Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

An Automated Theorem Proving in First-Order Predicate Calculus using Resolution

Moko Anasuodei
Department of Computer Science and Informatics,
Faculty of Science, Federal University Otuoke, PMB 126,
Yenagoa, Bayelsa State

James Godbless Tamaraebi
Department of Computer Science and Informatics,
Faculty of Science, Federal University Otuoke, PMB 126
Yenagoa, Bayelsa State.

***Abstract*— A key concern of Automated Theorem Proving (ATP) research is the development and use of systems that derives conclusions following inevitably from facts. This capability lies at the heart of many important computational tasks. The project's focus was on the design and implementation of a program for automated theorem proving in first-order logic using the resolution proof procedure for first-order logic. The justification for implementing such a program using logic is that, proofs derived by humans in natural languages are ambiguous and vague but with logic the ambiguity can be eliminated. The process model used in this project is a hybrid of the "waterfall" model and the evolutionary development model. The program was implemented using the Python programming language. After rigorous analysis and testing, proofs produced by the automated theorem prover were more concise, less ambiguous and conformed more to the principles of sound reasoning compared to proofs produced by humans.**

Keywords—Theorem; first-order; logic; reasoning; conjecture;

I. INTRODUCTION

Automated Theorem Proving is the use of computer programs to prove or disprove mathematical, non-mathematical theorems and logical statements. Such statements can express properties of hardware or software systems or facts about the world that are relevant for applications such as natural language processing and planning. Automated theorem proving (ATP) systems are used in a wide variety of domains. For example, a mathematician can prove conjecture that "groups of order two" are commutative, from the axioms of group theory, a management consultant can formulate axioms that describe how organizations grow and interact, and from those axioms prove that organizational death rates decrease with age, a hardware developer can validate the design of a circuit by proving a conjecture that describes a circuit's performance, given axioms that describes the circuit itself; or a student can formulate the jumbled faces of a Rubik's cube as a conjecture and prove, from axioms that describe legal changes to this cube's configuration, that the cube can be rearranged to the solution state. All these are tasks that can be performed by an automated theorem proving system, given an appropriate formulation of the problem as either axioms, hypothesis, or a conjecture (Sutcliffe, 2012). Theorem provers are used for software and hardware verification, information management, combinatorial reasoning, and more. They are also the most powerful means of proof automation in interactive proof assistants. In most applications, the theorem checked by a theorem prover is generated by an external software tool and not given by a human.

"Automated theorem proving is concerned with the task of automating mathematical (or logical) reasoning. Proofs of mathematical theorems that are performed by a computer program, analogously to the way arithmetical problems are solved by a calculator." (Harrison, 2009).

The proofs produced by ATP systems describe how and why the conjecture follows from the axioms and hypotheses in a manner that can be understood and agreed upon by everyone including other computer programs. The proof outputted may not only be a convincing argument that the conjecture is a logical consequence of the axioms and hypotheses but also describes a process that maybe implemented to solve some problem. For example, in the Rubik's cube example mentioned above, the proof would describe the sequence of moves that need to be made to solve the puzzle (Sutcliffe, 2012). Algorithms of automated theorem proving are implemented in computer programs called theorem provers. A theorem prover takes a logical conjecture as input and tries to either construct its proof or demonstrate that the conjecture is invalid. Theorem provers can be classified by the logic they support. Propositional, first-order and higher-order logic are among the logics that received the most attention in automated theorem proving. Reasoning in propositional logic, i.e. solving the problem of SAT (propositional satisfiability), is implemented in SAT solvers, such as Lingeling (Biere et al, 2010) and Minisat (Sorensson & Een, 2005). Solving the problem of satisfiability modulo theory (SMT) is implemented in SMT solvers, such as Z3 (Mendonça de Moura & Bjørner, 2008) and CVC4 (Barrett et al, 2011). Reasoning in first-order logic is implemented in first-order theorem provers, such as Vampire (Kovács & Voronkov, 2013), E (Schulz, 2013) and iProver (Korovin, 2008). Reasoning in higher-order logic is implemented in higher-order theorem provers, such as Satallax (Brown, 2012) and Leo-II (Benzmüller et al, 2008). The more expressive a logic is, the harder it is to reason in it. For example, satisfiability of a propositional problem can always be established, albeit possibly at a high computational cost. Modern SAT solvers implement elaborate algorithms that guarantee good performance characteristics in the average case. In contrast, satisfiability or un-satisfiability of a first-order problem cannot be in general established by any algorithm (Kotelnikov, 2016). Implementers of first-order provers face the challenge of making the provers succeed on as many real-world problems as possible.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

