Logic Programming Languages for Expert Systems

James D. Jones
Center for Advanced Intelligent Sytems, Texas Tech University, USA
Computer Science, Angelo State University, USA

INTRODUCTION

"Expert systems" are a significant subset of what is known as "decision support systems" (DSS). This article suggests a different paradigm for expert systems than what is commonly used.

Most often, expert systems are developed with a tool called an "expert system shell." For the more adventurous, an expert system might be developed with Prolog, a language for artificial intelligence. Both Prolog and expert system shells stem from technology that is approximately 30 years old.[1] There have been updates to these platforms, such as GUI interfaces, XML interfaces, and other "bells and whistles." However, the technology is still fundamentally old.

As an analogy, the current technology is akin to updating a 30-year-old car with new paint (a gooey interface), new upholstery, GPS, and so forth. However, the car is fundamentally still a 30-year-old car. It may be in far better shape than another 30-year-old car without the updates, but it cannot compete from an engineering perspective with current models.[2] Similarly, the reasoning power of current expert system technology cannot compete with the reasoning power of the state of the art in logic programming. These advances that have taken place in the logic programming community since the advent of Prolog and expert system shells include: a well developed theory of multiple forms of negation, an understanding of open domains, and the closed world assumption, default reasoning with exceptions, reasoning with respect to time (i.e., a solution to the frame problem and introspection with regard to previous beliefs), reasoning about actions, introspection, and maintaining multiple views of the world simultaneously (i.e., reasoning with uncertainty).

This article examines a family of logic programming languages. This article in conjunction with a companion article this volume, Knowledge Representation That Can Empower Expert Systems, suggest that logic programs employing recent advances in semantics and in knowledge representation provide a more robust framework in which to develop expert systems. The author has successfully applied this paradigm and these ideas to financial applications, security applications, and enterprise information systems.

BACKGROUND

Logic programming presents us with an excellent tool to develop a variety of intelligent systems. While there are still serious issues to be addressed and while there may be additional nonlogical techniques to complement logic-based systems, it is almost a self-evident truth that logic will form the cornerstone of any serious machine intelligence in the future. Consider that our goal is to build "HAL," the all-knowing, self-sufficient computer of the science-fiction movies.[3] To this end, it behooves us to understand, use, and further refine this paradigm.

In this article we shall present a family of logic programming languages called the Stable Model Semantics (Gelfond & Lifschitz, 1988, 1991), or more recently known (and hereafter known) as the Answer Set Semantics. These languages are purely declarative languages with roots in logic programming (Kowalski, 1974, 1979), the syntax and semantics of standard Prolog (Clark, 1978; Colmerauer, Kanoui, Paser, & Russel, 1973) and in the work on nonmonotonic logic (Moore, 1985; Reiter, 1980).

These semantics are arguably the most well-known and most well-developed semantics in logic programming. That there are other competing semantics is not of concern. Other semantics will differ from the Answer Set Semantics primarily at the extremes. Also, the Answer Set Semantics is conservative: a system built upon these semantics believes only what it is forced to believe. Further, the Answer Set Semantics is the most popular semantics in the logic programming research community.

It is our goal that the languages presented here and the ideas presented in the companion article will be adopted by the practitioner community. The material presented here is self-contained. It is hoped that all that is required is only a very careful reading in order to understand this very powerful paradigm.

LOGIC PROGRAMMING LANGUAGES[4]

In this section, an overview of a family of five logic programming languages will be given. These languages do not encompass the entire field of logic programming, but rather represent a particular flavor of logic programming: the answer set semantics. Not only is this "flavor" the author's own preference, but it also seems to be the preference of the logic programming community at large.

That there are five languages discussed is not important, and the names of these languages are not important. They are presented here only to show a progression from simple to complex reasoning. They are presented for pedagogical purposes. At the end of this section, one of these languages will be identified as the preferred language for general use.

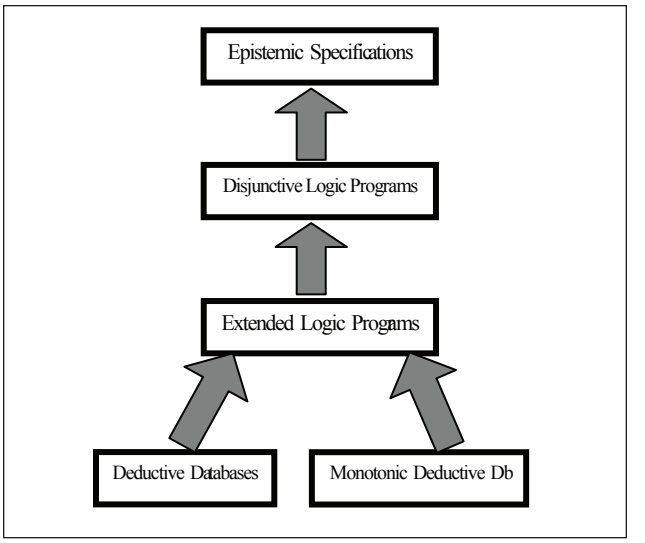
These five languages are not competing languages. Rather, these languages form a strict hierarchy of expressiveness and complexity. Each level of this hierarchy is more expressive than the previous level. Each level completely subsumes the expressiveness of the previous level.

Overview

Figure 1 presents the hierarchy of stable model languages. The topmost part of the figure represents the highest, most expressive, and most complex level of the hierarchy. Conversely, the lowest part of Figure 1 represents the lowest, least expressive, and least complex level of the hierarchy. That which can be expressed at the lower levels can be expressed in each of the higher levels.

For a moment, let us relate current technology with this hierarchy. Rule-based systems, expert system shells, and Prolog slightly blur the boundaries, but belong to the class of programs at the lowest level. At best, they could be classified as deductive databases. (Some hair-splitting issues we want to avoid begin to arise here. As an example, Prolog does have an operator for negation as failure, which is a feature belonging to the next highest level. Yet, Prolog does not properly or completely implement the semantics of negation as failure.)

Figure 1. Hierarchy of languages



Deductive Databases

The simplest semantics that we will discuss are deductive databases (Gelfond & Lifschitz, 1988; Lifschitz, 1989).[5] This language is important because this is the foundation upon which other logic programming languages are built. It is also the foundation upon which Prolog and rule-based systems were built.

Definition: A deductive database is a set of rules of the form:

$$A_0 \leftarrow A_1, ..., A_n$$

where A_i are ground atoms, $n >\_ 0$.

A_0 is called the head of the rule, and $A_1, ..., A_n$ is called the body of the rule. The A_i in the body of the rule is treated as a conjunction. The intended meaning of such a rule is that if the body is true (that is, if each A_i in the body is true), then the head is true. It is a rule of deduction. We deduce the head, if the body is true.

Let us digress a moment and explain in loose terms some of these concepts used in the definition above. An