Application of First-Order Logic in Knowledge Based Systems

Article · January 2013

CITATIONS
4

READS
5,141

2 authors:

Ibiyinka Temilola Ayorinde
University of Ibadan
14 PUBLICATIONS **6** CITATIONS
SEE PROFILE

Babatunde Akinkunmi
University of Ibadan
37 PUBLICATIONS **84** CITATIONS
SEE PROFILE

Vol 6. No. 3, September 2013
African Journal of Computing & ICT

www.ajocict.net

Application of First-Order Logic in Knowledge Based Systems

I.T. Ayorinde & B.O. Akinkunmi
Computer Science Department
University of Ibadan
Ibadan, Nigeria.
temiayorinde@yahoo.com, ope34648@yahoo.com

ABSRACT

For a system to be "artificially intelligent," it must contain a component that can be understood as linguistic. In order words, it can be expressed in some languages such that this component contains the knowledge of the system and drives its intelligent behaviour. This paper describes how first order logic can be used as a representational language for a knowledge base and inferences from it can be used to drive the intelligent behaviour of the knowledge based system.

Keywords: Artificial Intelligence, Knowledge Representation, Knowledge Based System, First Order Logic.

African Journal of Computing & ICT Reference Format:
I.T. Ayorinde and B.O. Akinkunmi (2013). Application of First-Order Logic in Knowledge Based Systems.
Afr J. of Comp & ICTs. Vol 6, No. 3. Pp 45- 52.

1. INTRODUCTION

The major goals of artificial intelligence are to construct computer programs that perform at high levels of competence in cognitive tasks and to understand and develop computational models of human intelligence [10]. These can be achieved with the help of knowledge representation and reasoning.

Virtually all knowledge representation (KR) languages are based, in some way, on formal logic like propositional logic, predicate logic and temporal logic, among others. Formal logic shares with English the advantage that it can express facts about the world. It shares with computer programming languages the advantage of being clear and unambiguous.

A logic is a formal language that includes Syntax (rules for describing legal sentences), Semantics (how sentences represent facts in the world) and Proof theory (rules for inferring sentences from other sentences).

The language of classical logic that is most widely used in the theory of knowledge representation is the language of first-order (predicate) formulas [5]. First-order logic (FOL) is a powerful tool for knowledge representation and reasoning. Unlike propositional logic which deals with simple declarative propositions, FOL additionally covers predicates and quantifications, allows reasoning about properties that are shared by many objects through the use of variables and also allows more flexible and compact representation of knowledge.

A KR system is to provide a fact management service. That is, the job of a KR system is to manage a knowledge base (KB) in some representation language (including both syntax and semantics) and to answer questions about what semantically follows from the current KB [8]. KR system must not only be sound, it must also be complete as discussed in section 5.

Section 2 gives a synopsis of knowledge based system while the syntax and semantics of first order logic are examined in sections 3 and 4 respectively. Properties of formal systems are discussed in section 5 while section 6 shows inferences in first order logic and section 7 gives the conclusion.

2. KNOWLEDGE-BASED SYSTEM

Knowledge-Based System (KBS) consists of a Knowledge Base (KB) and an inference engine. A knowledge base is a set of sentences that describe the world in some formal (representational) language (e.g. first-order logic) while an Inference Engine is a set of procedures that work upon the representational language and can infer new facts or answer KB queries (e.g. resolution algorithm and forward chaining among others) [6]. The knowledge base has domain specific knowledge and the inference engine is domain independent. Basic KBS architecture is shown in figure 1 below:

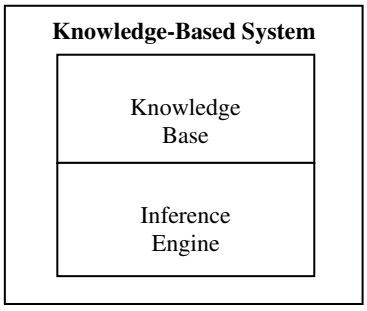

Figure 1: Knowledge-Based System

Knowledge-Based system can be described with four major characteristics:

