Using Prolog for Developing Real World Artificial Intelligence Applications

Athanasios Tsadiras
Technological Educational Institute of Thessaloniki, Greece

INTRODUCTION

Artificial Intelligence Applications are becoming crucial for enterprises that want to be successful by having the advantage of using high information technology. The development of such applications is assisted by the use of high level computer programming languages that are closer to the programmer than to the computer. Such a programming language is Prolog.

Prolog is a logic programming language (Clocksin & Mellish 2003) that was invented by Alain Colmerauer and Phillipe Roussel at the University of Aix-Marseille in 1971. The name *Prolog* comes from *programmation en logique* (i.e., "programming in logic" in French). Together with LISP, they are the most popular Artificial Intelligence programming languages. Prolog was generated by an attempt to develop a programming language that extensively uses expressions of logic instead of developing a program by providing a specific sequence of instructions to the computer. Theoretically, it is based on a subset of first-order predicate calculus that allows only Horn clauses (Bratko, 2000). The control of the program execution is based on Prolog's built-in search mechanism that in fact is an application of theorem proving by first-order resolution.

BACKGROUND

Prolog has a clear contribution in solving a series of **Artificial Intelligence** (**AI**) problems (Sterling & Shapiro, 1986). There are many features that make **Prolog** suitable as a programming language for developing AI applications (Luger, 2002). Some of them are:

- **Declarative nature:** Programming in Logic using Prolog, remove the imperative (serial order) nature of other languages and allow programmer to solve a problem by describing the problem itself rather than defining a solution. The programmer writes programs by declaring the facts and the rules that apply to the problem in hand and then makes queries in order for Prolog to return valid solutions. This high level of abstraction makes **Prolog** suitable for AI applications where the programmers should give emphasis on the problem itself rather than on the computer idiosyncratic commands that they should impose to the computer system. **Prolog**'s built-in search mechanism takes the control of the program execution, leaving the programmer to concentrate on the problem itself.
- **Backtracking:** Even when a search path ends at a dead end, the backtracking mechanism of **Prolog** retreats back down the search path to try another path. This feature makes Prolog exceptionally suitable for a number of search problems that AI faces. It also provides the additional advantage of finding more than one solution of the problem, in the case that backtracking is forced, after finding a first solution. Because a great number of AI problems can be represented as a problem of finding the right path in the search space, the built-in depth first mechanism of Prolog, accompanied with backtracking, make **Prolog** suitable for such applications.
- **Don't care and don't know nondeterminism:** In the execution of a Prolog program, the nondeterminism feature is really apparent. Although the rule that will execute is the first one that matches the goal, we can ask for more than one solution. Using the backtracking mechanism, alternative rules will apply and other valid solutions will be found. This introduces: a) "don't know nondeterminism" implying that all possible ways to find the solutions will be followed because the execution does not know how to find the solution, or b) "don't care nondeterminism" meaning that we just need one solution and we do not care which solution is that among the many that exist. Both of these types of nondeterminsm are considered useful in AI applications, especially for those dealing with logic.
- **Recursion**, instead of **Iteration:** Because iteration constructs are not provided in **Prolog**, recursion should be used instead. The simplicity of solving problems recursively makes Prolog programs smaller and understandable even when coping with large, real life **AI** problems. Additionally, many AI problems are recursive in nature, increasing the suitability of Prolog.
- **List handling mechanism:** The data structure of List is very important for handling AI problems (e.g., LISP which is also used for solving AI problems, stands for

"LISt Processing"). Lists are built-in in Prolog while in most other languages they are not, making faster and easier the writing of Prolog programs that require list handling. List's recursive nature allows the extensive use of recursion in problem solving, providing an additional advantage for solving AI problems with Prolog.

- **Pattern matching** and **unification:** The use of unification to find the most general common instance of two formulas or patterns makes pattern matching a build-in feature of **Prolog**. This intelligent feature can assist AI problem solving where in many cases the decisions that are made are based on situation matching. This Prolog ability can be found really helpful in specific areas of AI, such as natural language processing, computer vision and intelligent database search.

