PROLOG AND ARTIFICIAL INTELLIGENCE: A STUDY

Deep Inder Mohan (IMT2017013)
Gandharv Suri (IMT2017017)
Lakshay Agarwal (IMT2017026)

CS306: PROGRAMMING LANGUAGES

June 2020

International Institute of Information Technology, Bangalore

PROLOG AND ARTIFICIAL INTELLIGENCE: A STUDY

Submitted to International Institute of Information Technology, Bangalore
in Partial Fulfillment of
the Requirements for the completion of
CS306: Programming Languages course

by

Deep Inder Mohan (IMT2017013)
Gandharv Suri (IMT2017017)
Lakshay Agarwal (IMT2017026)

International Institute of Information Technology, Bangalore
June 2020

Report Certificate

This is to certify that the report titled **Prolog and Artificial Intelligence: A Study** submitted to the International Institute of Information Technology, Bangalore, for the completion of the **CS306: Programming Languages** course is a bona fide record of the course work done by **Deep Inder Mohan (IMT2017013), Gandharv Suri(IMT2017017), Lakshay Agarwal(IMT2017026)**, under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

_____

Sujit Kumar Chakrabarti

Bangalore,
The 7th of June, 2020.

Contents

CHAPTER 1

INTRODUCTION TO PROLOG AND AI

1.1 What is Prolog?

Prolog stands for **programming in logic** - an idea that emerged in the early 1970s to use logic as a programming language. It was created around 1972 by Alain Colmerauer with Philippe Roussel, based on Robert Kowalski's procedural interpretation of Horn clauses. It was motivated in part by the desire to reconcile the use of logic as a declarative knowledge representation language with the procedural representation of knowledge.

The first implementation of Prolog was an interpreter written in Fortran. Later the first Prolog compiler was credited to David Warren, an expert on Artificial Intelligence at the University of Edinburgh.

Prolog is a computer language that uses many of the representational strengths of the First-Order Predicate Calculus because of which it can express general relationships between entities. The program logic is expressed in terms of relations and represented as facts and rules. Further, a computation is initiated by running a query over these relations.

Example -

Facts build up the database -

```
likes(ram, sita)
likes(ram, puja)
likes(ram, seema)
```

Rule - (which represents shyam likes everything that ram likes.)

```
likes(shyam, Something) :- likes(ram, Something)
```

Now a query can be -

```
likes(shyam, Something)
```

Then Prolog will return -

```
Something = sita
Something = puja
Something = seema
```

1.2 What is Artificial Intelligence?

To understand Artificial intelligence we need to first know what is intelligence. Intelligence is defined as the ability to acquire and apply knowledge, and as the capacity for logical reasoning and self-awareness. It has been related to understanding, solving, planning, and several other concepts and tasks.

Artificial intelligence (AI), sometimes called machine intelligence, is intelligence demonstrated by machines. Basically it describes machines that mimic ”cognitive”

functions that humans associate with the human mind, such as learning and problem-solving.

Artificial Intelligence rests on two basic ideas :

1. **Representation** or the use of symbol structures (for eg - lists, trees, etc) to represent problem-solving knowledge (state). The symbolic computing methods defined on these symbolic structures form the foundation of Artificial intelligence.
2. **Search**: The systematic consideration of sequences of operations on the knowledge structures to solve complex problems. To control the complexity of the search operation, AI examines the use of heuristics.

1.3 Use of Prolog in AI

As an implementation of logic programming, Prolog makes many important contributions to AI problem-solving -

- **The declarative semantics of prolog is a means of directly expressing problem relationships in AI.** The declarative meaning of a program determines what will be the output of the program and the procedural meaning of a program determines both the output and how the output was obtained. The declarative semantics of prolog allows the programmer to be concerned with only the declarative meaning of the program whereas the procedural details are worked out by prolog itself.
- **Prologs direct and transparent representation and interpretation of predicate calculus expressions.** In AI, predicate calculus is one of the many important representational schemes which is used in many fields of AI such as automated reasoning, robotics research, etc. Prolog extracts its major power as a programming language in the sense that it uses predicate calculus to define its relations.

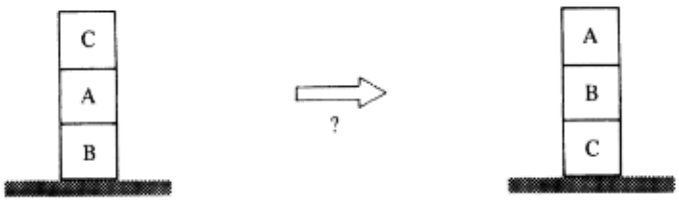
- **Prologs ability to create meta-predicates or predicates that can constrain, manipulate, and interpret other predicates.** Prolog is ideal for creating meta-interpreters or interpreters written in Prolog that can interpret subsets of Prolog code. Using this property we are able to write interpreters for expert systems, machine learning using explanation-based learning models, etc, which are all huge contributions in the field of AI.

