Lesson #2: About Prolog

What's It All About?

1. Five ideas that serve well to characterize Prolog are presented.
2. The syntactic forms that constitute the basis of programs in Prolog, called "terms", are defined by means of BNF.
3. The notion of pattern matching, called "unification" in Prolog, is introduced.
4. Two very different perspectives of Prolog programming are presented, the "procedural" perspective and the "declarative" perspective.

Five Things to Know about Prolog

1. Prolog is a computer programming language that was invented by Alan Colmereaur in Marseille, France, in 1971. The name stands for "Programming in Logic" (the French version of that phrase).
2. Prolog was motivated by an interest in performing natural language processing (NLP), and it is closely associated with a variant of context free grammar (CFG) known as definite clause grammar (DCG). The interesting thing about this grammar is that its rules can be automatically translated into parsers on which semantic procedures can be hung.
3. The featured data type in Prolog is the relation, which happens to be one of the standard knowledge representations associated with mental representation and cognitive processing. Thus, in Prolog, the featured data type is a classic knowledge representation.
4. Computation in Prolog takes the form of logical inference. On the surface, Prolog programs take the form of facts and rules. Under the hood, Prolog operates on sets of Horn clauses (Afred Horn) according to Robinson's Resolution Principle (John Alan Robinson).
5. Programming in Prolog can be viewed as as two step process: (1) Establish a knowledge base, and (2) query the knowledge base. The knowledge base will consist of structured facts and rules.

Prolog Terms

Prolog programs are made up of Prolog terms. There are three kinds of terms in Prolog: constants, variables, and compound terms.

1. The simplest kind of term is a constant: an integer, a real, or a symbolic atom.
   (a) An integer is a string of digits, possibly preceded by a plus sign or a minus sign. For example, the following are integers: `121`, `+4`, `-96`, `0`.
   (b) A real number can take the form of an optional plus or minus sign, followed by a sequence of digits, followed by a dot, followed by another sequence of digits. For example, the following are integers: `1.44`, `+5.9`, `-101.6666`, `0.0`.

(c) A symbolic atom is a string of characters starting with a lower case letter, or a string of special symbols. For example, the following are symbolic atoms: `apple`, `r2d2`, `***`, `[]`.

2. A variable is a name beginning with an upper case letter or the underscore. For example, the following are variables: `Result`, `Number`, `A1`, `_`, `_this_or_that`.
3. A compound term takes the form of a of a symbolic atom followed by a left parenthesis, followed by a comma separated list of terms, followed by a right parenthesis. For example: `number(three)` and `name(cookie, monster)` are compound terms.

BNF Description of Terms

```
<term> ::- <constant> | <variable> | <compound-term>
<constant> ::- <integer> | <real> | <atom>
<compound-term> ::- <atom> ( <term-list> )
<term-list> ::- <term> | <term> , <term-list>
```

Unification

Prolog makes extensive use of "pattern matching", which in Prolog is called "unification". Two terms are said to **unify** if there is some way of binding their variables that makes them identical. For example:

1. The terms
   `friends(elmo,Friend)`
   `friends(elmo,abby)`
   can be unified by binding the variable `Friend` to the atom `abby`.
2. The terms
   `friends(Friend,abby)`
   `friends(elmo,abby)`
   can be unified by binding the variable `Friend` to the atom `elmo`.

Perspectives on Prolog Programming

Prolog programs can be looked at in two very different ways:

1. procedurally
2. declaratively

In the **procedural** take, you think thoughts like: "To prove `color(X)`, find a value of `X` for which `color(X)` is true", *mindful of Prolog's model of execution.* This means, in turn, that you will have to find a value of `X` for which `primary(X)` is true, or a value of `X` for which `secondary(X)` is true.

In the **declarative** take, you think thoughts like: "For all `X`, if `primary(X)` is true, then `color(X)` is true, or if `secondary(X)` is true, then `color(X)` is true." That is, *you think in terms of logical content, not program execution.*

This dual perspective property of Prolog is highly valued, and is considered to be one of the charms of the language.