FO2L – A First-Order Logic Language for Knowledge Base System Implementation

C. Kara-Mohamed (*alias* Hamdi-Cherif), N. I. Al-Osily, R. A. Al-Marshad, M. S. Al-Twijary, J. Al-Robe'an, A. Hamdi-Cherif

***Abstract*—** FO2L is a novel First-Order Logic programming Language. Although FO2L can be used as a traditional logic programming language, such as Prolog, it represents an initial step towards the development of a complete environment for knowledge-based system (KBS) design and implementation. The choice of predicate calculus is dictated by the fact that it allows the use of variables in the definition of rules, and represents one of the keys to powerful knowledge bases development, *i.e.* the most important component of any KBS. Parsing is done using recursive descent for easy implementation. Additionally, a friendly graphical user interface (GUI) is provided allowing both the user and the expert to easily interact with the system by introducing and/or modifying their knowledge bases.

***Keywords*—** Theory of languages, Parsing, Propositional logic, First-order logic, Knowledge base system.

I. 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 a complete 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. From a computational point of view, knowledge is derived from information in a more complex way than information is derived from data. It also includes a description about both real world entities and relationships between them. Knowledge base systems (KBSs) are computational systems that use organized information, or knowledge, instead of raw data, and an inference process instead of procedural programs [9].

C. Kara-Mohamed (*alias* Hamdi-Cherif) is Assistant Professor at Computer College, Qassim University, PO Box 6688, 51452 Buraydah, Saudi Arabia, (email: smhmd@qu.edu.sa)

N. I. Al-Osily is Teaching Assistant at Computer College, Qassim University, now on leave absence preparing for graduate studies in Cleveland, OH, USA.

R. A. Al-Marshad is Computer Science Instructor at Buraydah College of Technology, Qassim, Saudi Arabia.

M. Al-Twijry and J. Al-Robe'an are Computer Analysts at Al-Rajhi Bank, Buraydah, Qassim, Saudi Arabia.

A. Hamdi-Cherif is Professor at Computer College, Qassim University, PO Box 6688, 51452 Buraydah, Saudi Arabia, and with Computer Science Department, College of Science, Ferhat Abbas University, Setif, Algeria (corresponding author phone: 9666-3500050 e-mail: shrief@qu.edu.sa)

In rule-based systems, knowledge is usually divided into rules and facts. The inference can be undertaken in forward or backward chaining. In forward chaining, the inference process checks the validity of a rule starting with the premises and then infers the validity of the conclusion part; this process is inversed in the case of backward chaining. According to Turing Test, knowledge representation is one of the key components of any intelligent system. Therefore, when designing artifacts, knowledge has to be represented in a codifiable and maintainable way. Different approaches have been used to code the human experts' knowledge. We find knowledge represented as a set of IF-THEN rules. These rules can eventually be affected by probabilities to manage uncertainty with Bayesian reasoning. They can also be written with fuzzy words and then use the fuzzy reasoning for approximate reasoning. Frames, association of syntax with semantic, are also used to represent objects and concepts of the knowledge as classes – the main concept in object oriented programming (OOP) paradigm. Frames can highlight inheritance relationship between elements of knowledge. Alternatively, as a (very) coarse mimicry of the human brain neural structure, knowledge can also be represented by artificial neural networks (ANNs). These structures permit to the system not only to reason but also to learn from different experiences or to discover unpredicted patterns in large datasets.

To allow machines to acquire knowledge, several knowledge representation-oriented programming languages have been developed over the last five decades, or so. For instance, Prolog, developed about four decades ago, but popularized much later, uses propositions and first-order logic (FOL), and can derive conclusions from known premises using backward chaining [17] and the closed world assumption. 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. The advances made by 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 [16]. 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 challenging space-efficiency. Thus, to avoid excessive space complexity, first-order or predicate calculus is commonly used as a mathematical basis for knowledge representation.

However simple predicate systems might appear to be, they can nonetheless be used to represent data that is well beyond the processing capability of current computer systems [3]. Predicate-based systems 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 [12]. 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 [15]. Owing to its structure, FO2L can be integrated within this body of knowledge with the characteristic of being open source, easily upgradeable, and made available online with a user-friendly interface.

In the next section we describe related works and concepts. Section 3 summarizes the design of the proposed language. In Section 4 and Section 5, we use FO2L to represent knowledge for a simple example followed by a second example incorporating both propositional logic and FOL. Section 6 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.

II. Related works and concepts

A. *From syllogism to inference*

Historically, the first type of inference most carefully studied by logicians since Classical Antiquity is the syllogism. Extensively investigated by Greek and early mathematics, the syllogism included elements of FOL, 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 axiom schemas, were the focus of the natural deduction approach [13].

The invention of clausal form was a crucial step in the development of a deep mathematical analysis of FOL. Although the use of FOL was primarily suggested in the late 1950's for representation and reasoning in artificial intelligence (AI), the first such systems were developed by logicians interested in mathematical theorem proving.