PROLOG AND ARTIFICIAL INTELLIGENCE APPLICATIONS

The features described above make Prolog suitable for developing applications that solve **AI** Problems. Such an area is that of **Decision Support Systems**. Rules that support decisions can be expressed as Prolog rules, declaratively in pure logic, making development and maintenance of these systems mush easier. An example of a successful decision support system is the "Options Trading Analysis System" (OTAS, Lassez, McAloon, & Yap, 1987), used for the analysis of stock options and investment strategies. OTAS automatically generates and analyzes investment strategies based on standard vertical option combinations. Its main elements are: a portfolio maintenance module that creates and updates portfolios and provides expert recommendations, a numeric database containing stock market data, a symbolic database containing rules describing standard options combinations and a linear algebra module for the analysis of options combinations.

Except from financial decisions, medical decisions can be supported by similar systems written in Prolog. The OSM medical decision support system for general practitioners (Fox, Glowinski, Gordon, Hajnal, & O'Neill, 1990) is such an example. OSM supports a number of knowledge and information retrieval functions, providing the user with rapid access to textual information from text sources, knowledge bases or patient database. The system incorporates a version of the Oxford Textbook of Medicine, a 300-author general medical reference work, and uses its indexes to retrieve text.

Decision support systems can also take the form of a computer based advisor. For example, the RoadWeather Pro (Reiter, 1991) is used as an Expert Weather Advisor written in Prolog that permits mouse point-and-click manipulation of weather "objects," thereby allowing forecast upgrades based upon recent observational data received from sensors or human observers. This decision support system estimates weather-related effects on highway maintenance operations, as well as on airports, transportation, recreational activities, agribusiness and so forth. Its purpose is to be used as a user-interactive 24-hour weather prediction system for snow and ice control.

Another major **AI** area that Prolog contributes is that of **Natural Language Processing.** Pattern matching capabilities and the declarative nature of grammar definitions, make Prolog a handy and powerful tool for processing natural language. For example, CAT2 (Sharp, 1991) is a unification-based natural language processing system, designed for analysis, generation and translation of natural language sentences. CAT2 is used for multilingual machine translation and for automatic translation of informative texts although the emphasis has been on European Commission texts, as well as general purpose texts. It embodies a particular formalism for natural language processing, as well as a grammar development environment. Grammars have been written for English, German, French, Spanish, with experimental versions for Russian, Greek, and Japanese.

The Logic-based Machine Translation system, LMT (McCord, 1982, 1986) is another such application. This is a machine translation system for translating English to German. The system is based on a grammatical formalism called Modular Grammars based on slot filling techniques, which includes some automatic semantic translation and handling of metagrammatical rules. The principle aim of the system is to translate computer manuals from English into German.

Among the various **AI** applications using Prolog, many **Knowledge-based Systems** can be found. This is because, in most of the cases, knowledge in such systems is expressed in the form of rules. These rules can be easily expressed in **Prolog** syntax. After that, by using unification and Prolog search mechanism, inference can be done. AGATHA Electronic Diagnosis Knowledge Based System (Allred, Lichtenstein, Preist, Bennett, & Gupta, 1991) is such a system. Its aim is to test and diagnose complex printed circuit boards. Agatha uses a suite of smaller subsystems, each one of them customized to diagnose a particular kind of test, something that is necessary due to the diversity and complexity of the various tests. Agatha could reason about the test results as well as suggest further tests to run.

Moreover, Gunga Clerk (Woodin, 1989) is an example of a Legal Knowledge-based System written in Prolog. This system is a substantive legal knowledge-based advisory system in New York State Criminal Law, advising on sentencing, pleas, lesser included offences and elements. Gunga Clerk is designed to accept key facts of a criminal case and provide guidance to attorneys and judges as to statutory rules affecting sentence parameters, regulation of plea bargaining, identification of lesser included offences,