- Symbolic: It incorporates knowledge that is symbolic (as well as numeric).
- Heuristic: It reasons with judgemental, imprecise and qualitative knowledge as well as with formal knowledge of established theories.
- Transparent: Its knowledge is simply and explicitly represented in terms familiar to specialists and is separate from its inference procedures. It provides explanations of its line of reasoning and answers to queries about its knowledge.
- Flexible: It is incrementally refinable and extensible. More details can be specified to refine its performance; more concepts and links among concepts can be specified to broaden its range of applicability.

Generally speaking in KBS, the program is not instructed on what to do but what to know. It keeps the knowledge in the KB and chooses a representation with high level of transparency. The task is elucidating and debugging knowledge, not writing and debugging a program. This makes this process to be called knowledge programming [10].

3. SYNTAX OF FIRST-ORDER LOGIC

The syntax determines which collections of symbols are legal expressions in first-order logic [1,7].

3.1 Symbols:

Constant: A | John | Car
Variable: x | y | z |...
Predicate: Brother | Owns | ...
Function: father-of | plus | ...
Connectives: ¬, ∧, ∨, →, ↔ (Negation, Conjuction, Disjunction, Implication, If and only if (iff))
Quantifiers: ∀, ∃ (For all, There exists at least one)

In first order logic, there are three kinds of constants namely object constants, function constants and relation constants [3]. Object constants refer to objects in a universe of discourse. Objects can be anything we want to say something about like integers, people, real numbers and geometric shapes among others. Function constants refer to functions like mother, age, plus and times. Each function constant has an associated arity indicating its number of arguments. Example: mother(Jane) - here, mother has arity 1 (unary) while in times(3,2) - times has arity 2 (binary).

An object constant is really a special case of a function constant with arity 0. Relation constants refer to relations between or properties of objects. Example: loves, betterthan and ishappy among others. Each relation constant also has an associated arity. Loves has arity 2 while ishappy has arity 1.

Predicate and Function symbols have an arity (number of arguments) as seen in the discussion above. 0-ary predicate is equivalent to propositional logic atoms while 0-ary function is a constant. Objects are represented by terms. Terms are simply names for objects. Logical functions are not procedural as in programming languages. They do not need to be defined, and do not really return a value. They allow representation of an infinite number of terms. An *n*-ary function maps a tuple of *n* terms to another term. Example: father-of(John), successor(0), plus(plus(1,1),2).

A predicate represents a property of or relation between terms that can be true or false. For example: Brother(John, James), Left-of(Square1, Square2), GreaterThan(plus(2,1), plus(0,1)). In a given interpretation, an *n*-ary predicate can be defined as a function from tuples of *n* terms to {True, False} or equivalently, a set of tuples that satisfy the predicate: {<John, James>, <John, Tolu>, <Ade, Ayo>, ...}

3.2 Sentences

A sentence can either be an atomic sentence or a complex sentence. An atomic sentence is simply a predicate applied to a set of terms as shown below:

Owns(John, Car)
Sold(Ade, Car, Ayo)

The standard propositional connectives (¬, ∧,∨, →, ↔) can be used to construct complex sentences as shown below:

Owns(John, Car) ∨ Owns(James, Car)
Sold(Ade, Car, Ayo) → ¬Owns(Ade, Car)

4. SEMANTICS OF FOL

Semantics is True or False depending on the interpretation. For example, it can be determined if the predicate is true of the arguments shown in complex sentences above. An interpretation of a first-order language assigns a denotation to all non-logical constants (predicates (relations), functions and constants) in that language. It also determines a domain of discourse that specifies the range of the quantifiers [5]. The interpretation of an *n*-ary predicate symbol is a set of *n*-tuples of elements of the domain of discourse. This means that, given an interpretation, a predicate symbol, and *n* elements of the domain of discourse, one can tell whether the predicate is true of those elements according to the given interpretation. For example, an interpretation *I(P)* of a binary predicate symbol *P* may be the set of pairs of integers such that the first one is less than the second. According to this interpretation, the predicate *P* would be true if its first argument is less than the second.