II. LOGICAL BACKGROUND

Logic is concerned mainly with two concepts: truth and provability (Gallier, 2003). These concepts have been investigated extensively for centuries, by philosophers, linguists, and mathematicians. Every logical system consists of a language used to write statements also called propositions or formulae. Normally, when one writes a formula, one has some intended interpretation of the formula in mind. For example, a formula may assert a true property about the natural numbers, or some property that must be true in a data base. This implies that a formula has a well-defined meaning or semantics. But how do we define this meaning precisely? In logic, we usually define the meaning of a formula as its truth value. A formula can be either true (or valid) or false. (Gallier, 2003). A logical language can be used in different ways. For instance, a language can be used as a deduction system (or proof system); that is, to construct proofs or refutations. This use of a logical language is called proof theory (Gallier, 2003). In this case, a set of facts called axioms and a set of deduction rules (inference rules) are given, and the object is to determine which facts follow from the axioms and the rules of inference. When using logic as a proof system, one is not concerned with the meaning of the statements that are manipulated, but with the arrangement of these statements, and specifically, whether proofs or refutations can be constructed. In this sense, statements in the language are viewed as cold facts, and the manipulations involved are purely mechanical, to the point that they could be carried out by a computer (Gallier, 2003). This does not mean that finding a proof for a statement does not require creativity, but that the interpretation of the statements is irrelevant. This use of logic is like game playing. Certain facts and rules are given, and it is assumed that the players are perfect, in the sense that they always obey the rules (Gallier, 2003). Occasionally, it may happen that following the rules leads to inconsistencies, in which case it may be necessary to revise the rules. However, the statements expressed in a logical language often have an intended meaning. The second use of a formal language is for expressing statements that receive a meaning when they are given what is called an interpretation (Gallier, 2003). In this case, the language of logic is used to formalize properties of structures and determine when a statement is true of a structure. This use of a logical language is called model theory (Gallier, 2003). To summarize, a logical language has a certain syntax, and the meaning, or semantics, of statements expressed in this language is given by an interpretation in a structure. Given a logical language and its semantics, one usually has one or more proof systems for this logical system.

A. *First-order Logic*

Every logic comprises a (formal) language for making statements about objects and reasoning about properties of these objects (Harrison, 2009). One might ask why a special language is needed at all, and why English (or any other natural language) is not adequate for carrying out logical reasoning. The first reason is that English (and any natural language in general) is such a rich language that it cannot be formally described. The second reason, which is even more serious, is that the meaning of an English sentence can be ambiguous, subject to different interpretations depending on the context and implicit assumptions. If the object of our study is to carry out precise rigorous arguments about assertions and proofs, a precise language whose syntax can be completely described in a few simple rules and whose semantics can be defined unambiguously is required. Another important factor is conciseness. Natural languages tend to be verbose, and even simple mathematical statements become exceedingly long (and unclear) when expressed in them (Gallier, 2003).

Definition 2.1.1 (Fitting, 1996): A first-order language is determined by specifying:

1. A finite or countable set **R** of relation symbols, or predicate symbols, each of which has a positive integer associated with it. If P ϵ **R** has the integer n associated with it, we say P is an n-place relation symbol.
2. A finite or countable set **F** of function symbols, each of which has a positive integer associated with it. If *f* ϵ **F** has the integer n associated with it, we say f is an n-place function symbol.
3. A finite or countable set **C** of constant symbols.

