Introduction to Prolog. Unification. Sicstus Prolog

In this introductory session you will get acquainted with the Prolog data types and unification rules.

0.1 Prolog Data Types

Prolog's single data type is the *term*. Terms are either: *atoms*, *numbers*, *variables* or *compound terms (structures)*. The figure below presents a classification of the data types in Prolog:

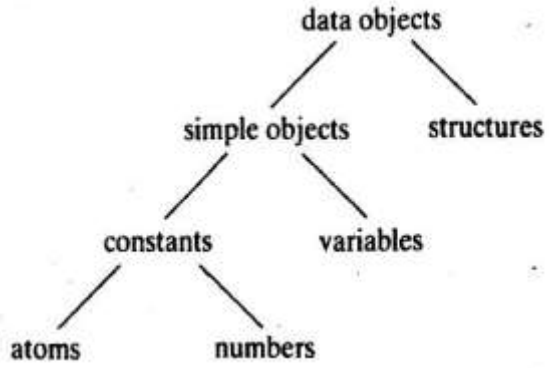

Figure 1 – Prolog data types[1]

0.1.1 Atoms and numbers

An atom is a general-purpose name with no inherent meaning. It is composed of a sequence of characters that is parsed by the Prolog reader as a single unit. Atoms are usually bare words in Prolog code, written with no special syntax. However, atoms containing spaces or certain other special characters must be surrounded by single quotes. Atoms beginning with a capital letter must also be quoted, to distinguish them from variables. The empty list, written [], is also an atom. Other examples of atoms include:

x, blue, 'Taco', and 'some atom'.

Numbers can be integers or real numbers (not used very much in typical Prolog programming).

0.1.2 Variables

Variables are denoted by a string consisting of letters, numbers and underscore characters, and beginning with an upper-case letter or underscore. Variables closely

[1]Ivan Bratko – *Prolog Programming for Artificial Intelligence*

resemble variables in logic in that they are placeholders for arbitrary terms. A variable can become instantiated (bound to equal a specific term) via unification. A single underscore (_) denotes an anonymous variable and means "any term". Unlike other variables, the underscore does not represent the same value everywhere it occurs within a predicate definition.

0.1.3 Compound Terms

A compound term is composed of an atom called a "functor" and a number of "arguments", which are again terms. Compound terms are ordinarily written as a functor followed by a comma-separated list of argument terms, which is contained in parentheses. The number of arguments is called the term's arity. An atom can be regarded as a compound term with arity zero.

Examples of compound terms are: truck_year('Mazda', 1986) and 'Person_Friends'(zelda,[tom,jim]). Users can declare arbitrary functors as operators with different precedence to allow for domain-specific notations. The notation f/n is commonly used to denote a term with functor f and arity n.

Special cases of compound terms:

- **Lists** are defined inductively. The list [1, 2, 3] would be represented internally as '.'(1, '.'(2, '.'(3, []))). A syntactic shortcut is [H | T], which is mostly used to construct rules. A list can be processed by processing the first element, and then the rest of the list, in a recursive manner.

  Lists can be constructed and deconstructed in a variety of ways:
  - Element enumeration: [abc, 1, f(x), Y, g(A,rst)]
  - Prepending single element: [abc | L1]
  - Prepending multiple elements: [abc, 1, f(x) | L2]
  - Term expansion: '.'(abc, '.'(1, '.'(f(x), '.'(Y, '.'(g(A,rst), [])))))

- **Strings**

0.2 Prolog Unification

- **Atoms** unify if and only if they are the same atom.
- **Numbers** unify if and only if they are the same number.
- **Strings** unify if and only if they are the same string.
- **Lists** unify if and only if
  1. their heads unify, and
  2. their tails unify.
- **Structures** unify if and only if
  1. their names unify,
  2. they have the same number of arguments, and
  3. their arguments unify.

- **Variable** *V* unifies with **Term** *T* just in case one of the following conditions is satisfied:
    - *V* is an instantiated variable.
        - If *T* is not a variable, then *V* and *T* unify if and only if the term instantiated on *V* unifies with *T*.
        - If *T* is an instantiated variable, then *V* and *T* unify if and only if the term instantiated on *V* unifies with the term instantiated on *T*.
        - If *T* is an uninstantiated variable, then *V* and *T* are unified by instantiating on *T* the term that is instantiated on *V*.
    - *V* is an uninstantiated variable.
        - If *T* is not a variable, then *V* and *T* are unified by instantiating *T* on *V*.
        - If *T* is an instantiated variable, then *V* and *T* are unified by instantiating on *V* the term that is instantiated on *T*.
        - If *T* is an uninstantiated variable, then *V* and *T* unify and become synonyms for the same variable.

Exercise 1.1: Check Sicstus Prolog manual for:

a. Terms (4.1.2)
b. Compound Terms (4.1.3)
c. Unification (4.8.1.2)

0.3 Quiz exercises

0.3.1 Which is the nature of the following Prolog terms:

a. X	d. hello	g. [a, b, c]
b. 'X'	e. Hello	h. [A, B, C]
c. _138	f. 'Hello'	i. [Ana, are, 'mere']

0.3.2 Look up the following built-in predicates in the Sicstus Prolog manual: var(Term), nonvar(Term), number(Term), atom(Term), atomic(Term) – section 4.8.1.1 *Type Checking*. Test the validity of your answers from exercise 1.3.2, by using them.

0.3.3 Execute the following unification queries. Explain the results in a text file:

a. ?- a = a.
b. ?- a = b.
c. ?- 1 = 2.
d. ?- 'ana' = 'Ana'.
e. ?- X = 1, Y = X.
f. ?- X = 3, Y = 2, X = Y.
g. ?- X = 3, X = Y, Y = 2.
h. ?- X = ana.
i. ?- X = ana, Y = 'ana', X = Y.
j. ?- a(b,c) = a(X,Y).
k. ?- a(X,c(d,X)) = a(2,c(d,Y)).
l. ?- a(X,Y) = a(b(c,Y),Z).
m. ?- tree(left, root, Right) = tree(left, root, tree(a, b, tree(c, d, e))).
n. ?- k(s(g),t(k)) = k(X,t(Y)).
o. ?- father(X) = X.
p. ?- loves(X,X) = loves(marsellus,mia).
q. ?- [1, 2, 3] = [a, b, c].
r. ?- [1, 2, 3] = [A, B, C].
s. ?- [abc, 1, f(x) | L2] = [abc | T].
t. ?- [abc, 1, f(x) | L2] = [abc, 1, f(x)].