4.1 Quantifiers

Universal Quantification Syntax: ∀ [1].

Example:

The statement "Everyone studying in Unibadan is wise" is logically represented as:

∀x StudiesAt(x, Unibadan) → Wise(x)

Universal Quantification Semantics:

∀x P is true in a model if and only if for all domain elements d in the model, P is true in the model when x is interpreted by d. In other words, universal quantifier (∀x) asserts that a sentence is true for all values of variable x. Other examples are shown below:

∀x Loves(x, Coke) - For all values of x, whatever x may represent, x loves Coke.

∀x Whale(x) → Mammal(x)- For all kinds of x where x represents Whales, implies that x is a mamma.

Universal quantification naturally uses implication.

Existential Quantification Syntax: ∃

Example:

The statement "Someone studying in Unibadan is wise" is logically represented as:

∃x StudiesAt(x, Unibadan) ∧ Wise(x)

Existential Quantification Semantics:

∃x P is true in a model iff there is a domain element d in the model such that P is true in the model when x is interpreted by d. Existential quantifier (∃x) asserts that a sentence is true for at least one value of a variable x as shown below:

∃x Loves(x, Coke) - There exists at least a value of x where x loves Coke.

∃x(Cat(x) ∧ Color(x, Black) ∧ Owns(Mary,x)) - there exists at least a variable x where x is a Cat with colour Black and Mary owns the Cat.

Existential quantification naturally uses conjunction.

4.2 General Identities

Examples of general identities are shown below:

- ∀x P → ¬ ∃x ¬P (Example, ∀x Likes(x, Fish) is the same as ¬∃x ¬Likes(x, Fish)
- ∃x P → ¬ ∀ x ¬P (Example, ∃x Likes(x, Meat) is the same as ¬ ∀ x ¬ Likes(x, Meat)

Others are:

- ∀x ¬P → ¬∃x P
- ¬ ∀x P → ∃ x ¬P
- ∀x P(x)∧Q(x) → ∀xP(x) ∧ ∀xQ(x)
- ∃x P(x) ∨ Q(x) → ∃xP(x) ∨
- ∃xQ(x)

4.3 Nesting Quantifiers

The order of quantifiers of the same type doesn't matter. This is shown in the example below:

∀x ∀y(Parent(x,y) ∧ Male(y) → Son(y,x)) - For all values of x and y where x is the parent of y and y is male, implies that y is always the son of x.

∃x ∃y(Loves(x,y) ∧ Loves(y,x)) - There exists at least a person x and at least a person y, where x loves y and y loves x.

Unlike the same quantifiers, the order of mixed quantifiers does matter as seen in the examples below:

∀x ∃y(Loves(x,y)) - Says everybody loves somebody, i.e. everyone has someone whom he /she loves.

∃y ∀x(Loves(x,y)) - Says there is someone who is loved by everyone in the universe.

∀y ∃x(Loves(x,y)) - Says everyone has someone who loves them.

∃x ∀y(Loves(x,y)) - Says there is someone who loves everyone in the universe.

4.4 Proof And Entailment

A proof is a sequence of sentences such that every sentence in the sequence is an axiom or can be derived from sentences earlier in the sequence by applying rules of inference. Proof is a syntactic notion. In constructing proofs, we don't consider the meaning of sentences. KB |- p means we can prove/derive p from KB. Entailment is a semantic notion. It depends on the meaning we give to logical connectives. KB |= p means that p is entailed by KB; that is, whenever KB is true, the sentence p is also true [9,5].