B. *Post-resolution period*

After the development of resolution, work on FOL 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 considering negation as failure. A subset known as MICRO-PLANNER was implemented and used in the SHRDLU natural language understanding system.

C. *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. As a result, forward chaining has been used in a wide variety of systems, ranging from Nevins's geometry theorem prover to the R1 expert system for DEC-VAX™ computers configuration. The production system language OPS-5 [4] was used for R1 and for the SOAR cognitive architecture [11]. OPS-5 incorporated the rete match process. SOAR, which generates new rules to cache the results of previous computations, was able to 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 [20] is another popular example of production system as a C-based language developed at NASA. CLIPS allowed better integration with other software, hardware, and sensor systems and was used for spacecraft automation and several military applications.

D. *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 with forward inference [14]. 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 [5].

Logical inference complexity has mainly come from the deductive database community. In [7], it was first showed that matching a single non-recursive rule, or a conjunctive query in database terminology, can be NP-hard. In [10] data complexity is defined 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. In [2], the authors make use of ontology for intelligent database describing the concepts, operations and restrictions of these databases with an implementation using Protégé.

E. *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 [8] describe the early SAM theorem prover, which helped to solve an open problem in lattice theory. In [19], an overview is given about 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, called OTTER, in solving open problems. The work in [18] describes SPASS, one of the strongest current theorem provers.

III. Design of the FO2L environment

In the design of the new language, 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 both users and experts to easily interact with the environment. In this regard, FO2L plays a key role in knowledge representation on the basis of production rules. The relation between FO2L and the complete KBS is described in Figure 1 below representing the class diagram.

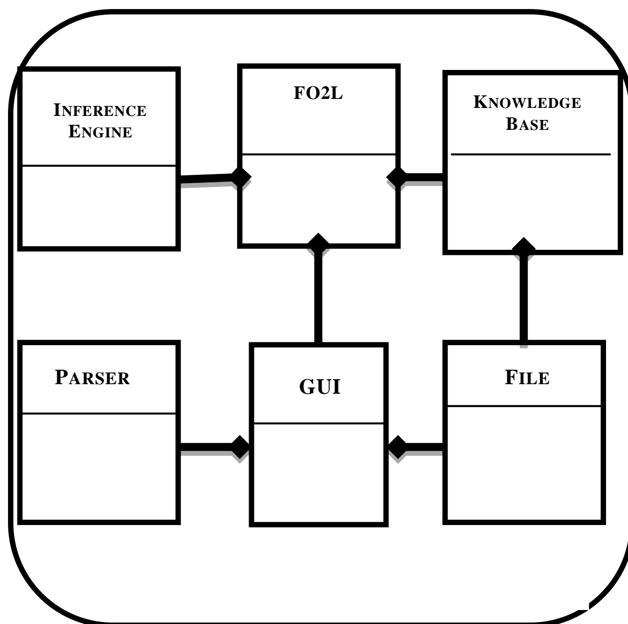

Figure 1 Class Diagram of the overall KBS

Based on the architecture presented in Figure 1 above, we describe FO2L and follow the steps of its implementation from grammar writing to programming of its lexical and syntactic analyzers.

A. *Logical language characteristics*

Knowledge bases are written using different formalisms such as frames, production rules and neural networks, among others. Production rules are the most natural way to represent knowledge whenever it can be understandably expressed in the form of IF-THEN rules. A production rule can be written using propositional (zero-order logic) or predicate calculus (FOL). In FO2L, the user can write knowledge bases using both types of logic.

We have adopted the rule-based solution for the following reasons:

- *Readability*: the use of a rule can be intuitively explained to the system user.
- *Flexibility*: Developers and users can modify some rules without breaking the entire system.
- *Extensibility*: new knowledge can be incorporated into the system simply by adding new rules without any concern of how these might fit into the overall knowledge base.

Moreover, the separation or independence of knowledge from processing allows us to postpone implementation of the inference engine.

