Development of FO2L – A First-Order Logic Language

CHAFIA KARA-MOHAMED (alias HAMDI-CHERIF) (1), ABOUBEKEUR HAMDI-CHERIF (1),(2)  
Computer College  
Computer Science Department  
Qassim University  
PO Box 6688 – 51452 Buraydah  
SAUDI ARABIA  
(2) Permanent Address: Université Ferhat Abbas Setif (UFAS)  
Faculty of Science  
Computer Science Department  
19000 Setif  
ALGERIA  
(1) email: smhmd@qu.edu.sa , chafikmo62@gmail.com  
(2) email: shrief@qu.edu.sa , elhamdi62@gmail.com

Abstract: - A first-order logic language, called FO2L, is proposed as an initial step towards the development of a complete knowledge-based system (KBS). Predicate calculus is used because it allows the use of variables in the definition of rules, and represents one of the keys to powerful knowledge bases development – the most important component of any KBS. Parsing is done using recursive descent for easy implementation. Additionally, a friendly user-interface is provided allowing both the user and the expert to easily interact with the system by introducing and/or modifying the knowledge base.

Key-Words: - Theory of languages, Parsing, Logic programming, First-order logic, Knowledge base system

1. Introduction
The present paper reports the design and implementation of a first-order logical language, called FO2L, as a first step in the implementation of an environment for a knowledge-based system (KBS). Knowledge is considered as human understanding of a subject matter that has been acquired through proper study and lifelong experience. It is derived from information in a similar way as information is derived from data. It also includes information about both real world entities and relationships between them. Knowledge base systems are computational systems that use organized data, or knowledge, instead of raw data, and an inference process instead of procedural programs [7]. In rule-based systems, knowledge is usually divided into rules and facts. The inference can be undertaken in forward or backward chaining. According to Turing Test, knowledge representation (KR) is one of the key components of any intelligent system. Therefore, when using artifacts, knowledge has to be represented in a codifiable way. Several KR-oriented programming languages have been developed for the last five decades, or so. For instance, Prolog, developed about four decades ago, but popularized much later, represents propositions and basic logic, and can derive conclusions from known premises using backward chaining [14]. The

Japanese Fifth Generation project, a 10-year research effort beginning in 1982, was completely based on Prolog as the means to develop intelligent systems. KL-ONE, developed in the eighties, is more specifically aimed at knowledge representation per se, followed by Dublin Core standard of metadata. In electronic document processing, vital for the Internet, languages concentrate on the structure of documents. Languages such as SGML, a precursor of HTML, and later XML play a vital role in information retrieval and data mining processes. Development of the Semantic Web, has included development of XML-based knowledge representation languages and standards, including Web Ontology Language (OWL), Ontology Inference Layer (OIL), DARPA Agent Markup Language (DAML), RDF, RDF Schema, and Topic Maps [13]. When XML or similar, as low-level syntax language, is used, the output of knowledge representation languages is made easy for machines to parse, but with a problematic human readability and often bad space-efficiency. To avoid excessive space complexity, first-order or predicate calculus is commonly used as a mathematical basis for knowledge representation. However simple predicate calculus-based systems might appear to be, they can be used to represent data that is well
beyond the processing capability of current computer systems [2]. They have been successfully used in theorem proving with the spectacular result obtained by the automatic proof of Robbins’ conjecture not proved by humans for decades. In addition, predicate calculus through productions or rule base systems has been used in various expert systems in medicine since Mycin, in legal advice and innumerable other applications [12]. Owing to its structure, FO2L can be integrated within this body of knowledge with the characteristic of being open source and made available online. As far as this paper is concerned, we concentrate on the grammar used and on the internal representation of the proposed language. Recursive descent is used for parsing, allowing easy implementation. Furthermore, the interface is user-friendly and allows KBS users and experts alike to easily interact with the environment. In this regard, FO2L plays a key role in knowledge representation on the basis of production rules.