1.4 Basic Problem Solving

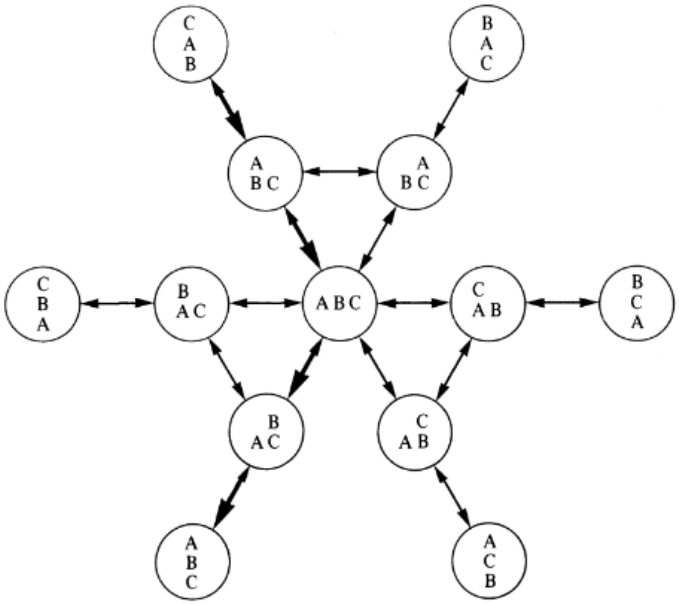
1.4.1 State Space strategy

Basic problem-solving strategies can include representing the problem in terms of state space. State space is a directed graph whose nodes correspond to problem situations and arcs to possible moves. A particular problem is defined by a start node (initial situation) and a goal condition (final situation). A solution to the problem then corresponds to a path in the graph. Thus problem-solving is reduced to searching for a path in a graph. [1]

For eg - A problem to rearrange the blocks shown below can be solved by forming a state space.



The state space for the above problem -



1.4.2 How Prolog is efficient in representing a state space

State spaces can be easily represented in Prolog by a relation - S(X, Y), which gives true if a move exists from node X to node Y in the state space, and Y is called the successor of X. This relation can be represented by facts in the Prolog program. The initial situation for the above problem can be represented as - $[[c,a,b],[],[]]$ The goal situation -

$$[[a,b,c],[],[]]$$
$$[[],[a,b,c],[]]$$
$$[[],[],[a,b,c]]$$

The goal condition can be represented using the rule - **goal(Situation) :- member([a,b,c], Situation)**.

The search algorithm can be represented by the relation - **solve(Start, Solution)**

In the case of the problem described above, on a successful search, the **Solution** is instantiated to a list of block arrangements.

We see that properties of the prolog language like the unification, declarative semantics, predicate calculus representation, make it very powerful in being able to represent the state spaces. Moreover, prolog perceives and solves the problem in a similar way a human would do so. Therefore prolog is very efficient in dictating human behavior to tackle a problem.

1.5 Key features of Prolog for AI

Some key features of Prolog that make it favourable over other languages in AI are [2] :-

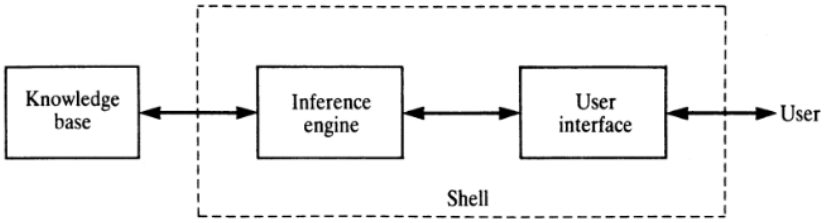
- **Unification / Pattern Matching** - Unification is the way in which variables are bound in prolog. Prolog performs syntactic matches on structures and does not evaluate expressions. It is powerful for pattern matching across sets of predicate calculus specifications. It offers an efficient implementation of the if/then/else constructs of lower-level languages: if the pattern matches, perform the associated action, otherwise consider the next pattern. It serves as a powerful tool that supports AI programming techniques.
- **Recursion** - Recursion works with unification to evaluate patterns in much the same way looping constructs are used in lower-level languages. Many of AI problems include searching indeterminate sized trees or graphs, where the naturalness of recursion serves very powerfully.