B. *Grammar*

A grammar is defined a 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 called the start symbol and is a distinguished symbol in N. Programming languages can be expressed using two kinds of grammars: regular and context free grammars (CFGs). We express the grammar of FO2L in Backus-Naur form as follows.

```
<KB> →  <Declaration>{<KB>}*
<Declaration> → <RDeclaration> | <FactDeclaration>
<RDeclaration> → rule <Id> <Rule> |  <Rule>
<Rule> → if <Conds> then <Concls>
<Conds> → [not](<Predicate>){<Conds>}*
<Predicate> →

         PredicName>{<Elements>}*{<Operation2>}*
         | <Operation>
<PredicName> → <Id>
<Operation2> → <ArithOp> {?<Id>3|Num}+2
<ArithOp> → +|-|*|/|%
<Operation> → <LogicOp> {<Operands>}+2
<LogicOp> → <|>|<=|>=|==|!=
<Operands> → ?<Id>3{?<Id>3 |<Id>3 |<Num>}
<Elements> → <Id>3 |?<Id>3 | ?- | &- | <Num>
<Concls>→ <FactRule> |<FactRule> <OtherConclus
<OtherConclusions>→ {add|infere|remove}
                    <FactRule><OtherConclusions>
            | execute <Statement><OtherConcls>
<FactRule>→(<Id>4{<Id>3|?<Id>3|<Num>|?-|&-}*
)
<Num> →integer or decimal with two digits
after the decimal point
<FactDeclaration> → facts <Fact> {< Fact>}*
<Fact> → ( <Id>4 { <Id>3 |<Num>}*)
```

```
<Statement> → print <List To Be Printed>
            | read ?<Id>
<List To Be Printed> →{"anything"|?<Id>}{+
{"anything"|?<Id>}}
```

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. The notation **<id>n** means an identifier with at most **n** characters. Although, our language is based on FOL, its use is transparent to the user. This characteristic allows its usability in a wide variety of applications.

C. Parser

A compiler is a program that translates a given program from one source language to a target language. This translation requires that the input program is syntactically and lexically correct. The compiler outputs some plausible errors if the source program is not written in the required lexical and syntactical rules of the source language. This translation goes through lexical, syntax, semantic analyzers, intermediate code generator, code optimizer with a handler for errors and a manager for user defined symbols. As FO2L belongs to the logical languages paradigm, the notion of semantics and of variables, as defined by Von-Newman, has no meaning.

Rather, in our case, the compiling step is reduced to the transformation of the knowledge base (our program) from FO2L language into a form (data structure) understandable by the inference engine. Therefore, only lexical and syntactic analysis steps are required here. The code generation required in classical compilers is replaced, in our case, by a generation of the equivalent data structure. We need a compiler-like feature for the lexical phase on the basis of different automata for different lexical entities. For the syntactical phase, the parser is responsible for the pre-compilation phase.

There are different methods for parsing such as top-down, bottom-up and recursive descent. This latter is used in the implementation of FO2L. The advantage of this approach is that it can be directly written based on the grammar [1]. In this approach, there is a function for each non-terminal. For example, this approach has been successfully applied in the development of the so-called ESPL programming language as a support for the development of real-time distributed applications, designed for embedded systems and proposes a comparative study between real-time features of many already existing programming languages [6].

However, the application of additional compiler techniques, such as type inference, made Prolog programs competitive with C programs in terms of execution speed.

D. 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, as shown in Figure 1.

1. Operations

FO2L uses the following operations:

i) **add**: when executed, the **add** operation allows the addition of a fact inside the knowledge base. Its syntax is **add(fact)**.

ii) **infer**: when executed, the **infer** operation allows the addition of a fact, not present in the Fact Base, to the working space (temporarily). Its syntax is **infer(fact)**.

iii) **remove**: the inverse operation of the **add** operation. When executed, the **Remove** operation allows the removal of a fact from the knowledge base. Its syntax is **remove(fact)**.

iv) **execute**: this operation starts a portion of program in another programming language. Its syntax is **execute(statement)**

2. Statement and argument

i) **statement**: instructions following **execute** operation.

ii) **argument**: any item used in a predicate, or a fact.

E. Data structures

1. Linked lists as basic data structures

Linked lists are the main data structure used in the implementation. Linked lists are characterized by their dynamicity and updating flexibility. These characteristics allow using just the required space while keeping modularity of production rules. The Rule Base is a linked list of rules, initially empty. Any new rule is added at the end of the list, based on the modularity of the rules. Usually the order of introducing rules in the KBS is irrelevant. But, if the order of rules is important, then the expert has to enter them in the required order. A node in the Rule Base is four-field structure comprising the following:

- name of the rule,
- list of conditions,
- list of conclusions, and
- link to next rule.

Figure 2 below shows a simple representation of the Rule Base.

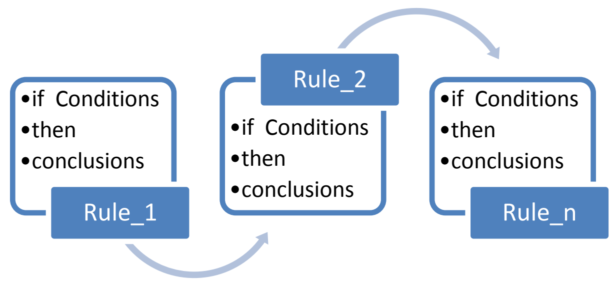

Fig. 2 List of rules in the Rule Base

2. Rule Base internal representation

Figure 3 shows the detailed structure of one rule in the memory. A Rule is an IF-THEN structure representing one of the most important pieces of knowledge and incorporates two lists for condition and conclusion parts.