We use the notation L**(R, F, C)** for the first-order language determined by **R, F**, and **C**. If there is no danger of confusion, we may abbreviate L**(R, F, C)** to just L.

Definition 2.1.2 (Fitting, 1996): The family of terms of L**(R, F, C)** is the smallest set meeting the conditions:

1. Any variable is a term of L**(R, F, C).**
2. Any constant symbol (member of **C**) is a term of L**(R, F, C).**
3. If *f* is an n-place function symbol (member of **F**) and $t_1, \ldots, t_n$ are terms of L**(R, F, C),** then $f(t_1, \ldots, t_n)$ is a term of L**(R, F, C).**

*A term is closed if it contains no variables.

*Definition 2.1.3 (*Fitting, 1996): An atomic formula of *L***(R, F, C)** is any string of the form $R(t_1, \ldots, t_n)$ where *R* is an *n*-place relation symbol (member of *R*) and $t_1, \ldots, t_n$ are terms of *L***(R, F, C)** ; also T and $\perp$ are taken to be atomic formulas of *L***(R, F, C)**

Definition 2.1.4 (Fitting, 1996): The family of formulas of *L***(R, F,C)**is the smallest set meeting the following conditions:

1. Any atomic formula of *L***(R, F, C)** is a formula of *L***(R, F, C).**
2. If A is a formula of *L***(R, F, C)**, so is ¬A.
3. For a binary connective o, if A and B are formulas of *L***(R, F, C)**, so is (A o B).
4. If A is a formula of *L***(R, F, C)**and x is a variable, then (∀x)A and (∃x)A are formulas of *L***(R, F, C).**

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

Sentence — AtomicSentence

| Sentence Connective Sentence

| Quantifier Variable, . . . Sentence

| ¬ Sentence

| (Sentence)

AtomicSentence — Predicate(Term, . . .) Term = Term

Term — Function(Term, . . .)

| Constant

| Variable Connective — ⇔ | ⇒ | ¬ | ∧ | ∨

Quantifier — ∀ | ∃

Constant — A \ X\ John … Variable — a | x | s …

Predicate — Before \ HasColor \ Raining \ …

Function — Mother \ LeftLegOf \ …

Figure 1: The syntax of first-order logic (with equality) in BNF (Backus-Naur Form). (Norvig and Russel, 2010).

*Definition 2.1.5 (*Fitting, 1996): A model for the first-order language *L*(**R, F, C)** is a pair **M** = (**D, I**) where:

1. **D** is a nonempty set, called the domain of **M.**
2. **I** is a mapping, called an interpretation that associates:

To every constant symbol **c** ϵ **C**, some member c^I ϵ **D.**

To every ***n***-place function symbol f ϵ F, some ***n***-ary function f^I: $D^n \longrightarrow D$.

To every *n*-place relation symbol P ϵ **R**, some *n*-ary relation $P^I \subset \mathbf{D}^n$.

Definition 2.1.6 (Fitting, 1996): An assignment in a model **M** = (**D, I)** is a mapping **A** from the set of variables to the set **D**. We denote the image of the variable *v* under an assignment **A** by v^A.

B. Resolution

Robinson (1965) introduced a simple calculus for automated theorem proving based on showing inconsistency of the negation of a conjecture and associated axioms using resolution. The problem of theorem proving is reduced to that of searching for the empty clause via a simple clause generation inference process. The starting point for a resolution-based proof is a conjecture expressed as a set of quantifier-free clauses in conjunctive normal form (CNF). Quantifier-free means that there are no existential quantifiers and all variables are assumed to be universally quantified so that P(x, y) for example is taken to be $\forall_x \forall_y P(x, y)$.