4.5 Validity, Satisfiability, And Logical Consequence

- If a sentence φ evaluates to True under a given interpretation *M*, one says that *M* **satisfies** φ; this is denoted as M |= φ . A sentence is **satisfiable** if there is some interpretation under which it is true. (Sentence P **entails** sentence Q, written as P |= Q, if Q is true whenever P is true)
- A formula is **logically valid** (or simply **valid**) if it is true in every interpretation. These formulas play a role similar to tautologies in propositional logic.
- A formula φ is a **logical consequence** of a formula ψ if every interpretation that makes ψ true also makes φ true. In this case one says that φ is logically implied by ψ.

5. FORMAL SYSTEM

A formal system consists of a formal language, a set of axioms, a set of inference rules and a proof (or derivation).

Properties of A Formal System

A formal system is sound, complete and decidable.

- *Sound*

If KB |- p, then KB |= p. An inference algorithm that derives only entailed sentences is called sound or truth preserving.

- *Complete*

If KB |= p, then KB |- p. An inference algorithm is complete if it can derive any sentence that is entailed.

- *Decidable*

There is an algorithm that can decide in finite time whether any proposition is a theorem or not

6. INFERENCES IN FIRST-ORDER LOGIC

Truth tables don't work in FOL because sentences with variables can have an infinite number of possible interpretations. Many inferences can be made in FOL through Proofs, Unification, Generalized modus ponens, Forward and backward chaining, Completeness, Resolution and Logic programming.

6.1 Unification in FOL

Given sentences

$P1 \vee ... \vee Pn$ and $Q1 \vee ... \vee Qm$

where each Pi and Qi is a literal, i.e., a positive or negated predicate symbol with its terms, if Pj and ¬Qk unify with substitution list θ, then, we can derive the resolvent sentence:

subst(θ, P1 ∨... ∨ Pj-1 ∨ Pj+1 ... Pn ∨ Q1 ∨ …Qk-1 ∨ Qk+1 ∨... ∨ Qm) as shown in the example below:

From clause P(x, f(a)) ∨ P(x, f(y)) ∨ Q(y) and clause ¬P(z, f(a)) ∨ ¬Q(z), we can derive resolvent clause P(z, f(y)) ∨ Q(y) ∨ ¬Q(z) using θ = {x/z}

6.2 Resolution Proofs

Resolution is a **sound** but not **complete** inference procedure for FOL. Resolution **won't always give an answer** since entailment is only semidecidable and you can't just run two proofs in parallel, one trying to prove Q and the other trying to prove ¬Q, since KB might not entail either one [2,4].

- *Unit resolution*

P ∨ Q, ¬Q |- P

Given P or Q **and** not Q, the resolution gives P

- *Resolution*

P ∨ Q, ¬Q ∨ R |- P ∨ R

Given P or Q **and** NOT Q or R, the resolvent is P or R

A resolution proof tree is shown below:

