5.6.1 Meta-Programming: Terms & Atomic Formulas

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
  
  \[
  \text{?- var(X).} \quad \text{?- X=2, var(X).}
  \]

  \[
  \text{true} \quad \text{false}
  \]

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

  \[
  \text{?- nonvar(a).} \quad \text{?- X=2, nonvar(X).} \quad \text{?- nonvar(X).}
  \]

  \[
  \text{true} \quad X=2 \quad \text{false}
  \]

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

  \[
  \text{?- atomic(a).} \quad \text{?- atomic(2).}
  \]

  \[
  \text{true} \quad \text{true}
  \]

  \[
  \text{?- atomic(X).} \quad \text{?- atomic(a(a)).}
  \]

  \[
  \text{false} \quad \text{false}
  \]

- `compound(t)` is true iff `t` is a term or an atom.
formulas 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 recognize certain terms.
The next pred. can also create or modify terms:

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

\[ t = [.. l \iff l \text{ is the list representation of the term } t \]

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

?- \( 1 + 2 = .. \) \( L \).
\( L = [+, 1, 2] \)

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

?- \( T = .. \) \( [f] \).
\( T = f \)
One must not ask queries where the leading fact symbol or its arity are not uniquely determined.

?- X =.. Y.  
  error

?- X =.. E Y, a, b].  
  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 \texttt{enlarge/3} where \texttt{enlarge(Fig, Factor, NewFig)} should hold if \texttt{NewFig} results from \texttt{Fig} by enlarging it by \texttt{Factor}.

**Naive Solution:**

\begin{align*}
\text{enlarge(square(Side), Factor, square(NewSide))} : & \text{ NewSide is Factor \times Side.} \\
\text{enlarge(rectangle(S1, S2), Factor, rectangle(NewS1, NewS2))} : & \text{ NewS1 is Factor \times S1, NewS2 is Factor \times S2.}
\end{align*}

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

**Improved Solution:**

\begin{align*}
\text{enlarge(Fig, Factor, NewFig)} : & \text{ Fig = \ldots \{Type | Param\},} \\
& \text{multiplylist(Param, Factor, NewParam),} \\
& \text{NewFig = \ldots \{Type \mid NewParam\}.}
\end{align*}

\begin{align*}
\text{multiplylist([I, -], [I]).} \\
\text{multiplylist([X | L], Factor, [NewX | NewL]):} & \text{ NewX is Factor \times X, multiplylist(L, Factor, NewL).}
\end{align*}
There are more predicates to construct or deconstruct terms:

\[ \text{functor/3 and arg/3} \]

\[ \text{functor}(t, f, n) \text{ is true iff } f \text{ is the leading symbol of the term } t \text{ and the arity of } f \text{ is } n. \]

\[ ?- \text{functor}(g(f(X),X,g), F, N). \]
\[ F = g, \ N = 3 \]

\[ ?- \text{functor}(T, g, 3), \]
\[ T = g(X, Y, Z). \]

\[ \text{arg}(n, t, a) \text{ is true iff } a \text{ is the } n\text{-th argument of } \]
\[ \text{the term } t \text{ (counting of arguments starts with index 1)}. \]

\[ ?- \text{arg}(3, g(f(X),X,g), A), \]
\[ A = g \]

\[ ?- \text{functor}(D, date, 3), \]
\[ \text{arg}(1, D, 3), \]
\[ \text{arg}(2, D, 7), \]
\[ \text{arg}(3, D, 2017). \]

\[ ?- \text{arg}(X, Y, N). \]
\[ \text{error} \]
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).

\[
ground(T) :\text{-- nonvar}(T),
T = \ldots [\text{Functor}(\text{Arglist}),
groundlist(\text{Arglist})]
\]

\[
groundlist([T]).
groundlist([T | Ts]) :\text{-- ground}(T), groundlist(Ts).
\]