Additionally, as the universal quantifier is distributive over the clauses in CNF, each variable may be considered local to the clause that it is in, which allows the renaming of variables to avoid clashes when combining clauses. For example, for two clauses φ(x) and ψ(x), ∀x (φ(x) ∧ ψ(x)) is equivalent to ∀x φ(x) ∧ ∀y ψ(y) where ψ(y) is ψ(x) with all occurrences of x replaced by y. The requirement for the conjecture to be in quantifier free CNF is not a restriction. Any well-formed formula in first order logic may be converted to CNF. Though a naive conversion to CNF may lead to an exponential increase in the size of the formula, there are efficient algorithms that perform the conversion (by the suitable introduction of new predicate symbols). Additionally, Heijenoort (1967) showed that existential quantifiers may be replaced by functions (named Skolem functions) whilst maintaining consistency (or, more importantly, inconsistency). That is any set of clauses in which this Skolemisation process has been used to eliminate quantifiers will be consistent if and only if the original set of clauses is consistent. Nonnengart and Weidenback (2001) discussed techniques for converting general first order logic into a suitable Skolemised CNF. The procedure for proving a conjecture in a theorem is to replace the conjecture with its negation, combine this with the axioms and place in CNF to form a set of clauses. The clauses are then combined in pairs using resolution to deduce new clauses. If the empty clause is reached, then inconsistency has been proved and the original conjecture is a theorem. For ground clauses (clauses without free variables) the process of generating new clauses will saturate so that after a point no new clause are generated. If this happens without the empty clause being reached, then the clauses are consistent (there is a model which satisfies them). With non-ground clauses (i.e. clauses containing variables) resolution is combined with unification. Unification is the process of substituting terms for variables to make two terms equal. This process is allowable because all terms, including variables, represent elements of a single domain and since variables are implicitly universally quantified, they will range over all values.

For non-ground clauses, the process of resolution is not guaranteed to saturate as this would violate Church's theorem (Robinson 1965) but for the inconsistent case the empty clause will eventually be found (provided factoring is also carried out), so the process is semi-decidable.

C. Proving methodologies

Automated theorem provers can determine if a well-formed formula (wff) A is a theorem, in some logic, using one of the following approaches:

i. The Refutation Approach

This approach is based on the following concept: A wff A is a theorem if and only if it is logically valid; A is logically valid if and only if A is unsatisfiable. Instead of showing that A is a theorem we may show that ¬ A is unsatisfiable. This can be done by refutation. Refutation is a process in which the wff ¬ A is added to the axioms of the theory, and then inference rules are applied to derive some contradiction. This contradiction indicates that ¬ *A* is unsatisfiable, which proves that *A* is a theorem. Many theorem provers use refutation as their proving approach. They use various versions of resolution as their inference rules. An example of

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

these provers is OTTER (Wos *et al*, 1992). Prolog is also considered as a refutation theorem prover (Manna and Waldinge, 1980).

ii. The Direct Proving Approach

In this approach A is proven as a theorem by deriving it from the axioms with the application of inference rules. These proofs can be done by forward or backward chaining. In forward chaining, inference rules are applied to the axioms to derive new theorems. These new theorems are used to derive additional theorems. This process continues until either A is derived (in which case it is a theorem), or until the time (or space) limit is exceeded. In backward chaining, the *wff* A is reduced to simpler *wffs*. These *wffs* are then reduced further and further. This process continues until all the new *wffs* are reduced to axioms (in which case A is a theorem), or until the time (or space) limit is exceeded. Some of the provers that use this approach are: the Boyer-Moore prover (Boyer and Moore, 1988), IMPLY (Bledsoe, 1983), and LCF (Paulson, 1987).

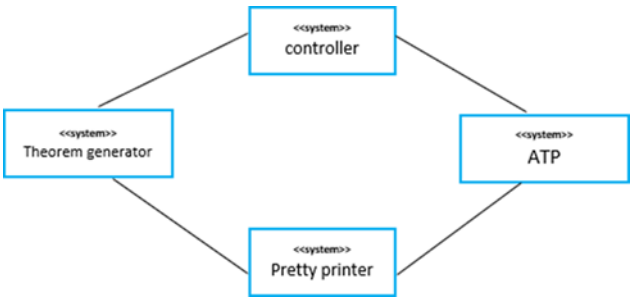

Figure 2: ATP Context diagram

D. Applications Fields of ATP

ATP is a rapidly growing field in computer science. There are many research areas that are currently under investigation (Maghrabi, 2015). These areas vary between improving the status of the field, e.g. enhancing some of the existing inference rules, and/or developing new techniques that will produce better results. Another aspect of current research is completeness versus effectiveness. Completeness refers to the process of developing theorem provers which can solve problems from different areas, i.e., general, although inefficiently, while effectiveness refers to the process of developing theorem provers which can solve specific problems, i.e., special-purpose, but efficiently. There are several areas for which theorem provers have proved to be good and productive tools.

