5.6.1 Meta-Programming: Terms & Atomic Formulas

Dienstag, 27. Juni 2017 09:30

5.6. Meta-Programming

Prolog can operate on terms, formulas, and even programs (in particular, a program can also operate on its own clauses).

5.6.1. Handling of Terms and Atomic Formulas

Pre-defined predicates for inspection and manipulation of terms and formulas, e.g., number/1 (see Sect. 5.1)

- var(t) is true iff t is an uninstantiated variable

```
?- var(X).
true

?- X=2, var(X).
false
```

- nonvar(t) is true iff t is not a variable.

```
?- nonvar(a).
true

?- X=2, nonvar(X).
X=2

?- nonvar(X).
false
```

- atomic(t) is true iff t is a fct/pred symbol of arity 0 or a number

```
?- atomic(a).
true

?- atomic(2).
true

?- atomic(X).
false

?- atomic(a(a)).
false
```

- compound(t) is true iff t is a term or an atom.

formula that is not just a symbol of arity 0 or a number or a variable

```
?- compound(a).
false

?- compound(X).
false

?- compound(a(a)).
true

?- compound(1+2).
true
```

These predicates only <u>recognize</u> certain terms.
The next pred. can also create or modify terms:

Idea: every term $f(a,b)$ (← f/2) could also be written as a list $[f, a, b]$. This transformation is done by the pre-def. pred. symbol $=..$ (f/0 →)

$t =.. l$ iff

l is the list representation of the term t

```
?- f(a,b) =.. L.
L = [f, a, b]

?- 1+2 =.. L.
L = [+, 1, 2]
```

```
?- T =.. [f, a, b].
T = f(a, b)

?- T =.. [f].
T = f
```

One must not ask queries where the leading fct. symbol or its arity are not uniquely determined.

```
?- X =.. Y.
error
```

```
?- X =.. [Y, a, 5].
error
```

```
?- X =.. [f | L].
error
```

=.. only converts between term and list representation on the top level:

```
?- p(f(X), 2, g(X,Y)) =.. L.
L = [p, f(X), 2, g(X,Y)]
```

arguments are not converted to lists

Ex. for the use of =..

Program for representation of geometrical figures:

```
square(Side)
rectangle(Side1, Side2)
triangle(Side1, Side2, Side3)
circle(Radius)
   :
```

We want to implement a predicate enlarge/3 where enlarge(Fig, Factor, NewFig) should hold if NewFig results from Fig by enlarging it by Factor.

Naive Solution:

```
enlarge(square(Side), Factor, square(NewSide)) :- NewSide is Factor * Side.
enlarge(rectangle(S1, S2), Factor, rectangle(NewS1, NewS2)) :-
        NewS1 is Factor * S1, NewS2 is Factor * S2.
        .
        .
        .
```

Drawback: many similar clauses that essentially do the same, all geometrical figures have to be known when implementing "enlarge".

Improved Solution:

```
enlarge(Fig, Factor, NewFig) :- Fig =.. [Type | Param],
        multiplylist(Param, Factor, NewParam),
        NewFig =.. [Type | NewParam].

multiplylist([], _, []).
multiplylist([X | L], Factor, [NewX | NewL]) :-
        NewX is Factor * X, multiplylist(L, Factor, NewL).
```

There are more predicates to construct or deconstruct terms:

functor/3 and arg/3

functor(t, f, n) is true iff f is the leading fct. symbol of the term t and the arity of f is n.

```
?- functor(g(f(X),X,g), F, N).
F = g, N = 3

?- functor(T, g, 3).
T = g(X, Y, Z).
```

arg(n, t, a) is true iff a is the n-th argument of the term t (counting of arguments starts with index 1).

```
?- arg(3, g(f(X),X,g), A).
A = g
```

```
?- functor(D, date, 3),
   arg(1, D, 3),
   arg(2, D, 7)
   arg(3, D, 2017)
```

```
?- arg(X, Y, 1).
error
```

arg(3, D, 2017).

D = date(3, 7, 2017)

Now we can implement meta-predicates ourselves.

Implement ground/1 where ground(t) is true iff t is a ground term (i.e., if t does not contain variables).

```
ground(T) :- nonvar(T),
             T =.. [Functor | Arglist],
             groundlist(Arglist).

groundlist([]).
groundlist([T | Ts]) :- ground(T), groundlist(Ts).
```