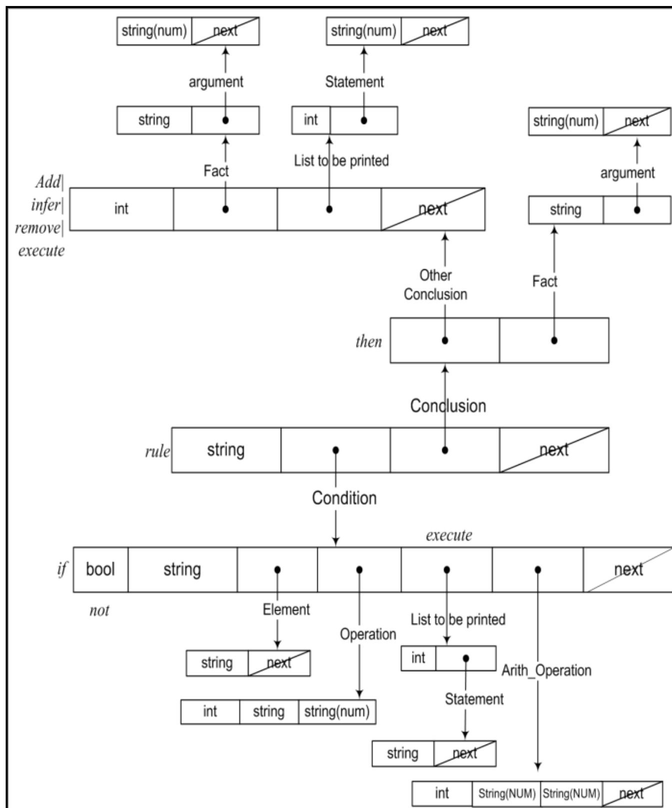

Fig. 3 Internal representation of a Rule

3. Fact Base internal representation

The Fact Base is also stored in memory as a list of facts. In Figure 4, a fact is a list with two fields; the first is the predicate and the second is a list of different attributes of the predicate.

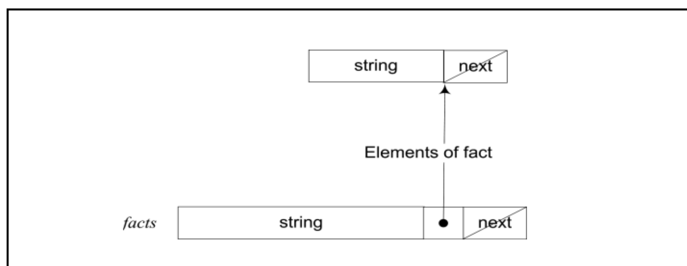

Fig. 4 Internal representation of a Fact

4. Condition part of a Rule

Each node in the condition list is structured as follows:

```
struct  ICondition
{ bool not; // for negation
string condName; // condition name
IElement* pIElement; //  element list
IOperation*pIOperation;// logical operation
IListStmt* PIStm; // ListToBePrinted
IOperation2* PIOp2; // arithmetic operation.
ICondition* next; // next condition
};
```

5. Conclusion part of a Rule

Each node in the conclusion list is structured as follows:

```
struct  IConclusion
{    IFact* pIFact; // fact data structure.
   IOtherCon* pIOtherCon; //
OtherConcolusion
 };
```

IV. EXAMPLE

Let us consider the following sentence:

"If an animal flies and lays eggs then it is a bird".

This sentence represents a unit piece of knowledge to be stored in the knowledge base. Suppose as a fact that a given animal has the two required characteristics. Translation of both rule and fact give the following knowledge base content with their corresponding internal representation.

A. Knowledge Base content

Rule Base content
rule r1 **if** (FLIES ?x) (LAYS ?x eggs) **then add** (IS ?x bird)
Fact Base content
facts (FLIES z) (LAYS z eggs)

B. Internal representation

1. Internal representation of the Rule

Figure 5 shows the internal representation of the Rule.