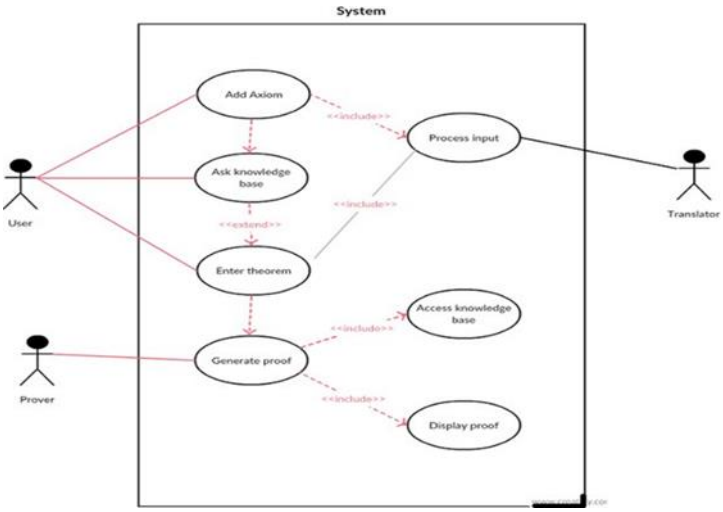

Figure 3: Use case diagram of the ATP

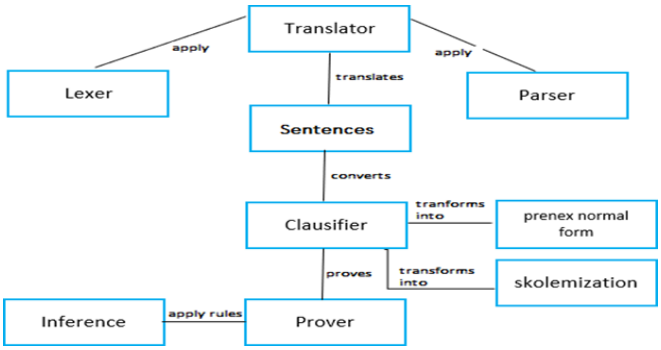

Figure 4: ATP class diagram

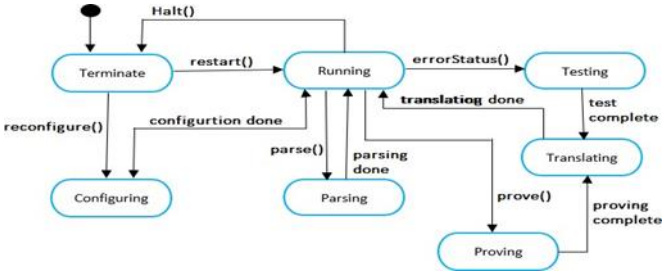

Figure 5: state diagram of ATP

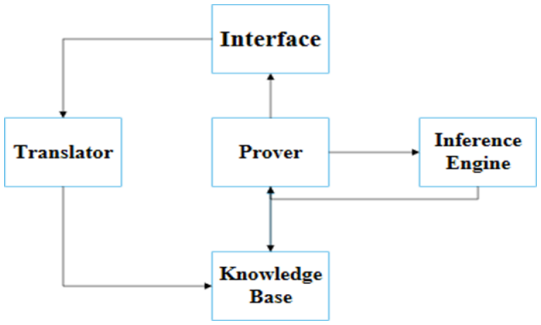

Figure 6: ATP Architecture

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

III. SYSTEM IMPLEMENTATION

This section discusses the program implementation decisions and issues, and the various functionalities step by step under each module with their outputs.