1.6 Development of an Expert System

1.6.1 What is an Expert System?

An expert system is a program that behaves like an expert in some domain. It is also answerable for its decisions and explanation of any topic of that domain. It is basically a substitute for the human expert in that domain.

1.6.2 Structure of an Expert System



- **Knowledge base** - This part of the expert system comprises the knowledge specific to the domain, facts about the domain, and the rules describing relations in the domain
- **Shell** - This comprises both the inference engine, which harnesses the knowledge by the knowledge base and the User interface, which provides a medium for communication between the user and the system.

This scheme of separating the knowledge and the algorithms that use this knowledge adopted by the expert system presents an excellent model because the knowledge is domain-dependent and the shell which uses this knowledge is fairly domain independent.

This kind of separation is similar to what we see in Prolog. Prolog separates the logic for the program specification (comprising facts and rules) from the execution or control of the use of that specification. The rules of the knowledge base are equivalent to the facts, rules in prolog. Therefore this property of prolog makes it favorable over other traditional languages, where both these components are mixed together for the creation of the expert systems.

The most popular or efficient formalism for representing knowledge is the language of **if-then rules** (also called production rules) which forms the propositional calculus. We also know that predicate calculus is built upon some form of this propositional calculus and Prolog is built upon the predicate calculus. Therefore Prolog serves as a good programming language for implementing such a representation.

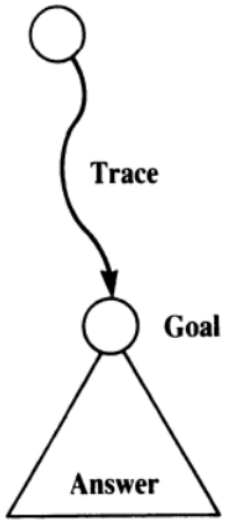
Simply answering the question using only the knowledge base by the prolog program doesn't qualify as an expert behavior for two reasons :

1. No information about how the result was obtained is provided
2. All relevant information had to be input into the system (as Prolog facts) before any question was asked.

To overcome these drawbacks the expert system must include a medium of communication between the user and the system, which is done by the user interface part. The system asks questions to the user where the information gained by the user's answers is not found in the knowledge base or could be derived from other information.

The best way of providing information on how the result was achieved is by displaying the evidence: that is rules and subgoals from which the result was achieved. This evidence is satisfied by using an AND/OR solution tree. Therefore the expert system while producing the actual result also produces an AND/OR solution tree consisting of rule names and subgoals.

1.7 Prolog Implementation of Expert Systems



The shell is implemented in prolog using the following predicates -

- explore(Goal, Trace, Answer): finds an answer Answer to a question Goal
- useranswer(Goal, Trace, Answer): gives solutions for an askable Goal by asking the user about the Goal.
- present(Answer): displays the result and answers ‘how’ questions.

Where,

Goal is the question to be investigated

Trace is a chain of ancestor goals and rules between Goal and the top-level question

Answer is an AND/OR-type solution tree for Goal

CHAPTER 2

IBM WATSON

This chapter is a case study of IBM Watson, a question analysis system developed by IBM, which uses Prolog as a core element in it's pipeline.

2.1 Introduction

In February 2011, IBM Watson competed in the famous classic quiz game show 'Jeopardy!' (USA) in a special 'Machine vs Man challenge' defeating the two former grand champions. Watson faltered in a few areas of knowledge and was touted by observers as one of the biggest computing advancements in the past several decades. The reason behind it was that Watson could do more than calculations, it could answer verbal questions posed by humans within a short time.

IBM Watson question answering system was designed to answer complex natural language questions over a broad domain of knowledge. Moreover, have enough confidence (probability of the answer being correct) in its answer that too in a very short time.

Question Answering (QA) requires going beyond just keyword matching in the documents, and to correctly interpret the question to figure out what exactly is being asked. The QA system also requires to be self-reliant, as it does not depend on human aid to

read through returned documents.

2.2 Overview

2.2.1 Basic Idea

Watson's QA receives an unstructured text question as input and begins with the question analysis phase to determine what the question is asking. The syntactic and semantic elements of the question are identified and encoded as structured information. This structured information is later used by other elements of Watson.

In order to explain the elements of Watson's processing the following example would be used.

> "He was a bank clerk in the Yukon before he published 'Songs of a Sourdough' in 1907"

The focus is the part of the question that references the answer. Like in this example, the *focus* would be"he".

The *lexical answer types* (LATs) are the terms in the question which indicate the entity type of the answer. In the example"he", "clerk" and "poet" would be the *LATs.*

Question Classification identifies the question as belonging to one or more of several broad types. The example belongs to Factoid, there are multiple question classes as definition, puzzles, etc.