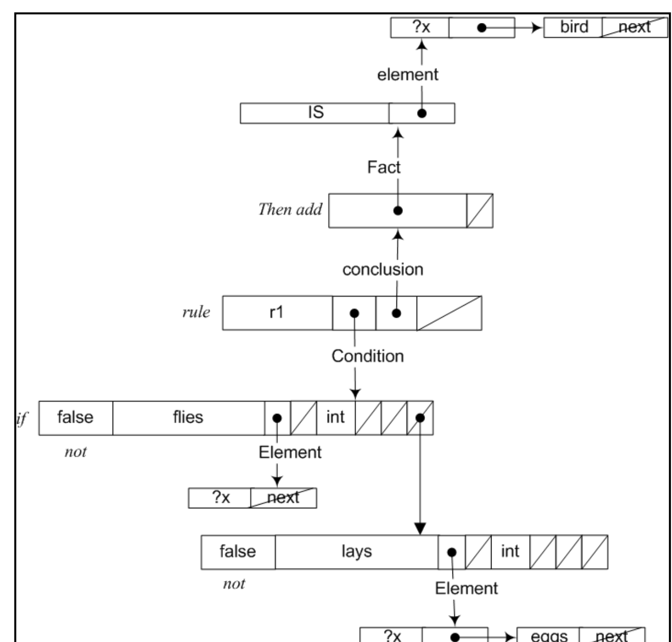

Fig. 5 Internal representation of the above rule

2. Internal representation of the Fact

Figure 6 shows the internal representation of the Fact

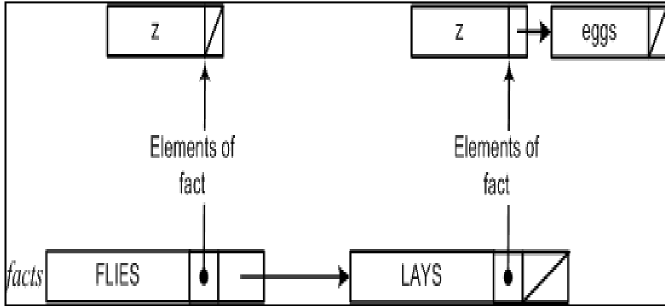

Fig. 6 Representation of the above Fact

V. Further Example of Knowledge Base in FO2L

As FO2L supports both propositional and predicate logic, we show how this can be done through an additional example. First, we give a brief description of the problem and then we write some rules in both types of logic as done by FO2L.

A. Wumpus world

The Wumpus world consists of a cave with 16 rooms, arranged as a 4x4 matrix and where an intelligent agent is initially in bottom-left room, labeled Room11 [15]. There are 3 pits but only one Wumpus in an unknown room and it doesn't move. One room contains gold to be discovered by the agent avoiding the Wumpus and all pits. Different percepts allow the agent to infer these dangerous locations. For example, if the agent perceives a breeze in one room, it can infer that there is one pit in at least one of the directly adjacent rooms, *i.e.* sharing walls with the actual agent's room, but not diagonally. Also, perception of stench in one room indicates the presence of the Wumpus in one of the adjacent rooms. For example, S_{12} indicates stench in Room 12 and therefore inferring the presence of the Wumpus in either Room 22 (W_{22}??) or Room 13 (W_{13}??) where the "??" represents a plausibility. Similarly, when the agent is breezy Room 21, (B_{21}) then the possible inferences are either a Pit in 22 (P_{22}??) or a Pit in Room 31 (P_{31}??), or perhaps in both rooms. The 16-room initial world is described as follows:

14	24	34	44
13 Wumpus W_{22}??	23	33	43
12 Stench (S_{12})	22 Pit (P_{22})?? Wumpus W_{22}??	32	42
11 Initial position of agent OK	21 Breeze (B_{21})	31 Pit P_{31}??	41

Fig. 7 The Wumpus world

B. Propositional logic

In propositional logic, propositions are sentences that can be assigned either a value "True" or "False" and nothing else. We start by writing rules in propositional logic describing one type of the general rules of such an agent. Some of the rules are expressed in the following table. In this example, we suppose that breeze can be perceived by the agent and therefore breeze is considered first as a premise.

Rule #	Rule form	Premise part of the rule	Conclusion part of the rule
Rule r1	$B_{11} \Leftrightarrow (P_{12} \cup P_{21})$	`(B 1 1)`	`Add (P 1 2)` `Add (P 2 1)`
...	...	...	...

Rule r7	$B_{32} \Leftrightarrow (P_{31} \cup P_{33} \cup P_{22} \cup P_{42})$	`(B 3 2)`	`Add (P 3 1)`
			`Add (P 3 3)`
			`Add (P 2 2)`
			`Add (P 4 2)`

...		...	....
Rule r16	$B_{44} \Leftrightarrow (P_{43} \cup P_{34})$	`(B 4 4)`	`Add (P 4 3)` `Add (P 3 4)`

These propositions tell the agent that when breeze is perceived in a room, at least a pit exists in one of the adjacent rooms.

C. Predicate Logic

As propositional logic fails in the description of general properties with variables and mathematical quantifiers, predicate logic provides a more concise description. We can rewrite the previous rules (from r1 to r16) above using predicates as described in the following table.

