Logic Programming for Intelligent Systems

James D. Jones
Liberty University, USA

INTRODUCTION

In what seem to be never-ending quests for automation, integration, seamlessness, new genres of applications, and "smart systems", all of which are fueled in part by technological changes, intellectual maturity (or so one thinks), and out-of-the-box thinking that says "surely, there must be a better way", one dreams of a future. That future consists of many different implications and dimensions, but materially, part of that future consists of devices that compute. These devices are not limited to small gadgets, but include all machinery that makes any kind of decision, whether that decision is autonomous, or imposed by an external act or agent. We dream of a future of endless computation, convenience, and sophistication, orders of magnitude above where we are today.

Limiting the present discussion to non-trivial complex computations, this paper suggests a different paradigm for "intelligent systems" than what is commonly perceived as being sufficient. That paradigm is *Logic Programming*, which is the use of logic to represent and reason about knowledge. Further, this paradigm has been blessed with reasoning formalisms specified in the language of Logic Programming which endow us with the ability to do complicated, human-like reasoning.

These reasoning formalisms that provide incredible reasoning power, 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 about current beliefs (as well as past beliefs), and maintaining multiple views of the world simultaneously (which enables reasoning with uncertainty.)

This paper suggests that logic programs employing recent advances in semantics and in knowledge representation formalisms provide a more robust framework in which to develop very intelligent systems in any domain of knowledge or application. The author has performed work applying this paradigm and these reasoning formalisms in the areas of financial applications, security applications, and enterprise information systems.

BACKGROUND

This is a primer for computing professionals, and not an authoritative work on the state-of-the-art of Logic Programming. Logic programming presents us with an excellent tool to develop a variety of powerful intelligent systems. What will be discussed is a particular version of logic programming called "ASP", which means "Answer Set Programming". "Version" in this context is defined by the semantics of the language, and not by performance issues. ASP is currently the most popular and the most well researched Logic Programming language. (To demonstrate the importance that ASP is receiving in the field, an entire issue of "AI Magazine" has been devoted to ASP. This journal is the flagship journal for the Association for the Advancement of Artificial Intelligence, the premier scholarly and practitioner association in the field of Artificial Intelligence. The issue mentioned is the most current issue at the time of this writing, Volume 37 Number 3, Fall 2016.)

While there are still issues that need to be addressed in Logic Programming, and while

DOI: 10.4018/978-1-5225-2255-3.ch411

there may be additional non-logical techniques to complement logic-based systems, it is reasonable to believe that logic will form the cornerstone of any serious machine intelligence in the future. Consider that the goal of the field of artificial intelligence (in fact, one of the early goals of computing itself) is to build "HAL", the all-knowing, self-sufficient computer of science-fiction movies (Clarke, 1968). To this end, it behooves us to understand, use, and further refine this paradigm.

In this paper, the syntax and of semantics of ASP shall be presented. The early forms of this language were initially called the Stable Model Semantics, and later, Disjunctive Logic Programming (Gelfond & Lifschitz, 1988; Gelfond & Lifschitz 1991). This language is a purely declarative language (as opposed to procedureal languages which dominate industry, and as opposed to functional languages which are the main competitor in the field of AI.) Declarative languages have their roots in logic programming (Kowalski, 1974; Kowalski, 1979), they are defined in the syntax and semantics of standard Prolog (Colmerauer, et. al., 1973; Clark 1978), and they employ the work on nonmonotonic logic (Reiter, 1980; Moore, 1985).

The semantics of ASP 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 ASP primarily at the extremes. Also, the semantics of ASP are conservative: a system built upon these semantics believes only what it is forced to believe. As already stated, the semantics of ASP are the most popular semantics in the logic programming research community.

It is the hope here that ASP and the accompanying knowledge representation techniques presented will be appreciated by the practitioner community to the extent that ultimately, reasoning systems will be based upon these ideas. The material presented here is self-contained.

ANSWER SET PROGRAMS (ASP)

The language of ASP is mathematically precise, and well understood among researchers and those "in the know". A correct discussion of this would necessarily consists of several unambiguous, mathematical definitions. However, for the purposes of this work, definitions will be kept to a minimum, and will favor an intuitive bent, rather than a rigorous mathematical bent. Nonetheless, some formalities are necessary. An intuitive discussion may be viewed as "sloppy" in the sense of mathematical rigor, but it is necessary and easier to understand for those not acquainted with the field. It is this audience to which this paper is being written.

The presentation here will be "backwards" in the sense of starting with the final result, and explaining the component pieces. A mathematical presentation would do the opposite: build a foundation, such that when the final result is presented, all the pieces have already been discussed.

Syntax

The syntax of any language is the grammar that defines how to construct legal sentences in that language. The syntax of ASP is as follows. A rule (the only building block of an ASP program) is of the form:

$l_0 \, or ... \, or \, l_i \leftarrow l_{i+1}, ..., l_m, not \, l_{m+1}, ..., not \, l_n$ (Gelfond & Kahl, 2014)

where the ls are "literals", $i \geq 0$, $m \geq 0$, and $n \geq 0$.

The symbol l is just an abstraction for a "predicate" with the "terms" appropriate for that predicate. (All these predicates and terms are defined merely by their use in an ASP program.) These would be some examples of literals:

male(john)
married(john, sue)
employee(john, accounting, 1/25/2012)
itIsHotToday()

The terms of the last two literals need further technical explanation, but suffice it to say that even these literals *could* be correct.