Some questions require to be broken in multiple subsections. *QSections* are the fragments that require to be handled separately.

The rule-based question analysis components were implemented in Prolog.

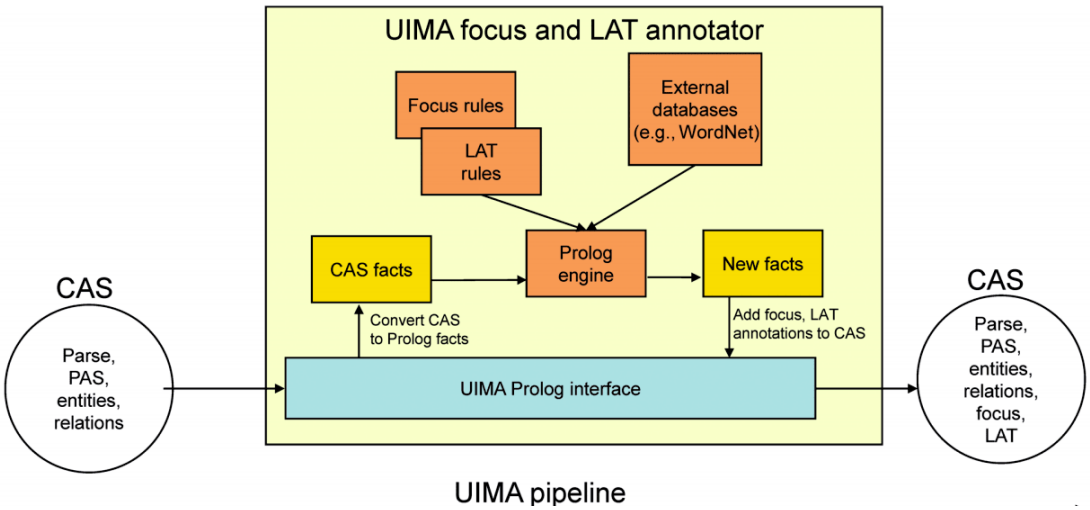
In this document, we focus on how the rule-based portion of question analysis was implemented using Prolog and why Prolog was chosen for this purpose. A. Lally et al., 2010 [3] describe a detailed explanation of the whole IBM Watson model.

2.2.2 Question Analysis

The parse and semantic analyzer consist of slot grammar parser (ESG), predicate-argument structure (PAS), a named entity recognizer (NER), co-reference recognizer, and a relation extraction component.

The ESG parses the content into a tree with a surface as well as a deep login structure. The node consists of the word or multiword component and its logical predicates. The list of features and left and right modifiers. In the example above, among many identifications. `publication(e1,he,"songs of a Sourdough")` and `in(e2,e1,1907)` indicating that the publish event e1 occurred in 1907.

2.2.3 Converting to Prolog Rules



Question analysis in Watson is implemented as a pipeline of components assembled us-

ing the **Unstructured Information Management Architecture (UIMA)**. The question analysis tasks are implemented as rules over the predicate-argument structure (PAS) and external databases such as WordNet. Prolog was the ideal choice because of its simplicity and could conveniently express a large number of rules over dependency-based parse.

In UIMA the common analysis structure (CAS) is a dynamic data structure that contains unstructured data (text whose meaning is not inferred yet). Structured data from unstructured data is encoded as feature structures. Converting into Prolog fact, for each CAS feature a unique id is assigned. For each feature of that feature structure, a fact in the form `feature_name(id, value)` is made. In case the value of a feature is another feature structure then the id of the latter is used in place of the value.

So for the example used earlier, the PAS nodes produced by the parsing and semantic analysis annotators represented as Prolog facts as:

```
lemma(1,"he")
partOfSpeech(1,pronoun)
lemma(2,"publish)
partOfSpeech(2,verb)
lemma(3,"Songs of a Sourdough")
partOfSpeech(3,noun)
subject(2,1)
object(2,3)
```

Such facts are added to the Prolog system and several rule sets were executed to detect the focus of the question, the LAT, and several relations between the elements. To detect the `authorOf` relation in Prolog is as follows.

```
authorOf(Author,Composition) :-
```

```
    createVerb(Verb),
    subject(Verb,Author),
    author(Author),
    object(verb,Composition),
    composition(Composition).

createVerb(Verb):-
    partOfSpeech(Verb,verb),
    lemma(Verb,verbLemma).  %[\write","publish"....]
```

The `author` and `composition` apply constraints on the nodes to rule out that are not valid fillers for the author and composition roles in the relation.
This rule applied to the example results in a new fact `authorOf(1,3)`, which are later passed to downstream components in the Watson pipeline.