A. Translator

This is a class in the program responsible for carrying out the necessary translations needed by the proving engine to generate proofs inn first-order logic.

1. *Lexical Analysis*
   Lexical analysis is the first phase of the ATP. It takes the user inputted first-order formulae from the screen. Characters in user input are classified into to broad groups: Spaces, identifiers and symbols. The lexical analyzer breaks these formulas into a series of tokens by removing any whitespace. If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the formula, checks for legal tokens, and passes the data to the syntax analyzer when it demands.

2. *Syntax Analysis*
   It takes the token produced by lexical analyzer as input and generates a parse tree (or syntax tree). In this phase, token arrangements are checked against the syntax of fist-order logic, i.e., the parser checks if the expression made by the tokens is syntactically correct.

B. Clausifier

This is a class in the program responsible for carrying out the necessary translations needed by the proving engine to apply the resolution algorithm.

1. *Conjunctive prenex normal form*
   Definition: A formula ϕ is in prenex normal form (PNF) if it has the form $Q_1x_1 \cdots Q_nx_n\psi$
   where each Qi is a quantifier (either $\exists$ or $\forall$) and ψ is a quantifier-free first-order formula. Moreover, if ψ is a conjunction of disjunctions of literals (atomic or negated atomic formulas), then ϕ is in conjunctive prenex normal form.

   So, a formula is in prenex normal form if all its quantifiers are in front

   Example: $\forall_y\exists_x(f(x)= y)$ is in PNF, and $\neg\forall_x\exists_yP(x, y, z)$ and $\exists_x\forall_y \neg P(x, y, z) \wedge \forall_x\exists_yQ(x, y,z)$ are not.

   Theorem: For any formula of first-order logic, there exists an equivalent formula in conjunctive prenex normal form.

   Proof: Let ϕ be an arbitrary formula. First, we show that there exists an equivalent formula ϕ' in prenex normal form. We prove this by induction on the complexity of ϕ

   If ϕ is atomic, then ϕ is already in PNF, so we can just let ϕ' be ϕ. Suppose ψ and θ are formulas and there exist ψ' and θ' in PNF such that $\psi \equiv \psi'$ and $\theta \equiv \theta'$. Clearly, if $\phi \equiv \psi$ then we can let ϕ' be ψ'. To complete the induction step, we must consider three cases corresponding to $\neg$, $\wedge$, and $\exists$. First, suppose ϕ is the formula $\neg\psi$. Then $\phi \equiv \neg\psi'$. Since ψ' is in PNF, ψ' has the form $Q_1x_1 \cdots Q_mx_m\psi0$ for some quantifier-free formula $\psi0$ and quantifiers $Q_1 \cdots Q_m$. So $\phi \equiv \neg Q_1x_1 \cdots Q_mx_m\psi0$. This is equivalent to $Q_1x_1 \cdots Q_mx_m\neg\psi0$

   where {Qi, Qi1} = {$\exists$, $\forall$}. This formula is in PNF, and so it may serve as ϕ'.

2. *Skolem normal form*
   Definition 3.2: A formula is in Skolem normal form (SNF), if it is universal and in conjunctive prenex normal form.
   Given any formula ϕ of first-order logic we define a formula ϕS that is in SNF. ϕ is satisfiable if and only if ϕS is satisfiable. The formula ϕS is called a Skolemization of ϕ. The following is a step-by-step procedure for finding ϕS.
   - First, we find a formula ϕ' in conjunctive prenex normal form such that $\phi' \equiv \phi$. So ϕ' is $Q_1x_1 \cdots Q_mx_m\phi0(x_1, \ldots, x_m)$ for some quantifier-free formula $\phi0$ and quantifiers $Q_1 \cdots Qm$.
   - If each Qi is $\forall$, then ϕ' is a universal formula. In this case let ϕ be ϕ'.
   - Otherwise, ϕ' has existential quantifiers. In this case we define a formula s(ϕ') that has fewer existential quantifiers than ϕ'. (So, if ϕ' has just one existential quantifier, then s(ϕ') is universal.) Let *i* be least such that Q_i is $\exists$.