Rule #	Rule in logical form	Premise part of the rule in FO2L	Conclusion part of the rule expressed in FO2L

r1	$B_{x,y} \Leftrightarrow (P_{x,y+1} \cup P_{x+1,y})$
	$x=1$
	$y=1$
		`(B ?x ?y)`
		`(= ?x 1)`
		`(= ?y 1)`
			`Add`
			`(P ?x ?y+1)`
			`(P ?x+1 ?y)`

...	...	...	...

r7	$B_{x,y} \Leftrightarrow$
	$(P_{x,y+1} \cup P_{x,y-1} \cup P_{x+1,y} \cup P_{x-1,y})$
	$1<x<4$
	$1<y<4$
		`(> ?x 1)`
		`(< ?x 4)`
		`(> ?y 1)`
		`(< ?y 4)`
			`Add`
			`(P ?x ?y+1)`
			`(P ?x ?y-1)`
			`(P ?x+1 ?y)`
			`(P ?x-1 ? y)`

...	...	...	...

r9	$B_{x,y} \Leftrightarrow (P_{x,y-1} \cup P_{x-1,y})$
	$x=4$
	$y=4$
		`(B ?x ?y)`
		`(= ?x 4)`
		`(= ?y 4)`
			`Add`
			`(P ?x ?y-1)`
			`(P ?x-1 ?y)`

For instance, Rule r7 replaces 4 propositional rules describing the Wumpus' position in the 4 central rooms, *i.e* those surrounded by rooms in all directions. There are 9 FOL rules instead of 16 in propositional logic.

VI. Graphical User Interface

Our implementation relies on advanced tools for graphical user interface (GUI). The available tools greatly helped in the implementation of the architectural concepts exposed above. The GUI is used in comparing, storing, retrieving, modifying and deleting different versions of knowledge bases. We show how a user can interact with our system *via* the proposed GUI. The figures are self-explanatory. See Figure 8 below, illustrating the main window, up to Figure 14.