In the next section we describe the design of the proposed language. In Section 3, we use it to represent knowledge from an example. Section 4 is devoted to the description of a friendly graphical user interface (GUI). The paper ends with a conclusion summing up the main results and pointing towards some potential future developments.

2. Related works and concepts

2.1 From syllogism to inference

Historically, the first type of inference most carefully studied by logicians since Classical Antiquity was the syllogism, extensively studied by Greek and early mathematics that included elements of first-order logic, such as quantification, although restricted to unary predicates. The axiomatic style of exposition, using Modus Ponens plus a number of logically valid schemas, was employed by a number of logicians. Inference rules, as distinct from axiomatic schemas, were the focus of the natural deduction approach. The invention of clausal form was a crucial step in the development of a deep mathematical analysis of first-order logic. Although the use of first-order logic was primarily suggested in the late 1950’s for representation and reasoning in AI, the first such systems were developed by logicians interested in mathematical theorem proving.

2.2 Post-resolution period

After the development of resolution, work on first-order inference proceeded in various directions. In AI, resolution was adopted for question-answering systems as implemented, in a somewhat less formal approach, in PLANNER language which was a precursor to logic programming and included directives for forward and backward chaining and for negation as failure. A subset known as MICRO-PLANNER was implemented and used in the SHRDLU natural language understanding system.

2.3 Production systems

Because AI applications usually rely on a large number of rules, it is therefore important to develop efficient rule-matching methods along with the underlying ad hoc technology, particularly for efficient incremental updates. The technology for production systems was developed in the early 1970’s to support such applications and forward chaining has been used since then to become a well established technique in AI as an easily understandable alternative to resolution. It has been used in a wide variety of systems, ranging from Nevis's geometry theorem prover to the R1 expert system for DEC-VAX™ computers configuration. The production system language OPS-5 [3] was used for R1 and for the SOAR cognitive architecture [10]. OPS-5 incorporated the rete match process. SOAR, which generates new rules to cache the results of previous computations, can handle very large rule sets over 8,000 rules in the case of the TACAIR-SOAR system for controlling simulated fighter aircraft. CLIPS, followed by FuzzyCLIPS [17] is another popular example of production system as a C-based language developed at NASA that allowed better integration with other software, hardware, and sensor systems and was used for spacecraft automation and several military applications.

2.4 Deductive databases

The deductive databases, as an area of research aims to integrated relational database technology, mainly designed for retrieving large sets of facts, with Prolog-based inference technology, which typically processes one fact at a time. This approach has also contributed a great deal to forward inference [11]. In terms of expressibility, deductive databases are half-way between relational databases and logic programming. Indeed, deductive databases are more expressive than relational databases but less expressive than logic programming systems. In recent years, deductive databases such as Datalog have found new application in data integration, information extraction, networking, program analysis, security, and cloud computing [4]. Logical inference complexity has mainly come from
the deductive database community. In [5], it was first shown that matching a single non recursive rule, or a conjunctive query in database terminology, can be NP-hard. [8] defined data complexity as a function of database size, viewing rule size as constant and showed that it can be used as a suitable measure for query answering.

2.5 Theorem provers
Research into mathematical theorem proving began even before the first complete first-order systems were developed. Since the late 1950’s, Gelernter’s Geometry Theorem Prover used heuristic search methods combined with diagrams for pruning false sub-goals and was able to prove some quite elaborate theorems in Euclidean geometry. The authors in [6] describe the early SAM theorem prover, which helped to solve an open problem in lattice theory. [16] give an overview of the contributions of the AURA theorem prover toward solving open problems in various areas of mathematics and logic. In [9]), the authors follow up on this, recounting the accomplishments of AURA’S successor, OTTER, in solving open problems. [15] describes SPASS, one of the strongest current theorem provers.

3. Design of the FO2L environment
In this section, we present FO2L and follow its implementation from the grammar writing to the programming of its lexical and syntactic analyzers.