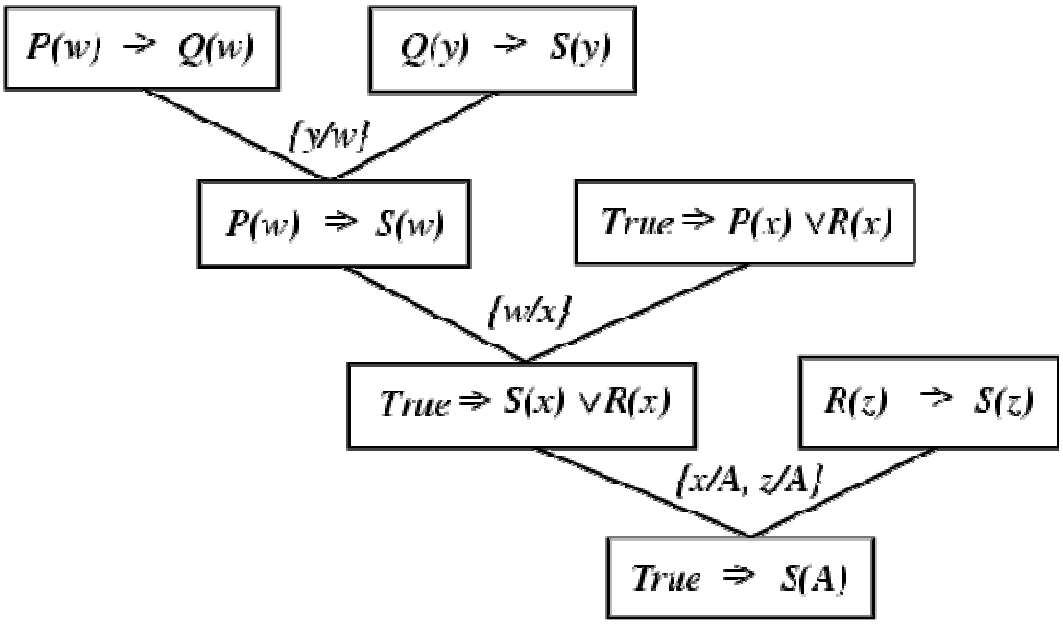

Fig. 2: A RESOLUTION PROOF TREE
(Source: Finin T. et. al.,2011)

6.3 Resolution Refutation Proofs

By itself, resolution is a sound rule of inference but not complete. However, a technique called resolution refutation is sound and complete. Resolution refutation is a form of proof by contradiction. It shows that KB |= p by showing that the set of clauses KB v {¬p} is unsatisfiable.
For example, given a consistent set of axioms KB and goal sentence Q, we can show that KB |= Q. This can be achieved with proof by contradiction as shown below:

Add ¬Q to KB and try to prove false i.e., (KB |- Q) ↔ (KB ∨ ¬Q |- False) .

To use resolution refutation, we first convert sentences to conjunctive normal form(CNF). Note that a **literal** is a propositional variable or its negation. A **clause** is a disjunction of literals and a sentence written as a conjunction of clauses is said to be in **conjunctive normal form.** Example: (¬P ∨ Q ∨ R) ∧ (P ∨ ¬R ∨ ¬S) ∧ (P ∨ ¬Q)
Any sentence in propositional logic can be written in conjunctive normal form (CNF) and resolution refutation requires sentences to be in conjunction normal form. To convert to CNF, **Remove biconditional** using the equivalence: P → Q becomes (P → Q)∧(Q → P)
Remove implication using the equivalence: P → Q becomes ¬P∨Q
Apply DeMorgan's laws:
¬(P∨Q) becomes ¬P∧¬Q
¬(P∧Q) becomes ¬P ∨¬Q
Apply Distributive laws
P∧(Q∨R) becomes (P∧Q)∨(P∧R)
P∨(Q∧R) becomes (P∨Q)∧(P∨R)
Remove double negation by associative law:
¬¬P becomes P.
Move negation inwards
¬∀x P becomes ∃x ¬P
¬∃x P becomes ∀x ¬P
Standardize variables
Each quantifier gets unique variables e.g.
∃xP(x) ∧ ∃xQ(x) becomes ∃xP(x) ∧ ∃yQ(y)
Move quantifiers to the left
∀xP ∨ ∃yQ becomes ∀x∃y P∨Q
Eliminate ∃ by Skolemization.
∃xP(x) becomes P(A)
∀x∀y∃z P(x,y,z) becomes ∀x∀y P(x,y,F(x,y))
∀x∃yPred(x,y) becomes ∀xPred(x,Succ(x))

To prove that a sentence p can be derived from a set of sentences KB (To prove KB |- p):

- Convert ¬p and the clauses in KB to CNF and combine into a single set of clauses, *Γ*
- Loop
- Find two clauses in *Γ* to which the resolution rule applies, but have not previously been used. If these can't be found, return "p cannot be proven."
- Apply the resolution rule to create a new clause. If the new clause is the empty clause (that is, a contradiction), return "p has been proven."
- Add the new clause to *Γ*.

For example, we can express the following statements in propositional logic and use it to prove that "it is raining"(R). The clause ¬ R will be added to the KB.

