Knowledge Representation to Empower Expert Systems

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

INTRODUCTION

Knowledge representation is a field of artificial intelligence that has been actively pursued since the 1940s.[1] The issues at stake are that given a specific domain, how do we represent knowledge in that domain, and how do we reason about that domain? This issue of knowledge representation is of paramount importance, since the knowledge representation scheme may foster or hinder reasoning. The representation scheme can enable reasoning to take place, or it may make the desired reasoning impossible.

To some extent, the knowledge representation depends upon the underlying technology. For instance, in order to perform *default reasoning with exceptions*, one needs *weak negation* (aka *negation as failure*. In fact, most complex forms of reasoning will require weak negation. This is a facility that is an integral part of logic programs but is lacking from expert system shells. Many Prolog implementations provide negation as failure, however, they do not understand nor implement the proper semantics. The companion article to this article in this volume, "Logic Programming Languages for Expert Systems," discusses logic programming and negation as failure.

BACKGROUND

Current expert system technology is 30 years old. Expert systems are developed with one of the following tools: expert system shells or Prolog (a language for artificial intelligence.) Expert system shells find their origins in the work of early expert systems, most notably, *MYCIN* which was developed at Stanford in the early to mid 1970s. Prolog was developed in the mid to late 1970s. It was based on the use of logic as a programming language.

The logic programming community (from which both expert systems and Prolog arose) has made notable advances since those times. These advances are lacking from current expert system technology. These advances 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. Each of these areas can be considered a form of knowledge representation.

This article examines knowledge representation issues that implicate the kinds of reasoning that can be performed. In particular, we will discuss two forms of negation, default reasoning with exceptions and the closed world assumption. This article in conjunction with a companion article in this volume, "Logic Programming Languages for Expert Systems," suggest that logic programs employing recent advances in semantics and in knowledge representation provide a more robust framework with which to develop expert systems. While there are still serious issues to be addressed, and while there may be additional non-logical 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 the goal is to build "HAL," the all-knowing, all-controlling computer of the science-fiction movies.[2]

The focus here is upon practicality. It is our goal that the languages and ideas presented here and 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.

KNOWLEDGE REPRESENTATION ISSUES[3]

Negation

One of the most fundamental and most powerful knowledge representation issues is the use of negation. Describing that which is false is a crucial component of human intelligence. Quite often we wish to describe things that are false. "It is not raining" and "the car will not start" are simple examples. To more directly see the negation, these can be recast as "it is false that it is raining" and "it is false that the car will start". Further, things may be described by attributes which are false, just as easily as with attributes that are true. When asked to describe a ballpoint pen, one may say things like "it is not easily erasable like a lead pencil," "it does not have a very large supply of ink," "it does not cost much money," or "it is not very big."

Theoreticians propose various forms of negation (Apt & Bol, 1994). However, there are two forms of negation that are particularly useful, even to the exclusion of all other forms. These two forms of negation are also very well accepted and understood by the research community. One form is *strong negation*. This is the form of negation that states that something is definitively false. The other form of negation is *weak negation* (hereafter, *negation as failure*.) This form of negation is merely the acknowledgment that something is *unknown.*[4]

Negation-as-Failure

Historically, the first form of negation is called *negation as failure* (Lifschitz, 1989). It is denoted by the symbol *not* which precedes a logical formula. Intuitively, it means "I don't know." *Not* merely means that this information is not part of our set of beliefs. As an analogy, consider a database. *Not* means that the tuple does not currently belong to the relation. In terms of logic programming (the paradigm we are most concerned with in this article), *not* means that this information is not provable. An example of this form of negation is

not out_of_business(*verizon*)

This example means that it is not known whether or not *Verizon* is "out of business." "Out of business" could mean that *Verizon* went bankrupt or merely ceased operations, etc.) Negation as failure is a powerful feature of symbolic reasoning. It can be used to perform inferences. Consider the following example.

can_perform_task(*verizon*, *engineering*) ←
    engineering_firm(*verizon*),
    not out_of_business(*verizon*)

This rule states that *Verizon* can perform an engineering task if *Verizon* is an engineering firm, and it is not believed that *Verizon* has gone out of business.[5]

Strong Negation

The other form of negation is called *strong negation* (or *classical negation*) (Gelfond & Lifschitz, 1991). It is denoted by the symbol ¬ . This form of negation states that a fact is explicitly *known to be false.*[6] For example, consider

¬*out_of_business*(*verizon*)

This formula means that it is absolutely false and will always be false (with respect to the current beliefs of the program) that *Verizon* has gone out of business.

The distinction between that which is not known and that which is known to be false is a crucial distinction.[7] These two forms of negation allow us to represent some very powerful information. Further, they allow us to correctly implement default rules as defined in (Reiter, 1980) and to reason nonmonotonically[8] (Baral & Gelfond, 1994; Brewka, 1991; Lukaszewicz, 1990; Marek & Truszczynski, 1993).

"Tricks" to Represent Strong Negation

Expert systems technology and Prolog do not provide for strong negation. However, there are two very simple tricks that allow us to represent strong negation. Gelfond and Lifschitz (1991) present us with the basic idea. Their original idea states that for every predicate *p*, introduce a corresponding predicate *p*', which is interpreted as meaning ¬*p*. For our example, we could have the predicate *male* to represent the positive statement, as in the following

male(*john*)