## Prolog

## Prolog is a declarative programming language. This means that in prolog, we do not write out what the computer should do line by line, as in procedural languages such as C or Java. The general idea behind declarative languages is that we describe a situation. Based on this code, the interpreter or compiler will tell us a solution. Programming in Prolog is very different from programming in a traditional procedural language like Pascal etc. In Prolog we don't say how the program will work. Prolog (PROgramming LOGic) rose within the realm of Artificial Intelligence (AI). It originally became popular with AI researchers, who know more about "what" and "how" intelligent behavior is achieve dects.

## Prolog can be separated in two part

#### The Program

The program, sometimes called Database is a texted file (\*.pl) that contain the facts and rules that will be used by the user of the program. It contains all the relations that make this program.

#### The Query

When we launch a program we are in query mode. This mode is represented by the sign? - At the beginning of the line. In query mode we ask questions about relations describing the program.

#### Loading a program

Loading first we have to launch our Prolog compiler, for this report we used the SWI-Prolog which is a freeware .When Prolog is launched the ?- should appear meaning we are in query mode. The manner to launch a program depends of our compiler. For SWI-Prolog we can load a program by typing the command [file]. Then the file of our program is file”.pl”. If our compiler is not SWI-Prolog we can also try the command reconsult(file). When we have done this we can use all the facts and rules that are contained in the program.

## Backtracking

## Fail

## The fail/1 predicate is provided by Prolog. When it is called, it causes the failure of the rule. And this will be forever; nothing can change the statement of this predicate. We can ask us what its utility is. We can associate it to the cut/1 predicate, described in the next subsection in this report. It allows us to include the negation in a rule. A typical use of fail is a negation of a predicate. We can resume the fail with this scheme

goal(X) :- failure(X),!,fail.

goal(X).

failure(X) are the conditions that make goal(X) fail.

* Not

For logical purity, it is always possible to rewrite the predicates without the cut/1. This is done with the built-in predicate not/1. Some claim this provides for clearer code, but often the explicit and liberal use of 'not' clutters up the code, rather than clarifying it. When using the cut/1, the order of the rules becomes important. Let's try to rewrite compare\_cut\_2/1 with the not/1 predicate.

#### not\_2(X) :- X = pentiumIII.

not\_2(X) :- not(X = pentiumIII).

We can now see the difference between not/1 and cut/1, but the code is here to simple to see really the difference of clarity. Try with compare\_cut\_3:

#### not\_3(X,Y) :- X = pentiumIII,data(Y).

not\_3(X,Y) :- not(X = pentiumIII),not(Y).

Now, we can imagine what will be the difference between the algorithms based on the cut/1 and those on the not/1. The result is the same; it's just a way of thinking. It is interesting to note that not/1 is defined using the cut/1. It also uses call/1, another built-in predicate that calls a predicate.

#### not(X) :- call(X), !, fail.

not(X).

**How to Query**

Once we have entered the facts in a program we can ask prolog about it. An example program can be:

eats(fred,oranges). /\* 'Fred eats oranges' \*/

eats(tony,apple). /\* 'Tony eats apple' \*/

eats(john,apple). /\* 'John eats apple' \*/

If we now ask some queries we would get the followings things:

?- eats(fred,oranges).

yes /\* yes, that matches the first clause \*/

?- eats(john,apple).

Yes

**Characteristics of PROLOG**

There are a number of reasons why Prolog is so suitable for the development of advanced software systems:

* Logic programming:

Prolog is a logical language; the meaning of a significant fraction of the Prolog language can be completely described and understood in terms of the Horn subset of first-order predicate logical (Horn-clause logic). So, this is the mathematical foundation of the language, explaining its expressiveness.

* A single data structure as the foundation of the language:

Prolog offers the term as the basic data structure to implement any other data structure. The entire language is tailored to the manipulation of terms.

* Simple syntax:

The syntax of Prolog is much simpler than that of the imperative programming languages. A Prolog program is actually from a syntactical point of View simply a sequence of terms. For example, the following Prolog program

p(X) :- q(X).

Corresponds to the term:- (p(X), q(X)). Whereas the definition of the formal syntax of a language such as Java or Pascal requires many pages, Prolog's syntax can be described in a few lines. Finally, the syntax of data is exactly the same as the syntax of programs.

* Program-data equivalence:

Since in Prolog, programs and data conform to the same syntax, it is straightforward to interpret programs as data of other programs, and also to take data as programs. This feature is of course very handy when developing interpreters for languages.

* Incremental program development:

Normally, a C or Java program needs to be developed almost fully before it can be executed. A Prolog program can be developed and tested incrementally. Instead of generating a new executable, only the new program fragments need to be interpreted or compiled. It is even possible to modify a program during its execution. This offers the programmer the possibility of incremental program development, a very powerful and exible approach to program development. It is mostly used in research environments and in the context of prototyping.

**Advantages**

* Logic based languages are able to represent the real world more accurately.
* Prolog is able to derive new rules from the existing rules contained within the knowledge base.
* It has built-in list handling, very useful for representing sequences, trees, and so on.
* It is easy to read and write programs which build structures.
* Although Prolog is not a complete implementation of logic, it is much closer to it than C, and is more like mathematical notation.
* It is easy to build tables and databases while a program is running, as I did with “memo\_fib”. You don't need a lot of programming effort.

**Disadvantages**

* It can be very difficult to design a database that accurately represents relationships.
* Prolog is not best suited to solving complex arithmetical computations.
* Prolog programs are not best suited to the current PC architecture (sequential execution) and are best optimized on parallel architectures.
* It tempts you to write things that look logically correct, but that won't run.
* The obvious way to write a predicate is unlikely to be efficient. We must know when a predicate needs to be optimized.
* Because it lacks functional notation, predicates can become cumbersome.

f( N, F ) :-

N1 is N - 1,

f( N1, F1 ),

F is N \* F1.

It would be much nicer to write

f( N ) = N \* f(N-1).

* Input and output is not always easy.
* There are some features which have not been standardized, and differ between implementations. For example: formatted input and output, file-handling, sorting predicates.
* You can't re-assign to parts of data structures. This makes it impossible to implement arrays. However, functional programmers have developed a number of fast-access data structures which do almost as good a job.

**LISP**

LISP is originated by John McCarthy in 1959 as an implementation of recursivefunction theory.

The first language to have:

* Conditionals- if-then-else constructs
* Recursion
* Typed values rather than typed variables
* Garbage collection
* Programs made entirely of functional expressions that return values
* A symbol type
* Built-in extensibility

Most of these features have gradually been added to other languages

**LISP** **Applications**

AI programs often need to combine symbolic and numeric reasoning. Lisp is the best language I know for this.

* Writing SHOP (my group's AI planning system) took a few weeks in Lisp
* Writing JSHOP (Java version of SHOP) took several months
* Lisp is less used outside of AI, but there are several well-known LISP applications:
* AutoCAD - computer-aided design system
* Emacs Lisp - Emacs's extension language
* IT a Software's airline fare shopping engine - used by Orbitz
* Parasolid - geometric modeling system
* Remote Agent software - deployed on NASA's Deep Space 1 (1998)
* Script-Fu plug-in for GIMP (GNU Image Manipulation Program)
* Yahoo! Merchant Solutions - e-commerce software