- If John carries an umbrella(U) and it is humid(H), then it is raining(R).- U ∧ H → R
- If it is humid, then John carries an umbrella.
  - H → U
- It is humid.
  - H

By applying the rules stated above under conversion to CNF,
U ∧ H → R becomes ¬(U ∧H) ∨ R and finally becomes ¬U ∨ ¬H ∨ R
H → U becomes ¬H ∨ U
The resolution refutation tree is shown below:

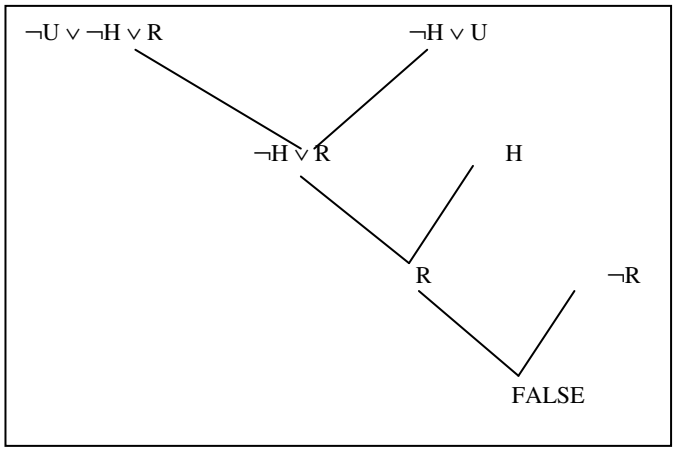

Fig. 3: Resolution Refutation Proof Tree (1)

Generally speaking, resolution refutation will terminate with the empty clause if KB ⊨ p. It will terminate without producing the empty clause if KB ⊭p. Resolution is a state-space search algorithm. A state is a set of clauses. The operator is resolution on two clauses. A proof has a tree structure with p as the root and clauses in the original KB as leaves. Intelligent search strategies can find proofs efficiently.

Another example is given with the scenario:
Anyone passing his Physics exams and winning the lottery is happy. But anyone who studies or is lucky can pass all his exams. John did not study but John is lucky. Anyone who is lucky wins the lottery. Is John happy?
First convert this to FOL:
1. Anyone passing his Physics exams and winning the lottery is happy.

$\forall x$ Pass(x, Physics) $\wedge$ Win(x, Lottery) $\rightarrow$ Happy(x)

2. But anyone who studies or is lucky can pass all his exams.

$\forall x \forall y$ Study(x) $\vee$ Lucky(x) $\rightarrow$ Pass(x,y)

3. John did not study, but John is lucky

$\neg$ Study(John) $\wedge$ Lucky(John)

4. Anyone who is lucky wins the lottery.

$\forall x$ Lucky(x) $\rightarrow$ Win(x, Lottery)

Then, convert to CNF. To do this, first eliminate the implications as shown below:
1. $\forall x \neg$ (Pass(x, Physics) $\wedge$ Win(x, Lottery)) $\vee$ Happy(x)
2. $\forall x \forall y \neg$ (Study(x) $\vee$ Lucky(x)) $\vee$ Pass(x,y)
3. $\neg$ Study(John) $\wedge$ Lucky(John)
4. $\forall x \neg$ Lucky(x) $\vee$ Win(x, Lottery)

Also, move $\neg$ inward as shown below:
1. $\forall x \neg$ Pass(x, Physics) $\vee \neg$ Win(x, Lottery)) $\vee$ Happy(x)
2. $\forall x \forall y$ ($\neg$ Study(x) $\wedge \neg$Lucky(x)) $\vee$ Pass(x,y)
3. $\neg$ Study(John) $\wedge$ Lucky(John)
4. $\forall x \neg$ Lucky(x) $\vee$ Win(x, Lottery)