   If $i = 1$, then ϕ' is $\exists x_1Q_2x_2 \cdots Q_1x_1 \cdots Q_mx_m\phi(x_1, \ldots, x_m)$.

   Let s(ϕ') be $x_1Q_2x_2 \cdots Q_1x_1 \cdots Q_mx_m\phi o(c, x_1, \ldots, x_m)$.where c is a constant symbol that does not occur in ϕ'.

   If I > 1, then ϕ' is $\forall x_1 \cdots \forall x_i{-}1\exists x_iQ_i{+}1x_i{+}1 \cdots Q_mx_m\phi(x_1, \ldots, x_m)$.

   Let s(ϕ') be the formula

   $\forall x_1 \cdots \forall x_i{-}1\exists x_iQ_i{+}1x_i{+}1 \cdots Q_mx_m$
   $\phi(x_1, \ldots, x_i{-}1, f(x_1, \ldots, x_i{-}1), xi{+}1, \ldots, x_m)$,

   where f is an $(i - 1)$-ary function symbol that does not occur in ϕ'.
   So, if the first quantifier in ϕ' is $\exists$, we replace x1 with a new constant.

   And if the *ith* quantifier in ϕ' is $\exists$ and all previous quantifiers are $\forall$, replace xi with $f(x_1, \ldots, x_{i-1})$ where f is a new function symbol.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 02, February-2020

Since s(ϕ') has fewer existential quantifiers than ϕ', by repeating this process, we will eventually obtain the required universal formula ϕs. That is, ϕs is sn(ϕ') = s(s(s ⋯ s(ϕ'))) for some

CONCLUSION

With the ever-increasing desire of Mankind to automate both mundane and complex tasks using intelligent agents, artificial intelligence (involves the study and building of intelligent agents) has become a universal field where scientists in other fields are moving gradually into, to find the tools and vocabulary to systematize and automate the intellectual tasks on which they have been working all their lives. Also, Experts in artificial intelligence can choose to apply their techniques and principles to many areas of human intellectual endeavour.

Automated theorem proving techniques can be used by computer scientists to axiomatize structures and prove properties of programs working on these structures. Another recent and important role that logic plays in computer science, is its use as a programming language and as a model of computation.

ACKNOWLEDGMENT

All gratitude goes to God almighty for his infinite provisions, guidance and inspiration.

REFERENCES

[1] Armin Biere. Lingeling, p. p. (2010). FMV Report Series Technical Report.

[2] Ashutosh Gupta, L. K. (2014). Extensionality Crisis and Proving Identity. Proceedings of ATVA, volume 8837 of LNCS, 185–200.

[3] Bjørner., L. M. (2008). Z3: an efficient SMT solver. Proceedings of TACAS, volume 4963 of LNCS, 337–340.

[4] Bledsoe, W. (1983). "The UT Prover". Math Department Memo ATP-17B, University of Texas at Austin.

[5] Bloch, E. D. (2011). Proofs and Fundamentals: A First Course in Abstract Mathematics (2ed). New York: Springer .

[6] Christoph Benzmüller, L. P. (2008). LEO-II — A Cooperative Automatic Theorem Prover for Higher-Order Logic. Proceedings of IJCAR, volume 5195 of LNAI, 162–170.

[7] Church, A. (1936.). An unsolvable problem of elementary number theory. American Journal of Mathematics, 58(2):345–363.

[8] Clark Barrett, C. L. (2011). CVC4. Proceedings of CAV, 171–177.

[9] Doré, M. (2017). ELFE – An interactive theorem prover for undergraduate students. London: Imperial College London.

[10] Duffy, D. (1991.). Principles of Automated Theorem Proving. New York: John Wiley & Sons, .Een, N. S. (2005.). Minisat v1.13 – a SAT solver with conflict-clause minimization.

[11] SAT, 53. fitting, M. (1996). First-order logic and automated theorem proving (2ed). New York, NY.:Springer Inc.