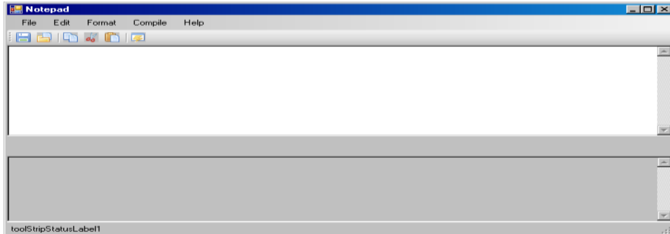

Fig. 8 Main Window

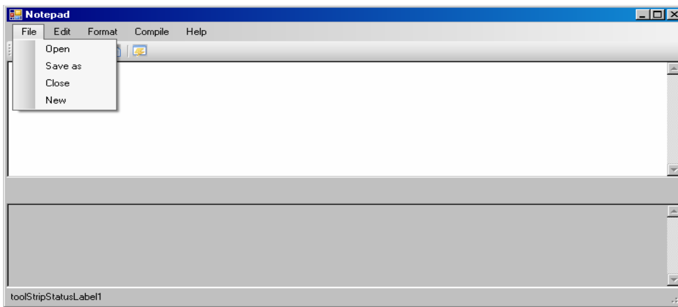

Fig. 9 Open-Save-Close-New / File menu

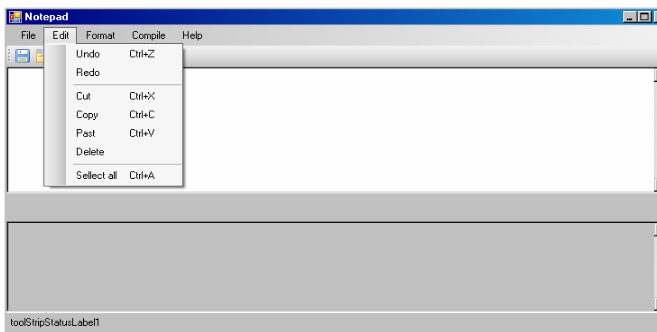

Fig. 10 Edit Menu

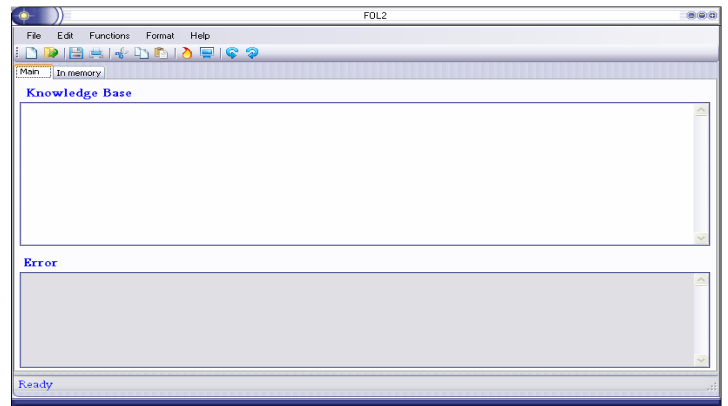

Fig. 11 Knowledge Base Main Menu

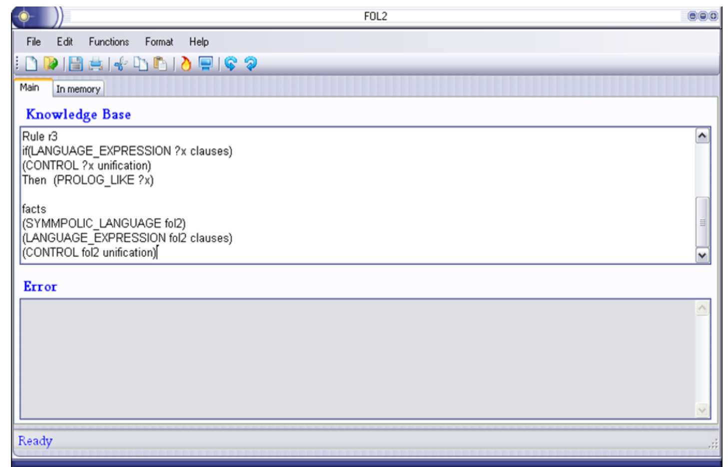

Fig. 12 Editor for writing a new knowledge base

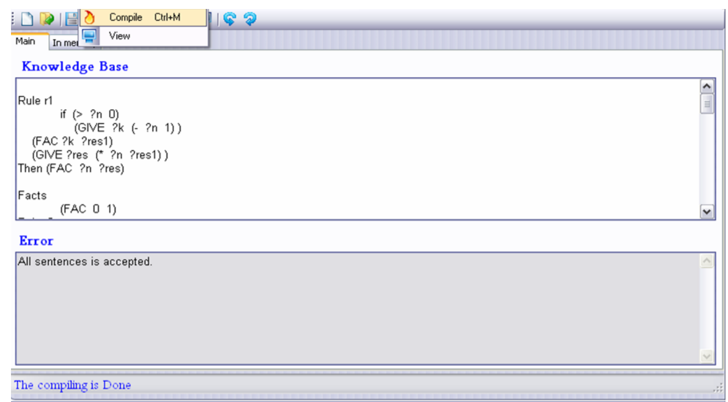

Fig. 13 Pre-compiling step

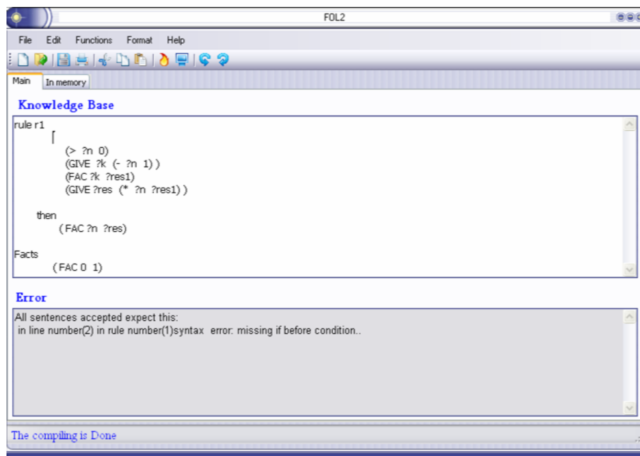

Fig. 14 Example with syntax errors

VII. Conclusion

We have described the most important methodological steps needed to design and implement a programming language and more specifically a logical language. As it stands now, FO2L is a language that offers powerful assistance to both experts and users in writing, parsing, compiling, storing, retrieving, modifying or deleting knowledge bases of concern. Although FO2L is a relatively simple language, it is powerful enough to allow knowledge base systems management in either propositional or in predicate logic. It also allows, *via* its simple set of arithmetic and logical operations, to combine between the declarative and the procedural approaches. 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 development environment. Further, to allow the system to deal with gradual human perception, we aim to add a fuzzy logic component.