Distribute $\wedge$ over $\vee$
1. $\neg$ Pass(x, Physics) $\vee \neg$ Win(x, Lottery)) $\vee$ Happy(x)
2. ($\neg$ Study(x) $\vee$ Pass(x,y)) $\wedge$ ($\neg$ Lucky(x) $\vee$ Pass(x,y))
3. $\neg$ Study(John) $\wedge$ Lucky(John)
4. $\neg$ Lucky(x) $\vee$ Win(x, Lottery)

Now, state as a set of disjunction of literals
1. $\neg$ Pass(x, Physics) $\vee \neg$ Win(x, Lottery) $\vee$ Happy(x)
2. a. $\neg$ Study(x) $\vee$ Pass(x,y)
2. b. $\neg$Lucky(x) $\vee$ Pass(x,y)
3. a. $\neg$ Study(John)
   b. Lucky(John)
4. $\neg$ Lucky(x) $\vee$ Win(x, Lottery)

Finally, Standardize variables apart as shown below to make it in CNF
1. $\neg$ Pass(x1, Physics) $\vee \neg$ Win(x1, Lottery) $\vee$ Happy(x1)
2. a. $\neg$ Study(x2) $\vee$ Pass(x2,y1)
2. b. $\neg$Lucky(x3) $\vee$ Pass(x3,y2)
3. a. $\neg$ Study(John)
   b. Lucky(John)
4. $\neg$ Lucky(x4) $\vee$ Win(x4, Lottery)

We now follow the resolution proof procedure by asserting negation of goal. Since the goal is Happy(John), the clause $\neg$ Happy(John) will be added to the KB. These clauses will be resolved together until FALSE is derived. This is shown in Fig. 4 below.