2.3 Why Prolog?

Prolog was chosen for its flexibility and ease in pattern matching rules. Prolog brings in the major element required for Jeopardy!, quick processing. A. Lally et al., 2010 claim custom pattern-matching didn't replicate all the features of Prolog and lacked efficiency. Post using Prolog, the productivity significantly improved in developing new rules.

CHAPTER 3

PRACTICAL APPLICATION OF PROLOG IN AI

3.1 Introduction

One of the objectives we wanted to explore in this study was to build some kind of functional AI system entirely on Prolog. But the honest truth is, Prolog as a language is simply not designed for that kind of task. The way that the people at IBM used it was to make it a part of their pipeline, interfacing it with languages that could accomplish other tasks, such as linear algebra better.

But why is this the case? Is it truly impossible to use prolog to build a Machine Learning system? There was only one way to find out, and that is to try building one ourselves! So we decided to try building a linear regressor from the ground up solely using prolog. The rationale behind this decision was simple: Linear regression is the first fundamental Machine Learning approach learned by everybody in the field. Hence, it became the most feasible target for us in the timeframe.

3.2 Normal Equation in Bivariate Linear Regression

Given a series of data points of the form (x_i, y_i), create matrices X, Y, A, and E s.t.

$$Y = XA + E \text{ , where}$$

$$Y = [[y_1], [y_2], ..., [y_n]]$$
$$X = [[1, x_1], [1, x_2], ..., [1, x_n]]$$
$$A = [[b], [m]] \text{ , where b is intercept, m is slope}$$
$$E = [[e_1], [e_2], ..., [e_n]] \text{ is the matrix of errors}$$

Then, knowing X and Y, we can find A as:-

$$A = (X^T X)^{-1}(X^T Y)$$

Then, using X, Y and A, E can be calculated as:-

$$E = Y - XA$$

From E, the sum of squared errors (SSE) can be calculated as:-

$$SSE = E^T E$$

The reason we limited ourselves to bivariate regression was because of the matrix inversion problem. Although there are known algorithms for computationally inverting $n \times n$ matrices, the mathematical scope of those is well beyond this course, and more importantly, they are incredibly complex and at our level of comfort with prolog, impossible to implement.

3.3 Practical Challenges/Advantages

One of the main challenges in working with prolog for a major project is the general lack of support for the language. This is understandable, prolog being as old as it is. Nonetheless, a strong community of programmers that have asked and answered questions pertaining to problems one might encounter while using a particular language is invaluable to beginners in that language, and its lack for prolog made our lives harder.

The next major complaint is the fact that most of the time, anything you want to accomplish in prolog must be done manually, due to the extremely limited library support. For instance, Python, C, C++, and Java all have libraries containing functions for carrying out matrix operations like addition, subtraction, and multiplication. However, these 'functions' had to be manually implemented in our case.

The problem with existing libraries is what I thought to be poor documentation. Most of the time, it is very hard to understand what a module does just from looking at the docs. Additionally, I/O operations in prolog are hard to manage as well.

However, the language does come with a few advantages. Implementing mathematical 'functions' like matrix multiplication becomes much easier through the use of inbuilt meta-predicates such as maplist and foldl. The code, therefore, becomes much more compact than in languages like C and C++. The unification properties of prolog can be very helpful in dealing with matrices. Additionally, memory management and type mismatch errors become non-concerns for the programmer, where they can be the source of much grief in the case of a language like C.

To code this bivariate linear regressor, we used the modules *csv* and *clpfd*.

```
% Using inbuilt library for reading CSV files
:- use_module(library(csv), [csv_read_file/3]).
:- use_module(library(clpfd), [transpose/2]).
```

3.4 Code

The code can be found at this Github link:-

`https://github.com/deep-guy/PL_project/tree/master/code`

The information on how to run the code and the general structure of the code is in a README file in this repository.

Bibliography

[1] George F. Luger and William A Stubblefield. *AI Algorithms, Data Structures, and Idioms in Prolog, Lisp, and Java for Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Addison-Wesley Publishing Company, USA, 6th edition, 2008.

[2] Ivan Bratko. *PROLOG Programming for Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., USA, 2nd edition, 1990.

[3] A. Lally, J. M. Prager, M. C. McCord, B. K. Boguraev, S. Patwardhan, J. Fan, P. Fodor, and J. Chu-Carroll. Question analysis: How watson reads a clue. *IBM Journal of Research and Development*, 56(3.4):2:1–2:14, 2012.