[12] Gallier, J. H. (2003). Logic For Computer Science: Foundations of Automatic Theorem Proving. Philadelphia: University of Pennsylvania press.

[13] Ganzinger, L. B. (1994). Rewrite-based equational theorem proving with selection and simplification. Journal of Logic and Computation, 4(3):217–247.

[14] Geoff Sutcliffe, S. S. (2012). The TPTP Typed First-Order Form with Arithmetic.

[15] Proceedings of LPAR, volume 7180 of LNCS, 406–419.

[16] Harrison, J. (2009). Handbook of Practical Logic and Automated Reasoning. New York: Cambridge University Press.

[17] Hedman, S. (2006). A First Course in Logic: An introduction to model theory, proof theory, computability, and complexity. New york: Oxford University press.

[18] Heijenoort., J. v. (1967). From Frege to Godel a source book in mathematical logic 1879- 1931. New York: Harvard University Press.

[19] Korovin, K. (2008). iProver — An Instantiation-Based Theorem Prover for First-Order Logic (System Description). . Proceedings of IJCAR, 292–298.

[20] Kotelnikov, E. (2016). Automated Theorem Proving in a First-Order Logic with First Class Boolean Sort. Gothenburg: Author.

[21] Larry Wos, G. A. (1967). The Concept of Demodulation in Theorem Proving. J. ACM, 14(4):698–709.

[22] Maghrabi, T. (1992). "The Synthesis Tableau: A Deductive Framework for Program Synthesis Based on Sequent Calculus",., August 1992. PhD Thesis, Arizona State University, Tempe, Arizona, U.S.A.

[23] Maghrabi, T. (1997). Automated theorem proving: An overview. Handbook of automated reasoning.

[24] Moore, R. B. (1988.). A Computational Logic Handbook. Boston: Academic Press,. Overbeek, E. 1. (1984). "The Automated Reasoning System ITP" A1984. NL-84-27.

[25] Argonne, 11: Argonne National Laboratory.

[26] Paskevich, J. C. (2013). TFF1: The TPTP Typed First-Order Form with Rank-1 Polymorphism. roceedings of CADE, volume 7898 of LNCS, 414–420.

[27] Paulson, L. (1987). Logic and Computation: Interactive Proofs with Cambridge LCF. UK: Cambridge University Press.

[28] Putnam., M. D. (1960). A computing procedure for quantification theory. Journal of the ACM (JACM), 7(3)201–215.

[29] Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. J. ACM,, 12(1):23–41.

[30] Rubio., R. N. (2001). Paramodulation-Based Theorem Proving. In A. Robinson and A. Voronkov, editors, Handbook of Automated Reasoning, volume I, chapter 7, pages 371–443. Elsevier Science.

[31] Schulz, S. (2013). System Description: E 1.8. Proceedings of LPAR, volume 8312 of LNCS, 735–743.

[32] Simon., A. N. (1956.). The logic theory machine–A complex information processing system, Information Theory. J. ACM., 2(3):61–79,.

[33] Sommerville, I. (2011). Software engineering (9th ed.). New York: Pearson.

[34] Sutcliffe, G. (2009). The TPTP Problem Library and Associated Infrastructure. Journal of Automated Reasoning, 43(4):337–362,.

[35] Suttner., G. S. (2006). The State of CASC. AI Communications. Proceedings of LPAR, 19(1):35–48.

[36] Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungs problem. Journal of Math, 58(345-363):5.

[37] Voronkov, L. K. (2013). First-Order Theorem Proving and Vampire. Proceedings of CAV, volume 8044 of LNCS, 1–35.

[38] Waldinger, Z. M. (1980). "A Deductive Approach to Program Synthesis". ACM Transactions on Programming Languages and Systems, pp. 90-121.

[39] Weidenback., A. N. (2001). Computing small clause normal forms.. In Handbook of Automated Reasoning, volume I, pages 335–367. Elsevier Science and MIT Press,.

[40] Wos., G. R. (1983). Paramodulation and Theorem-Proving in First-Order Theories with Equality. Automation of Reasoning, 2:298–313.