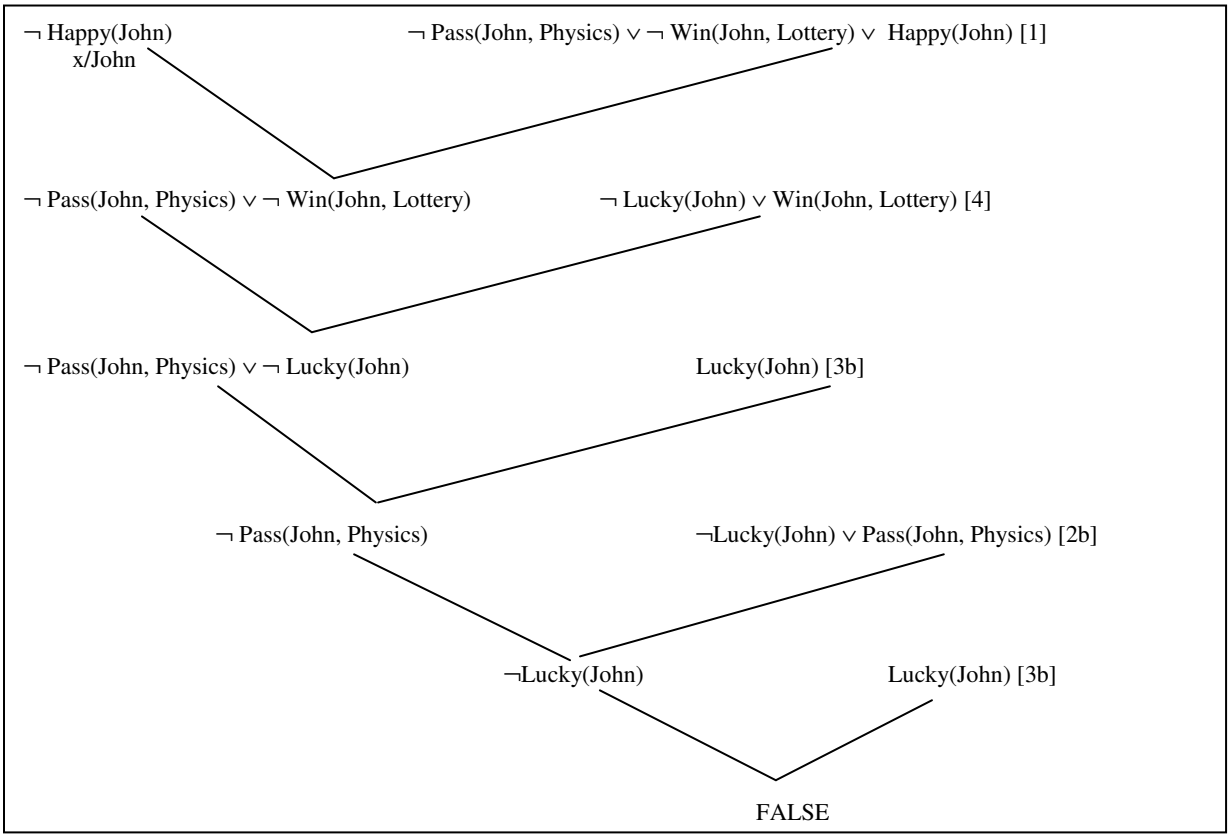

Fig. 4: Resolution Refutation Proof Tree (2)

7. CONCLUSION

This paper has shown how first order logic can be used as a representational language for knowledge based systems. This is made possible because it allows reasoning about properties that are shared by many objects through the use of variables and also allows more flexible and compact representation of knowledge.

8. FUTURE DIRECTIONS

We intend to explore other types of inferences like Generalized modus ponens, Forward and backward chaining, among others to show how first order logic can be used as a representational language for knowledge based systems.

REFERENCES

[1] Beckert B. (2004) "Introduction to Artificial Intelligence – First Order Logic (Logic, Deduction, Knowledge Representation).

[2] Burgard W., Nebel B., and Riedmiller M. (2011). "Foundations of Artificial Intelligence - Predicate Logic".

[3] Dillig I. (2012) "Automated Logical Reasoning – Lecture 6: First Order Logic Syntax and semsntics"

[4] Finin, T., DesJardins M., Geyer-Schulz A. and Dyer C. (2011). "CS 63 – Logical Inference" PowerPoint Presentation.

[5] Harmelen, F. V., Lifschitz, V. and Porter, B. (2008) " Handbook Of Knowledge Representation" Elsevier, 2008. ISBN 978-0-444-52211-5

[6] Hauskrecht M. (2012). Lecture note on "Introduction to AI: Inferences in First Order Logic, Knowledge-based System".

[7] Kaneiwa K. (2004). "On The Semantics Of Classical First-Order Logic With Constructive Double Negation". National Institute of Informatics (NII) Technical Report

[8] Patel-Schneider P.F. (1985) "A Decidable First-Order Logic for Knowledge Representation" IJCAI, Page 455-458, Morgan Kaufmann (1885).

[9] Russell, S. and Norvig, P. (2003). " Artificial Intelligence". *A modern Approach. Second Edition*, 2003.

[10] Smith R.G.(1985). "Knowledge-Based Systems: Concepts, Techniques, Examples". *Canadian High Technology Show* Ottawa, ON, Canada, May 8, 1985 (half-day course).

Author's Brief

Mrs Ayorinde Ibiyinka T. is a lecturer at the department of Computer Science, University of Ibadan. She holds a Bachelor of Technology (B.Tech.) degree in computer science from the Federal University of Technology, Akure, Ondo State in 1995 and Master of Science (M.Sc.) degree (Computer Science) from the University of Ibadan in 2005. She is presently a doctoral student in the department of computer science, University of Ibadan. Her research interests include: Knowledge Representation, Ontology, Software Engineering and Data Mining. She can be reached on phone number: +2348035289814 and email address: temiayorinde@yahoo.com.

Babatunde O. Akinkunmi is a member of the academic staff at the Dept of Computer Science University of Ibadan. He has authored over twenty five research articles in computer science. His research interests include Knowledge Representation, Formal Ontologies and Software Engineering. Mrs Ayorinde is under his direct supervision for her doctorate degree.