INTRODUCTION TO PROLOG

Many developers prefer to use programming languages such as Prolog and Lisp. Both languages have a basis which cannot be traced back to the early ideas of computing as state change of variables, as holds for the imperative languages, but instead to more abstract, mathematical notions of computing. It is this formal foundation which explains why these two languages are enormously expressive, much more than the imperative languages. As a consequence, it is straightforward to create and manipulate complex data structures in these languages, which is not only convenient but really essential for AI, as AI people tend to solve complicated problems. Although much of what will be said in this introduction also holds for Lisp, we will not pay further attention to this latter programming language.
Prolog is frequently used for the development of interpreters of particular (formal) languages, such as other programming languages, algebraic or logical languages, and knowledge representation languages in general.

Although Prolog and AI are often mentioned in one phrase, this does not mean that Prolog is only suitable for the development of AI programs. In fact, the language has and is being used as a vehicle for database, programming language, medical informatics and bioinformatics research. And finally, the language is used by many people simply to solve a problem they have to deal with, resulting in systems that from the user's point of view are indistinguishable from other systems, as in particular commercial Prolog systems offer extra language facilities to develop attractive GUIs.2

LOGIC PROGRAMMING

Logic programming is a programming paradigm based on mathematical logic. In this paradigm the programmer specifies relationships among data values (this constitutes a logic program) and then poses queries to the execution environment (usually an interactive interpreter) in order to see whether certain relationships hold. This style of programming is popular for data base interfaces, expert systems, and mathematical theorem provers.

ADVANTAGES OF USING 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 (lists, arrays, trees, records, queues, etc.). 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.

WEAK TYPING: Types of variables in Prolog do not have to be declared explicitly. Whereas in C or Java, the type of any object (for example int, real, char) needs to be known before the compiler is able to compile the program, the type of a variable in Prolog only becomes relevant when particular operations are performed on the variable. This so-called weak typing of program objects has as a major advantage that program design decisions can be postponed to the very last moment. This eases the programmer's task. However, weak typing not only comes with advantages, as it may make finding program bugs more difficult. Expert programmers are usually able to cope with these problems, while in addition most Prolog systems come with tools for the analysis of programs which may be equally useful for finding bugs.

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 excitable approach to program development. It is mostly used in research environments and in the context of prototyping.

EXTENSIBILITY: A Prolog system can be extended and modified. It is even possible to modify the Prolog language, and to adapt both its syntax and semantics to the needs. For example, although Prolog is not an object-oriented language, it can be extended quite easily into an object-oriented language, including primitives for inheritance and message passing. Compare this to the task of modifying a Java interpreter into a Prolog compiler, something which would require an almost complete redesign and re-implementation of the original Java system.

SWI-Prolog

What is SWI-Prolog

If you want to use Prolog you need a compiler. There are many compiler downloadables on internet. For this report we used SWI-Prolog. SWI-Prolog is free Prolog compiler licensed under the GPL, targeting primarily the research and education.

Author

SWI-Prolog is designed and implemented by Jan Wielemaker from the University of Amsterdam, department of Social Science Informatics (SWI).

Platforms

SWI-Prolog is written in ANSI C and should configure for most Unix machines having an ANSI C Compiler.
Ports have been made for :

1. MS Win32
2. MS Win3.1
3. MS DOS
4. OS/2

FTP Sites

The main ftp-site to download SWI-Prolog is :
ftp://swi.psy.uva.nl/pub/SWI-Prolog/

Latest version of the sources :
ftp://swi.psy.uva.nl/pub/SWI-Prolog/pl.tar.gz

Latest binary for use with with Win32 :
ftp://swi.psy.uva.nl/pub/SWI-Prolog/w32pl.exe

You can obtain more informations about SWI-Prolog by visiting the site :
http://www.swi.psy.uva.nl/projects/SWI-Prolog/

Books about Prolog

These subjects have been described extensively in the literature. See *Bratko, 1986*, *Sterling & Shapiro, 1986*, and *Clocksin & Melish, 1987*. For more advanced Prolog material see *O'Keefe, 1990*. Syntax and standard operator declarations confirm to the `Edinburgh standard'. Most built in predicates are compatible with those described in *Clocksin & Melish, 1987*. SWI-Prolog also offers a number of primitive predicates compatible with Quintus Prolog (1) *Qui, 1997* and BIM_Prolog (2) *BIM, 1989*. ISO compliant predicates are based on ``Prolog: The Standard'', *Deransart et al., 1996*, validated using *Hodgson, 1998*.

Compliance to the ISO standard

SWI-Prolog 3.3.0 implements all predicates described in ``Prolog: The Standard'' *Deransart et al., 1996*.

Exceptions and warning are still weak. Some SWI-Prolog predicates silently fail on conditions where the ISO specification requires an exception (functor/3 for example). Some predicates print warnings rather than raising an exception. All predicates where exceptions may be caused due to a correct program operating in an imperfect world (I/O, arithmetic, resource overflows) should behave according to the ISO standard. In other words: SWI-Prolog should be able to execute any program conforming to *Deransart et al., 1996* that does not rely on exceptions generated by errors in the program.