References

[1] A.V. Aho, and J.D. Ullman, "Principles of Compiler Design", Pearson Education, 2007
[2] M. Ana, A. Jose, "A General Ontology for Intelligent Database", *Int. J. of Comp.,* 3(1):102-108, 2007
[3] R.J. Brachman, and H.J. Levesque, "*Knowledge Representation and Reasoning*", Morgan Kaufmann, 2004
[4] L. Brownston, R. Farrell, E. Kant and N. Martin, *"Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming"*, Addison-Wesley, 1985
[5] S. Ceri, G. Gottlob and L. Tanca, *"Logic Programming and Databases"*, Springer-Verlag, 1990
[6] H. Ciocarlie and CM Vacarescu, "Considerations regarding the implementation of the ESPL programming language", *Int. J. of Comp*., 4(2):401-410, 2008
[7] A.K. Chandra and P.M. Merlin, "Optimal implementation of conjunctive queries in relational databases" In *Proc. of the 9th Annual ACM Symp. on Theory of Computing*, pp. 77-90, 1977
[8] J. Guard, F. Oglesby, J. Bennett, and L. Settle, "Semi-automated mathematics", *Journal of the ACM*, 16:49-62, 1969
[9] S. Kendal and M. Creen, *"An Introduction to Knowledge Engineering"*, Springer, 2007
[10] G.M. Kuper and M.Y. Vardi, "On the complexity of queries in the logical data model", *Theoretical Computer Science*, 116(1):33-57, 1993
[11] J.E. Laird, A. Newell and P.S. Rosenbloom, "SOAR: An architecture for general intelligence", *Artificial Intelligence*, 33(1):1-64, 1987
[12] W. McCune, "Solution of the Robbins problem", *J. of Automated Reasoning*, 19(3), 263-276, 1997
[13] M. Negnevitsky, *"Artificial Intelligence. A guide to Intelligent Systems"*, Addisson Wesley, 2nd Ed. 2005
[14] R. Ramakrishnan and J.D. Ullman, "A survey of research in deductive database systems", *J. of Logic Programming*, 23(2), 125-149, 1995
[15] S. Russel and P. Norvig, *"Artificial Intelligence – A Modern Approach"*, Prentice Hall, 2nd Edition, pp. 51-54 & pp.310-315, 2003
[16] J.F. Sowa, *"Knowledge Representation: Logical, Philosophical, and Computational Foundations"*, Brooks/Cole, New York, 2000
[17] E.S.L. Shapiro, *"The Art of Prolog", Advanced Programming Techniques"*, MIT Press, 2nd Edition, 1994
[18] C. Weidenbach and D. Dimova, A. Fietzke, R. Kumar, M. Suda and P. Wischnewski, "SPASS Version 3.5", In *22nd Int. Conf. on Automated Deduction, CADE 2009*, LNCS 5663, pp. 140-145, 2009
[19] L. Wos, R. Overbeek, E. Lusk and J. Boyle, *"Automated Reasoning: Introduction and Applications"*, 2nd Ed., McGraw-Hill, 1992
[20] R.M. Wygant, "CLIPS- a powerful development and delivery expert system tool", *Computers and Industrial Engineering*, 17,546-549, 1989

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

Chafia Kara-Mohamed (*alias* Hamdi-Cherif) was born in Bordj-Bou-Arreridj, Algeria. She received "Diplôme d'Ingénieur d'État" in Computer Science from Ferhat Abbas University, Setif, (UFAS), Algeria, and "Magister" in Computer Science (Artificial Intelligence) from USTHB, Algiers, in 1988 and 1994, respectively and "Doctorat d'État" in Computer Science (Artificial Intelligence) from UFAS in 2012.
She worked as demonstrator at USTHB from 1988 to 1994 and taught at UFAS Computer Science Department from 1994 to 2001. In 2002, she joined Computer Science Department, Qassim University, where she is Instructor. She supervised 15 BS student projects and collaborated in 8 research projects. Her currents interests are grammatical inference, machine learning and bioinformatics. Dr. Chafia is member of IEEE and ACM.

Noura Ibrahim Al-Osily was born in Qassim, Saudi Arabia. She received Bachelor of Science in Computer Science from Qassim University, in 2010 with First Class Honors. She is Teaching Assistant with Computer College at Qassim University, Saudi Arabia. She is now on leave of absence in Cleveland, Ohio, USA, preparing for graduate studies. Her actual interests are knowledge-based systems and programming languages, particularly logical languages.

Rayy Abdulaziz Al-Marshad was born in Jeddah, Saudi Arabia. She received Bachelor of Science in Computer Science from Qassim University, Saudi Arabia, in 2010 with First Class Honors. She is now with Buraydah College of Technology, Qassim, Saudi Arabia. Her actual interests are programming languages.

Modhi Saad Al-Twijary received Bachelor of Science in Computer Science from Qassim University, in 2010. She is now Relationship Manager with Bank Al-Rajhi, Buraydah, Qassim. Her actual interests are software engineering and human relations.

Johara Al-Robe'an received Bachelor of Science in Computer Science from Qassim University, Saudi Arabia, in 2010. She is now Computer Analyst with Bank Al-Rajhi, Buraydah, Qassim, Saudi Arabia. Her actual interests are programming languages.

Aboubekeur Hamdi-Cheri was born in Setif, Algeria. He received BSc and MSc in Electrical Engineering, both from Salford University, Manchester, England, in 1976 and 1978, respectively, and PhD degree in Computer Science from Université Pierre et Marie Curie Paris 6, (UPMC) Paris, France, in 1995.
He worked with Algerian Petroleum Company SONATRACH. He taught at École Supérieure des Transmissions, Algiers, at Université de Bretagne Occidentale, Brittany, France, at ESLSCA, Paris, and at UFAS, Algeria.
In 2001, he joined Qassim University, Saudi Arabia, where he is Professor and Deputy Head of Computer Science Department. He supervised about 95 BS student projects, and 10 master and doctoral students, and directed 12 research projects. He is currently interested in machine learning, intelligent control and bioinformatics. Dr. Aboubekeur is member of IEEE and ACM.