3.1 Logical language characteristics
A production rule can be written using propositional (zero-order logic) or predicate calculus (first-order logic FOL). In our system, we use first order logic to describe SPASS, one of the strongest current theorem provers. In [9]), the authors follow up on this, recounting the accomplishments of AURA’S successor, OTTER, in solving open problems. [15] describes SPASS, one of the strongest current theorem provers.

3.2 Grammar
Chomsky has defined a grammar as the quadruple

\[ G = (N, T, P, S) \]

where \( N \) is a finite set of non-terminal symbols, \( T \) is a finite set of terminal symbols \( P \) is a finite set of production rules and \( S \) is a distinguished symbol in \( N \) and is called the start symbol. Programming languages can be expressed by two kinds of grammars: regular and context free grammars.

We express here our logical language in Backus-Naur form.

The language is not case sensitive. Identifiers are written under the ordinary rules of many programming language, i.e. first letter of predicate names must be capital.

3.3 Parser
The parser is responsible of the pre-compilation phase. There are different methods for parsing such as top-down, bottom-up and recursive descent. In our implementation, we use the latter one. In this approach, there is a function for each non-terminal. The advantage of this approach is that it can be directly written based on the grammar [1]. The application of additional compiler techniques, such as type inference, made Prolog programs competitive with C programs in terms of speed.

3.4 Internal representation
In imperative programming languages, the result of compiling a program is an optimized object code written in the machine language for the machine where it will be executed. Because FO2L is a logical language, therefore the result of the pre-compiling step is an internal representation of the knowledge
base into some form acceptable by the inference engine.

3.5 Data structures
Linked lists are the main data structure used. In Figure 1 above, we show the structure of a rule in the memory. A rule is a structure consisting of main information with two lists for condition and conclusion part.

In Figure 2 below, a fact is a list with two fields, the first is the predicate and the other is a list of different attributes of the predicate.

4. Example
Let us consider the following sentence, representing a unit piece of knowledge to be stored in the knowledge base: “If an animal flies and lays eggs then it is a bird”. Suppose as a fact that we have an animal that has the two characteristics. Translation of both rule and fact give the following knowledge base content:

<table>
<thead>
<tr>
<th>Rule base content</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule r1 if (FLIES ?x ) (LAYS ?x eggs ) then add (IS ?x bird )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fact base content</th>
</tr>
</thead>
<tbody>
<tr>
<td>facts (FLIES z ) (LAYS z eggs)</td>
</tr>
</tbody>
</table>

The following figures show the internal representation of this rule and fact respectively.

5. Graphical User Interface
Our approach is also related to applications of user interfaces as used in architectural facilities such as
development environments, and to work on comparing different versions of knowledge base. In this section we show how a user can interact with our system via the interface. The figures are self-explanatory.

Figure 5 Main window

Figure 6 Open-Save-Close-New/File menu

Figure 7 Edit menu

Figure 8 Main menu in new style

Figure 9 Editor for writing a new knowledge base

Figure 10 Pre-compiling step

Figure 11 Example with syntax errors

6. Conclusion
As it stands now, FO2L is language that offers powerful assistance to both experts and users in writing, parsing, compiling, storing and modifying knowledge bases of concern. We have described all the important methodological steps needed to design and implement a programming language and more
specifically a logical language. Although our language FO2L is relatively simple, it is powerful enough to allow knowledge base system design. The main features are based on an improved grammar and internal representation, an efficient parsing method, and a friendly GUI. Adding a reasoning component integrating both forward and backward chaining is under process to bring about a novel user-friendly complete knowledge base system.

7. Acknowledgments
The authors wish to thank Ray Al-Marshad and Nora Al-'Osily for the extensive testing and debugging of the final version of the implemented system.

References:


Websites accessed as of January 2013
CLIPS: http://clipsrules.sourceforge.net/
Dublin Core:
http://dublincore.org/documents/dcmi-terms/