawdza się następująco:

- Jeśli X jest zmienną nieukonkretnioną, a Y nie jest powiązana z żadnym termem, to X i Y są równe. Poza tym X ukonkretniana jest wartością Y. Na przykład poniższe zapytanie zawiedzie, poza tym zmienna X zostanie ukonkretniona strukturą

```
jedzie(student,rower).
?- jedzie(student,rower) = X.
```

- Liczby całkowite i atomy zawsze są sobie równe.

Przykłady:
```
policjant = policjant uzgodniony
papier = kredka zawodzi
1066 = 1066 uzgodniony
1206 = 1580 zawodzi
```

- Struktury są sobie równe, jeśli mają taki sam funktor oraz taką samą liczbę składników, zaś odpowiadające sobie składniki są sobie równe. Na przykład poniższe zapytanie nie zwiedzie, poza tym zmienna X zostanie ukonkretniona wartością rower.

```
jedzie(student,rower) = jedzie(student,X).
```
Operatorem działającym przeciwnie do równy jest:
X \= Y

Zadania do samodzielnego rozwiązania

Zadanie 1. Sprawdź działanie oraz wysuń wnioski.

Sprawdzić działanie:
```
?- X is 2 + 2.
?- Y is 2.5 + ( 4 / 2).
?- Z is 2 + 0.001.
```
Uwaga:
```
?- A is 3.
?- B is A + 4.
?- A is 3, B is A + 4.
```

Operacje arytmetyczne:

```
?- X is 2 + 2.
?- X is 2 * 3.
?- X is 4 / 2.
?- X is 4 / 3.
?- X is 4 // 3.
```

Uwaga na ,,podstawianie":

```
?- X is 2 + 5.
?- X = 2 + 5.
?- 2 + 5 =:= 1 + 4.
?- 2 + 5 =:= 3 + 4.
?- 2 + 5 =:= 4 + 4.
```

Przećwiczyć użycie operatorów:

```
?- 2 < 3.
?- 2 > 3.
?- 3 > 3.
?- 3 >= 3.
?- 3 =< 3.
```

Zadanie 2.

Zaprojektuj strukturę danych weryfikującą poprawność składni prostych zdań orzekających. Dla uproszczenia ogranicz się do trzywyrazowych zdań składających się wyłącznie z rzeczowników i czasowników.

- Utwórz bazę wiedzy zawierającą kilkanaście przykładowych słów (rzeczowniki i czasowniki).
- Skonstruuj regułę weryfikującą poprawność składni zdania (Po podaniu dowolnego trzywyrazowego zdania składającego się ze znajdujących się w bazie wiedzy rzeczowników i czasowników program powinien odpowiedzieć czy zdanie jest napisane poprawnie, czy też błędnie).

Zadanie 3.

Rozbuduj bazę wiedzy z poprzedniego ćwiczenia tak, aby obsługiwała pięciowyrazowe zdania orzekające składające się z rzeczowników, czasowników i przymiotników.

Zadanie 4.

Baza wiedzy zawiera fakty postaci:

```
długość(kontener1, 20).
szerokość(kontener1,30).
wysokość(kontener1, 15).
długość(kontener2, 25).
szerokość(kontener2,9).
wysokość(kontener2, 10).
```

Napisz regułę obliczającą objętość dowolnego kontenera z bazy wiedzy.

Zadanie 5.

Znajdź NWD dwóch liczb: dla dwóch liczb całkowitych, dodatnich X i Y, największy wspólny dzielnik D:

- równa się X, jeżeli X i Y są równe,
- równa się największemu wspólnemu dzielnikowi X i Y-X, jeżeli X<Y,
- równa się największemu wspólnemu dzielnikowi Y i X-Y, jeżeli Y<X.