**Objective: Working with PROLOG (a Logic Programming) language associated with AI.**

**Notes:**

Prolog is a general-purpose logic programming language associated with artificial intelligence and computational linguistics.

Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is intended primarily as a declarative programming language: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations.

In Prolog, program logic is expressed in terms of relations, and a computation is initiated by running a query over these relations. Relations and queries are constructed using Prolog's single data type, the term. Relations are defined by clauses.

Prolog (and other logic programming languages) particularly useful for database, symbolic mathematics, and language parsing applications. Because Prolog allows impure predicates, checking the truth value of certain special predicates may have some deliberate side effect, such as printing a value to the screen.

The difference between facts and rules is that rules are conditional, and use Prolog's "if" operator.

Mainly three operators in Prolog we use are:

|  |  |
| --- | --- |
| Operator | Meaning |
| :- | if |
| , | and |
| ; | or |

1. ***Knowledge Base Queries-***

Knowledge Base is simply a collection of facts and rules. Facts are used to state things that are unconditionally true of some situation of interest, and Rules state information that is conditionally true of the situation of interest.

**Example 1-**

we can state that Mia, Jody, and Yolanda are women, that Jodyplays air guitar, and that a party is taking place, using the following five facts:

woman(mia). woman(jody). woman(yolanda).

playsAirGuitar(yolanda). party.

**QUERIES:**

1. woman(jody).



1. playsAirGuitar(jody).



1. party.

Example 2-

Here is KB2, our second knowledge base:

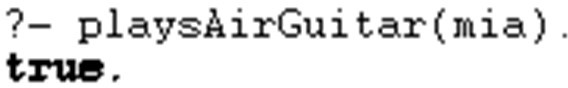
happy(yolanda). listens2Music(mia).

listens2Music(yolanda):- happy(yolanda). playsAirGuitar(mia):- listens2Music(mia). playsAirGuitar(yolanda):- listens2Music(yolanda).

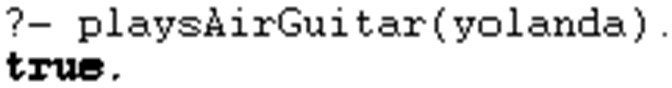
There are two facts in KB2, listens2Music(mia) and happy(yolanda) . The last three items it contains are rules.

**QUERIES:**

1. playsAirGuitar(mia).



1. playsAirGuitar(yolanda).



***Types of terms in Prolog-***

Prolog's single data type is the term. Terms are either atoms, numbers, variables or compound terms.

* An **atom** is a general-purpose name with no inherent meaning. Examples of atoms include x, red, 'Taco', and 'some atom'.
* **Numbers** can be floats or integers. ISO standard compatible Prolog systems can check the Prolog flag "bounded". Most of the major Prolog systems support arbitrary length integer numbers. Their Prolog syntax is the obvious one: 23 , 1001 , 0 , -365 , and so on.
* **Variables** are denoted by a string consisting of letters, numbers and underscore characters, and beginning with an upper-case letter or underscore. Variables closely resemble variables in logic in that they are placeholders for arbitrary terms. For example, X , Y , Variable , \_tag , X\_526 , List , List24 , \_head , Tail , \_input and Output are all Prolog variables.
* 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. An examples of compound terms is 'Person\_Friends'(zelda,[tom,jim]), hide(X,father(father(father(butch))))

Special cases of compound terms:

A **List** is an ordered collection of terms. It is denoted by square brackets with the terms separated by commas or in the case of the empty list, []. For example, [1,2,3] or [red,green,blue].

**Strings**: A sequence of characters surrounded by quotes is equivalent to either a list of (numeric) character codes, a list of characters (atoms of length 1), or an atom depending on the value of the Prolog flag double\_quotes. For example, "to be, or not to be".

1. ***Operators and Arithmetic:***

# **Arity -**

You have probably noticed that Prolog's error messages always refer to a predicate name along with a number; for example, likes/2 . The number given with each predicate is called its arity.

The arity of a predicate is simply the number of arguments it takes.

The reason Prolog always refers to the arity is that Prolog allows you to have different predicates with the same name, but different arity. Thus you could define two totally different predicates with the same name but a different number of "parameters"; when you called one of them, Prolog would count the number of arguments, and reference the appropriate definition. It's not really a good idea to do this (as it can be confusing), but it might help explain some seemingly strange errors in your input!

# **Comments –**

As you write more knowledge bases, you may want to comment them for your own reference; two forms of comment are allowed in Prolog:

1. The character "%" followed by any sequence of characters up to end of line.
2. The symbols "/\*" followed by any sequence of characters (including new lines) up to "\*/"

**Simple I/O in Prolog –**

We'll be looking at I/O in a little more detail in later tutorials, but for the moment you should know about the following predicates:

* nl which moves to a new line on screen
* write(X) which writes X on screen

# **Built-In Predicates**

The built-in arithmetical predicates are the obvious ones: <, >, >=, =<, = etc. A simple example of their use would be the following two predicates:

positive(N) :- N>0.

non\_zero :- N<0 ; N>0

Note that Prolog's "=" relation is equality (not assignment); it is the same as the "==" relation in C.

**Arithmetic Operators**

Prolog also has arithmetic operators like +, -, \*, / and also the usual collection of functions like sqrt, exp, cos. However, these do not work exactly as expected!

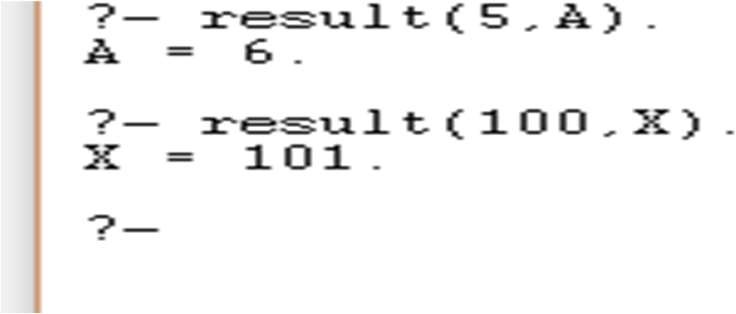
The important point here is to realise that writing "2+3" in Prolog is not an instruction to carry out the addition (remember, Prolog is not an imperative language). Rather it represents "the addition of 2 and 3". It is thus a completely different term to "1+4", or "3+2", and certainly different from "5\*1" etc.

1. **Exercises –**

Define predicates to calculate the following:

1. The result of adding 1 to a number-

result(X,Y) :- Y is X+1.



1. Maximum of 2 numbers-

maximum(X,Y,X) :- X>Y. maximum(X,Y,Y) :- Y>X.

